Evolutionary Algorithms for Optical Network Design: A Genetic-algorithm/Heuristic Hybrid Approach

Mark C. Sinclair

A thesis submitted for the degree of Ph.D.

Department of Electronic Systems Engineering

University of Essex

July, 2001

Summary

Multi-wavelength all-optical transport networks can potentially provide the huge bandwidths necessary for broadband services. However, they present a challenging range of difficult network design and optimisation problems. Recently, evolutionary algorithms (EAs) have attracted growing interest for use on similar complex problems in many fields, including telecommunications. However, while EAs are good general-purpose search and optimisation procedures, they are almost never the best approach for any specific problem. A powerful alternative is the development of problem-specific EAs—genetic-algorithm (GA)/heuristic hybrids. These employ both problem-specific encoding and operators, combining the best existing heuristics for the problem within an overall GA framework. Moreover, operator-probability adaptation is used so that the most appropriate blend of operators is applied at each stage in a run.

This thesis describes the application of GA/heuristic hybrids to two of the main problems in the design of multi-wavelength all-optical transport networks: mesh network topology design, and wavelength-path routing, fibre choice and wavelength allocation. It starts with an overview of EAs, a tutorial on GAs, a near-exhaustive survey of evolutionary telecommunications, and surveys of both GA/heuristic hybrids and operator-probability adaptation. A software toolset for optical network optimisation, modelling and design (NOMaD), implementing the author's GA/heuristic hybrid approach, is described, including the design-assessment metrics, overall GA framework, genetic operators and the operator-probability adaptation mechanism. Experimental results for both mesh network topology design, and wavelength-path routing, fibre and wavelength allocation are reported. An investigation into NOMaD's operator-probability adaptation mechanism is described. Finally, the successes and limitations of the research are discussed, and numerous suggestions for further work made. Overall, the hybrid approach is shown to provide excellent solutions, of comparable or higher quality than those obtained with either heuristics or simple GAs alone, and requiring less computational effort than GAs.

Contents

Abbreviations

ACO	ant colony optimisation
ACTS	Advanced Communications Technologies and Services
ADM	add drop multiplexer
BON	Business Object Notation
CS	classifier system
DCR	different common ring
DNA	deoxyribose nucleic acid
DQDB	Dual Queue Dual Bus MAN (IEEE 802.6)
EA	evolutionary algorithm
EC	evolutionary computation
EH	evolutionary hardware
EON	European Optical Network
EP	evolutionary programming
ES	evolution strategy
FSM	finite state machine
FTEL	Fujitsu Telecommunications Europe Ltd.
GA	genetic algorithm
GP	genetic programming
GUI	graphical user interface
id	identification number
LAN	local area network
LCS	learning classifier system
MAN	metropolitan area network
MCA	microcanonical annealing

NGOS	network grade-of-service
NOMaD	optical network optimisation, modelling and design
NP	non-polynomial
NWR	network wavelength requirement
OOA&D	object-oriented analysis and design
OPEN	Optical Pan-European Network
OXC	optical cross connect
PON	passive optical network
RCA	random cost algorithm
SA	simulated annealing
SGA	Goldberg's Simple Genetic Algorithm
SWP	sparse wavelength path
TS	tabu search
TSP	travelling salesman problem
VWP	virtual wavelength path
WOTAN	Wavelength-agile Optical Transport and Access Network
WP	wavelength path

List of Figures

List of Tables

Chapter 1

Introduction

In the beginning . . .

Genesis 1:1a, NIV

Telecommunications is a vital and growing area, important not only in its own right, but also for the service it provides to other areas of human endeavour. Moreover, there currently seems to be a demand for an ever-expanding set of telecommunication services of ever-increasing bandwidth. One particular technology that has the potential to provide the huge bandwidths necessary if such broadband services are to be widely adopted is multi-wavelength all-optical transport networks [1]. However, the development of such networks presents a challenging range of difficult design and optimisation problems.

There are many different potential solution approaches to these, including mathematical programming, problem-specific heuristics, and modern heuristic search methods: for example, evolutionary algorithms (EAs) [2–4], simulated annealing (SA) [5] and tabu search (TS) [6, 7]. In particular, EAs have attracted growing interest in recent years for use on similar complex problems in many fields, including telecommunications. Applications in this latter area have included network design, call routing, frequency assignment, wavelength allocation, capacity planning, admission control, network management, and many others [8].

Now, EAs exist in many forms, although the distinction between these is blurring as practitioners adopt the best of one another's ideas. The once distinct historical areas include: genetic algorithms (GAs), evolutionary programming (EP), evolution strategies (ESs), genetic programming (GP) and classifier systems (CSs), amongst others [4].

However, while EAs are good general-purpose search and optimisation procedures, they are almost never the best approach for any specific problem [9]. A powerful alternative is the development of problem-specific EAs—GA/heuristic hybrids. These adopt a problem-specific encoding and employ problem-specific operators that together combine the best existing heuristics for the problem within an overall GA framework.

This thesis describes the application of GA/heuristic hybrids to two of the main problems in the design of multi-wavelength all-optical transport networks: mesh network topology design, and wavelength-path (WP) routing, fibre choice and wavelength allocation. The hybrid approach is shown to provide excellent solutions, of comparable or higher quality than those obtained with either heuristics or simple GAs alone, and requiring less computational effort than GAs.

1.1 Assumed Background Knowledge

In writing this thesis, certain background knowledge has been assumed on the part of the reader, rather than being provided in the form of introductory chapter(s).

Familiarity with EAs in general, and bit-string GAs in particular, has been assumed, although a brief tutorial is provided in §2.2. A noteworthy introduction to GAs and CSs has been provided by Goldberg [2]. A more recent and slightly less introductory source by Mitchell [3], and a definitive collection of important and historical papers in the field, with editorial context, by David Fogel [4], represent alternative points of entry.

Some understanding of both network design and multi-wavelength optical optical networks is expected. An excellent introduction to wide-area network design is provided by Cahn [10], and to telecommunications network design algorithms by Kershenbaum [11,12]. For a tutorial introduction to multi-wavelength optical networks, the reader is referred to either Mukherjee [1] or Ramaswami & Sivaraja [13]. In addition, as the focus in the thesis is on EAs in general, and the GA/heuristic hybrid approach in particular, rather than on the application domain, surveys of non-EA work on topology design, routing and wavelength allocation have not been provided. However, for the interested reader, the 1977 survey of 'classical' work on topology design by Boorstyn & Frank [14] is a good starting point, complemented by the more recent books by Kershenbaum [11] and Cahn [10]. Moreover, routing and wavelength allocation are well covered in the books by Mukherjee [1] and Ramaswami & Sivaraja [13].

Knowledge of object-oriented analysis and design (OOA&D), sufficient to appreciate Booch class diagrams, and some understanding of the C++ programming language have also been assumed. A good introduction to OOA&D, and a description of the Booch Notation used to record designs (particularly in Chapters 3 and 4, and Appendix E), is given by Booch [15]. Information on C++ can be found in numerous works, including those by Stroustrup [16] and Lippman [17].

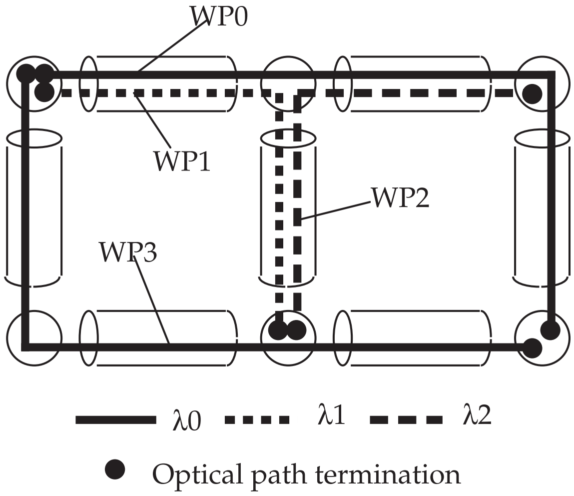
1.2 Multi-wavelength Optical Networks

In selecting the design of multi-wavelength all-optical transport networks as the application area for this thesis, it was not the author's intent to argue for the adoption of a particular network architecture. Rather the intent was to select a promising architecture, with potential for adoption in future telecommunications transport networks, that would offer sufficiently challenging design problems for the application of the GA/heuristic hybrid approach.

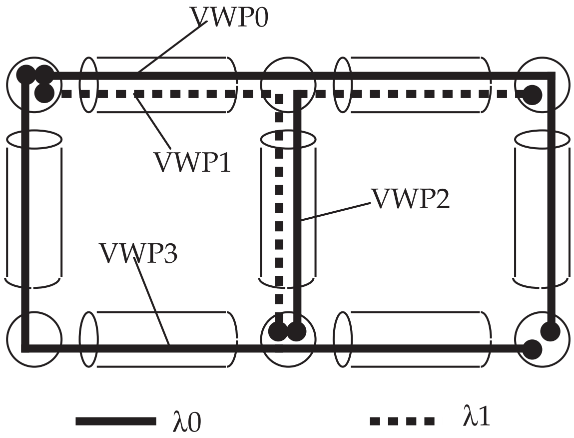
Indeed, many different kinds of optical network have been developed [1], including both broadcast (local) and wavelength-routed (wide-area) optical networks. The focus here is on the latter, over national or international distances, but even given that, numerous choices remain. There are a wide variety of potential architectures. Should the network use regular topologies (*e.g.* torus, hypercube, *etc.*), arbitrary mesh or (multiple) rings? Should it be all-optical or partitioned into optical islands separated by electronic regeneration? Which layering structure should be adopted: 'flat' or hierarchical? Should multiple fibre cables be used, multiple wavelengths utilised in each fibre, or both? If multiple wavelengths are employed, should the wavelength channels be routed on one wavelength end-to-end (WP routing), or avoid wavelength blocking by using (possibly expensive) wavelength conversion in the nodes to access any free wavelength on a fibre (virtual-wavelength-path routing)?

For example, Fig. 1.1(a) shows a simple 6-node, 7-link network, with single-fibre cables, on which four wavelength channels have been placed using WP routing. Three different wavelengths have had to be used to ensure not only that each channel uses only one wavelength end-to-end, but also that on any given fibre, no wavelength is used by more than one channel. In contrast, Fig. 1.1(b) shows the same network and traffic requirements, but due to the use of virtual-wavelength-path (VWP) routing, and hence wavelength conversion, only two wavelengths are needed.

In this thesis, the choice is a flat, all-optical, arbitrary-mesh multiple-fibre-link topology, with WP routing—an architecture that is expected to have its place in future ultra-high-capacity transport networks [18]. While hierarchical networks have many notable benefits, particularly in simplifying network management [19], the less-constrained flat architecture still has potential. Similarly, although more modular networks (*e.g.* multi-ring [20–24]) have several desirable qualities,



(a) Wavelength path



(b) Virtual wavelength path

Figure 1.1: Wavelength-routing schemes

the arbitrary-mesh topology may well form the core of future transport networks. Also, whereas the use of wavelength converters can result in greater flexibility under failure conditions, careful WP routing remains competitive during normal operation. Moreover, in all three areas, the choices adopted lead to more potential networks and hence larger search spaces during both topology design and network allocation, particularly in employing WP rather than VWP routing. Overall, then, these problems will provide a suitable testbed for development of problem-specific EAs.

1.3 Choice of Evolutionary Algorithms

Design and optimisation problems such as those investigated here often require solution algorithms that are non-polynomial (NP) in their time complexity with respect to growth in problem size [25]. Non-trivial instances of such problems are difficult to tackle with the exact methods of mathematical programming, due to their large time and computer memory requirements [20, 23]. Consequently, researchers frequently resort to either problem-specific heuristics or modern heuristic search techniques. Of the latter, EAs are enjoying growing popularity amongst practitioners, with more than 2,000 papers published each year [26]. However, the focus of this thesis is on combining the best aspects of both problem-specific heuristics and EAs into a GA/heuristic hybrid approach.

In selecting EAs as the solution approach adopted, it was not the author's intent to claim their superiority over other approaches, such as mathematical programming or other modern heuristic search techniques, including SA [5] and TS [6, 7]. Consequently, no comparison with these alternative approaches will be provided. Rather EAs were chosen primarily on the basis of the author's prior expertise and enthusiasm for the area, combined with their growing popularity amongst practitioners. Nevertheless, despite the absence of comparisons with other methods, the thesis will allow detailed study into how to apply EAs in general, and the GA/heuristic hybrid approach in particular, to the design of multi-wavelength all-optical transport networks. Moreover, it is anticipated that many of the outcomes will also be useful in the application of the GA/heuristic hybrid approach to other domains.

1.4 Genetic-algorithm/Heuristic Hybrid Approach

The decision to focus on problem-specific EAs was motivated by limitations encountered by the author in applying simple GAs to network design problems. These were primarily difficulties in developing suitable bit-string encodings and poor results in comparison with problem-specific heuristics.

For example, in [27] the author applied a simple GA to design routing tables for a circuit-switched network subject to link failures. The aim was to develop tables with the lowest network grade-of-service (NGOS), *i.e.* blocking caused by both network congestion and link failures. Designing an efficient bit-string encoding of the routing tables proved difficult. The encoding adopted included redundancy[1], as well as having a relatively poor mapping of the building blocks of the problem, which were doubtless dependent on the two-dimensional nature of the routing table itself. On a trial network, the GA gave an NGOS of only 4.89% after 11.6 hours compared to an NGOS of 2.43% obtained in just 23 seconds by the author's earlier heuristic [28]. Similarly, with Tan [29], the author had difficulty in devising a successful bit-string encoding for WP routing in an 11-node network with single-fibre links. Rather, the most successful approach (Strategy 3) combined a bit-string encoding of the routes with a wavelength-allocation heuristic. More recently, with Moore [30], the author returned to routing for circuit-switched networks, this time aiming for survivability of military communications. A simple bit-string GA was again adopted, and while markedly superior results were obtained compared to the existing table-design heuristic, even better results followed from using an iterative stochastic hill climber employing a problem-specific operator.

In contrast, the GA/heuristic hybrid approach adopted in this thesis, inspired by Davis' work [9], employs an object-oriented network model, an adaptive overall GA framework, and heuristic operators. Such an object-oriented network model would appear to be a natural representation for network problems [9,16], rather than a linear bit-string encoding. Thus, in this approach, the object-oriented network models are themselves the structures undergoing adaptation—a highly-structured and very problem-specific representation that requires no decoding or interpretation. Using heuristic operators allows problem-specific knowledge to be applied, while operator-probability adaptation should allow the blend of operators to be adjusted according to their relative productivities during execution. Moreover, by retaining an overall GA framework, the robustness of a population-based heuristic should be retained.

Although the GA/heuristic hybrid approach is a promising one, it does have some limitations that should be mentioned. As this is a form of guided stochastic search, there is no guarantee that the global optimum will be discovered; rather, the aim is to obtain a 'good enough' solution, given the time and computational resources available. In addition, as both a population of potential solu-

[1] Although some authors advocate redundancy (*cf.* DNA), this is usually on the basis of a *planned* re-introduction of redundancy after the development of an efficient encoding!

tions and a problem-specific encoding are employed, large amounts of computer time and memory may be required in comparison with other techniques. Finally, the development of a GA/heuristic hybrid often requires the investment of considerable time and ingenuity by its designer; this must be offset against actual or anticipated improvement in the quality of the solutions obtained.

1.5 Thesis Aims and Objectives

The overall aim of the thesis then, is to explore the suitability of the GA/heuristic hybrid approach for optical network design.

How does this approach compare to simple bit-string GAs and heuristics? What are its advantages and disadvantages? How successful is holding candidate network designs as object-oriented models, rather than decoding them from a binary representation? What are the advantages and disadvantages of operator-probability adaptation? What particular approach should be taken to adaptation? Can the performance of individual heuristic operators aid designers in the construction of better overall heuristics?

It is hoped that in the course of the thesis these questions, and related ones, will be answered.

1.6 Outline of Thesis

The overall structure of the thesis is as follows. First, after an overview of EAs and an introductory tutorial on GAs, Ch. 2 provides a near-exhaustive survey of evolutionary telecommunications, *i.e.* the application of EAs to the telecommunication problem domain, up to December 1998. In addition, less comprehensive surveys of earlier research into GA/heuristic hybrids and operator-probability adaptation are also included. These surveys provide the context for the evolutionary aspects of the research in this thesis, as well as establishing the foundation on which it was built.

A software toolset for optical network optimisation, modelling and design (NOMaD), developed by the author, is described in Ch. 3. It is aimed at two key design problems for multi-wavelength all-optical transport networks: mesh network topology design, and routing, fibre choice and wavelength allocation. NOMaD implements the author's GA/heuristic hybrid approach, and was used for all the experimental work reported.

The description of the GA/heuristic hybrid approach itself is provided in Ch. 4. The metrics used for design assessment, the overall GA framework, genetic operators and operator-probability adaptation mechanism are all reviewed.

Experimental results are reported in Ch. 5 for mesh network topology design, as well as WP routing, fibre choice and wavelength allocation. In addition, an investigation into NOMaD's operator-probability adaptation mechanism is described.

The successes and limitations of the research in this thesis are discussed in Ch. 6, which also provides numerous suggestions for further work, and records the author's conclusions and the contributions of the thesis.

The Appendices provide a variety of supporting information. Appendix A includes a copy of the author's published papers reporting directly on the research incorporated in this thesis. A complete bibliography of all the author's published work to date is recorded in Appendix B. Details of the wavelength-allocation heuristics implemented in NOMaD are provided in Appendix C. Several detailed tables supporting the near-exhaustive survey of evolutionary telecommunications in Ch. 2 are included in Appendix D. Further description of NOMaD judged inappropriate for inclusion in Chapters 3, 4 and 6, are given in Appendix E. Finally, Appendix F records additional experimental results excluded from Ch. 5.

1.7 Contributions of the Thesis

The key contributions of the thesis are:

- A tutorial introduction to GAs and GA/heuristic hybrids (§2.2).
- A near-exhaustive survey of evolutionary telecommunications, *i.e.* the application of EAs to the telecommunications problem domain, up to December 1998 (§2.3) [8].
- The design of the network model and GA/heuristic hybrid framework incorporated in NOMaD (§E.3 and §4.2) [31, 32].
- NOMaD's operator-probability adaptation mechanism (§4.2.4 and §5.3) [33].
- The GA/heuristic hybrid algorithms for optical mesh network topology design, particularly the new link-group crossover operator (§5.1) [34].
- The GA/heuristic hybrid algorithms for minimum network wavelength requirement routing, fibre-choice and wavelength allocation, particularly the new problem-specific operators: wavelength-allocation random k-shortest paths, k-shortest path mutation, path re-route, path shift-out and k-shortest path single-point crossover (§5.2.1) [35].

- The GA/heuristic hybrid algorithms for minimum cost routing, fibre choice and wavelength allocation, using the same new problem-specific operators (§5.2.2) [36, 37].

Chapter 2

Evolutionary Algorithms

Evolutionary computation has a long history. Unfortunately, this history is virtually unknown to many practitioners in the field.

David Fogel, 1998

This chapter includes a brief overview and historical perspective on evolutionary algorithms (EAs), an introduction to genetic algorithms (GAs), as well as three literature surveys. These provide the context for the evolutionary aspects of the author's research, as well as establishing the foundation on which it was built.

The chapter is divided into six sections: §2.1 is a brief overview of EAs. A tutorial introduction to GAs, Goldberg's Simple Genetic Algorithm and GA/heuristic hybrids is given in §2.2. A near-exhaustive survey of evolutionary telecommunications, *i.e.* the application of EAs to the telecommunications problem domain, up to December 1998 is presented in §2.3. Then §2.4 and §2.5 provide more limited surveys of prior work on GA/heuristic hybrids and operator-probability adaptation, respectively, in areas other than evolutionary telecommunications. Finally, the chapter is summarised in §2.6.

2.1 Overview of Evolutionary Algorithms

Early work

The history of EAs is longer than most practitioners in the field realise. However, this has been remedied in recent times by both a concise history and summary of the state of the art by Bäck *et al.* [38] in 1997, as well as by David Fogel's 1998 edited volume of both early and important papers in evolutionary computation [4]. The origin of EAs dates back to pioneering work in the 1950s, even before adequate computing power for them was available [38]. In particular, David Fogel [4] has highlighted early research by Friedman [39], Fraser [40], Box [41] and Friedberg [42,43].

In his 1956 master's thesis, Friedman [39] described how control circuits employing 'basic neural elements' could be evolved for autonomous robots using random selection and mutation. His work anticipated by decades more recent efforts in robot control, evolving artificial neural networks, and evolving electronic circuits. Friedman's research was theoretical—there was no practical implementation—but by analysis of fitness landscapes, he endeavoured to show the superiority of his suggested approach over purely random search.

Then, as part of a computer-based simulation study of genetic systems in 1957, Fraser [40] introduced an approach which clearly anticipated the bit-string GA later studied by Holland [44], Goldberg [2] and others. Fraser's algorithm employs diploid bit-strings and a multi-point crossover operator for recombination, with different crossing probabilities at different points along the genotype so as to vary linkage between the genes. His algorithm is generational, with a selection mechanism that produces more offspring than required and then removes the excess according to some function of their phenotype fitness. This foreshadowed the (μ, λ) selection method of evolution strategies (ESs).

Box [41], also in 1957, introduced evolutionary operation as an online approach to improving the productivity of an industrial process. During an agreed period, a systematic set of mutations is applied to the current production settings, with performance data gathered from the running plant. At the end of the period the best settings found become the new current production settings, and systematic experimentation recommences. While Box's approach has been criticised by others (*e.g.* Goldberg [2]), it nevertheless represents a successful early application of both mutation and fitness-based selection.

Finally, Friedberg *et al.* [42, 43] in 1958 developed some of the earliest research on the evolution of computer programs. These are represented as binary instructions, and a credit-assignment mechanism is used to apportion credit to individual instructions. The approach suffered from a number of limitations motivated and exacerbated by the restricted computing power available to Friedberg *et al.*. Nevertheless, the work clearly had similarities to the later development of both GAs and classifier systems (CSs). Moreover, concepts such as Holland's implicit parallelism and schemata [44] were anticipated, the importance of incorporating domain knowledge into the algorithm was recognised, and a number of insights obtained that were subsequently rediscovered by the evolutionary computation (EC) community [4].

Genetic algorithms (GAs)

GAs were introduced by Holland in 1962 [44,45], originally intended as a general model of adaptive processes [38], but subsequently widely adopted as optimisers [2,3,9]. They are inspired by natural genetic processes, and thus have a population of chromosomes, each analogous to the DNA in a natural organism [46]. The population evolves by simulated evolution, employing genetic operators modelled on those in nature. The overall process is driven by fitness, determined by decoding the chromosome and applying a problem-specific objective function, analogous to the reproductive success of an individual organism matured from its DNA.

Their popularity can arguably be traced back to two extensively cited books by Goldberg [2] and Davis [9] in 1989 and 1991 respectively. As well as a clear description of the basics of GAs, Goldberg [2] provides a complete Pascal implementation, a survey of applications, advanced operators and techniques, and an introduction to CSs. In his book, Davis [9] describes the step-by-step development of a GA/heuristic hybrid from a simple GA, and then includes a wide-ranging edited collection of application case studies of hybrid GAs.

While the historical distinctions between EAs have blurred [4], GAs are now, by far, the strongest research area. However, to make this assertion clearly, GA/heuristic hybrids [9] must also be regarded as GAs. Some researchers would argue [47], though, that such algorithms bear little resemblance to the canonical GA, and should simply be described as EAs

Evolutionary programming (EP)

EP originated with Lawrence Fogel in 1962 [48–50] as an attempt to create artificial intelligence [38]. The population of individuals being evolved are typically finite state machines (FSMs), and the emphasis is on the evolution of behaviour from parents to offspring, rather than on modelling genetic operations as in GAs [46]. The inspiration for EP is an abstraction of evolution at the species level, consequently recombination (crossover) is not usually included, although even in the early days, this was not always so [47]. Although originally more focused, EP does not now impose any restriction on the representation of an individual in the population [4, 46]: rather a ‘natural’ representation for the problem is selected (*cf.* hybrid GAs). Mutation is stochastic, usually determined such that small variations are common and large variations less so; in addition, adaptation of mutational variation is now often applied during a run. Selection, although originally deterministic, is now typically tournament-based [4].

Evolution strategies (ESs)

ESs were developed by Rechenberg and Schwefel in 1965 [51, 52], with the aim of solving mainly experimental parameter optimisation problems [38]. An individual in the population is an array of real-valued parameters: there is no encoding. ESs were originally based on a 'population' of size one, with the parent competing each time for survival with a single offspring: the (1+1)-ES. Now, however, ESs typically use a larger population, size μ. From this, λ offspring are obtained by mutation (and sometimes recombination). The best μ individuals survive deterministically, selected from either the combined pool of parents plus offspring, termed $(\mu + \lambda)$, or just from amongst the offspring, (μ, λ). Mutation of an individual's parameters is Gaussian (normal) about the current position. In addition, strategy parameters are usually included in the individual, one for each problem parameter; each of these determine the mutation rate (standard deviation) for the corresponding parameter. The strategy parameters are themselves mutated, although according to a global log-normal distribution.

While ESs have achieved a measure of success, particularly within civil engineering [46], its popularity has been limited by much of the early literature being in German. This was remedied in 1995 with a book by Schwefel [52].

Genetic programming (GP)

As has already been noted, efforts to evolve computer programs date back to the earliest days of EC, with the work of Friedberg *et al.* in 1958 [42, 43]. However, these early machine-code based efforts were not particularly fruitful. Better results awaited more powerful computers, and also more expressive representations. In particular, the FSMs of early EP (see above), the if-then rules of CSs (see below), and the LISP s-expressions of what came to be known as GP have all had a measure of success. While several researchers in the 1980s suggested the use of LISP s-expressions for program evolution, including Hicklin, Fujiki and Cramer [4], it was the work of Koza from 1989 [53] which established the GP field.

In GP [54], an individual in the population is usually a single LISP s-expression (*i.e.* a parse tree), composed from a user-specified, problem-specific set of functions and terminals. GP crossover, usually the only genetic operator, is based on the exchange of sub-trees. Fitness of the individuals is established by evaluating an individual program, either directly or indirectly, as a solution to the problem. Selection was originally stochastic and fitness-proportional [54], but subsequently tournament-based selection has been preferred [55].

In the last decade, GP has been developed by Koza [55, 56] and others [57–59], and has also diversified to include a wide-range of representations and operators [60]. It has even returned to the successful evolution of machine-code programs [61].

Classifier systems (CSs)

CSs[1] were introduced by Holland and Reitman in 1978 [62] as an attempt to develop adaptive behaviour by evolving a set of classifiers (*i.e.* condition-message pairs). In a CS, a set of detectors (*i.e.* the program input) posts messages to a message list [2]. Each classifier (in parallel) is activated when it detects a matching condition, and competes to post its message. While messages are binary, the classifier conditions can include 'wild cards' (usually represented as '#'), allowing an individual classifier to respond to a range of conditions. The message list in turn can trigger effectors, resulting in program output. Evolution of the set of classifiers is achieved by assigning credit to successful classifiers; they in turn pass some of this credit on to other classifiers that contributed to their success (the 'bucket brigade'). Periodically, accumulated credit is used as fitness to replace a limited subset of the population with a modified GA, hopefully resulting in an improvement of the overall adaptive behaviour of the system as a whole.

Subsequent development of CSs have included evolving entire sets of classifiers (the 'Pittsburgh' approach) rather than individual classifiers (the 'Michigan' approach) [4], and a wide variety of different credit-assignment mechanisms [46].

Evolutionary hardware (EH)

A relatively recent development in EAs has been the emergence of the EH field. This includes EAs where the evolution occurs in purpose-built processors (rather than in software on general-purpose processors), EAs where the fitness evaluation occurs in hardware (rather than *e.g.* software simulation), or EAs that combine both of these. It could be argued that EH is not really a distinct field, as the only difference is in the implementation of the algorithms or their fitness evaluation. Indeed, much of the work on ESs could be considered EH. Nevertheless, there is a growing community focusing on this specialised aspect of EAs, and the interested reader is referred to the recent books by Sipper [63], Thompson [64] and Mange & Tomassini [65].

[1] CSs have been given a variety of names over the years, including learning classifier systems (LCSs) [46].

Conference	Years
Congress on Evolutionary Computation (CEC)	1999, 2000
Genetic and Evolutionary Computation Conf. (GECCO)	1999, 2000
Intl. Conf. on Genetic Algorithms (ICGA)	1985, 1987, 1989, 1991, 1993, 1995, 1997
IEEE Intl. Conf. on Evolutionary Computing (ICEC)	1994, 1995, 1996, 1997, 1998
Parallel Problem Solving from Nature (PPSN)	1991, 1992, 1994, 1996, 1998, 2000
Conf. on Evolutionary Programming (EP)	1992, 1993, 1994, 1995, 1996, 1997, 1998
Genetic Programming Conf. (GP)	1996, 1997, 1998
Foundations of Genetic Algorithms (FOGA)	1991, 1993, 1995, 1997, 1999
Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA)	1995, 1997
Intl. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA)	1993, 1995, 1997, 1999

Table 2.1: Major evolutionary computation conferences

Conferences & books

In completing this overview, it is worth noting that the growth in popularity of EAs has been accompanied by a growth in the number and size of the conferences in the area and in the number of books published. Some of the major conferences, and the years they took place, are recorded in Table 2.1. In addition, based on the bibliography assembled by Koza [66], Table 2.2 records EC books published in chronological order[2], whereas Table 2.3 shows them by subject area.

2.2 Tutorial on Genetic Algorithms

This section provides a tutorial introduction to GAs. Its main purpose, following some background on both GAs in general and Goldberg's Simple Genetic Algorithm in particular, is to introduce GA/heuristic hybrids. This latter approach is a central theme of the thesis. However, the tutorial does not give sufficient detail to fully appreciate the literature surveys later in this chapter (*cf.* §1.1).

[2] In Tables 2.2 and 2.3, only English language books are included, and books are recorded only in their most recent editions, so *e.g.* Holland's *Adaptation in Natural and Artificial Systems* is represented by the 1992 edition [45] rather than the 1975 [44].

Year	Count	Books
1966	1	[49]
1983	1	[67]
1986	1	[68]
1987	1	[69]
1989	2	[2], [70]
1990	1	[71]
1991	3	[9], [72], [73]
1992	3	[74], [45], [54]
1993	1	[75]
1994	5	[76], [77], [78], [57], [55]
1995	5	[79], [80], [81], [82], [52]
1996	11	[83], [84], [85], [86], [87], [3], [88], [89], [90], [58], [91]
1997	15	[92], [93], [94], [95], [96], [97], [98], [99], [100], [101], [102], [103], [104], [61], [63]
1998	10	[105], [4], [106], [107], [60], [108], [109], [110], [64], [65]
1999	15	[111], [112], [113], [114], [115], [116], [117], [118], [119], [120], [121], [56], [59], [122], [50]
2000	1	[123]
Total	76	

Table 2.2: Evolutionary computation books – chronological order

Subject	Count	Books
EC	14	[67], [79], [80], [83], [92], [93], [94], [95], [96], [105], [4], [111], [112], [113]
GAs	38	[69], [2], [70], [71], [9], [74], [45], [75], [76], [77], [78], [81], [82], [84], [85], [86], [87], [3], [88], [89], [90], [97], [98], [99], [100], [101], [102], [103], [106], [107], [114], [115], [116], [117], [118], [119], [120], [121]
EP	4	[49], [72], [109], [50]
ESs	1	[52]
GP	13	[54], [57], [55], [58], [91], [104], [61], [60], [108], [56], [59], [122], [123]
CSs	3	[68], [73], [110]
EH	3	[63], [64], [65]
Total	76	

Table 2.3: Evolutionary computation books – by subject area

2.2.1 Genetic Algorithms: An Introduction

Goal of optimisation

When engineers attempt to optimise the parameters of a problem, or search for the global maximum (or minimum) of an objective function, what is their actual goal? In a large and complex problem space, where it may never be possible to find the true global optimum, surely a 'good' solution is sufficient? In thinking about optimising a problem, it is possible to be biased into thinking that only the true, global optimum is sufficient. However, in complex systems, as in the natural world, a solution that is 'good enough' may suffice. The goal is not so much to find the global optimum solution, but rather a solution that is the best that can be achieved with the time and resources available. It is with complex problems such as these, that the robustness and generality of modern heuristic search methods, such as EAs, becomes apparent. As Wasserman has observed [124]: "Sam and Ed are on a camping trip when they see a charging bear. Sam immediately starts to lace up his running shoes. Ed says, 'Sam, don't bother; you'll never outrun that bear.' Sam replies, 'I don't have to outrun the bear. I only have to outrun you.'"

Evolutionary algorithms

EAs are based on a simplification of the mechanisms of natural evolution *i.e.* natural selection and genetics. As has already been noted, a variety of different EAs have been proposed [4]. However, they all share a common basis of simulating the evolution of individual structures using *selection*, *reproduction* and *mutation*. These processes in turn depend on the apparent *fitness* of the individuals in the context of their environment. Thus EAs maintain a *population* of structures that reproduce under the influence of selection and other genetic operators, such as mutation and, in some cases, *recombination*. Reproduction focuses attention on high-fitness individuals, thus exploiting the available fitness information. Recombination and mutation modify those individuals, thus promoting exploration of new possibilities. Although EAs are simplistic from a biologist's point of view, nevertheless, they are still sufficiently complex to act as robust and powerful adaptive search techniques [46].

Genetic algorithms

Against this general background of EAs, GAs can be distinguished by two key features. First of all, they operate in a search space using genetic operators on fixed-length *strings* that encode points in the problem space, rather than on the problem space directly. In addition, they use a particular recombination operator on these fixed-length strings that is unique to GAs, called *crossover*.

Chromosomes

The use of fixed-length strings in GAs is inspired by the encoding of the genetic inheritance of an individual biological organism in the DNA that makes up its chromosomes. The strands of DNA in the cells of, say, a human being, encode its entire genetic heritage in a four-'letter' alphabet. Thus, in a GA, each individual in the population contains a (fixed-length) string (also sometimes called a chromosome), usually, but not always, using a binary alphabet. The use of such a binary representation follows from schema theory [2], which suggests that a two-symbol encoding promotes more effective search.

Fitness

The fitness of a biological organism is measured by its interaction with the environment. Its fitness determines (to some extent) how long it will survive, what opportunities it will have to reproduce, and consequently, how many offspring (or children) it will have. In a GA there is not usually an environment as such, but rather the fitness of an individual is determined by decoding its chromosome (*i.e.* the fixed-length binary string) and thereby interpreting it as a point in the problem space (Fig. 2.1). The (possibly modified) value of the objective function for the problem is then fed back to the GA as the fitness of that individual. The fitness value is then used to determine the extent to which that individual is allowed to reproduce into the next generation.

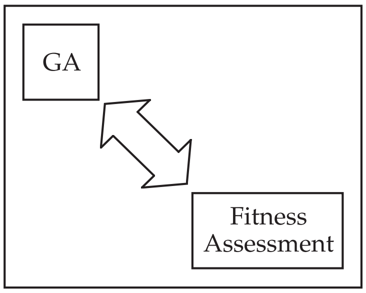


Figure 2.1: Genetic algorithm and fitness assessment

Crossover

What then of the evolution of the population from one generation to the next? As well as reproduction as a function of fitness (which in a GA may simply be *fitness-proportional*), many biological organisms use sexual reproduction to allow the blending of genetic characteristics from two organisms to be reflected in their offspring, such that at least some of the children will be 'fitter' than either of their parents. Thus in GAs, parents that are selected to reproduce will (with a certain probability) not simply duplicate their chromosomes in their children (*i.e.* cloning), but instead two parents will recombine their chromosomes using the crossover operator such that the chromosomes of their children are a combination of both their parents.

Mutation

In the natural world, another factor that operates on biological organisms' genetic heritage is mutation—the apparently random modification of some of the genetic material inherited by an organism. Most such mutations would appear to be harmful[3], but occasionally one will be beneficial, resulting in increased fitness for the organism, and consequently the mutated genetic material will be passed on to more and more organisms as the generations pass. Inspired by this, GAs include the possibility of the mutation of a child's chromosome during reproduction by randomly changing (at low probability) each of the bits (or letters for a non-binary alphabet) of its chromosome. For GAs though, this operator is regarded as much less important than fitness-proportional reproduction and crossover, and so the probability of any single bit being mutated is usually kept very low [2].

Population and generations

In contrast with natural populations, the population of individuals in a GA is often maintained at a constant size (for simplicity). The initial starting population (which has no analogue in the natural world) is usually randomly generated, although it may be seeded with a few individuals encoding better-than-average attempts at the problem. In addition, many GAs are *generational*—a new population of individuals is created from the old by reproduction, crossover and mutation, and then the old generation is discarded (thankfully this is not done with human populations!).

Distinctive qualities

What then are the main differences between GAs and other search algorithms? Goldberg [2] identifies four. GAs work with a coding of the problem's parameters, not with the parameters directly. They employ the objective function values themselves, rather than derivatives. They

[3]Unless there is considerable redundancy in the chromosome, as there often is in nature, in which case the majority will actually be neutral, *i.e.* no observable impact on fitness; it is the majority of *expressed* mutations that are actually harmful.

search using a population of points, instead of a single point. Finally, GAs incorporate probabilistic choices, rather than purely deterministic.

Because the problem parameters are coded as finite-length (binary) strings, and the GA then operates on these directly using only objective function values, *i.e.* without problem-specific information (at least for a 'pure' GA) or even derivatives, a very broad range of problems can be tackled successfully. Although it should be noted that considerable thought can still be required to develop a suitable binary encoding for a complex problem domain. Clearly, using a 'database' of points in the problem space (*i.e.* the population of strings), rather than a single point, immediately reduces the chance of getting trapped in a local optimum. Finally, the probabilistic choices in the heart of the GA, far from indicating directionless search, are actually used to guide the algorithm to explore areas of the search space most likely to lead to improvement. Further, by re-running a GA several times with the same parameters (apart from the random seeds), the probability of finding a solution of a given quality can be greatly enhanced.

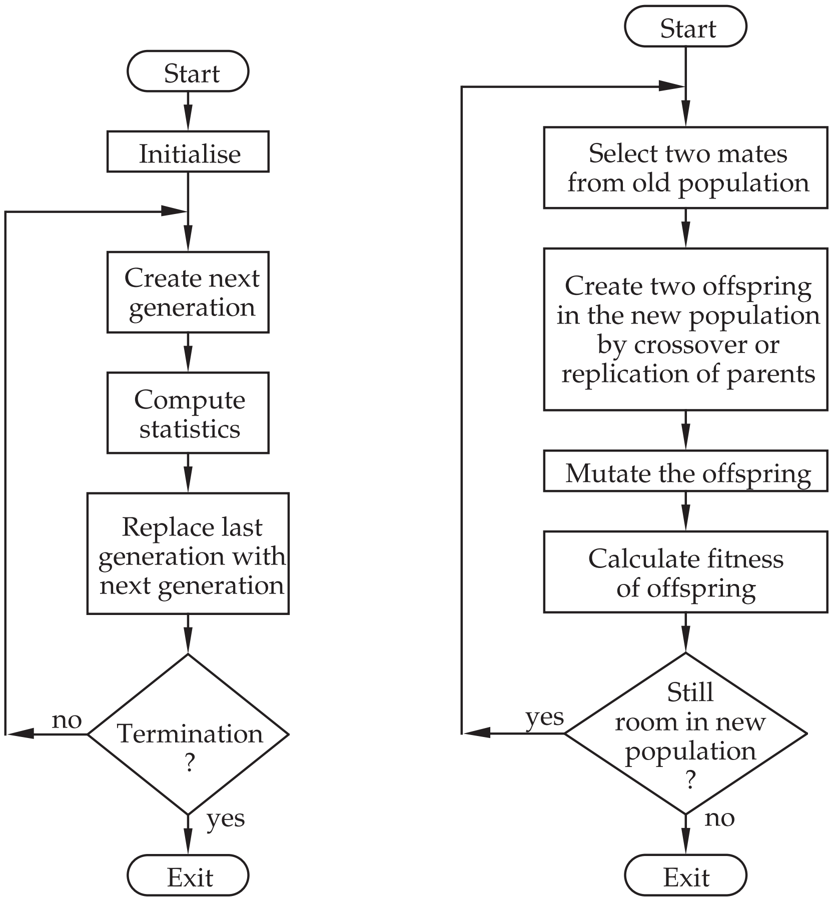
2.2.2 The Simple Genetic Algorithm

Overall algorithm

To make the description of the genetic algorithm above a little more concrete, it is perhaps beneficial to examine Goldberg's Simple Genetic Algorithm (SGA) [2]. An overall flowchart for the algorithm is given in Fig. 2.2(a). After creating the initial population (random binary strings), assessing the fitness of each individual and calculating overall population statistics, the algorithm enters its main loop. Each generation is created from the one before, using selection, reproduction, crossover and mutation. The population statistics are computed again, including calculating average fitness and identifying if a new best-of-run individual has been found. The old generation is then replaced by the new. The algorithm continues to loop until a termination condition, usually a maximum number of generations, is met.

Create next generation

The 'Create next generation' subroutine can be further expanded, as shown in Fig. 2.2(b). First two individuals are selected from the old population. From these, two new offspring are created which, with a certain probability, are recombined using crossover. Then the new individuals, whether simple clones of their parents or not, are subject to mutation. Finally, the fitness of the new



(a) Overall (b) Create next generation

Figure 2.2: Simple Genetic Algorithm flowcharts

individuals is determined by decoding them and applying the objective function. The subroutine continues to loop until the new population has been completely filled with offspring.

Another way of looking at the details of the SGA is in terms of the relationship of one generation to the next (the new population to the old). The SGA is generational, *i.e.* it uses non-overlapping populations. Each generation is created entire from the one before, and then the old one is discarded. In contrast with natural populations, the population size is kept constant: each generation has the same number of individuals as the preceding one. This is illustrated in Fig. 2.3, which shows generation t of N individuals transformed into generation $t+1$ by the genetic operators: selection, reproduction, crossover and mutation.

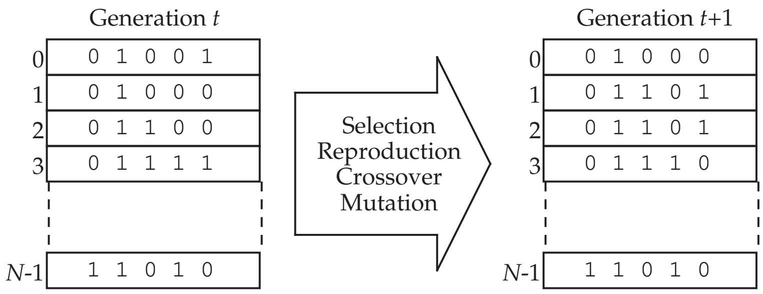


Figure 2.3: From one generation to the next (after [2])

Selection & reproduction

By way of illustration, consider a tiny example population of six strings, each of length 5 bits. The strings encode unsigned integers (x), and the objective function (fitness) is $f(x) = 2x^2 + x$ (Table 2.4). The simplest form of selection is fitness-proportional. The probability of a string being selected is simply its fraction of the sum-of-fitness of the whole population (Fig. 2.4). Fitness-proportional selection can be imagined to operate by analogy with a roulette wheel (or a 'wheel of fortune'). For every vacancy in the new population, say, the roulette wheel is spun, and where the ball lands (by randomly selecting a number between 0.0 and 1.0, representing the circumference of the wheel), that individual is reproduced in the new population. The chance an individual has of being selected and reproduced is thus proportional to its share of the wheel, and hence to its fitness. Selection is done 'with replacement' *i.e.* just because an individual is selected once does not mean it cannot be selected again. The wheel remains the same, and is simply spun again until the new population is filled. Therefore, the number of offspring each individual in the old population can expect to have in the new, is just that fraction of the new population that corresponds to its proportion of the total fitness in the old. In practice though, because of stochastic effects, an individual may have more (or less) than the expected number of offspring—in the limit, a very fit individual might have no offspring, or a very unfit individual might have many!

Crossover

For every pair of individuals in the new population (which are just clones of their parents at this stage), the crossover operator is applied with a certain probability (typically 0.6). Simple crossover operates by selecting a crossover point at random somewhere along the strings, *e.g.* if the strings are length k, a crossover point j would be selected between 1 and $k-1$. All the bits between 0 and $j-1$ inclusive would remain unaffected, but the bits between j and $k-1$ would be swapped between strings (Fig. 2.5). Between two strings from the tiny example population, say:

```
0 1 1 0|0
0 1 1 1|1
```

with $k = 5$ and $j = 4$ (indicated by the '|'s), the resulting crossover would yield:

```
0 1 1 0 1
0 1 1 1 0
```

Mutation

For GAs, fitness-proportional reproduction and crossover are the primary genetic operators, but something more is still needed. As a result of selection and crossover, the population will gradually converge on more promising areas of the search space. However, in any given generation, all members of the population may come to have the same bit value (say `1`) at a particular bit position—perhaps all those individuals with a `0` at that location proved unfit and died out. But what if later in the evolution of the population, better fitness could be found by incorporating a `0` at that bit position (as other locations have changed over time)? Selection and crossover alone can never bring back a `0` at that bit position—hence the need for mutation. In a binary string alphabet, mutation simply consists of, at low probability (say 0.001), toggling the bit at each position in a new individual—a `0` to a `1`; a `1` to a `0`. For non-binary alphabets, some variation must be used—perhaps selecting a letter at random from the alphabet, excluding the letter currently at that location. Although mutation brings needed diversity back into a population that might otherwise converge prematurely on less than ideal points in the search space, it must be used sparingly [2]. Just as mutation in

No.	String	x	Fitness $2x^2 + x$	% of Total
0	`01001`	9	171	5.2
1	`01000`	8	136	4.2
2	`01100`	12	300	9.2
3	`01111`	15	465	14.2
4	`10100`	20	820	25.1
5	`11010`	26	1378	42.1
Total			3270	100.0

Table 2.4: Sample problem strings and fitness values

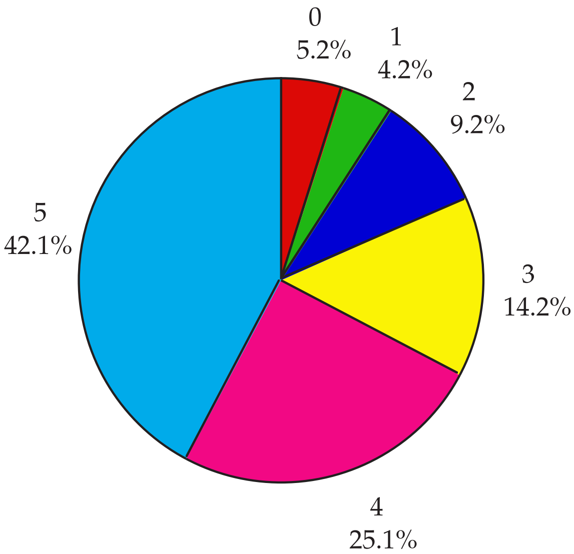


Figure 2.4: Roulette-wheel selection

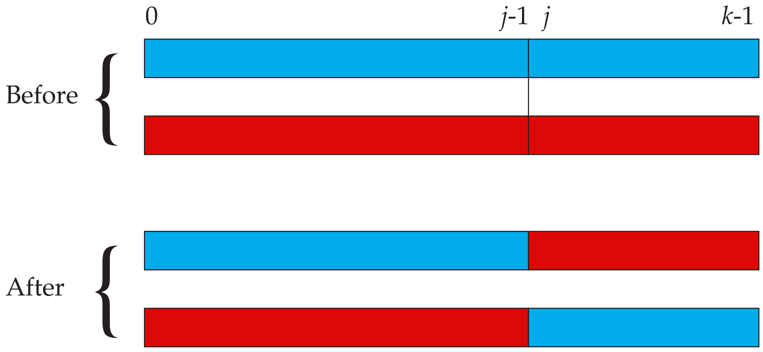


Figure 2.5: Crossover

the natural world is often harmful, so mutation in GAs often reduces the fitness of the individuals affected—it is after all just a form of random walk.

Improving the SGA

Although the SGA is a reasonable starting point, a wide variety of ways have been suggested to extend it [2]. These are usually inspired by either additional aspects of natural evolution, or by integrating other search algorithms or domain knowledge into the GA.

Instead of fitness-proportional selection, for example, fitness *ranking* can be used, where an individual's rank in the fitness-sorted population determines its selection probability, rather than its absolute fitness value. Another possibility is *tournament selection* where, whenever an individual is to be selected, a small number of structures, chosen randomly, but *not* according to fitness, compete directly in terms of their fitness values for the right to mate.

Alternatively, rather than abandoning fitness-proportional selection, the fitness values themselves can be suitably modified to maintain adequate selection pressure and avoid premature convergence. Variations include *windowing*, *i.e.* rescaling fitness values using the history of the last few generations, or using *linear*, *sigma truncation* or *power-law rescaling* mechanisms.

The SGA is strictly generational, but this too can be altered. One, or a few, of the better individuals can be passed unchanged into the new population (called *elitism*). At the other extreme, the population as a whole can be left largely unchanged, and only a few (usually the worst) individuals replaced (known as *steady-state reproduction*).

Modified or additional genetic operators can also be introduced. Instead of single-point crossover, used in the SGA, *multi-point* (usually two-point) *crossover* can be employed. In the limit, *uniform crossover* can be used to create an offspring by taking a bit at random, for each position along the string, from either parent.

The possibilities for extending the SGA are considerable, of which just a few have been mentioned, but some caution should be exercised. While for any particular problem, some of these changes will be beneficial, others may prove counter-productive. The next section describes a more promising alternative.

2.2.3 Genetic-algorithm/Heuristic Hybrids

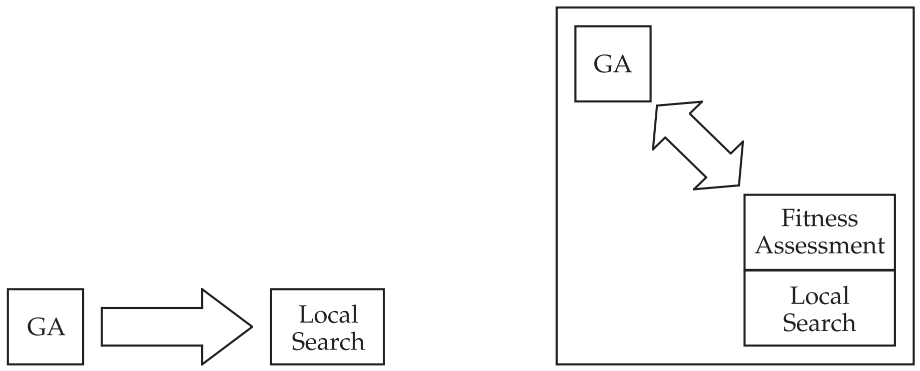
Incorporating other search algorithms

How else then can the SGA be improved? Most of the extensions to the SGA mentioned above were biologically inspired. What about making use of other optimisation techniques or domain-specific knowledge? One straightforward approach is based on the recognition that GAs are good for generalised search, but having located optimal or near-optimal regions in the search space, can find it difficult to locate the optimal point itself. Calculus, greedy algorithms and problem-specific heuristic search can, however, be very good at local search. By combining GAs and local search algorithms, the overall hybrid algorithm can outperform its individual components—"the genetic algorithm finds the hills and the hill-climber goes and climbs them" [2].

There are two main approaches. Either run the GA for a reasonable period and then run the local search algorithm on the best (or better few) individuals (Fig. 2.6(a)), or incorporate the local search into the overall algorithm (Fig. 2.6(b)). In the latter case, the local search can itself be used as an operator (alongside crossover and mutation, say), with the results of the local search re-encoded (if necessary) as an offspring—essentially acting like an enhanced mutation operator (*i.e.* Lamarkian evolution). Alternatively, local search can be applied before function evaluation, but no change made to the string itself (*i.e.* Baldwinian evolution). The fitness of each individual is simply its fitness after the application of local search each time. Because the results of the local searches are only fed back into the population in the form of fitness values, not actual strings, this approach (at the expense of considerable additional processing) helps to reduce the risk of a super individual being found by local search, whose genetic material would then dominate future generations, leading to premature convergence.

Domain-specific knowledge

As discussed above, one way of incorporating problem-specific knowledge into a GA is by, say, adding an enhanced mutation operator which decodes a string to a point in the problem space, applies a local search heuristic, and then re-encodes the result. This approach is a comparatively minor extension to a traditional GA, that nevertheless can bring considerable performance enhancement. However, it is possible to go a good deal further! There is a pragmatic 'school of thought' in GA research [9] that sees GAs solely as a means to the solution of engineering problems. Here, the approach is to adopt a problem-specific encoding, and then develop and employ problem-specific



(a) After the GA

(b) Within the GA

Figure 2.6: Incorporating local search

operators that together combine the best existing heuristics for the problem with an overall GA framework—a hybrid algorithm. The motivation for this approach is simple, according to Davis [9]: "Although genetic algorithms using binary representation and single-point crossover and binary mutation are robust algorithms, they are almost never the *best* algorithms to use for any [specific] problem". It is almost always possible to develop a problem-specific GA/heuristic hybrid that will outperform either the heuristic(s) or a traditional GA alone.

Domain-specific encoding

Rather than using a binary string, the encoding chosen is the same as the current algorithm (*i.e.* the best known algorithm used on the problem before the development of the hybrid) [9]. However, if the problem is a new one, then a 'natural' encoding would be used. For example, if the problem is to optimise the ten real-valued parameters that control the overall design of an aeroplane, then an individual in the GA would simply consist of a list of ten real numbers, perhaps constrained to the physical limits of the parameters. By contrast, in a traditional GA, each parameter would be quantised and encoded as a binary string, with an individual consisting of the concatenation of the ten encoded individual parameters. In other problems the contrast might be even more marked, especially where the natural encoding was, say, two-dimensional or some complex pointer-based structure in computer memory.

Domain-specific operators

The method then proceeds to developing domain-specific versions of the traditional crossover and mutation operators [9]. For the aeroplane example, one offspring might be constructed by selecting parameters at random (with equal probability) from either parent; the other offspring would get the complementary set (uniform crossover). Another possibility might be *average crossover*, with one offspring generated by averaging the values of the corresponding parameters in the two parents. Possible mutation operators include simply choosing a single parameter at random and giving it a random value (within established limits); or each of the parameters on an individual (with a certain probability) could be modified by a small random amount—*real-number creep*. Finally, operators based on existing domain- or problem-specific heuristics can be developed and incorporated as enhanced mutation operators, as was described previously for the traditional GA. In terms of the aeroplane example, these might include well-established relationships between, say, wing and tail area, or overall engine power and fuselage length—whatever could be obtained from domain experts and existing design software. However, the choice of encoding and the development of problem-specific operators requires considerable ingenuity. It is an iterative process, involving online experimentation, consultation with domain experts, and experience drawn from the growing body of literature on hybrid algorithms. Clearly, efforts to improve a hybrid algorithm should only continue as long as the fitness of the problem solutions discovered by the hybrid remain less than required. Remember, as Sam said, "'I don't have to outrun the bear. I only have to outrun you.'" [124]

2.3 Evolutionary Telecommunications

This section provides a (nearly) exhaustive survey of evolutionary telecommunications, *i.e.* the application of EAs to the telecommunications problem domain. To the best of the author's knowledge, there has been no prior publication of a thorough survey in this area, although there is a brief prior survey by Proestaki & Sinclair [125]. In addition, two (unpublished) technical reports, by Smith [126] and Smith *et al.* [127], were consulted. A summary of the survey was published at the GECCO'99 Workshop on Evolutionary Telecommunications: Past, Present and Future [8]. It should be noted that the survey was completed in December 1998, and has not been updated subsequently.

2.3.1 Survey Scope

In considering the scope of the survey, it was decided to restrict it to the application of purely evolutionary approaches to problems that are unique (or nearly unique) to telecommunications, such as network design.

As a result, not only other modern heuristic methods, such as simulated annealing (SA) [5] and tabu search (TS) [6, 7], but also other natural algorithms were excluded. An example of the latter is ant colony optimisation (ACO) [128], a search and optimisation procedure inspired by the cooperative problem-solving behaviour of natural ant colonies. This has been applied, amongst other areas, to static routing (load balancing) in circuit-switched telecommunications networks [129] and dynamic routing in packet-switched networks [130].

Limiting the problem domain to those applications unique to telecommunications also removed many papers. Examples of these include areas as diverse as signal processing [131,132], ground-water monitoring network design [133], distributed computing [134, 135] and distributed databases [136].

Furthermore, it was decided to focus exclusively on published papers, rather than include project reports, theses or technical reports. These are not necessarily of the same standard as published material, and where they are, this is best demonstrated by subsequent peer-reviewed publication.

A final limitation is that there are some papers, of which the author is aware, that could not be located in the timescales and with the means available. As a result, these too have been omitted from both the survey and bibliography of the thesis.

In total, 112 papers were judged within the scope of the survey, and these are listed (in alphabetical order of first author surname) in Appendix D, Table D.1. For each paper, as well as its own reference number in the thesis bibliography, those papers it cites and which in turn cite it, from within the survey, are also recorded. In addition, classification of each paper in terms of application area (and problem element), solution representation and genetic operators (including population dynamics) are provided.

2.3.2 Historical Growth

Fig. 2.7 shows the chronological distribution of the papers (see also Table D.2 on p. 242). From the earliest in 1987, there has been a long-term pattern of steady growth, overlaid by the two-year cycle of important conferences for the area, particularly the Intl. Conf. on Genetic Algorithms (ICGA) series. The slightly depressed total in 1998 is believed to be partially due to this latter effect, as well as to the relative difficulty of locating recent conference papers.

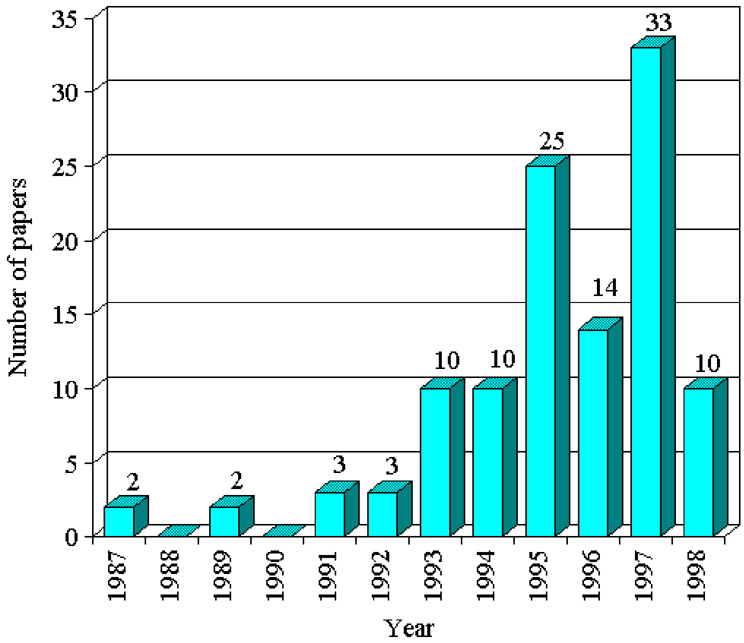


Figure 2.7: Evolutionary telecommunications papers – historical growth

2.3.3 Early Papers

The ten earliest papers in evolutionary telecommunications addressed a variety of application areas and employed a wide range of representations and operators. Apart from citing their own work, none of them cites other evolutionary telecommunications papers (in common with many later authors), and so each represents an independent entry into the area. While some have been cited by several subsequent papers [137–141], others have received comparatively little attention [142–144] and some have not been rediscovered at all [145, 146].

In their three papers from 1987 and 1989, Davis & Coombs [137–139] selected link capacities for wide-area packet-switched networks using a hybrid GA with both an integer representation and several problem-specific operators. They also applied efficient approaches to incorporate their numerous design constraints. Then in 1992, Potter *et al.* [142] combined both a hybrid GA and a bit-string GA for the design of military networks, including network element selection, dimensioning and battlefield location. Michalewicz [143] evolved minimum-spanning-tree topologies for computer networks in 1991 using a two-dimensional binary adjacency matrix representation and problem-specific versions of mutation and crossover. The next year, Kumar *et al.* [145] tackled three constrained computer network topology problems, aiming for maximum reliability, minimum network diameter or minimum average hop count. Their GA was bit-string, but used problem-specific crossover, as well as a repair operator to correct for redundancy in their network representation.

Also in 1992, Sobotka [146] augmented a survivable military communications network with a fixed number of satellite links selected by GA. He employed elitism, problem-specific recombination and fitness based on network damage assessment. For the Steiner tree problem, Hesser *et al.* [144] used a simple bit-string GA in 1989 for arbitrary Steiner point locations, although applying several heuristics during decoding. For network routing, Cox *et al.* [140] used a permutation-based GA in 1991 for the bandwidth-packing component of an heuristic for dynamic-anticipatory routing in circuit-switched networks. Also that same year, Pan & Wang [141] employed a GA for minimum-delay bandwidth allocation, *i.e.* link capacity assignment and routing, in an ATM network. Their GA encodes individuals as a string of indices into a combined look-up table of both link capacities and routes.

2.3.4 Applications

Fig. 2.8 illustrates the classification of papers into application groups (see also Table D.3 on p. 243). While papers have usually only been included in one overall application area (*e.g.* Computer networks), they have been assigned to as many problem elements as necessary (*e.g.* Dimensioning, Node location, *etc.*). Although there is reasonable overall coverage, with every feasible 'cell' containing some papers, some applications have resulted in more publications than others. For example, while there are many similarities between wavelength allocation in optical networks and frequency assignment in radio networks, the latter has almost four times the number of papers. Such imbalances may well indicate areas with potential for further research.

Dimensioning

Fully 33 papers consider dimensioning of telecommunications networks, although usually alongside another aspect of network design, such as node location, topology or routing. (The discussion of many of these [34,147–168] has therefore been deferred to other sections below.) Papers addressing dimensioning are amongst some of the earliest evolutionary telecommunications papers, including Davis & Coombs [137–139], Potter *et al.* [142] and Pan & Wang [141] (see §2.3.3 above).

Building on their earlier work Potter *et al.*, in three further papers [169–171], describe improvements to their battlefield communications network configuration work including efficiency gains, enhancements of the objective function(s) and the incorporation of additional heuristics into the hybrid GA. Davis *et al.* [172] apply a hybrid GA to obtain both link capacities and traffic routes in a network subject to link failures; their aim is minimum cost within a reliability constraint. Their

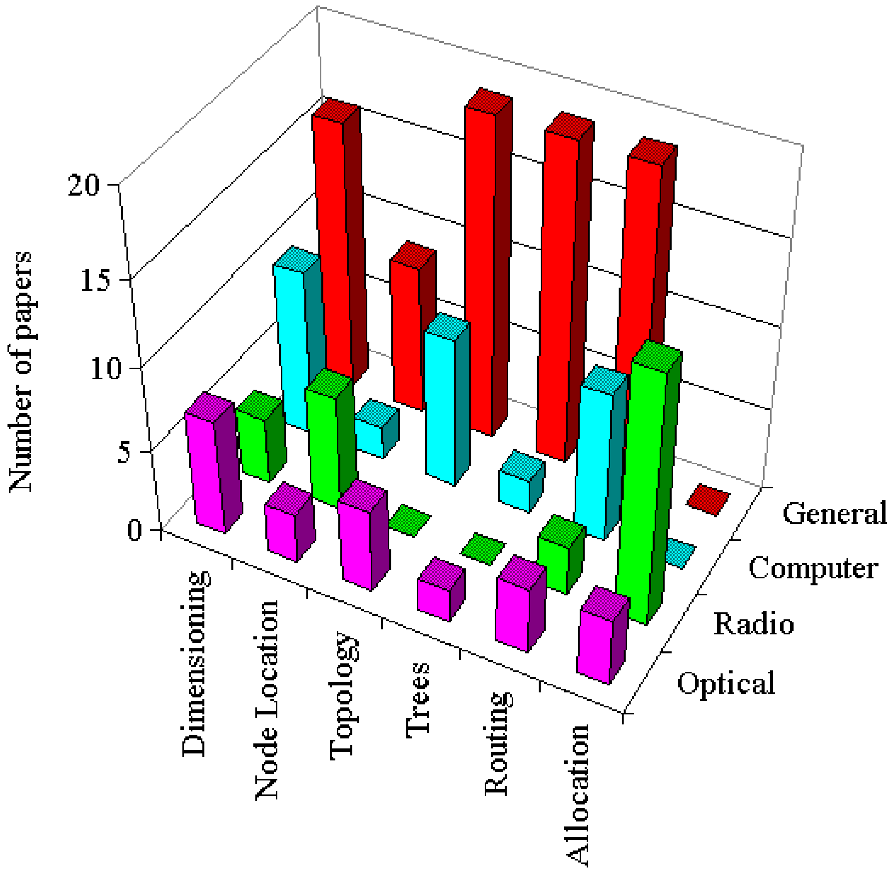


Figure 2.8: Evolutionary telecommunications papers – application groups

GA uses a combination of integer and order-based representations, problem-specific operators and a local greedy repair strategy. More recently, Garcia *et al.* [173] address the multi-period capacity expansion problem *i.e.* when and how much to increase network link sizes to meet forecast demand. They employ a steady-state GA with a hybrid integer-real representation combined with a variety of local search techniques.

Node Location

Network node location is tackled by 17 papers. Of these, a few also address topology design, and are discussed in the next section [161,162,174]. In addition, the work of Potter *et al.* [142,169–171] on battlefield location has already been discussed.

In a 1994 paper, Routen [175] considered the concentrator assignment, multipoint line topology[4] (*i.e.* constrained minimum spanning tree) and concentrator location problems in local access network design. His GAs use integer representation and some problem-specific operators. As well as reviewing earlier work on Steiner trees [176] and frequency assignment [177], Chardaire *et al.* [178] also tackled the concentrator location problem. They combined a bit-string GA for concentrator choice with a greedy heuristic for terminal assignment. Celli *et al.* [179] selected suitable multiplexor sites for metropolitan area networks (MANs) using a parallel GA with bit-string representation. Kumar *et al.* [180] developed their earlier topology work [145] (restated in [181]) to consider computer network expansion. Their GA both selects the new nodes and also determines their links to the existing network. To illustrate their distributed GA, Srivastava *et al.* [182] revisited Kumar *et al.*'s earlier work on both network topology design [145] and network expansion [180]. More recently, Webb *et al.* [183, 184] employed a bit-string GA with heuristic repair for backbone node selection in a ring-star transport network. They contrasted topologies resulting from both distance- and tariff-based cost models.

For radio networks, Gondim [185] has applied GAs to the location area problem in GSM cellular networks *i.e.* clustering cells around mobile switching centres. They used an elitist, integer-based GA with problem-specific recombination. Calégari *et al.* [186] used a multi-population GA to select basestation sites for a cellular network. Their GA, which is generational and employs a bit-string representation, clearly out-performed a uni-population variant. Finally, Tang *et al.* [187] applied a multi-objective GA to obtain both the number and location of basestations in a wireless local area network. They combined a hierarchical binary-real representation with appropriate genetic

[4] Denoted 'terminal layout' by Routen [175].

operators and Pareto ranking.

Topology

Network topology design is the focus of some 33 papers. Of these, three approach topology via routing [151–153], and are discussed below. Another three are early papers by Michalewicz [143], Kumar *et al.* [145] and Sobotka [146] (see §2.3.3 above). In addition, papers by Routen [175], Kumar *et al.* [180–182] and the more recent work of Webb *et al.* [183, 184] have already been considered in the context of node location.

For general networks, in 1997 Deeter & Smith [148] addressed minimum-cost topology design for small network sizes subject to an all-terminal reliability constraint. They determined not only which links to include, but also the level of reliability (and hence cost) associated with each. Their GA uses an integer representation, uniform crossover, mutation and rank-based selection. Dengiz *et al.*'s slightly earlier paper [188] also focused on minimum-cost topology design subject to a reliability constraint. In contrast, here all links shared a common reliability and their costs, while differing, were fixed at the outset. Consequently, only a bit-string GA was required, and they were also able to tackle far larger network designs by employing an approximate fitness function. Dengiz *et al.* developed their work in four further papers. In the first two [189, 190], they refined their estimate of fitness to give increasingly accurate assessment to the fitter individuals. As a result, they were able to attempt even larger networks. Then in [191, 192], while still retaining their focused fitness calculation, they obtained superior results by adopting a hybrid GA. This employs a variable-length integer representation, ranked selection, local-search mutation and repair of individuals with poor connectivity. Furthermore, although their major focus has been network trees, in 1994 Abuali *et al.* [193] tackled minimum-cost terminal assignment using a GAs with two different permutation encodings combined with different initialisation approaches, permutation crossovers and both fixed and adaptive mutation probabilities. The same authors [194] later applied a bit-string GA to the subset interconnection problem. This is a constrained minimum-cost network topology design problem, whereby every one of the given node subsets also has to be internally connected, as well as the network as a whole. Again they applied a variety of operators, demonstrating the superiority of their GA over problem-specific heuristics. Tanaka & Berlage [167] tackled the design of video-on-demand distribution networks, determining not only the network topology, but also which nodes would provide video storage. Their GA uses a binary matrix representation, as well as problem-specific crossover and repair. Meanwhile, Saha & Chakraborty [195] employed a bit-string GA to address the link enhancement problem *i.e.* adding new links to an existing network topology (*cf.* Kumar *et al.* [180]). However, their approach is somewhat simplistic, as they assume not only the cost, but also the profit resulting from the addition of any link is known in advance. Nakamura & Oda [196] also tackled link enhancement, but over multiple planning periods, however their focus was limited to installation costs. They combined an heuristic routing and restoration algorithm with a simple GA to select from a range of possible additional links.

For optical networks, Sinclair has worked on multi-wavelength all-optical transport networks, applying a simple bit-string GA [166], a hybrid GA [34] and, with Aiyarak & Saket, three different GP approaches [147] to mesh network design. While Sinclair *et al.*'s EAs only select which links to include in the network topology, routing and link dimensioning are accomplished by their fitness function, which also includes a connectivity constraint. Mikac & Inkret [157] developed Sinclair *et al.*'s work with bit-string GAs [166] to consider both minimum cost and minimum unavailability. Their algorithm employs two fitness functions: in odd generations it focuses on minimum cost, applying an unavailability penalty; whereas in even generations unavailability is the focus, with an associated cost penalty. In contrast, both Paul *et al.* [161, 162] and Brittain *et al.* [174] address constrained minimum-cost passive optical network (PON) topology design for local access[5]. However, while problem-specific representations are used by both, as well as problem-specific genetic operators by Paul *et al.*, only Brittain *et al.* provide full details of their algorithm. This employs a two-part chromosome comprising a bit-string and a permutation, with each part manipulated using appropriate standard genetic operators.

For computer networks, Hewitt *et al.* [150] studied minimum-cost mesh network design subject to both connectivity and packet-delay constraints. They applied a hybrid GA with a linear link-data-structure representation and problem-specific crossover and mutation. Later, a three-stage GA for minimum-cost design of a biconnected computer network including topology, routing and capacity assignment was described by Ko *et al.* [154]. Their GAs are tailored to each stage, employing bit-string, integer and hybrid representations, as well as suitable operators, respectively. However, Pierre & Legault employed a bit-string GA for computer mesh network topology design under connectivity and delay constraints [163,164]. Although their GA only directly selected network links, the fitness function included link capacity allocation (*cf.* Sinclair [166]). Finally, Qin *et al.* [165] also used a bit-string GA, this time for ATM mesh network topology design. Their algorithm considers

[5]Strictly, Paul *et al.* consider only *tree* topologies, but [161, 162] are discussed here alongside the other optical network papers.

capacity assignment in the fitness assessment, but makes no explicit calculation of network delay, and applies a biconnectivity constraint only by repair.

Trees

Network trees are the focus of 24 papers. Of these, the early paper by Hesser *et al.* [144] on Steiner trees, as well as the more recent work by Routen [175] on multipoint line topology design, and by Paul *et al.* [161, 162] on PON topology design, have already been discussed. In addition, papers on multipoint routing by Christensen *et al.* [197] and Zhu *et al.* [198] have been deferred to a later section.

While papers in this area address a number of aspects, nevertheless, it would appear that the central issue in successful application of EAs to network tree design is the encoding of the trees themselves.

Several authors have researched minimum Steiner trees. In contrast to Hesser *et al.* [144], who tackled arbitrary Steiner point locations, Kapsalis *et al.* [176,178] used a bit-string GA to focus on the more limited problem of selecting Steiner nodes from a given graph. Julstrom [199], however, tackled the minimum rectilinear Steiner tree problem with a hybrid integer-binary encoding and problem-specific crossover. His encoding exploited the constraints on Steiner point locations in the rectilinear problem, although empirical performance of the GA was poor compared to problem-specific heuristics. Subsequently, Esbensen & Mazumder returned to the selection of Steiner nodes for an arbitrary given graph [200–202]. Their hybrid GA [200] uses a modified bit-string encoding (allowing inversion) and repair to bias a simple problem-specific heuristic. In a subsequent paper, Esbensen [201] applied their GA to a wide range of large test networks, demonstrating its superiority over both Kapsalis *et al.* [176] and several powerful problem-specific heuristics. Then in [202], Esbensen improved the GA by adopting a steady-state population dynamic and bit-string encoding (hence no inversion), employing uniform crossover and including both adaptation of the mutation rate and local search.

Considering trees more generally, Abuali *et al.* have published some five papers [203–207] in addition to the two on network topology design already reviewed. In the first [203] in 1993, they considered multipoint line topology design *i.e.* the constrained minimum spanning tree problem, investigating eight different GAs. These were formed from combinations of two encodings (integer and permutation), with two different crossover operators for each encoding, and both steady-state and generational approaches. Their permutation-encoded GAs performed better than the well-known Esau-Williams heuristic [10, 11], particularly for larger networks. Others applying GAs to this problem (independently of Abuali *et al.*) include Routen [175] in 1994, who provided only a summary of his disappointing results using an integer-based encoding, and Zhao *et al.* [208] with a more successful hybrid GA in 1995. This used an object-oriented representation, as well as both problem-specific recombination and mutation incorporating repair. Once again, they were often able to obtain better results than either the Prim or Esau-Williams heuristics [10,11]. Meanwhile, in 1994 Abuali *et al.* [204] moved to consider the probabilistic minimum spanning tree problem *i.e.* finding the minimum-expected-cost spanning tree in a network where each subset of nodes only require to be connected with a certain probability. They adopted a Prüfer number encoding and tested a wide variety of operators. However, that same year Palmer & Kershenbaum [159, 160] argued against Prüfer encoding (as well as predecessor-vector encoding), instead using a GA to select node-cost biases for a standard minimum spanning tree heuristic in a capacitated network (*cf.* Esbensen & Mazumder [200] above). They obtained results superior to those of a problem-specific heuristic, particularly for more unusual test problems. Abuali *et al.* [205] also later abandoned Prüfer encoding, instead comparing two graph-theoretical integer encodings, determinant factorisation and cycle basis, for the probabilistic minimum spanning tree problem. In a subsequent paper [207], Abuali *et al.* tested their determinant-factorisation encoding against both Prüfer encoding and Palmer & Kershenbaum's node-cost bias encoding [159, 160]. Overall, both determinant factorisation and node-cost bias encoding performed well, although the latter appears superior for larger networks, as well as having the important advantage of conceptual simplicity. A further paper by Abuali *et al.* [206] addressed the more constrained problem of three-star trees, where all tree branches consist of a chain of three nodes attached to the root. Once again, their determinant encoding performed well, although an alternative permutation-based encoding used to bias a greedy heuristic appears to be superior. The minimum spanning tree problem in a capacitated network was tackled independently by Berry *et al.* [209] in 1995. They adopted the predecessor-vector encoding deprecated by Palmer & Kershenbaum [159], although avoiding its limitations through heuristic recombination and mutation. However, their paper offers little experimental evidence in support of their GA. Also that year, Walters & Smith [168] independently developed a hybrid GA for (directed) minimum spanning trees in capacitated networks using a problem-specific integer encoding combined with heuristic recombination and mutation (*cf.* Dengiz *et al.* [191, 192]). Again, it is unclear how their approach would compare to those of other researchers. Finally, Elbaum & Sidi [210, 211] considered the topological design of a local area network (LAN) for minimum average network delay. As the

LAN is divided into clusters interconnected by transparent bridges, the network topology is limited to a spanning tree. Individuals are encoded using a hybrid three-part chromosome to define the network tree (based on a 'virtual Huffman tree'), match clusters to tree nodes, and assign users to clusters.

Routing and Restoration

Routing and restoration is tackled by 32 papers. Two of these are early papers by Cox *et al.* [140] and Pan & Wang [141] (see §2.3.3 above). Two more, by Davis *et al.* [172] and Ko *et al.* [154], have already been discussed in the contexts of dimensioning and topology, respectively. In addition, three papers on optical routing and wavelength allocation, by Tan & Sinclair [29] and Sinclair [35, 36], are discussed in a later section.

As well as Davis *et al.* [172], other routing papers from 1993 were by Munakata & Hashier [158], Sinclair [27] and Shimamoto *et al.* [212]. To address the maximum flow problem in a directed capacitated graph, Munakata & Hashier [158] combined an integer representation of the flow matrix with both a problem-specific recombination and an adaptive problem-specific mutation. While their hybrid GA obtained good results, its computational complexity is relatively poor compared with conventional heuristic algorithms. Meanwhile, to design static-routing tables for minimum network grade-of-service (NGOS) in an unreliable circuit-switched network, Sinclair [27] used a compact bit-string encoding of the routing tables with conventional operators, but employed a penalty function to eliminate infinite-routing loops. As with Munakata & Hashier [158], his algorithm also proved uncompetitive with an earlier heuristic. In contrast, Shimamoto *et al.* [212] tackled dynamic routing in reliable circuit-switched networks. An individual in their algorithm was represented as a string of indices into a look-up table of the k-shortest paths for each node pair (*cf.* Pan & Wang [141]). They used a form of steady-state GA to exploit the relatively slow changes in traffic distribution, and obtained grade-of-service that compared well with a fixed routing algorithm (although in both cases only a single route was available between node pairs).

In 1995, Mann *et al.* [213] addressed static routing, using a representation very similar to that of Shimamoto *et al.* [212], but encoded as bits rather than integers. Their aim was to minimise network cost while also balancing the traffic load on the network links. Further detail of their work was given in [214]; they made a detailed comparison with a variety of algorithms, both GA and SA, obtaining similar results from the best variant of each, although with much less computational effort by SA. A subsequent study by Mann & Smith [156] focused on routing in ring networks, aiming for balanced load and hence a minimal ring capacity requirement for rerouting in the event of failure. Their representation consists of one bit per node pair, with each bit simply indicating whether the traffic is routed clockwise or anti-clockwise. They again applied a wide range of algorithms, both GA and SA, this time finding GA to be more robust to algorithm and parameter choice.

Two papers by Hoelting *et al.* [215, 216] in 1996 addressed variations on more standard routing problems[6]. Their first study [215] considered finding an investigator tour for a network, *i.e.* the route taken by a network management agent attempting to locate a failed link by travelling to and interrogating an appropriate subset of the network nodes. Their permutation-based GA was shown to obtain superior tours to an heuristic approach. In the second study [216], Hoelting *et al.* tackled the minimum broadcast time problem, *i.e.* where a message must be distributed to all the nodes in a network in a minimum number of time steps, but at any one time step each node can only transmit the message to one of its neighbours. They applied a permutation-based GA, and once again showed it to be superior to a problem-specific heuristic. Almost the same group of authors went on to tackle the point-to-multipoint routing problem, *i.e.* routing a set of circuit-switched connection requests, each of which has a single source and one or more destinations [197,198]. First Christensen *et al.* [197] extended the earlier work of Cox *et al.* [140] to provide dynamic-anticipatory routing for multiple point-to-multipoint connections. They combined a permutation-based GA to order the requests and an heuristic Steiner tree algorithm to find their minimum cost routing trees. Subsequently, Zhu *et al.* [198] extended their work to also consider subsets of the outstanding requests where routing the full set is infeasible.

An unusual variation on applying GAs for routing was considered by Servet *et al.* [217], who used both binary and real-valued GAs with a variety of fitness functions to invert a single-moment traffic model of a network and so estimate the size of traffic streams starting from only the offered (or carried) traffic on the links.

Several authors in recent years have applied EC to network restoration rather than just primary routing. In 1996, Huang *et al.* [151] evolved two minimum-cost node-disjoint routes between one or more node pairs in a communications network, and used these to guide both topology selection and link dimensioning. Their algorithm used a node-index representation with heuristic recombination and mutation. In two later papers [152, 153], they explored a variety of parallel computing approaches to enhance the speed of their GA. In 1997, Bentall *et al.* [218] used a GA with a two-

[6]These two papers share common authors, Schoenfeld and Wainwright, with the papers by Abuali *et al.* on network trees [203–207] and network topology [193, 194].

dimensional order-based representation to obtain restoration paths for a heavily-loaded network. Their aim was not so much to apply the GA to real-time restoration, but rather for the results obtained to serve as a benchmark in assessing rapid heuristic algorithms with only local knowledge. Kozdrowski *et al.* [155] used a hybrid GA to allocate traffic streams and link capacities in a network subject to both link failures and a number of possible constraints on restoration paths. Meanwhile, Kirkwood *et al.* [219, 220] applied GP to obtain both primary and restoration paths in a network subject to single-link failures. They evolved programs that determined near-optimal routes from one node to all others under all failure conditions. However, their approach does not scale well to larger networks.

Focusing on computer networks, in two 1995 papers, Carse *et al.* [221, 222] evolved a network routing agent, employing a Pittsburgh-style fuzzy classifier system, for adaptive distributed routing in a datagram packet network. They applied their algorithm to a small network, and obtained reduced mean packet delay, for realistic measurement intervals, compared to a lowest-delay routing algorithm. Then in 1997, Munetomo *et al.* [223] tackled much the same problem, real-time dynamic source routing of packets, but in far larger networks, by employing path-based genetic operators to directly manipulate alternative routes in the node routing tables. In comparison with two established Internet routing algorithms, their scalable approach demonstrated both considerably reduced delay and more balanced link loads, particularly under heavy traffic conditions. In contrast, He [149] addressed static route and link capacity allocation in large packet-switched networks. His GA encodes individuals as a string of indices into look-up tables of link capacities and routes, independently replicating the representation used much earlier by Pan & Wang [141]. Also in 1997, Tanterdtid *et al.* [224, 225] applied Shimamoto *et al.*'s GA for dynamic routing in reliable circuit-switched networks [212] to the related problem of virtual-path routing in ATM networks. Finally, Markaki *et al.* [226] developed a hybrid GA with a two-dimensional binary encoding and local repair for connection-oriented channel selection and reuse in a DQDB MAN. Their GA allowed many more active connections, and obtained higher channel reuse, than an earlier graph-colouring based approach.

Call Admission

A series of three papers by Yener & Rose [227–229] are the only ones to address call admission using EAs, in their case for cellular radio networks. In 1994 [227] they evolved admission policies for a small network using a bit-string GA. The admission policy decides which classes of call to admit to each cell in each network state. Their GA compared well to the 'admit if possible' heuristic, and employed a random codeword based bit-string compression technique to cope with the large bit strings that would otherwise be required. In their two subsequent papers [228,229] they considered admission policies based on local rather than global state information, and were thus able to tackle larger networks. Interestingly, examination of the structure of the evolved policies themselves suggested a novel heuristic local admission policy to Yener & Rose.

Wavelength Allocation

Only four papers consider wavelength allocation in optical networks. The earliest, by Tan & Sinclair [29] in 1995, describes several GA-based solution approaches for multi-wavelength all-optical transport networks. The most successful of these combines a bit-string GA for route selection with an heuristic for wavelength and fibre allocation. Subsequently, Sinclair [35, 36] developed a hybrid GA for routing, fibre choice and wavelength allocation using both an object-oriented representation and problem-specific operators. He applied it first to minimise the number of different wavelengths required [35], and then to minimise a simple model of network cost [36]. In contrast, Abed & Ghanta [230] applied a bit-string GA to design the logical topology of a local area network (LAN) or MAN using wavelength allocation on a single optical fibre.

Frequency Assignment

Frequency assignment in radio networks is the topic of 15 papers. Of these, seven have included Hurley as co-author: with Crompton & Stephens [231–233], with Smith [234] and Thiel & Smith [235], and most recently, with first Valenzuela & Jones [236] and then Valenzuela & Smith [237]. In the first of these, and also the earliest paper in the area, Crompton *et al.* in 1993 [231] applied a multi-population GA to frequency assignment for the air-ground-air problem using an integer representation. Their aim was to minimise the number of constraint violations and hence interference. Then, for the same problem, they reported improved results in [232] with a transmitter-set representation combined with heuristic recombination. Subsequently, in [233] they demonstrated the superiority of their GA to a backtracking heuristic even on an equal execution time basis. However, Hurley & Smith [234] demonstrated an SA [5] approach to be superior to the same GA, primarily because their SA was able to use an incremental fitness function, and hence perform up to 1,000 times more evaluations in the same execution time. Nevertheless, despite sharing the advantage of incremental fitness evaluation, TS [6, 7] was found to be inferior to SA by Hurley *et al.* [235]

(although still better than their GA). Moving to a slightly different problem in 1997, Valenzuela *et al.* [236] aimed to minimise the span (*i.e.* range) of frequencies required to satisfy all constraints, resulting in zero interference, in both regular and irregular cellular networks. Their steady-state GA biases a greedy assignment heuristic[7] by providing an initial ordering of transmitters. In addition, they subsequently demonstrated [237] the superiority of their GA over earlier SA and TS approaches to the same problem (although *cf.* Hurley *et al.* [235] above).

Another early paper on frequency assignment was in 1994 by Cuppini [238], representing an independent entry into the area. However, her bit-string algorithm uses no crossover, although very high rates of mutation, and is only applied to small problems. The following year, Kapsalis *et al.* [177, 178] used a bit-string GA for realistic problem sizes, testing a wide range of heuristic operators and adaptive fitness functions (*cf.* Crompton *et al.* [232]). Their aim was to minimise both interference and the number or span of frequencies. Subsequently, Kapsalis & Smith [239] applied a meta-GA to select from an even wider range of operators, as well as to tune the parameters of their primary GA. Although improved results were obtained, they were discouraged by the extensive execution times required. More recently, Crisan & Mühlenbein [240] applied a breeder GA to digital cellular networks. Their frequency assignments are represented as integers and several heuristic operators employed in evolving a minimum-interference solution. Their problem formulation is more complex than earlier authors (*e.g.* Crompton *et al.* [232] and Kapsalis *et al.* [177]), including not only hard constraints, but also interference probabilities used to define an 'interference cost'. Also in 1998, Ngo & Li [241] used a compact bit-string representation and applied problem-specific operators and local search to minimise interference and hence obtain a feasible assignment (although this is a more limited objective than other recent authors).

In addition, by incorporating a detailed propagation model and fuzzy logic in fitness assessment, Kaminsky [242] was able to tackle the hourly assignment of frequencies to military radio networks using a steady-state GA (*cf.* Potter *et al.* [142, 169–171]). In contract, Sandalidis *et al.* [243] addresses dynamic frequency assignment for cellular mobile radio. They applied a $(1,\lambda)$ ES with a bit-string representation and mutation based on channel swapping (thereby avoiding the need to repair individuals following GA crossover). Their results were superior to four problem-specific heuristics and a neural-net based approach, although the question of real-time operation of their algorithm was not addressed.

[7] *cf.* Esbensen & Mazumder [200–202] and Palmer & Kershenbaum [159, 160] above

2.3.5 Algorithms

Clearly, evolutionary telecommunications has been addressed using a wide-variety of algorithms (see Table D.1). Solution representations surveyed included binary, integer, real, permutation and tree-based (GP) encodings, as well as a few hybrid approaches, each modifying or combining more straightforward encodings, or in some cases employing object-oriented models. Population dynamics, although dominated by generational and elitist approaches, also included some steady-state, multi-population and breeder algorithms. Authors employed a variety of selection methods, and although favouring the random methods of fitness-proportional, tournament or rank-based selection, some ES-type deterministic selection was also used. Not surprisingly, many different genetic operators were applied, including single- and multi-point crossover, uniform crossover, permutation crossover, heuristic recombination in hybrid algorithms, tree crossover (GP), a wide range of mutation operators, inversion, repair and local search.

Overall, a few observations can be made. Firstly, biasing a greedy heuristic, either by using a GA to select its starting point, or to modify the local fitness landscape seen by the heuristic, seems a promising direction for future work (see, *e.g.* [159, 160, 200–202, 236, 237]). Secondly, while there are arguments for developing EAs elsewhere as general problem-solving tools, for the complex and demanding problems addressed by evolutionary telecommunications, incorporating domain knowledge seems vital, rather than just using, say, a simple GA (see §2.2.3). However, far too few of the researchers surveyed seem to be investigating and employing hybrid GAs. Next, there must be more comparative work. Many, although not all, of the papers examined describe a single EA for the solution of the problem presented, without apparently investigating any alternatives. Authors may well be going through an algorithm development and parameter tuning process, but this is not being reported. Instead, their ‘final’ algorithm is simply presented as a *fait accomplie.* In addition, there are often no comparisons with other non-EA algorithms. Now, while there are potential difficulties with such comparisons, including only benchmarking against a ‘straw man’ (*e.g.* random search, or a poorly-implemented modern heuristic search technique), nevertheless, too few authors are making the attempt. Finally, two further problems remain: citations and reproducibility.

Citations & Reproducibility

It is apparent from Table D.1 that papers in evolutionary telecommunications rarely cite earlier work in the field. Indeed, if citations of each author’s own prior papers are discounted, remarkably few remain. Examination of the individual papers reveals that many are ‘first entries’ into the evolutionary telecommunications domain. A typical paper will thus cite earlier research on the specific problem tackled, make reference to one or more books on EAs (such as those by Goldberg [2] or Davis [9]), before describing the particular EA used and reporting results. Little reference, if any, is made to any earlier work in applying EAs to this or similar problems. Clearly, given that there is now more than ten years of research reported in over 100 papers, such an approach fails to develop on the experience of previous researchers. However, this is not an uncommon omission in EC in general, as David Fogel has observed [4]: "Evolutionary computation has a long history. Unfortunately, this history is virtually unknown to many practitioners in the field."

In addition, many of the papers in the survey do not present sufficient information to allow the algorithm to be reproduced. Authors need to include detailed, unambiguous information on solution representation, population dynamics, selection methods, genetic operators and the parameters used. Without these details, it is very difficult to compare one EA with another.

Thus, as well as finding earlier work to cite and compare current work with, evolutionary telecommunications authors must try and ensure the reproducibility of their own work, so that others can cite and compare their work with it in turn. Indeed, it is only as researchers begin to build on previous work in these and other ways, that evolutionary telecommunications will begin to mature as a field.

2.4 Applying Genetic-algorithm/Heuristic Hybrids

To complement the more comprehensive survey of evolutionary telecommunications above, in this section a brief overview is given of some prior work from other areas using GA/heuristic hybrid algorithms, the solution approach adopted in the thesis, and the basics of which were explained in §2.2.3 above.

In 1989, Jog *et al.* [244] tackled the travelling salesman problem (TSP) using a generational GA/heuristic hybrid algorithm, and building on a number of even earlier papers in this area. Their algorithm used the usual permutation representation, but employed both an heuristic crossover operator and two local-search mutation operators applied with fixed operator probabilities. Their experiments focused on the effects of including or excluding operators from the hybrid, as well as varying the population size. They concluded that larger population sizes, using problem-specific crossover and including local-search heuristics of increasing sophistication were all beneficial to the final solution quality.

An early and influential reference in this area, as already noted, is the *Handbook of Genetic Algorithms* edited by Lawrence Davis in 1991 [9]. After a step-by-step development of a GA/ heuristic hybrid from a simple GA, the book includes a wide-ranging collection of application case studies of (mainly) hybrid GAs. These include the parametric design of aircraft [245], dynamic anticipatory routing in circuit-switched telecommunications networks [140], robot arm trajectory generation [246], international security modelling [247], aeroplane control strategy design [248], neural network design [249], industrial plant control [250], multiple-fault diagnosis [251], conformational analysis of DNA [252], interpretation of passive sonar traces [253], engineering design optimisation [254], scheduling [255] and the TSP [256]. Between them, the case studies employ a wide variety of representations and operators.

For example, the chapter by Bramlette & Bouchard [245] from [9] concerns the parametric design of a modern fighter aircraft. Their design required eight parameters to be optimised, including fuselage width, height and length, wing loading and engine size. They investigated not only an integer-coded generational GA, but also a wide variety of stochastic approaches including various types of SA, hill climbing and random search, used both individually and in combination. Overall, they found that iterated SA, and a hybrid GA using stochastic hill climbing as the mutation operator, obtained the best results on their test problem.

Also in [9], Davidor [246] developed trajectories for robot arms using a hybrid GA which represents the trajectories as order-dependent variable-length strings of (intermediate) arm positions, with each position specified by the inter-link angles in the arm configuration. A problem-specific two-point crossover is introduced which chooses two random arm configurations as crossover points in one parent (*cf.* Munetomo *et al.* [223]). It then locates the two most similar (*i.e.* analogous) configurations in the other parent. Then the configuration sequences between these points (and including the second crossover point) are exchanged between the parents to form the children. Mutation is guided by problem information to focus on those points in an arm trajectory that are least fit. In addition, two further problem-specific mutations are used that can add (duplicate) and delete arm configurations from the trajectories. In experiments on one particular target trajectory, Davidor's hybrid GA performed well in comparison with both random search and hill climbing, and was found to be further improved by directly incorporating hill climbing within the GA.

As a third example from [9], Grefenstette [248] sought to develop strategies for sequential decision tasks in a dynamic system, specifically evasive maneuvers allowing a plane to avoid a pursuing missile. Within his generational GA, each strategy is represented by a set of condition-action rules that relate the sensor inputs (missile range, missile bearing, missile speed, *etc.*) and the system response (*i.e.* the turning rate of the plane). The problem-specific crossover operator exchanges complete rules between individuals, whilst trying not to separate successful rule sequences. Of the two mutation operators provided, one specialises existing rules, and the other randomly modifies an individual clause within a rule. In addition, the GA can be seeded with rules generated by domain experts that are combined with randomly-generated rules within the individual strategies that form the initial population. In experimental runs, the GA performed well providing a high probability of missile evasion, although no direct comparison was made by Grefenstette with other non-trivial approaches.

More recently, in their 1997 paper, Schnecke & Vornberger [257] address constrained placement problems—both the facility layout problem and VLSI macro-cell layout generation—using a hybrid GA. Their steady-state algorithm uses a binary slicing-tree representation to express the relative placement of the blocks, three problem-specific mutations, and a form of gene-pool recombination. It employs a multi-population approach, with each sub-population using parameter adaptation, including the operator probabilities. Their hybrid GA produced competitive results for VLSI macro-cell layout generation, without the use of sophisticated routing algorithms, and obtained the best results yet reported on large facility layout problems.

A hybrid GA for computer-aided process planning for job shop machining, considering operation selection, machine selection, setup selection, cutting tool selection, and operations sequencing simultaneously, was presented by Zhang *et al.* in [258]. Their representation was a ordered list of operations, each one providing details of the corresponding machine, tool and tool-approach direction choices. A problem-specific form of cyclic crossover and three mutation operators for machines, tools and tool-approach direction were applied using fixed operator probabilities. For the one complex problem examined, Zhang *et al.* found that their hybrid GA was superior to an heuristic approach.

An adaptive evolutionary planner/navigator for mobile robots was described by Xiao *et al.* [259] in 1997. Their system uses the same GA/heuristic hybrid algorithm for both offline planning and online navigation. The steady-state hybrid represents paths as a linked list of knot points (*i.e.* intersection points) between straight line segments. It uses a problem-specific crossover and seven mutation operators, with their probabilities adapted during the course of a run (see below). Their system has been developed over a period of time, and demonstrates robust online and offline performance over a wide range of challenging environments. Their work was subsequently adapted by Śmierzchalski & Michalewicz [260] for modelling ship trajectories in collision situations. The path representation was extended to include ship speeds on the line segments, the concept of time introduced to allow for moving obstacles, due allowance made for dynamic as well as static constraints, and a new speed mutation operator added. However, apart from these, the earlier planner/ navigator was largely adopted unchanged, except that for simplicity Śmierzchalski & Michalewicz omitted operator-probability adaptation, preferring instead to use equal fixed operator probabilities.

In 2000, Magyar *et al.* [261] described a GA/heuristic hybrid for the three-matching problem. This problem requires a set of nodes to be clustered in disjoint groups of three (*i.e. triplets*), with the cost of each triplet simply that of the minimum-spanning tree required to interconnect its nodes, and the overall cost taken to be the sum of the triplet costs. Magyar *et al.*'s hybrid uses a permutation representation and incorporates several problem-specific recombination and local-improvement operators. It employs operator-probability adaptation (see below), and the results obtained were better than those of a local-search heuristic, SA and a grouping GA over twenty large problem instances.

Although this section has focused on GA/heuristic hybrids, it should perhaps be mentioned that many authors have hybridised GAs with other modern search heuristics. As one example, a paper in 1999 by Bhandarkar & Zhang [262] addresses the problem of image segmentation using EC. As well as developing a problem-specific generational GA, Bhandarkar & Zhang hybidise this with three different stochastic annealing algorithms in turn: SA, microcanonical annealing (MCA) and the random cost algorithm (RCA). In the GA, a two-dimensional integer representation is used, combined with problem-specific crossover, mutation and a local-search operator. All three operators follow deterministic changes in their operator probabilities during a run, with local search and mutation increasing, and crossover decreasing. The hybridisation is achieved by wrapping a stochastic annealing replacement mechanism around the GA framework (excluding the local search operator). Thus children that had been created using GA crossover and mutation only replace their parents if they satisfy the appropriate annealing condition. Overall, they found that the three hybrids all produced higher quality results than the GA, particularly so for the SA-GA combination.

Overall, as in evolutionary telecommunications, the GA/heuristic hybrid approach has been readily applied to a wide range of problem domains, only a few of which have been considered here. Authors have shown considerable ingenuity in developing problem-specific algorithms for difficult design and optimisation tasks, and the GA/heuristic hybrid approach clearly has considerable promise.

2.5 Operator-probability Adaptation

One aspect of GA/heuristic hybrid algorithms explored in this thesis is their potential use of operator-probability adaptation mechanisms. Consequently, in this section some attempts to classify the many approaches to parameter control in EAs will be mentioned, as well as briefly examining a selection of prior work, taken from areas other than evolutionary telecommunications, employing such mechanisms.

A detailed survey and classification of parameter setting was recently completed by Eiben *et al.* [263]. They based their work in part on their own earlier, although more limited, classification scheme [264]. In [263], they argue that there are only two main criteria for classifying adaptation mechanisms: what is changed, and how the change is made. For the first, they considered parameters controlling representation, evaluation function, mutation operators and their probabilities, crossover operators and their probabilities, parent selection, replacement operator (survivor selection) and the population itself as potential targets for adaptation. As to how the changes are made, they divided *parameter setting* into two main areas: *parameter tuning* and *parameter control*. In the former case, parameters remain fixed throughout the course of the EA run, although they can be tuned between runs by the experimenter. In the latter approach, however, the parameter values change during the run, with this mechanism further divided into three categories of parameter control. In the *deterministic* method, the changing parameter value follows a preset schedule—there is no feedback from the evolutionary process. With *adaptive* control, there is feedback, but this only changes the parameter value using some heuristic, rule-based approach. The final possibility is *self-adaptation*—the parameter values participate in the ongoing evolutionary process; thus the adaptation mechanism is itself evolutionary.

An earlier and more focused classification by Tuson [265, 266], considered only operator-probability adaptation algorithms. In his study, he classifies these into two groups: co-evolutionary methods (equivalent to Eiben *et al.*'s self-adaptive parameter control), that encode operator probabilities into each member of the population; and learning-rule methods (equivalent to adaptive parameter control), that adapt the probabilities based on measurement of the operator productivities.

Turning now from surveys to individual papers employing operator-probability adaptation[8], an early paper by Fogarty [267] in 1989 studied varying mutation probability for an industrial

[8]Here the focus is primarily on the adaptive approach and algorithms from the GA 'tradition', rather than ESs, EP or even GAs which employ self-adaptation to control operator probabilities.

optimisation problem, *i.e.* minimising combustion stackloss in the common flue of multiple burner furnaces. He compared two fixed mutation schemes (one of which applied reduced levels of mutation to more significant bits) with two corresponding deterministic schedules that gradually reduced mutation throughout the run. He found that deterministic variation of the mutation probability improved the GA's online performance in both cases when starting from a conservative initial population.

In the same year, Davis [268] presented an operator-probability adaptation mechanism incorporated into a steady-state GA/heuristic hybrid with an integer representation and five problem-specific operators. The adaptation mechanism was found to provide improved performance over both naive and tuned fixed probabilities, and to be competitive even with a deterministic schedule derived from a substantial number of experimental runs.

Davis subsequently described his approach in more detail in [9]. The operator productivity is computed by crediting operators with any improvement over the best member of the population; a decreasing fraction of this credit is also passed back to the operators responsible for creating the individual's parents, and so on, to a given depth. After a defined interval, the productivity of an operator is taken to be its total credit over the interval divided by the number of individuals it has created. Operator probabilities are then adjusted by reassigning a fraction of their probabilities in proportion to their productivity. Davis also mentions the possibility of preventing an operator's probability falling below an assigned minimum value, as well as the potential of his approach to measure the productivity of individual problem-specific heuristic operators.

Rather than using a credit-assignment approach, in 1994 Srinivas & Patnaik [269] adapted both crossover and mutation probabilities in a generational binary-string GA such that the smaller the gap between the best-in-population fitness and the average fitness, the higher the operator probabilities. However, to preserve the better individuals, the probabilities are also reduced the nearer an individual is to the best-in-population fitness. In particular, the mechanism ensures the best individual is (almost) always preserved unchanged (although a small background level of mutation prevents premature convergence). Their experimental results show that across a range of test functions their adaptive algorithm was in most cases significantly better than a GA with fixed operator probabilities, showing both more rapid convergence and a reduced tendency to get trapped in local optima.

In 1995, Julstrom [270] presented ADOPP, an operator-probability adaptation mechanism for steady-state GAs, in which all operators back to a given depth in an operator tree are credited with a decreasing fraction of any improvement over population median fitness. At all times, actual operator probabilities (and not just a fractional part) are determined by their recently earned credit. However, his work was limited to two-operator GAs, and provided no comparison with fixed operator probabilities.

Tuson & Ross [266, 271] describe an investigation of Corne's COBRA (COst Based operator Rate Adaptation) algorithm, which instead of adjusting individual operator probabilities, assigns operators to a fixed set of probabilities according to their operator-productivity rank over a given interval. The operator productivity is determined by the average improvement of a child individual over its parent(s). Although COBRA has shown itself to be beneficial for timetabling problems, Tuson & Ross found it provided only equal or worse solution quality over a wide range of other test problems, compared with carefully chosen fixed operator probabilities (for both steady-state and generational GAs). They concluded that operator productivity (at least as Corne defines it) may misguide COBRA as, in some cases, although an operator's productivity is high, its preferred probability is low.

In a 1997 paper, Xiao *et al.* [259] describe an adaptive evolutionary planner/navigator for mobile robots (see §2.4 above). Their adaptation algorithm not only takes account of improvements in path fitness over its parent(s) (*cf.* COBRA [271]), but modifies this by the time taken to execute an operator and any change in the number of line segments in the path (as more line segments require both more memory and processing time). Operator probability in one interval of a given number of trials directly reflects the operator's relative figure of merit over the previous interval. Overall, they found this adaptive algorithm gave better results than either equal or pre-tuned fixed operator probabilities.

Most recently, Magyar *et al.* [261] incorporated an operator-probability adaptation method into their GA/heuristic hybrid for the three-matching problem (see §2.4 above). Their GA is generational and the operators obtain credit by improvements of the children over their parents (*cf.* COBRA [271]), but they are also penalised to an equivalent extent if there is no improvement. However, no credit is given to chains of operators (*cf.* Davis [9]), although the credit (or debit) obtained by an operator is moderated by the time taken to execute it. The actual probabilities are adjusted using an exponential moving average similar to Davis' scheme [9]. Whilst Magyar *et al.* did obtain slightly poorer results than with a non-adaptive, but finely-tuned, set of fixed operator probabilities, this was only by a very small margin, and at a considerable saving in computer time over the substantial effort involved in parameter tuning.

Overall, while a wide range of different mechanisms have been proposed for operator-probability adaptation, in general authors seem to consider such a mechanism advantageous for GAs. In addition, although this approach has not always proved superior to carefully pre-tuned probabilities, nevertheless the advantages in computer time are clear, with one run using an adaptive mechanism producing results (almost) equivalent to the many runs required for parameter tuning.

2.6 Summary

This chapter has provided a brief overview and historical perspective on EAs, a tutorial introduction to GAs, as well as three literature surveys. The overview considered early work in the field, GAs, EP, ESs, GP, CSs and EH, as well as conferences and books on the area. The tutorial provided background on both GAs in general and Goldberg's Simple Genetic Algorithm in particular before introducing a central theme of the thesis, GA/heuristic hybrids. The first survey gave near-exhaustive coverage of evolutionary telecommunications, *i.e.* the application of EAs to the telecommunications problem domain, up to December 1998. It concluded with some comments on promising directions for future research and encouragement to future authors in regard to both citation of earlier work and the reproducibility of their research. The two remaining sections gave more limited surveys of prior work on GA/heuristic hybrids and operator-probability adaptation, respectively, in areas other than evolutionary telecommunications. Overall, the chapter has given the context for the evolutionary aspects of the author's research, as well as establishing the foundation on which it was built. Accordingly, in the next chapter, the focus moves to an overview of the author's software toolset for optical network optimisation, modelling and design (NOMaD).

Chapter 3

NOMaD

By the roadside you sat waiting for lovers,
sat like a nomad in the desert.
Jeremiah 3:2b, NIV

This chapter provides an overview of a software toolset for optical network optimisation, modelling and design (NOMaD) [31, 32] developed by the author. The toolset has been used by the author to support research into the application of genetic-algorithm (GA)/heuristic hybrid optimisation techniques [9] to optical network design, the topic of this thesis, as well as by others[1] for heuristic-based optical network design in three research projects. These included two under the European Commission funded research programme in Advanced Communications Technologies and Services (ACTS) [272, 273]: WOTAN (Wavelength-agile Optical Transport and Access Network) [125] and OPEN (Optical Pan-European Network) [18, 274]; and the Fujitsu Telecommunications Europe Ltd. (FTEL) "Future Broadband Networks" project [20–24].

Much of the material presented in this chapter was initially published by the author at the 12th UK Performance Engineering Workshop [31], when NOMaD was in the early stages of development, and then more recently at an IEE Colloquium on Multiwavelength Optical Networks: Devices, Systems and Network Implementations [32] (both reprinted in Appendix A).

The chapter is divided into ten sections: §3.1–3.3 describe in turn NOMaD's application areas, design metrics and solution approaches. Then §3.4 presents the overall architecture of the software. The next three sections, §3.5–3.7 describe different layers of that architecture: the object-oriented network toolset, the scripting layer and the graphical user interface (GUI). Next, in §3.8, the software testing & verification approach is reviewed. Then section (§3.9) gives a brief overview of

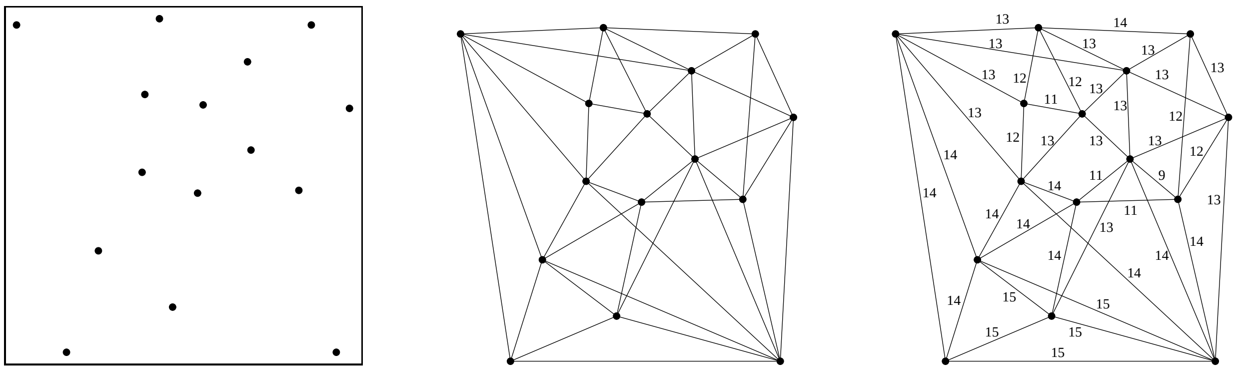
[1] Katerina Proestaki for the WOTAN and Fujitsu projects; Noel Parnis for OPEN

the main application program used to obtain the results reported in Ch. 5. Finally, the chapter is summarised in §3.10.

3.1 Application Areas

The design problems that NOMaD addresses are primarily those presented by optical transport networks over national or international distances. NOMaD's network model allows for a wide-range of network topologies, such as regular (*e.g.* torus, hyper-cube, *etc.*), arbitrary-mesh and (multiple) ring networks, as well as consideration of both 'flat' and hierarchical approaches. In addition, the network model can represent the cable, fibre and wavelength-routed structure of multi-wavelength all-optical transport networks. Two key application areas are directly supported: (i) network topology design, and (ii) routing, fibre choice and wavelength allocation.

Network topology design determines which node pairs are to be directly connected by a link (cable), given the node locations and static traffic requirements. However, while the network model supports both primary and restoration paths when used for topology design, fibre choice and wavelength allocation are not included. As an example, Fig. 3.1(a) shows a 15-node network design problem (this is actually Problem 1 from §F.2), for which an example flat mesh network topology design has been developed, given in Fig. 3.1(b).



(a) Network design problem (b) Mesh topology (c) Wavelength allocation

Figure 3.1: Example network design problem, mesh topology & wavelength allocation

Starting from a network topology, the second application area is routing, fibre choice and wavelength allocation. The static traffic requirement between each node pair in the network has to be partitioned into an appropriate number of separate wavelength channels. For each of these, a route through the network must be found, including which fibre and wavelength will be used on each link (cable) along the path. However, although restoration paths are supported during topology design, the network model provides no direct support for the fibre and wavelength allocation of restoration paths. An example wavelength allocation, for the network topology already developed, is illustrated in Fig. 3.1(c), where (assuming just a single fibre for each cable) the number of wavelengths used has been drawn next to each network link.

A more complete review of NOMaD's network model can be found in §E.3, and detailed descriptions of the specific versions of these design problems addressed by experimental work are given in §5.1 and §5.2 below.

3.2 Design Metrics

As a vital part of any network design process, it is necessary to define one or more objectives, to allow prospective designs to be compared, and the best selected. While it is possible to carry out a design with multiple objectives simultaneously [275–277], in NOMaD a single overall design metric is assumed. This approach is common to much evolutionary-algorithm (EA) research, as it allows a single 'fitness' value to be determined for each individual in the population. As a result, however, if multiple objectives must be considered, they will have to be combined as a weighted sum. In addition, any design constraints may be enforced either by repairing individual network designs that violate them [278], or adding a penalty [279,280] to the design metric for each violation (usually also dependent on the *extent* of the violation).

Within NOMaD, the author has developed three metrics, one for topology design and two for routing, fibre and wavelength allocation, and these are presented in §4.1 below. In addition, it is straightforward for users to incorporate their own individual metrics for network design assessment into the software.

3.3 Solution Approaches

Two main solution approaches to network design problems have been anticipated within NOMaD: overall design heuristics, and EAs.

Problem-specific, constructional heuristic algorithms employing the network model can be readily built by users. Moreover, while the author did not identify any suitable overall heuristics for minimum-cost topology design of optical transport networks, several routing, fibre and wavelength allocation heuristics from the literature were implemented, and these are described in Appendix C (additional algorithms have also been identified in §6.4).

Extensive support for EAs in the form of GA/heuristic hybrids is provided, with both an overall adaptive GA framework, and also the straightforward ability to add heuristic operators. This applies whether the latter are inspired by traditional genetic operators, based on elements from overall design heuristics, or developed specifically for a particular hybrid. Several heuristic operators for both network topology design, and routing, fibre and wavelength allocation have been devised by the author, and these are described in §4.2, as are the overall GA framework and NOMaD's operator-probability adaptation mechanism.

In addition to these two approaches, however, only a modest amount of effort would be required to further develop NOMaD to support other modern heuristic search methods, such as tabu search [6, 7] or simulated annealing [5] (*cf.* also Bhandarkar & Zhang [262]). However, as was explained in Ch. 1, these fall outside the scope of the thesis. More distant methods, such as (integer) linear programming [25], would not however be suited to NOMaD's object-oriented network representation.

3.4 Overall Architecture

NOMaD has been developed as an overall architecture to be extended by several people in a variety of directions to serve the needs of their individual projects [31, 32]. Consequently, its users have not just been given a single unchangeable piece of software, but rather a framework which they can actively add to for themselves. As a result, NOMaD has had to be easy to understand, flexible and extensible. The overall layered architecture is shown in Fig. 3.2.

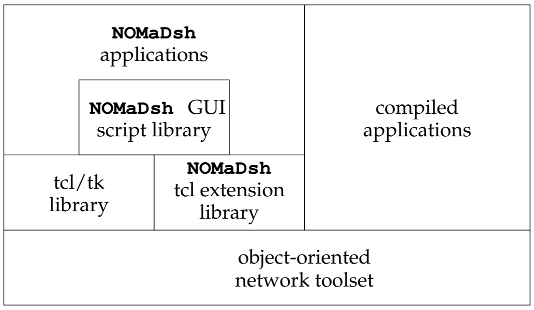


Figure 3.2: Overall NOMaD architecture

To aid both understanding and extensibility, the toolset has been developed using object technology. The analysis and design were documented using Booch Notation [15], and the implementation done in the C++ programming language [16, 17, 281, 282]. This process has been aided by the use of a CASE tool, Rational Rose/C++ [283], which generates skeletal C++ code directly from the diagrams and specifications created by the author.

Although the majority of the experimental work undertaken by the author and others using NOMaD has employed executables compiled directly from the object-oriented network toolset, for flexibility, rather than only providing a C++ interface, a NOMaD-specific tcl extension has also been developed (termed `NOMaDsh`). Tcl [284, 285] is a scripting language designed to be extended by the incorporation of application-specific commands, coded in C or C++. Consequently, `NOMaDsh` can be used either interactively to access the underlying network toolset—creating networks, modifying them, assessing their cost/performance, *etc.*—or via scripts written using a combination of tcl and the NOMaD-specific commands. It was hoped that this latter option would enable new high-level algorithms to be developed with much faster edit-compile-debug cycle times, because no compile time would be needed—the scripts could be immediately run by the `NOMaDsh` interpreter. Only if the user needed to extend the underlying C++ network toolset would the `NOMaDsh` itself need to be recompiled. It was envisaged that these benefits would be particularly important in a multi-person research environment, where the rapid prototyping of ideas is of paramount importance. In addition, a library of graphical user interface (GUI) scripts was developed using the tk toolkit (tcl/tk) to allow `NOMaDsh` scripts to display networks.

3.5 Object-oriented Network Toolset

The top-level structure of NOMaD's object-oriented network toolset is partitioned into five class categories, as shown in Fig. 3.3. Although included from an early design stage, the `NOMaD` class category contains no classes, but instead represents a 'place holder' for executables compiled directly from the network toolset, such as `ga` (see §3.9) and `NOMaDsh` itself. The remaining four categories are `NetworkModel`, `GA`, `Operators` and `Utilities`.

The `NetworkModel` class category contains classes used to represent networks for topology design as well as routing, fibre and wavelength allocation, plus metrics to allow the assessment of designs and several minor classes. For example, the main `Network` class diagram, given in Fig. 3.4(a), shows the `Network` class, composed of `Node`s, `Link`s, and `PathLossSeq`s (path-loss sequences, *i.e.* ordered sequences of `Path`s [286]), plus several implementation classes used to represent the network's adjacency matrix [25, 287], connection matrix (which records link levels in hierarchical networks) and the static traffic requirements. Further, Fig. 3.4(b) illustrates the wavelength-routed network

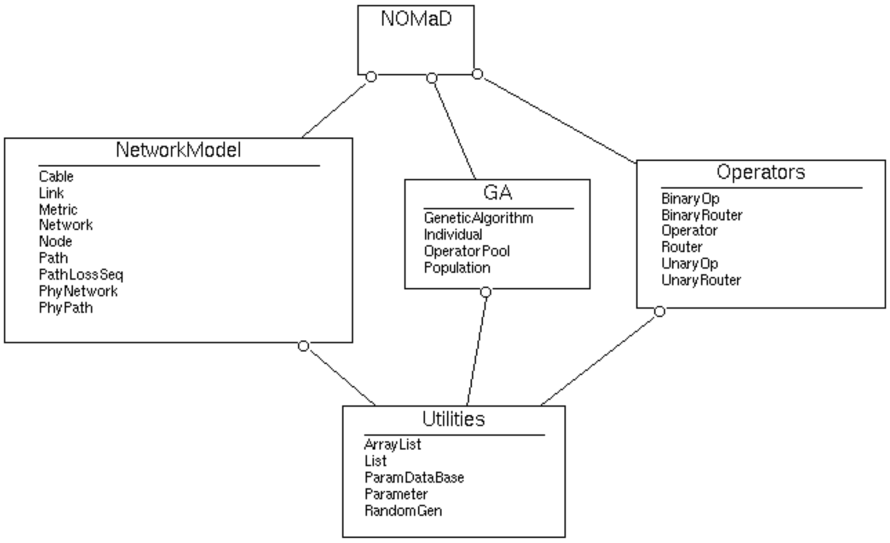
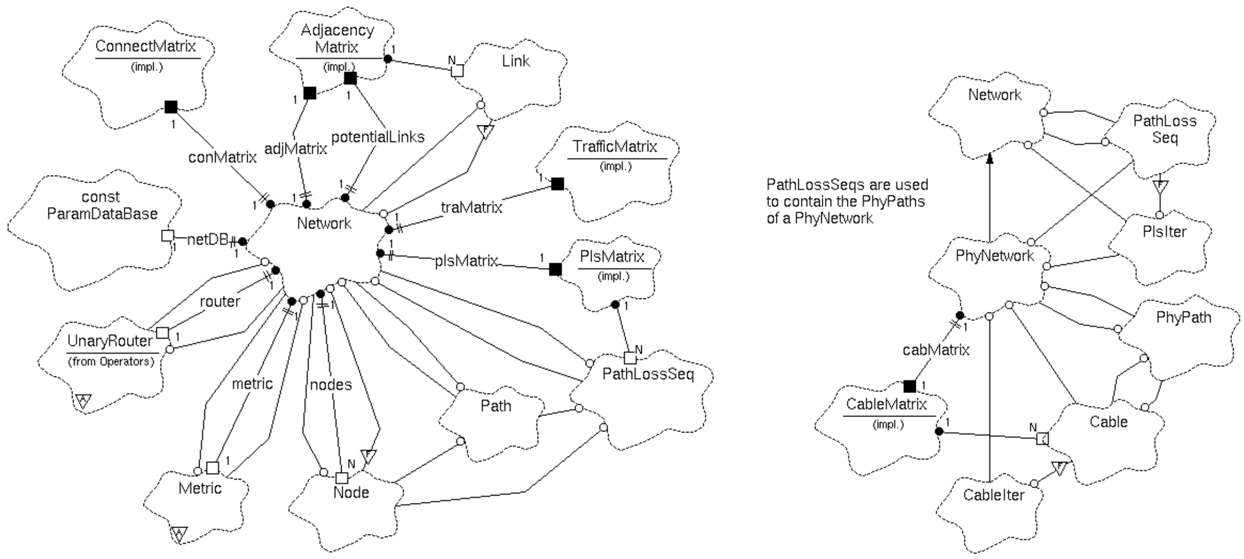


Figure 3.3: NOMaD class categories

class (`PhyNetwork`, derived from `Network`), which uses wavelength-routed paths (`PhyPath`s, derived from `Path`) and a matrix of `Cable`s to represent the cable, fibre and wavelength-routed structure of a multi-wavelength optical network.

The classes that implement NOMaD's GA framework are grouped into the `GA` class category, and illustrated in the main `GA` class diagram in Fig. 3.5—the `GeneticAlgorithm` class, with its `Population` of `Individual`s and its `OperatorPool` of `Operator`s. An `Individual` simply consists of a single `Network` (or `PhyNetwork`) object and some additional accounting information. Operators themselves are created by deriving classes from either unary operator (`UnaryOp`) or binary operator (`BinaryOp`) classes (not shown). In general, those derived from `UnaryOp` are analogous to mutation [2] in a traditional GA, but can be powerful heuristics that utilise arbitrary amounts of problem-specific information. Likewise, `BinaryOp` subclasses are analogous to traditional crossover [2], although once again, there is considerable potential for problem-specific heuristics.

As in Davis [9], the probabilities of individual operators being applied to an `Individual` adapt through the course of a run using a credit-assignment algorithm, rather than being chosen with a fixed probability throughout [33]. Whenever an individual, whose fitness exceeds that of the best individual up to the end of the previous generation (`bestIdv`), is created, the improvement in fitness is credited to the operator responsible. In addition, a decreasing fraction of the credit is awarded to the parent operator (*i.e.* the operator that created this individual's parent, `parentOp`) and grandparent operator (`grandParOp`), to ensure that two- and three-operator sequences are rewarded appropriately. After a few generations, a small proportion of the operator probabilities is



(a) `Network` (b) `PhyNetwork`

Figure 3.4: Network class diagrams

reassigned according to the credit earned.

As well as including the abstract classes representing operators (`Operator`, `UnaryOp` and `BinaryOp`) and routers (`Router`, `UnaryRouter` and `BinaryRouter`), the `Operators` class category includes a large number of concrete classes derived from these. The operators range from those of modest effect, analogous to traditional mutation or crossover, intended for use within a GA framework, through more powerful problem-specific operators, up to complete design heuristics, intended for use in isolation. Routers are those operators capable of completely routing a network, rather than merely modifying an existing routing (*i.e.* either simple routing, for topology design, or complete routing, fibre and wavelength allocation as appropriate).

The final class category, `Utilities`, contains a number of comparatively unrelated classes used by the other categories. These include dynamic arrays, linked lists, file handling and pseudo-random number generation.

3.6 `NOMaDsh`: The Tcl Scripting Layer

Tcl [284, 285] is a scripting language that allows for application-specific extension in a variety of ways. The approach adopted in creating `NOMaDsh` is an object-oriented rather than action-oriented one [284,285], with the user able to create objects that directly correspond to the underlying network toolset implemented in C++. Once an object is created, a new command is added to the interpreter

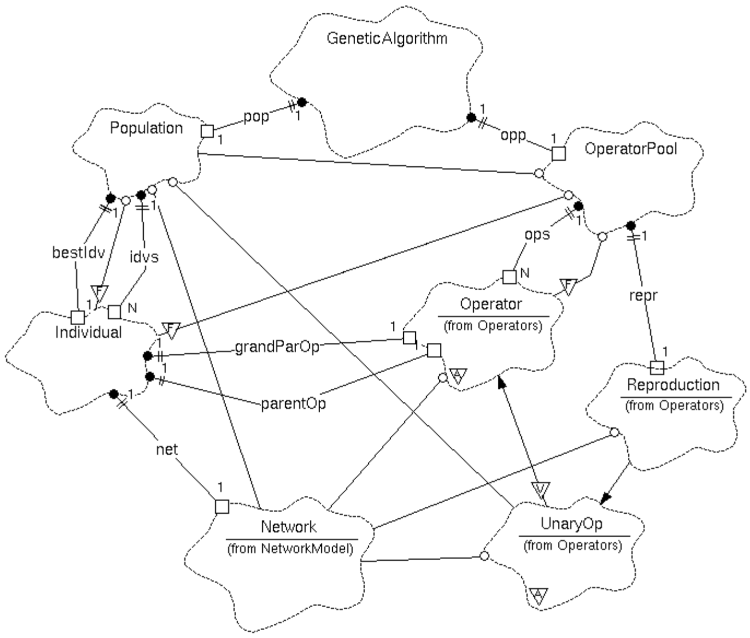


Figure 3.5: `GA` class diagram

that allows the user to send commands to that particular object, equivalent to invoking its member functions in C++. For example, a `Network` object, created in C++ by:

```
ifstream netifs("central.net");
ifstream nmdifs("central.nmd");
ParamDataBase* ndb = new ParamDataBase(netifs);
ParamDataBase* pdb = new ParamDataBase(nmdifs);
Network*       net = new Network(ndb, pdb);
```

could have been created in `NOMaDsh` with the following commands, where the NOMaD-specific extensions can be identified by the `nomad_` prefix:

```
set ndb [nomad_paramdatabase "central.net"]
set pdb [nomad_paramdatabase "central.nmd"]
set net [nomad_network $ndb $pdb]
```

In both the C++ and `NOMaDsh` commands, the two input files, one describing the network itself (`"central.net"`) and the other certain necessary parameters (`"central.nmd"`), are used to construct [16] two parameter databases (`ParamDataBase` objects `ndb` and `pdb`, respectively). It is from these that the `Network` itself is then constructed. Parameter databases are simply random access storage objects, which free other NOMaD classes from the complexities of direct file handling. Once it has been constructed, the `Network` object can then be queried for the number of nodes it contains, either in C++ as below:

```
int numNodes = net->getNumNodes();
```

or in `NOMaDsh`, by using the new command for the underlying `Network` object, whose name was stored in the `net` variable:

```
set numnodes [$net numnodes]
```

It was hoped that this semantic similarity (though partially masked by syntax!), would result in a short learning curve for users familiar with both C++ and tcl/tk. Consequently, the full benefits of the interactive and interpretive style of `NOMaDsh` scripts would be swiftly realised.

As an aid to the reader in understanding the relationship between a `NOMaDsh` script and the underlying C++, the code in `NOMaDsh` that allows access to a compiled `Network` object and the invocation of its `getNumNodes()` member function is shown below. For clarity, both error-handling code and the large number of alternatives to the `numnodes` command argument have been removed.

```
int Nomad_NetworkCmd(ClientData clientData,
          Tcl_Interp *interp, int argc, char *argv[]) {
  ostrstream rst(interp->result, TCL_RESULT_SIZE);
  Network* net = (Network*) clientData;
  if (strcmp(argv[1], "numnodes") == 0) {
    rst << net->getNumNodes() << '\0';
  }
  return TCL_OK;
}
```

First, an output string stream (`ostrstream`) is constructed around the interpreter's return buffer (`interp->result`). Then, a pointer to the underlying `Network` object is retrieved from the `clientData` parameter. If the first argument passed to the `$net` command was `numnodes`, then the `getNumNodes()` member function is invoked and the result placed in the return buffer. Finally, the function signals success by returning `TCL_OK`.

3.7 Graphical User Interface

The final element of the NOMaD architecture is the GUI. Although it was originally envisaged that a large library of GUI scripts would be built [31], including widgets for the display and editing of networks, the actual achievement in this area was relatively modest. Scripts were written for the display of network topologies (`netdisplay.tcl`), graph drawing (`graphdraw.tcl`) and the output of network topologies in `xfig` v2.1 format [288] (`netxfig.tcl`). From these, a genetic-algorithm application and a network-display application were developed, although both are suitable only for network topology design (as opposed to routing, fibre and wavelength allocation). In addition, a prototype network editor was implemented, but this was simply written in tcl/tk as a feasibility study, without employing the NOMaD-specific extensions to access the underlying object-oriented network toolset.

Screen shots of the genetic-algorithm (`ga.tcl`), network-display (`netdisp.tcl`) and prototype network-editor (`network.tcl`) `NOMaDsh` applications are given in Figures 3.6–3.8.

The first of these (Fig. 3.6) shows the genetic-algorithm application's control panel, where file names are entered and the GA run started, two graph windows, and a series of best individuals found at the end of successive generations. The network design problem tackled here was the minimum-cost topology design of the nine central nodes of the COST 239 European Optical Network (EON) [166], with the result found at the end of generation 14 matching the best known. The two graph windows have recorded the lowest & average network costs (`Network Assess` window) and the operator probabilities (`Operator Prob` window) changing as execution proceeded. In the latter window, the varying probabilities of the three operators actually used in this run *i.e.* link-group crossover (LGCO), single-link mutation (ML) and reproduction (REPR) can be seen; for a full description of these, see §4.2 below.

The second screen shot (Fig. 3.7) has captured two windows of the network display application: the control panel, where a succession of file names may be entered for display, and a network topology window, once again showing the nine central nodes of the EON. As with `ga.tcl`, the network topology window allows the network design to be saved to disk (using a member function of the underlying `Network` class) or an `xfig` format diagram generated and saved.

Finally, the NOMaD prototype network editor is shown in Fig. 3.8, with an 8-node network displayed. The vertical row of icons on the left-hand side switch the editor into the following modes (starting from the top): query, add-node, add-link, move-node (and attached links), traffic-display and delete. The status line along the bottom of the window prompts the user for the next expected action, and provides appropriate feedback during operations; in Fig. 3.8, it can be seen displaying the initial add-link prompt.

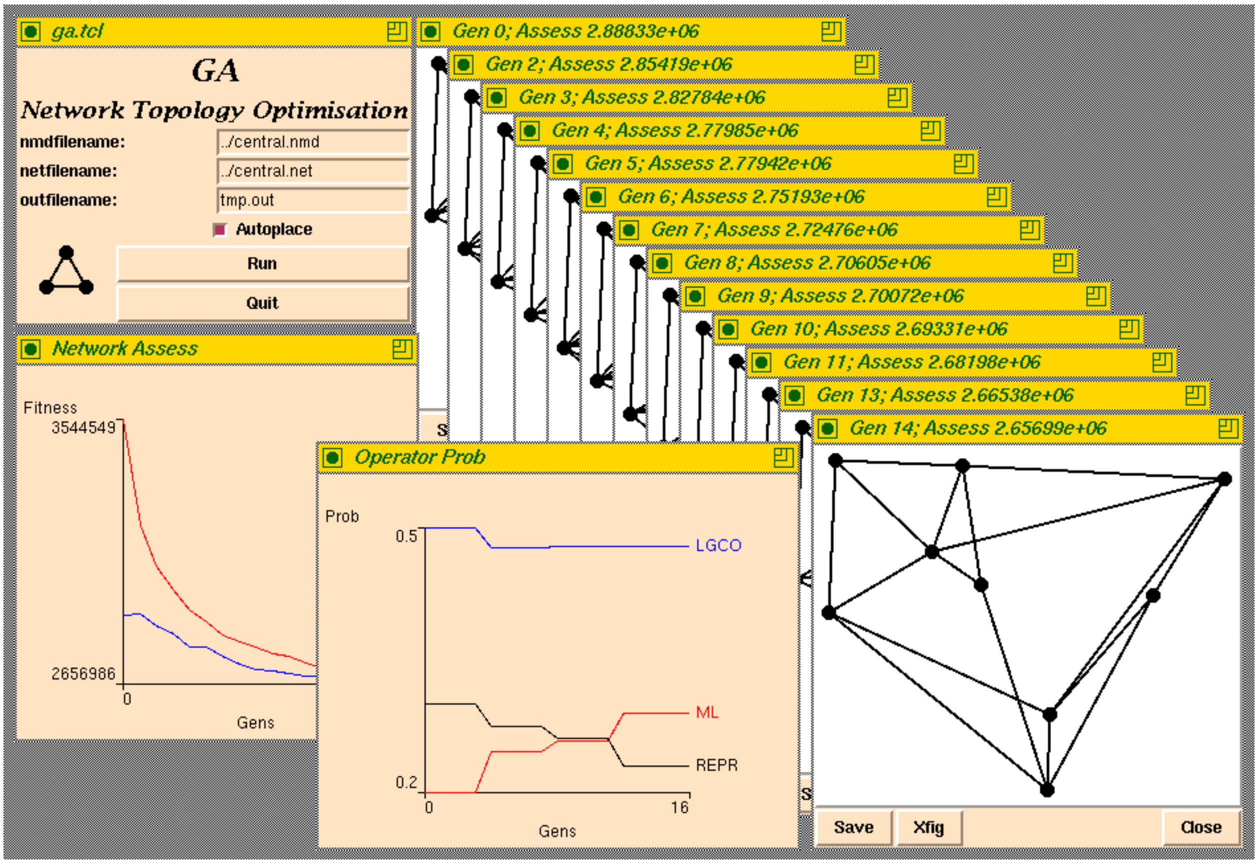


Figure 3.6: Genetic-algorithm (`ga.tcl`) application

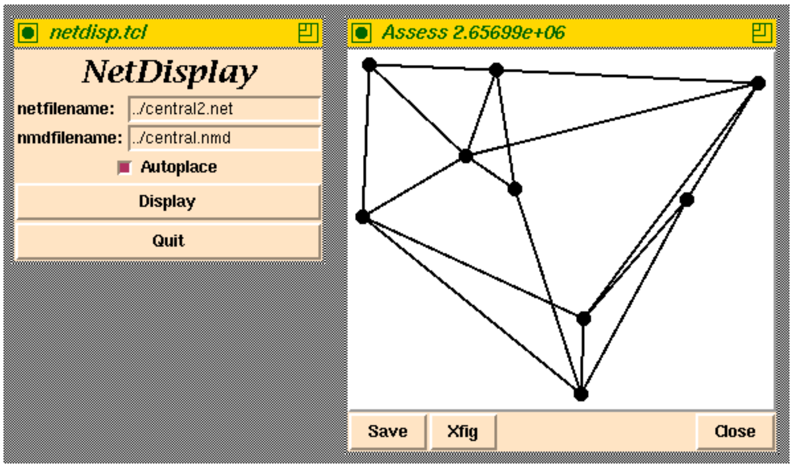


Figure 3.7: Network-display (`netdisp.tcl`) application

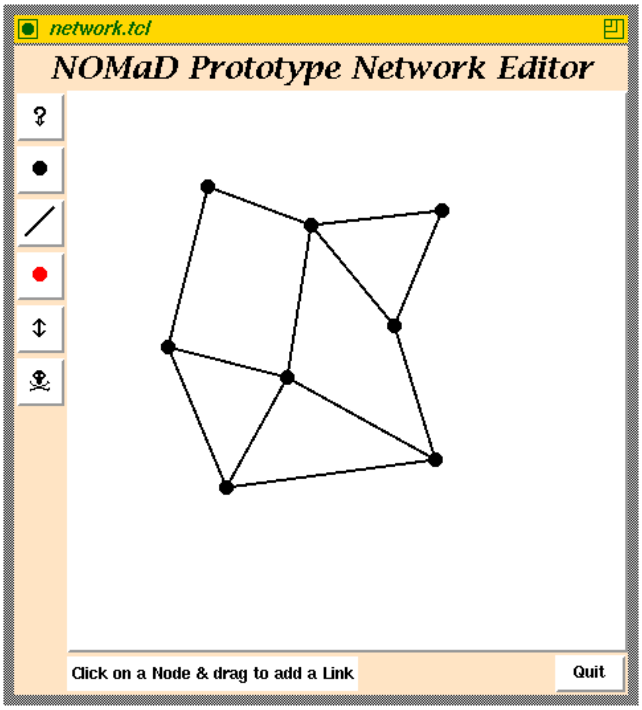


Figure 3.8: Network-editor (`network.tcl`) application

3.8 Testing & Verification

During the design and implementation of NOMaD's object-oriented network toolset, three different approaches were used to try and minimise errors in the software: Programming by Contract, dynamic memory tracking and regression testing.

Meyer's concept of Programming by Contract [289] introduces pre- and postconditions on class member functions: preconditions define what a class requires to do its job; postconditions, what it will provide if its preconditions have been met. In the object-oriented network toolset, explicitly stating these conditions in this way led to clarity in class design, a benefit which was further enhanced by run-time checking using appropriate assertions.

Now, as C++ requires the programmer to directly manage the allocation and release of all dynamic memory objects [16,282], this can be a significant source of error. In addition, the dynamic allocation of objects that are not subsequently released when no longer required ('memory bleed'), can lead to program failure due to exhaustion of machine memory. To combat problems in this area, dynamic memory tracking code (called `xmem`) for C [290] was rewritten by the author for C++. This allowed the detection of a variety of different problems during NOMaD's development.

The final approach to minimising software errors adopted was regression testing. A large test program (`testNOMaD`) was written, with a long series of assertions for many of the classes in the object-oriented network toolset. These checked the expected outcome of invoking the class member functions under a variety of different conditions. By running, and extending, the tests periodically throughout the development of NOMaD, several subtle errors were detected and eliminated.

Further details of all three elements of NOMaD's testing and verification strategy can be found in §E.7.

3.9 The `ga` Application Program

Although several application programs were developed by the author using NOMaD's object-oriented network toolset, the most important of these is `ga`, which has therefore been selected for description here. Others include, for example, `wavealloc`, used to apply complete design heuristics for routing, fibre and wavelength allocation (such as those described in Appendix C) to individual network topologies. The `ga` application program is partially equivalent to the `ga.tcl` `NOMaDsh` application mentioned in §3.7 above, but allows access to the full range of GA/heuristic hybrid techniques in NOMaD's object-oriented network toolset. Consequently, the majority of the experimental results reported in Ch. 5 were obtained using `ga`.

The `ga` program is invoked with four arguments: the name of the parameter file (`.nmd`), the name of a file containing a description of the network design problem (or network topology, as appropriate; `.net`), a filename to be used to output the results of the run (`.out`), and an initial value for the pseudo-random number generator.

A high-level flowchart for the `ga` application program is shown in Fig. 3.9. After checking and storing the command-line arguments, the major program objects are constructed. These consist of three parameter databases, two built from the input files and one to record data (in variable-value pairs *i.e.* parameter form) that will be output to file at the end of program; a pseudo-random number generator; and a genetic-algorithm object. Initial statistics are then recorded in the output parameter database, including the average and lowest fitness of the initial population, as well as the number and names of the operators, and their initial probabilities. The program then enters its main loop, which is controlled by a Boolean variable which, after initially being set `true`, is assigned the value returned by the genetic-algorithm object's `run()` member function. The latter is invoked to evolve the population for one generation, with the return value indicating whether further evolution is required. Similar statistics to those recorded for the initial population are then added to the output parameter database for this generation. Finally, at the end of program execution, both the best network design found and the contents of the output parameter database are written to file.

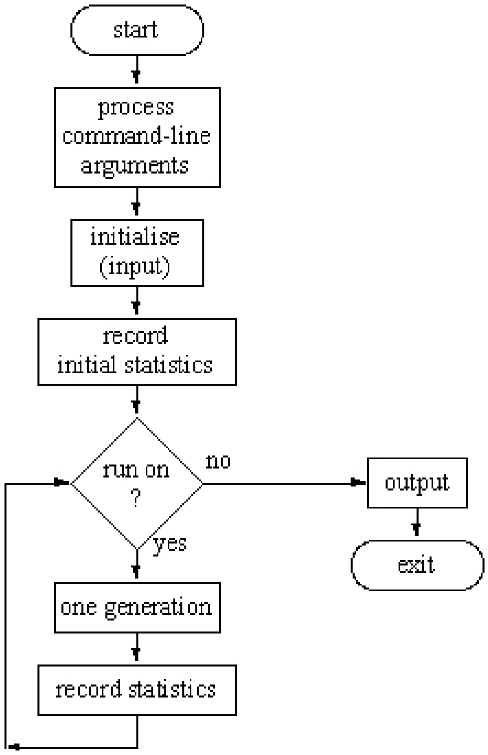


Figure 3.9: Flowchart of the `ga` application program

3.10 Summary

This chapter has provided an overview of the NOMaD software toolset, including its application areas, design metrics and supported solution approaches. The overall NOMaD architecture, its object-oriented network toolset, interactive scripting capabilities, graphical user interface, testing & verification strategy and an example compiled application have all been discussed. Building on this overview, the next chapter will provide greater detail of NOMaD's design metrics and EA framework. It is these elements, implemented in the NOMaD software toolset, that encapsulate the author's GA/heuristic hybrid approach to multi-wavelength all-optical transport network design.

Chapter 4

GA/Heuristic Hybrid Network Design

*Although genetic algorithms using binary representation and single-point crossover and binary mutation are robust algorithms, they are almost never the **best** algorithms to use for any [specific] problem.*
Lawrence Davis, 1991

This chapter presents key details of NOMaD's object-oriented network toolset, briefly introduced in the context of the overall NOMaD architecture in §3.5 above. A description of the central features of NOMaD's network model is provided in §E.3, but here the author's genetic-algorithm (GA)/heuristic hybrid approach to multi-wavelength all-optical transport network design is described.

Given the large size of the NOMaD software, discussion of the design and implementation, both here and in Appendix E, is necessarily restricted to a somewhat high-level view. However, where it is deemed appropriate, more detailed description will be given. As an aid to the reader in assimilating this, §E.2 contains class cross-reference tables, which cite class descriptions and diagrams throughout the thesis. Further, although the classes are discussed both in terms of their functionality and their underlying data structure, it should be noted that the users of a class can only interact with it through its public interface [15, 16]. In almost all cases, this consists entirely of public member functions [16], to ensure rigid enforcement of data hiding; to a limited extent, however, some classes use protected data members and class 'friendship' to allow non-public access [16].

Throughout the chapter, and in Appendix E, when a particular class is discussed, its parameters may be briefly mentioned. However, the tables of class parameter names and their default values (where appropriate) are provided in §E.1.2. Classes obtain their parameters from a parameter database object, itself constructed from a NOMaD parameter file. NOMaD's use of execution-time class parameters affords considerable flexibility, allowing the user to make significant changes to class (and hence program) functionality without recompilation. Further details on parameter databases can be found in §E.6, and NOMaD file formats are discussed in §E.1.

The chapter is divided into three sections: §4.1 provides details of three metrics for network design assessment: one for minimum-cost topology design, another for network wavelength requirement, and the final one for minimum-cost wavelength allocation. Then §4.2 presents NOMaD's evolutionary algorithm framework: the overall structure of the GA/heuristic hybrid; operators for network topology design; operators for routing, fibre and wavelength allocation; and the operator-probability adaptation mechanism. Finally, the chapter is summarised in §4.3.

4.1 Metrics for Design Assessment

To enable the assessment of network designs, expressed using the classes of NOMaD's network model (§E.3), the object-oriented network toolset provides an abstract base class, `Metric`, from which users can derive concrete classes for their individual design-assessment metrics. As was mentioned in §3.2, the author has developed three such metrics, described in the following subsections, and implemented as the classes: `AlesCost`, `NetWaveReq` and `PhyNetCost` (see Fig. 4.1).

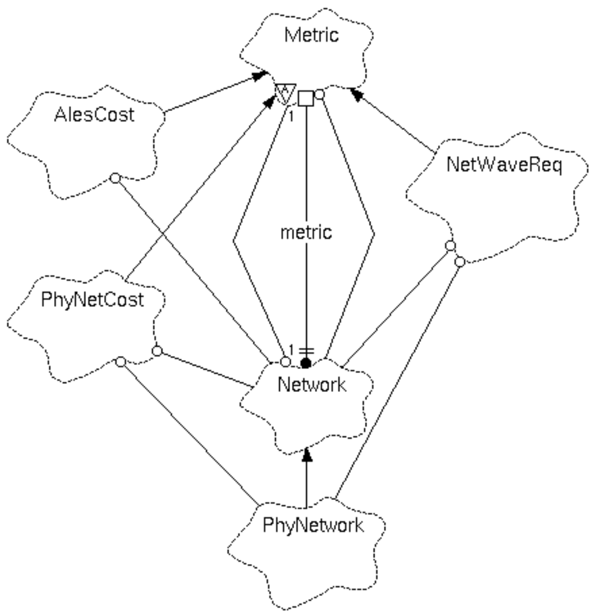


Figure 4.1: `Metric` class diagram

The `Metric` class constructor requires a single argument, the metric name, which can be up to four characters, and must be unique within the toolset. It is this name which is subsequently used to select the metric in a parameter file. For example, `AlesCost` uses the name `"ALCT"`, `NetWaveReq` uses `"NWR"`, and `PhyNetCost`, `"PNCT"`. The name is stored, and can be retrieved using a member function. The abstract virtual [16] member function `assess()`, which takes as a single argument the `Network` (or its derived class, `PhyNetwork`) to be assessed, is provided as a placeholder for concrete derived classes, which must implement it for themselves. In addition to being able to apply a `Metric` to a `Network` in this way, the `Network` class itself stores a pointer to a `Metric` object, which can thus be used for self-assessment: this ability is described (in the context of the `Network` class) in §E.3.1.

4.1.1 Network Cost Model A (`AlesCost`)

The original version of this cost model was developed by the author for minimum-cost topology design of the European Optical Network (EON) as part of COST 239 [291]. It was first presented at ICANNGA'95 in Alès [166], and as a result the corresponding NOMaD class was named `AlesCost`. However, the actual model presented here has been extended by the addition of the constant α in Eqn. 4.3 (below), which allows non-linear dependence of link cost on capacity, a possibility suggested as further work in the paper itself [166].

The model assumes that static two-shortest-node-disjoint-path routing [292] is used between node pairs, and that a reliability constraint is employed. The latter is to ensure that there are two, usually fully-resourced, node-disjoint routes between node pairs, thereby guaranteeing that the network will survive the failure of any single component (node or link). However, no attempt is made to do detailed modelling of the potential sharing of restoration capacity (if protection is not required for multiple simultaneous link or node failures), and this may lead to an overestimation of the required network capacity. Instead, the model simply provides a parameter (K_R) to set the desired level of capacity on the restoration routes; in the author's use of the model, this has usually been set to 1.0 (*i.e.* 100% restoration capacity) whereas a more restricted level of 0.4 (*i.e.* 40%) might well prove sufficient [293]. In addition, as the model was intended for network topology design, wavelength-allocation is not modelled, and as a result the model may underestimate the capacity required for wavelength-path (WP) routing [294], due to the presence of dark wavelengths.

To determine the cost of a given topology, separate models for both links and nodes are required. The intention is to approximate the relative contribution to purchase, installation and maintenance costs of the different network elements, while ensuring that the model is not too dependent on the details of the element designs, nor too complex for use in the 'inner loop' of a design procedure.

First, the two-shortest-node-disjoint routes are determined for all node pairs. In this, the 'length' of each path is taken to be the sum of the contributing link weights, with the weight of the bi-directional link between nodes i and j given by:

$$W_{ij} = 0.5N_i + L_{ij} + 0.5N_j \tag{4.1}$$

where N_i and N_j are the *node effective distances* of nodes i and j respectively (see below) and L_{ij} is the length of link (i, j) in km. Then, the link carried traffic is determined for each link by summing the contributions from all the primary and restoration routes that make use of it. As mentioned above, the restoration traffic is weighted by K_R, but restoration routes are usually taken to be fully-resourced, and thus are assumed to carry the same traffic as the corresponding primary routes *i.e.* the traffic they would be required to carry if their primary route failed. The capacity of link (i, j) is taken to be:

$$V_{ij} = \mathrm{ceil}_{K_G}(K_T T_{ij}) \tag{4.2}$$

where T_{ij} is the carried traffic in Gbit/s on the link, and $\mathrm{ceil}_x()$ rounds its argument up to the nearest x—here, the assumed granularity of the transmission links, K_G (say 2.5 Gbit/s). The factor of K_T is to allow for stochastic effects in the traffic[1]. The cost of link (i, j) is then taken to be:

$$C_{ij} = V_{ij}^{\alpha} L_{ij} \tag{4.3}$$

where α is a constant. Increasing capacity necessarily implies increased cost due, for example, to wider transmission bandwidth, narrower wavelength separation and/or increasing number and speed of transmitters and receivers. With $\alpha = 1$, a linear dependence of cost on capacity is assumed, but with $\alpha < 1$, the cost can be adjusted to rise more slowly with increases in the link capacity [295]. The linear link length dependency approximates the increasing costs of, for example, duct installation, fibre blowing and/or the larger number of optical amplifiers with increasing distance.

Node effective distance was used as a way of representing the cost of nodes in an optical network in equivalent distance terms. It can be regarded as the effective distance added to a path as a result of traversing a node. By including it in link weights (Eqn. 4.1) for the calculation of shortest paths, path weights reflect the cost of the nodes traversed (and half the costs of the end nodes). As a

[1]Following discussions in COST 239, K_G was taken to be 1.0 in the author's subsequent metrics, *i.e.* traffic taken to be identical to capacity requirement, and the factor quietly dropped.

result, a longer geographical path may have a lower weight if it traverses fewer nodes, thereby reflecting the relatively high costs of optical switching. The node effective distance of node i was taken to be:

$$N_i = K_0 + n_i K_n \tag{4.4}$$

where n_i is the degree of node i, *i.e.* the number of bi-directional links attached to it. The constants K_0 and K_n were taken to be say 200 km and 100 km, respectively. Node effective distance thus increases as the switch grows more complex. Node capacity is the sum of the capacities of all the attached links, *i.e.*

$$V_i = \sum_j V_{ij} \tag{4.5}$$

where V_i is the capacity of node i in Gbit/s. The node cost was taken to be:

$$C_i = 0.5 N_i V_i \tag{4.6}$$

The cost is thus derived as if the node were a star of links, each of half the node effective length, and each having the same capacity as the node itself. Further, if all the links attached to a node are of the same capacity, the node costs would grow approximately with the square of the node degree, corresponding, for example, to the growth in the number of crosspoints in a simple optical space switch. Overall, the relative costs of nodes and links can be adjusted by setting the values of K_0 and K_n appropriately.

The network cost is then taken to be the sum of the costs of all the individual links and nodes comprising the network. However, to ensure that the reliability constraint was met, the actual metric used also includes a penalty function [279, 280]. This adds P_R (say 250,000) to the network cost for every node pair that has no alternative path, and P_N (say 500,000) for every node pair with no routes at all between them. This avoids the false minimum of the topology with no links at all, whose cost, under the link and node cost models employed, would be zero. The penalties chosen must be large enough to ensure the final topology design meets the reliability constraint. However, they should not be much larger than this, or they will reduce the speed of (evolutionary) search, unnecessarily focusing the algorithm on meeting the constraint early in the run, rather than finding low-cost topologies with only minor violations.

The cost model and associated routing procedure were implemented in two NOMaD classes, `AlesCost` and `AlesRouter`, respectively. The parameters for these (obtained via a parameter database argument on each of their constructors) are given in Tables E.4 and E.5 on p. 248, and their relationships with other classes are illustrated in Fig. 4.1 on p. 67 and Fig. E.2(b) on p. 260. These two classes are also the default metric and router of class `Network` (see Table E.3 on p. 247).

4.1.2 Network Wavelength Requirement (`NetWaveReq`)

Many authors [29, 35, 296–300] have considered a low network wavelength requirement (*i.e.* the number of distinct wavelengths in the network) to be desirable as a design objective for wavelength-routed networks. This is due to the technological limitations, and resulting cost implications, of carrying more than a small number of wavelengths over national or even international distances [301].

However, the actual metric described here also includes a penalty [279, 280] added for each unsatisfied traffic requirement (*i.e.* wavelength channel that remains unrouted), to avoid the false minimum of carrying no traffic at all, which would otherwise have a network wavelength requirement of zero. A suitable penalty would be a number of wavelengths comparable with, but smaller than, the expected network wavelength requirement (*e.g.* 5 wavelengths for an expected network wavelength requirement in the range 6–10). This provides a balance between allocating all traffic requirements by the end of the (evolutionary) algorithm run, while avoiding undue focus on early satisfaction of the constraints.

This metric, including the node-to-node penalty, was first published by the author in *Electronic Letters* [35] (reprinted in Appendix A). Since then, an additional (optional) per-fibre penalty has been introduced, to penalise a network design with more than one fibre using the highest wavelength number. For every fibre using the current highest wavelength, a small penalty (say 0.01 wavelengths) is added to the overall assessment. For example, this enables two network designs to be distinguished, both with a wavelength requirement of say 8, but one with 5 fibres using the highest wavelength, the other only 2. The first would have an assessment of 8.05, the second, 8.02, indicating the latter's superior quality. The value of per-fibre penalty selected should be such as to avoid increasing the assessment by a full wavelength, even if many fibres use the highest wavelength number. A value comparable to the inverse of the total number of fibres in the network is recommended.

The `NetWaveReq` class was written to implement this metric, and its parameters (obtained via a parameter database argument to the constructor) are given in Table E.6 on p. 248; its relationships with other classes are illustrated in Fig. 4.1 on p. 67.

4.1.3 Network Cost Model B (`PhyNetCost`)

Network wavelength requirement was presented as a suitable design metric for wavelength-routed networks in §4.1.2 above. However, an objective that consists of a simplified assessment of the cost of the whole network including, at an appropriate level, the cost of exceeding the minimum network wavelength requirement, is arguably superior, and is the metric described here.

This second cost model is a further development of Network Cost Model A, described in §4.1.1 above (`AlesCost`). It was first published at the 6th IEE Conf. on Telecommunications [36] and subsequently in *IEE Proc. Communications* [37] (both reprinted in Appendix A), and makes Network Cost Model A specific to wavelength-routed all-optical transport networks [294]. It should be noted that although, in assessing similar designs, others have included failure restoration [297, 302] and restrictions on the number of wavelengths per fibre [302,303], for simplicity, these are not considered here.

Given the wavelength-routed network paths, the capacity of link (i, j) is taken to be:

$$V_{ij} = \lambda_{ij} V_\lambda \tag{4.7}$$

where λ_{ij} is the total number of wavelengths in use on the link (*i.e.* across all its fibres) and V_λ is the granularity of the wavelength channels (*e.g.* 10 Gbit/s). For example, in Fig. E.6 on p. 269, $\lambda_{ij} = 12$ and thus V_{ij} is 120 Gbit/s. The *wavelength-requirement* capacity of link (i, j) is taken to be:

$$V_{\lambda,ij} = \lambda_{\mathrm{req},ij} F_{ij} V_\lambda \tag{4.8}$$

where $\lambda_{\mathrm{req},ij}$ is the wavelength requirement of link (i, j) and F_{ij} is the number of fibres in use on link (i, j). The wavelength-requirement capacity is thus the capacity that would result if all the dark wavelengths on all the (partially-used) fibres of the link were filled, up to the current link wavelength requirement. For example, again in Fig. E.6, $\lambda_{\mathrm{req},ij} = 5$, $F_{ij} = 3$ and thus $V_{\lambda,ij}$ is 150 Gbit/s. Since capacity is one of the factors in the link cost calculation (see below), wavelength-requirement capacity thus allows the additional costs of uneven fibre loading and dark wavelengths to be assessed in equivalent capacity terms, and thus included at an appropriate level in the overall link cost.

The *capacity* cost[2] of link (i, j), *i.e.* the link cost omitting any contribution from wavelength

[2] Equivalent to link cost in Network Cost Model A, Eqn. 4.3.

requirement, is:

$$C_{V,ij} = V_{ij}^{\alpha} L_{ij} \tag{4.9}$$

where α is a constant and L_{ij} is the length of link (i, j) in km. Whereas, the *wavelength-requirement* cost of link (i, j) is:

$$C_{\lambda,ij} = V_{\lambda,ij}^{\beta} L_{ij} \tag{4.10}$$

where β is a constant. By setting $\beta \neq \alpha$, the link model can be adjusted to allow for a different dependency of cost on wavelength-requirement capacity. The overall cost of link (i, j) is then taken to be:

$$C_{ij} = \gamma C_{V,ij} + (1 - \gamma) C_{\lambda,ij} \tag{4.11}$$

where γ is a constant ($0 \leq \gamma \leq 1$). Different values of γ allow the model to reflect the relative contributions to link cost of the capacity cost and the wavelength-requirement cost. With $\gamma = 1$, the link model only takes account of the capacity used, and ignores the link wavelength requirement; whereas, with $\gamma = 0$, the actual capacity used is ignored, and the focus is entirely on the wavelength requirement. An intermediate value of γ, such as 0.5, is regarded as being more representative of the actual costs involved.

For the nodes, node effective distance was again used as a way of representing the cost of nodes in an optical network in equivalent distance terms. However, while in Network Cost Model A it was used to influence the selection of paths in minimum-cost network topology design, here it is only used as part of the node cost model. Although node effective distance is defined as in Eqn. 4.4 above, node capacity here is taken to be the sum of the *effective* capacities of all the attached links, *i.e.*

$$V_i = \sum_j V_{e,ij} \tag{4.12}$$

where V_i is the capacity of node i in Gbit/s and the effective capacity of link (i, j) is given by:

$$V_{e,ij} = \gamma V_{ij} + (1 - \gamma) V_{\lambda,ij} \tag{4.13}$$

The node capacity is therefore dependent to some extent on the wavelength requirements of its attached links. The node cost was then assessed following Eqn. 4.6.

The network cost is once again taken to be the sum of the costs of all the individual links and nodes comprising the network. However, only one penalty (P_N, say 250,000) was needed, which was added for each individual traffic requirement (*i.e.* wavelength channel) unsatisfied, so as to avoid the false minimum of carrying no traffic at all.

The cost model was implemented in a NOMaD class, `PhyNetCost`, the parameters for which (obtained, as usual, via a parameter database argument to the constructor) are given in Table E.7 on p. 248. `PhyNetCost`'s relationships with other classes are illustrated in Fig. 4.1 on p. 67.

4.2 Genetic-algorithm Framework

In this section the focus is on solution approaches, primarily GA/heuristic hybrids, but also, to some extent, overall design heuristics. Both these approaches, supported by NOMaD's object-oriented network toolset, were discussed briefly in §3.3, and GA/heuristic hybrids in particular were introduced in §1.4 and the tutorial of §2.2.3.

In the following subsections, first the overall GA framework (§4.2.1), then the various operators designed by the author for network topology design (§4.2.2), as well as routing, fibre choice and wavelength allocation (§4.2.3) are described. Finally, in §4.2.4, the details of the operator-probability adaptation mechanism, which aims to improve operator productivity within the overall GA framework, are presented.

4.2.1 Overall Structure

The classes discussed in this subsection form the overall GA framework, a brief overview of which was given in §3.5 above. As well as the central framework classes (`GeneticAlgorithm`, `Population`, `Individual` and `OperatorPool`), the abstract classes from which all NOMaD's genetic operators are derived (`Operator`, `UnaryOp` and `BinaryOp`) are described, plus one very special operator, `Reproduction`. The relationships between most of these were illustrated in a class diagram, Fig. 3.5 on p. 59.

`GeneticAlgorithm` class

The `GeneticAlgorithm` class is central to NOMaD's GA/heuristic hybrid approach. It supports a generational, rather than steady-state, algorithm [9], and depends heavily on two other classes: `Population`, which holds the population of evolving network designs, and `OperatorPool`, which not only manages the pool of genetic operators used for population evolution, but also runs the operator-probability adaptation mechanism.

The main `GeneticAlgorithm` class constructor takes three arguments: parameter databases built from both the network specification and the NOMaD parameter files, plus a pseudo-random number generator object (of class `RandomGen`, §E.6). After extracting the GA's one parameter, the maximum number of trials[3] (Table E.8 on p. 248), the constructor uses the parameter databases and random number generator to build both population and operator-pool objects. Both of these, as well as the maximum number of trials, can subsequently be retrieved with the appropriate member functions. In addition, a GA object maintains one additional data member, the current number of trials, which is initialised to zero, and can also be accessed via a member function.

The `GeneticAlgorithm` class' principal member function is `run()`. This takes the desired number of generations as an integer argument, and runs the GA for that number of generations. However, it will terminate prematurely, although always at the end of a complete generation, if either the number of trials exceeds the maximum specified or the appropriate Unix interprocess signal has been received. On completion, the function returns a Boolean indicating whether it would be able to continue for at least one more generation ('run on'). However, if the desired number of generations is specified as zero (the default), the function continues without returning until the maximum number of trials is exceeded *i.e.* the run is completed, or the signal has been received. Nevertheless, although it allows for runs of multiple generations in this way, if recording of population and operator statistics is required every generation (as it is in the `ga` application program, see §3.9), the function is only invoked for one generation at a time.

A flowchart of the `run()` member function is given in Fig. 4.2 below. This shows the overall generational nature of the process, represented by the main loop, including breeding of a new population and its replacement of the old, as well as the use of operator-probability adaptation. However, there are a number of decision points that can end the iteration. As well as receipt of the appropriate interprocess signal, exceeding the maximum number of trials or completing the desired number of generations, there is one further possibility.

To appreciate this, first note that network objects both cache their last assessment result, as is mentioned in §E.3, and also count the number of trials they have used in self-assessment since their construction. This count is made available via a member function, and so the number of trials (*i.e.* assessments) used in constructing a population can be obtained by summing over all its constituent network objects. Consequently, if a new population is bred without consuming any trials, then not

[3] A trial is an assessment of a network design *i.e.* application of the appropriate design metric. The number of trials can be used as a platform-independent measure of algorithm effort, as assessment is often the most computationally-intensive part of algorithm execution.

a single network has been changed in the breeding process: the GA is 'spinning' (*i.e.* has converged to identical or almost identical population members), and the run is terminated.

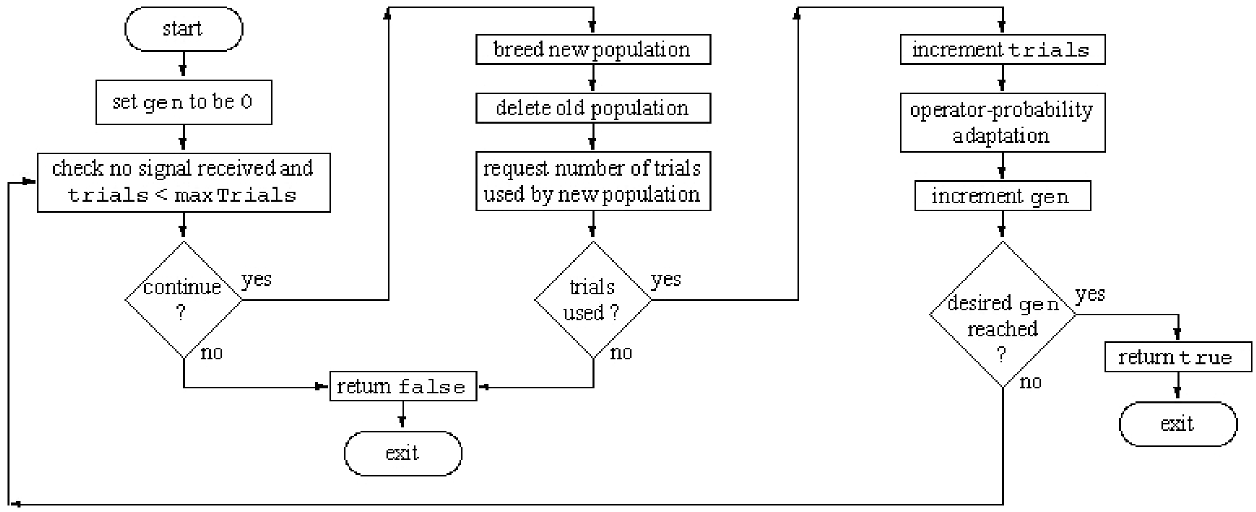


Figure 4.2: Flowchart of the `GeneticAlgorithm` `run()` member function

`Population` class

NOMaD's `Population` class is not only responsible for containing a population of individuals (each a network object plus additional accounting information), but also maintaining several important population statistics.

There are two main `Population` class constructors. One of these, taking both network specification and NOMaD parameter databases plus a random number generator, creates a full population. The other, omitting the network specification parameter database, creates an empty population, to which individuals can subsequently be added. The former constructor is used for initial population construction, whereas the latter is intended as part of breeding a new population from an old one. Both constructors extract and store several parameters (Table E.9 on p. 248): the population size, the window size (w) for best-in-window fitness, the selection function and, if the latter is tournament selection, the tournament size[4]. Member functions are provided to allow the population size, window size and selection function to be subsequently retrieved. In addition, for the former constructor, a choice of initialisation operator may also be given. This is applied to every network object in the population as it is constructed, and can be used to build a random initial population. For example, in network topology design, the default `RandomLinks` operator can be employed (see

[4]While NOMaD has been designed to allow for a choice of selection functions, only tournament selection has actually been implemented. However, there is strong support in the literature for choosing tournament selection [304, 305].

§4.2.2 below), and for routing, fibre choice and wavelength allocation, `WaveRandKSP` (§4.2.3).

Expanding a non-full population is achieved using a member function which enables an individual to be added to the population. At any point the current size of the population can be accessed via another member function, and then, when the population is full, a further member function is automatically invoked to refresh the population statistics. (The latter is also invoked at the end of initial population construction.)

The statistics stored include[5], at the end of the last generation, the highest fitness (*i.e.* assessment) of any individual in the population, the lowest fitness, the average fitness and the median fitness. Further, a record is maintained of the lowest fitness in each generation over a user-defined number of generations (see Table E.9). In addition, a copy of the best individual (*i.e.* of lowest fitness value) seen up to the end of the last generation is made. To access the statistics, member functions are provided for highest, lowest, average, median and best-in-window fitness, as well as one to access a `const` pointer to the best individual (and hence the best-so-far fitness).

Two further member functions are also available: an overloaded '`[]`' operator allows indexed access to the individuals in the population, and the total number of trials used by the population can be obtained, as was mentioned in the description of the `GeneticAlgorithm` class above.

`Individual` class

The `Individual` class is simply a `Network` (or `PhyNetwork`) with some additional accounting information to support operator credit assignment (see §3.5, and Fig. 3.5 on p. 59).

The main constructor takes two arguments: the parameter databases for network specification and NOMaD parameters. From the latter database, a single parameter is taken (Table E.10 on p. 248) which determines whether the individual constructs a `Network` or a `PhyNetwork` using the network database. However, construction of an individual from parameter databases in this way is computationally relatively expensive, and as a result, after the first, the remainder of the individuals in a population are actually created using the copy constructor (both in the initial population and when breeding).

An individual stores a pointer to its network object, as well as pointers to both its parent and grandparent operators (see §3.5, and §4.2.4 below). These latter objects are used by the credit-assignment mechanism to ensure that two- and three-operator sequences are properly rewarded.

[5]NOMaD assumes throughout that the *lowest* fitness value is best.

However, they are not actually maintained by the individual itself, but rather by its friend class, `OperatorPool` (see below).

For member functions, as well as allowing access to its network object as a `const` pointer, an individual also allows indirect retrieval of the underlying network fitness (*i.e.* self-assessment) using its own member function.

`OperatorPool` class

In addition to containing and owning the set of operators utilised in a particular GA/heuristic hybrid algorithm, the operator pool is responsible for both breeding the population and managing the credit-assignment and operator-probability adaptation mechanisms.

The main `OperatorPool` class constructor takes two arguments: a parameter database and a pseudo-random number generator. The random number generator is stored, and from the parameter database several parameters are retrieved (Table E.11 on p. 249). The number of operators (excluding simple reproduction) is stored, and then the names and initial probabilities of the expected number of operators is retrieved from the database. Each of these is constructed and stored, as is the reproduction operator object, whose initial probability is simply whatever remains to ensure the sum of the initial probabilities is 1.0. The remaining parameters, each of which is stored, influence the credit-assignment and operator-probability mechanisms, discussion of which is largely deferred until §4.2.4 below.

As well as a member function to obtain the number of operators and an overloaded '`[]`' operator to retrieve `const` pointers to the individual operators themselves, the `OperatorPool` class provides only two member functions: `breed()` and `adaptOpProb()`.

The first of these creates a new population from an old one. Until the new population is full, it first chooses an operator from the pool at random, according to their current probabilities. Next, for unary operators, it selects an individual from the old population using the designated selection function (*e.g.* tournament selection), creates a new individual from this, and then applies the chosen operator. Both the operator's children and trials counts are increased, by 1 and the number of trials actually used (which may be 0, 1 or more), respectively. Then, the operator itself, as well as the individual's parent and grandparent operators, are awarded any credit earned, according to the applicable credit type (§4.2.4). Next, the individual's grandparent and parent operator data members are updated (the parent operator is copied to the grandparent, and the current to the parent). Finally, the new individual, possibly modified, is added to the new population. A similar sequence, except that *two* parent individuals are used, is executed for binary operators.

The second member function periodically adjusts the individual operator probabilities according to one of the algorithms discussed in §4.2.4 below.

`Operator`, `UnaryOp`, `BinaryOp` and `Reproduction` classes

The abstract virtual base class `Operator` [16,17], and its two derived abstract classes, `UnaryOp` and `BinaryOp` have already been mentioned briefly in §3.5 (Fig. 3.5), and are discussed at more length in §E.3.1 on p. 259 (Fig. E.2(a)). Nevertheless, before reviewing their many concrete derived classes in the following subsections, it is appropriate to review their public interfaces and data structures (Fig. 4.3).

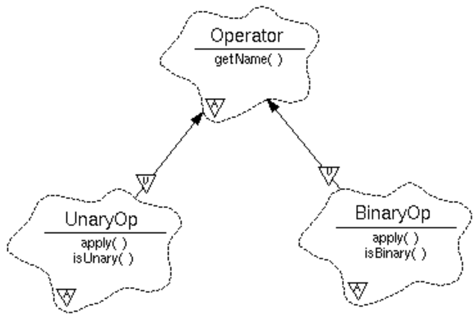


Figure 4.3: `Operator` class diagram

As with `Metric` (§4.1), the `Operator` class' main constructor requires a single argument, the operator name, which can be up to four characters, and must be unique within the toolset. For example, `DegreeTwo` uses `"DEG2"`, and `WaveKSPMutate`, `"WKM"`. In addition to its name, an operator has data members for: the number of children it has generated, the number of trials it has used, its probability of being applied, and the credit it has earned. All the latter four data members are maintained by the friend class `OperatorPool` and all five, including its name, can be accessed via member functions. In addition, the `Operator` class provides two virtual member functions, with the overloaded name `apply()`, that take either one or two network arguments, and return a Boolean to indicate success or failure. In the same way, it provides two virtual Boolean tests: `isUnary()` and `isBinary()`. All four virtual member functions are implemented in the base class to simply return `false`.

As the abstract base class of all unary operators, `UnaryOp` redefines as abstract virtual [16] the version of `apply()` with only one network argument, forcing all its derived classes to re-implement it for themselves. In addition, it also re-implements `isUnary()` to return `true`.

Class	Name	Description
`RandomLinks`	`RLS`	Random links
`MutateLink`	`ML`	Single-link mutation
`DegreeTwo`	`DEG2`	Degree-two operator
`DegreeDec`	`DEGD`	Degree-decrement operator
`RandomAddLink`	`RAL`	Random add link
`RandomRemLink`	`RRL`	Random remove link
`RandomChgLink`	`RCL`	Random change link
`RandomAddLinks`	`RALS`	Random add links
`RandomRemLinks`	`RRLS`	Random remove links
`RandomChgLinks`	`RCLS`	Random change links
`SPLinkCO`	`SPCO`	Single-point link crossover
`LinkGroupCO`	`LGCO`	Link-group crossover

Table 4.1: `Operators` for network topology design

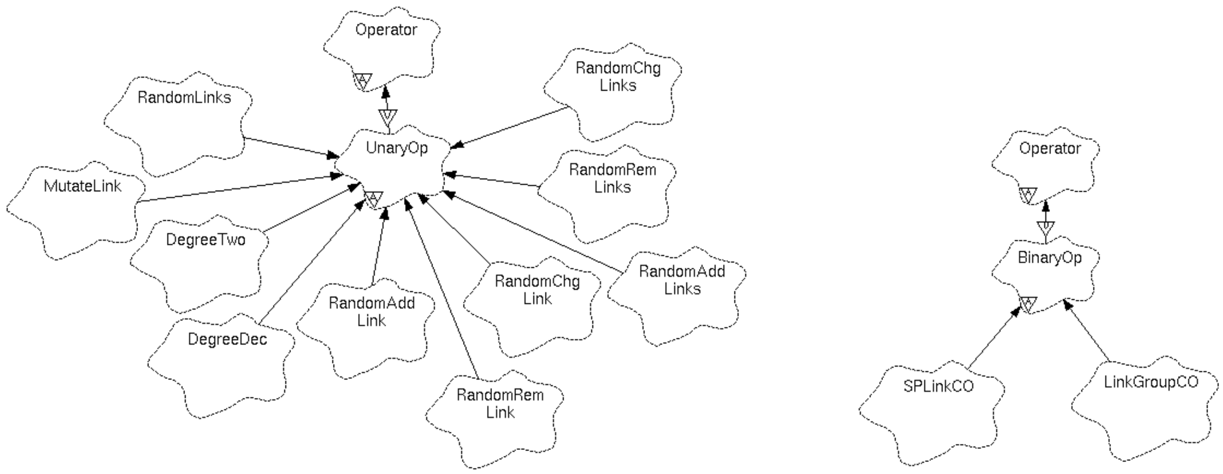
On the other hand, the `BinaryOp` class makes the complementary changes, redefining the other version of `apply()` as abstract virtual to force its own descendants to work with two networks at a time, and re-implementing `isBinary()` to return `true`.

`Reproduction`, derived from `UnaryOp`, is a very special concrete class: it does nothing, apart from return `true`, when applied! The reason for this is that in NOMaD's implementation of the overall GA framework, reproduction (*i.e.* cloning) of parent individuals is actually performed by the operator pool (see above). As a result, when simple reproduction is applied, as an operator, to the child individual, no further actions are required on its part. A reproduction operator object is still necessary, however, as it is capable of earning credit for its part in two- and three-operator sequences, just as any other operator, and is thus very much still part of the overall operator-probability adaptation mechanism (see §4.2.4 below).

4.2.2 Operators for Mesh Network Topology Design

The author has developed twelve operators for network topology design, ten of these unary and two binary. They are listed in Table 4.1, and the relationships between the classes illustrated in Fig. 4.4. Two of these (`RandomLinks` and `RandomAddLinks`) were mentioned in the author's paper at the 12th UK Performance Engineering Workshop [31], and more complete descriptions of another five (`MutateLink`, `DegreeTwo`, `DegreeDec`, `SPLinkCO` and `LinkGroupCO`) published by the author at the 3rd Intl. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97) [34] (both reprinted in Appendix A).

The *random links* operator is intended to establish a random network topology for use in the initial population of a GA/heuristic hybrid. For each node pair in the network, it flips an unbiased



(a) Unary (b) Binary

Figure 4.4: Topology-design operators class diagrams

coin to determine whether to directly connect the node pair with a link. It is implemented as NOMaD class `RandomLinks`.

Single-link mutation selects a random node pair and then, if they are already directly connected, removes their link, and if not, installs one. In NOMaD the operator is implemented as class `MutateLink`.

The *degree-two operator* alters a network by adding links to connect nodes to their nearest neighbours, where necessary, to make all nodes at least of degree two (*i.e.* connected to two other nodes). This ensures reasonable connectivity, perhaps avoiding a penalty, and hopefully thereby resulting in an improvement in the network assessment. It is implemented as NOMaD class `DegreeTwo`.

If there is at least one node of degree greater than two in the network, the *degree-decrement operator* removes a random link from the highest-degree node; if there is more than one such, then a random node is chosen from amongst these. Hopefully the removal of this potentially redundant link from such a highly-connected node will improve the overall network assessment. In NOMaD the operator is implemented as class `DegreeDec`.

The *random add link* operator randomly selects (*without* replacement) node pairs that are not directly connected until it finds the first pair between which adding a link improves the network assessment. If such a node pair is found, the link is added, and the operator has succeeded; if not, after all the pairs have been tried once, in random order, the operator has failed, and it leaves the network unchanged. The *random remove link* operator works in a complementary fashion, randomly selecting node pairs that *are* directly connected, until it finds the first pair between which removing their link improves the network assessment. A further variant is the *random change link* operator, which considers *all* node pairs, in random order, looking for the first change, either adding *or* removing a link, which results in an improvement. The three operators, which were inspired by branch exchange heuristics [11], are implemented as NOMaD classes `RandomAddLink`, `RandomRemLink` and `RandomChgLink`, respectively.

The next three operators augment the previous three by including iteration. *Random add links* initially acts identically to random add link, but after the first link is added, it 'forgets' which node pairs it has examined, and looks again for a pair between which adding a link results in an improvement. The operator is only complete when, having examined afresh all node pairs that are not directly connected, it cannot find a pair between which to successfully add a link. However, it is still regarded as having succeeded overall if, during the earlier iterations, it was able to add at least one link. Effectively, random add links is an iterative stochastic hill climber [29, 30]. In much the same way, *random remove links* extends random remove link, and *random change links*, extends the random change link operator. In NOMaD, the three operators are implemented as classes `RandomAddLinks`, `RandomRemLinks` and `RandomChgLinks`, respectively.

Link-group crossover selects a group of nodes in one network: those nodes contained within a circle, centred on a randomly-selected node, of radius defined by a second randomly-selected node on the circle's circumference. It then exchanges all the links used within that group with those used within the same group of nodes in the other network. It was believed that this would lead to better building blocks than a *single-point link crossover*, which assumes a linear encoding of the network topology, as has been employed earlier by some authors, including this one [150, 154, 166]. An example of link-group crossover in action, towards the end of a hybrid run, can be seen in Fig. 4.5. This shows two parent networks recombining by exchanging links between a (shaded) group of four nodes. One of the two resulting offspring (bottom right) is not only of lower network cost than either of its two parents, but was the best-of-run individual. The two operators are implemented as NOMaD classes `LinkGroupCO` and `SPLinkCO`.

4.2.3 Operators for Routing, Fibre & Wavelength Allocation

The author has also developed six routers (*i.e.* overall heuristics) for WP routing, fibre choice and wavelength allocation, plus one for virtual-wavelength-path (VWP). These are listed in Table 4.2, and the relationships between the classes illustrated in Fig. 4.6.

Wavelength-allocation random k-shortest paths is intended for the random, but reasonable ini-

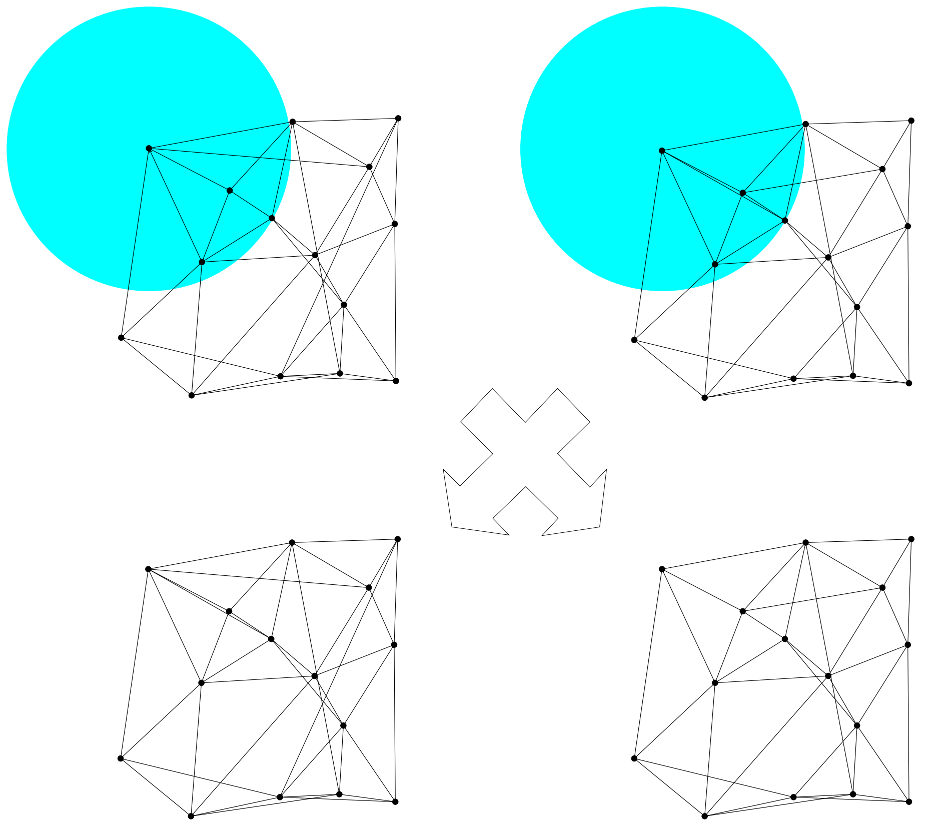


Figure 4.5: Example of link-group crossover

Class	Name	Description
`WaveRandKSP`	WRKS	Wavelength-allocation random k-shortest paths
`WaveNop`	WNOP	Wavelength-allocation no-op
`WaveWutti95`	WW95	Wuttisuttikulkij & O'Mahony (W&O) [298]
`WaveWauDijk`	WADI	Wauters & Demeester (W&D1) [306]
`WaveWauHRWA`	WAHR	Wauters & Demeester HRWA (W&D2) [306]
`WaveNagVWP`	WNVW	Nagatsu *et al.* VWP [296]
`WaveNagWP`	WNWP	Nagatsu *et al.* WP (NHS) [296]

Table 4.2: `Routers` for wavelength allocation

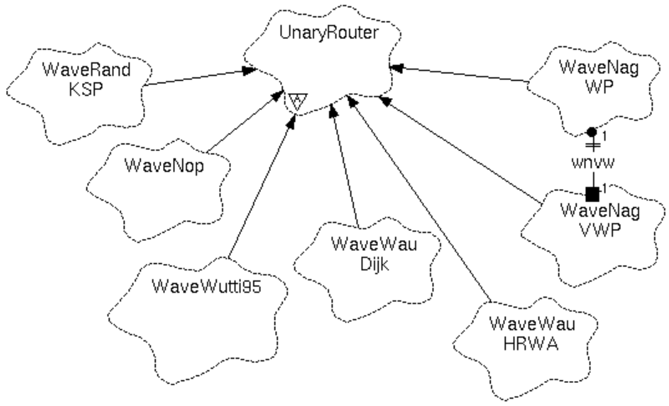


Figure 4.6: Wavelength-allocation routers class diagram

Class	Name	Description
`WaveKSPMutate`	`WKM`	Wavelength-allocation k-shortest path mutation
`WaveWauReroute`	`WWRR`	Wavelength-allocation path reroute operator
`WaveWauShiftOut`	`WWSO`	Wavelength-allocation path shift-out operator
`WaveKSPCO`	`WKCO`	Wavelength-allocation k-shortest path single-point crossover

Table 4.3: `Operators` for wavelength allocation

tialisation of WP routing, fibre choice and wavelength allocation for those networks in the initial population of a GA/heuristic hybrid. For each of the wavelength channels required between a node pair, a random path is selected from the k-shortest, and the wavelength channel routed along it on the lowest wavelength available end-to-end, irrespective of fibre choice (*i.e.* on each of the cables along the route, the lowest fibre id is used which has the required wavelength free). This router has been implemented in NOMaD as class `WaveRandKSP`, the parameters for which are given in Table E.12 on p. 249.

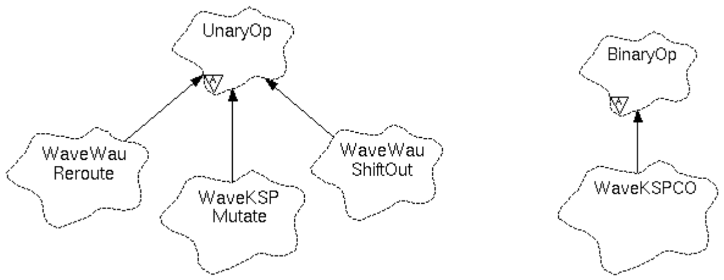
Wavelength-allocation no-op is the simplest of routers: it does nothing at all! It is implemented as class `WaveNop`, and is intended for use as the router data member of a wavelength-routed network object whose routing, fibre and wavelength allocation is to be determined by the external application of a router or GA/heuristic hybrid. If a wavelength-allocation no-op object is not installed in those circumstances, the network would reroute itself (possibly incorrectly) whenever it carried out a self-assessment (see §E.3, p. 253).

The remaining routers in Table 4.2 are all overall wavelength-allocation heuristics published in the literature. Details of these, and their implementation in NOMaD, can be found in Appendix C.

As well as the routers, described above, the author has developed four operators for WP routing, fibre choice and wavelength allocation, three unary and one binary. These are listed in Table 4.3, and the relationships between the classes illustrated in Fig. 4.7. All four were first published by the author in *Electronic Letters* [35] (reprinted in Appendix A).

For a random requirement (*i.e.* an individual wavelength channel of those required between one of the node pairs in the network), *wavelength-allocation k-shortest path mutation* reroutes the wavelength channel on a randomly chosen k-shortest path (where k is an operator parameter), using the lowest wavelength available end-to-end on that path (irrespective of fibre choice). The operator has been implemented as class `WaveKSPMutate`, the parameters of which are given in Table E.13 on p. 249.

The *wavelength-allocation path reroute operator* locates the first wavelength channel in the network which has the highest wavelength number (found by searching the cables in order of their



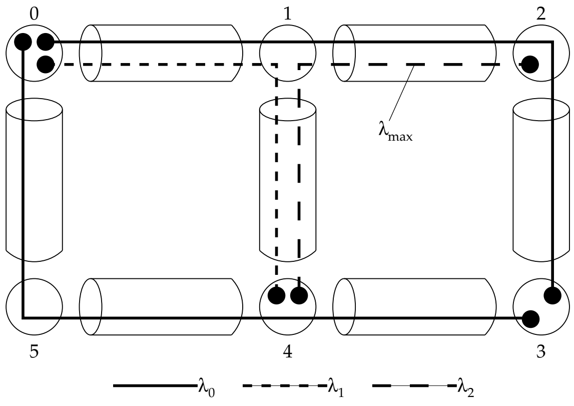
(a) Unary (b) Binary

Figure 4.7: Wavelength-allocation operators class diagrams

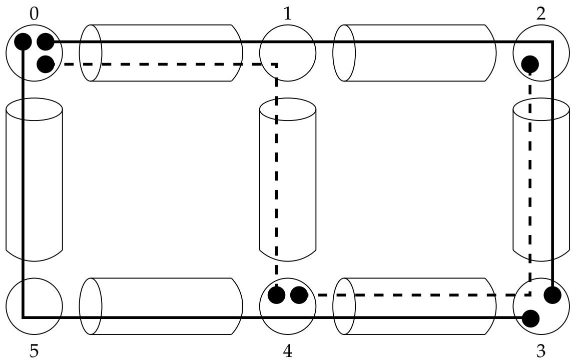
end-node ids), and attempts to reroute it on a lower wavelength number (and the lowest available end-to-end) using one of the k-shortest paths (where k is an operator parameter, and the shortest path is selected if there is more than one at the lowest available wavelength number). It is based on Step 2a in Wauters and Demeester's HRWA algorithm [306] (Appendix C). As an example, Fig. 4.8(a) shows a 6-node, 7-link network, with single-fibre cables, over which four WP wavelength channels have been routed. The wavelength channel using the highest wavelength number (λ_2) is labelled $\lambda_{\max}$. In Fig. 4.8(b), after applying the reroute operator, that channel has been successfully rerouted on a lower wavelength, λ_1. As a result, the network wavelength requirement has been reduced from three to two wavelengths. Within NOMaD, the operator has been implemented as class `WaveWauReroute`, the parameters of which are given in Table E.14 on p. 249.

The *wavelength-allocation path shift-out operator* locates the first wavelength channel in the network which has the highest wavelength number, and attempts to shift it to a lower wavelength number by shifting out (rerouting) all the paths blocking that particular wavelength number end-to-end. The wavelength chosen is the one that results in the smallest increase in overall path length, given the constraint that all rerouting must be to lower wavelength numbers than the original wavelength of the channel being shifted down. It is based on Step 2b in HRWA [306] (Appendix C). Illustrating this, Fig. 4.8(c) shows the original network of Fig. 4.8(a) after shift-out has been applied. By rerouting the blocking wavelength channel on λ_1, it has proved possible to reduce the wavelength number of channel $\lambda_{\max}$ itself to λ_1. This change has again lowered the network wavelength requirement from three to two. Within NOMaD, the operator has been implemented as class `WaveWauShiftOut`, the parameters of which are given in Table E.15 on p. 249.

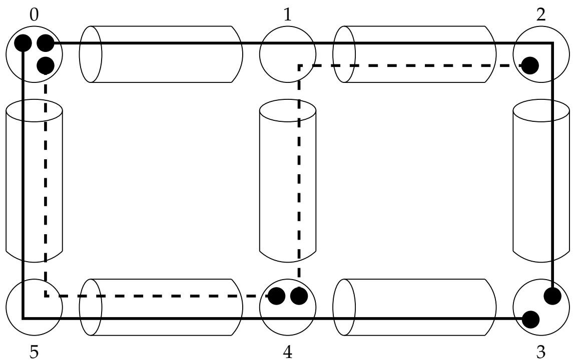
Wavelength-allocation k-shortest path single-point crossover operates on two networks. The first is modified such that up to a randomly selected individual requirement (considered in order



(a) Original network



(b) Reroute



(c) Shift-out

Figure 4.8: Example of reroute and shift-out operators

of their end-node ids), all the wavelength channels remain unchanged, but from then on the paths used are those in the second network, except that the lowest wavelength actually available end-to-end in the first network is used in each case. (However, if any requirement was unsatisfied in the second network, then a random k-shortest path is used instead.) The second network undergoes the complementary recombination. The operator has been implemented as class `WaveKSPCO`, the parameters of which are given in Table E.16 on p. 249.

4.2.4 Operator-probability Adaptation

As a variety of operators are incorporated in NOMaD's different GA/heuristic hybrid algorithms, it might seem necessary to determine appropriate operator probabilities for each of the distinct combinations of operators used. However, this implies a large computational effort. Instead, the probabilities of individual operators being applied to an individual adapt during the course of a run using a credit-assignment algorithm, rather than being fixed throughout, as was mentioned in §3.5 above.

As well as avoiding the need to determine appropriate fixed operator probabilities (except for the initial probabilities, which could be selected from a few trial runs), it was also hoped that there might be performance gains in allowing the probabilities to vary during the course of a run, such that the most appropriate blend of operators was used at each stage, as determined by the observed operator productivity.

The operator-probability adaptation algorithm adopted is based on that of Davis [9], and although mentioned in three papers by the author [34–36], was given its first detailed treatment at the IEEE Intl. Conf. on Evolutionary Computation (ICEC'98) [33] (all four reprinted in Appendix A). First, the basic description of the algorithm, as published in [33], is given, and then a number of further variants presented.

Basic Adaptation Algorithm

Whenever an individual is created whose fitness exceeds that of the best individual found up to the end of the previous generation, the improvement in fitness is credited to the operator responsible:

$$c'_u = c_u + (f_b - f_i), \quad f_i < f_b \tag{4.14}$$

where c'_u and c_u are the current and previous values of the accumulated credit of operator u, f_b is the best fitness found at the end of the previous generation, and f_i is the fitness of individual i. In addition, as has already been mentioned, a decreasing fraction of the credit (say, $\kappa = 0.5$ and $\kappa^2 = 0.25$) is awarded to the parent operator (*i.e.* the operator that created this individual's parent) and grandparent operator, to ensure that two- and three-operator sequences are rewarded appropriately. Thus:

$$c'_p = c_p + (f_b - f_i) \cdot \kappa, \quad f_i < f_b \tag{4.15}$$

$$c'_g = c_g + (f_b - f_i) \cdot \kappa^2, \quad f_i < f_b \tag{4.16}$$

where c_p and c_g are the accumulated credit of the parent (p) and grandparent (g) operators, respectively[6], and κ is the credit constant ($0 \leq \kappa \leq 1$).

It should be noted that, while Davis [9] maintained a complete tree (although of finite depth) of the operators responsible for the creation of an individual, here only a single (arbitrary) parent is recorded for binary operators. This greatly simplifies accounting, as instead of up to two parents, four grandparents, eight great-grandparents, *etc.*, only a single parent and grandparent operator have to be recorded. Indeed, the overhead of maintaining and crediting a complete operator tree has already been noted elsewhere [265]. Further, although the selection of a single arbitrary parent operator means that all the operators responsible for a superior individual are not rewarded, it would be expected that these unrewarded operators would have subsequent opportunities to acquire credit, and it is their relative accumulation of credit, rather than the absolute value, that is important.

After a few generations (say, $\epsilon = 4$), a small proportion (say, $\psi = 0.15$) of the operator probabilities is reassigned according to the average credit earned per child each operator has generated. In addition, operators are given a minimum probability (say, $p_{\min} = 0.05$ for less than 20 operators) to ensure they do not lose the opportunity to gain some credit if they, having decayed to the minimum level, are later found to be useful to further evolve the population. Thus, every ϵ generations, for all operators, first:

$$a_u = \begin{cases} c_u / o_u, & o_u > 0 \\ 0, & o_u = 0 \end{cases} \tag{4.17}$$

where, for operator u, a_u is the adaptation weight and o_u the number of children (*i.e.* offspring)

[6]The three operators, u, p and g are not necessarily different: the same operator may have been applied to a single individual two or three times in sequence.

generated. The total adaptation weight is then given by:

$$a = \sum_u a_u \tag{4.18}$$

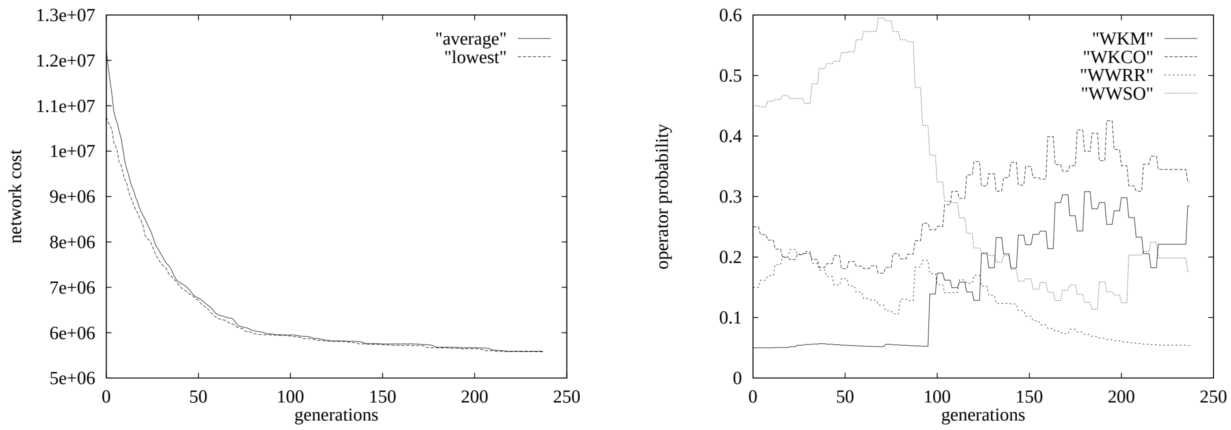
Now, provided some credit has been earned, and thus a is non-zero, probability reassignment can proceed. However, before this an adaptation scaling factor (K_a) must be calculated. This is the amount of the total probability that is to be reassigned, whilst still protecting the minimum probability level of all operators, rescaled using the total adaptation weight, thus:

$$K_a = \psi \cdot (1 - (s+1)p_{\min})/a, \quad a > 0, \quad (s+1)p_{\min} < 1 \tag{4.19}$$

where ψ is the adaptation proportion ($0 \leq \psi \leq 1$), s is the number of operators (excluding reproduction, hence the addition of 1) and $p_{\min}$ is the minimum operator probability level. All the operator probabilities can then be then reassigned as follows:

$$p'_u = (p_u - p_{\min}) \cdot (1 - \psi) + a_u K_a + p_{\min} \tag{4.20}$$

where p_u is the probability of operator u. The first term represents the unchanged part of the operator's probability, the second, that element reassigned according to adaptation weight, and the final one, the guaranteed minimum probability. It is straightforward to verify that, after the application of Eqn. 4.20, the $\sum_u p_u$ remains 1. Finally, before continuing, the accumulated credit (c_u), number of children (o_u) and number of trials (t_u, see below) of all operators are reset to zero.



(a) Progress over a single run

(b) Operator-probability adaptation

Figure 4.9: Example GA/heuristic hybrid run

As an example of the operator-probability adaptation mechanism, Fig. 4.9(a) illustrates the progress of a typical wavelength-allocation hybrid run, showing the lowest and average population cost against generation (Network Cost Model B). Fig. 4.9(b) gives the operator probabilities for the same run, where the effect of the adaptation algorithm can be clearly seen[7]. For example, wavelength-allocation k-shortest path mutation (WKM), which started with a low probability, became much more productive after generation 100; whereas the wavelength-allocation path shift-out operator (WWSO), which had the highest probability in early generations, was found to be much less useful towards the end of the run.

Variants on the Basic Algorithm

Variations to both the basic credit-assignment and operator-probability adaptation mechanisms have also been developed. First, in the basic credit-assignment algorithm, operators are rewarded as a result of improvement over the fitness of the best individual found up to the end of the previous generation. Instead, credit can be awarded for improvement over:

- best-in-window fitness (*i.e.* the best fitness achieved in the previous w generations), f_w [265, 271];
- the median fitness of the population at the end of the previous generation, f_m [270];
- or the fitness of the parent individual (or the fitter of the two parents, for binary operators), f_p [265, 271].

Thus, Eqn. 4.14 (and in a similar way, Eqn. 4.15 and 4.16) is altered to one of:

$$c'_u = c_u + (f_w - f_i), \quad f_i < f_w \tag{4.21}$$

$$c'_u = c_u + (f_m - f_i), \quad f_i < f_m \tag{4.22}$$

$$c'_u = c_u + (f_p - f_i), \quad f_i < f_p \tag{4.23}$$

where f_w is the best-in-window fitness at the end of the previous generation, likewise, f_m is the median fitness found at the end of the previous generation, and f_p is the (better) fitness of the parent(s) of individual i.

For the operator-probability algorithm, in the basic approach, operator credit was divided by the number of children each operator had generated before being used to reassign operator probabilities.

[7] The reproduction operator has been omitted for clarity.

However, if an operator uses more than one trial per child generated, this may unfairly favour operators that are expensive in their use of trials (*e.g.* `RandomChgLink`, `RandomChgLinks`, *etc.*). An alternative is to use the number of trials itself as the divisor before probability reassignment. Thus, Eqn. 4.17 is altered to:

$$a_u = \begin{cases} c_u/t_u, & t_u > 0 \\ 0, & t_u = 0 \end{cases} \tag{4.24}$$

where t_u is the number of trials used by operator u. However, this change also has an interesting secondary consequence, as not all operator applications require trial(s), due to assessment being cached by the network (§E.3.1 and §4.2.1): if the network is unchanged, the operator has been applied 'for free'.

4.3 Summary

This chapter has provided details of the design metrics used by the author for network assessment: one for minimum-cost mesh network topology design, another for network wavelength requirement, and the final one for minimum-cost wavelength allocation. Moreover, the evolutionary algorithm framework has been described, including the overall structure of the GA/heuristic hybrid; operators for network topology design; operators for routing, fibre and wavelength allocation; and the operator-probability adaptation mechanism. In the next chapter, experimental results will be reported exploring the suitability of the GA/heuristic hybrid approach for multi-wavelength all-optical transport network design.

Chapter 5

Experimental Results

There are three kinds of lies:
lies, damn lies, and statistics.
Mark Twain, Autobiography

This chapter reports experimental results exploring the suitability of the author's genetic-algorithm (GA)/heuristic hybrid approach for multi-wavelength all-optical transport network design.

As a basis for the experimental work described here, five network design problems were generated using the approach described in §F.1. Each problem has fifteen nodes, placed within a 1,000 km $\times$ 1,000 km area using a minimum clear radius constant of 0.5, and is intended to carry an overall static traffic requirement of 1,500 Gbit/s. The node distributions of the five problems are illustrated in Fig. F.3 and the node coordinates and traffic matrices have been recorded in Tables F.1–F.10 in Appendix F.

The chapter is divided into four sections. Results for mesh network topology design are presented in §5.1, and for wavelength-path (WP) routing, fibre choice and wavelength allocation in §5.2. An investigation into NOMaD's operator-probability adaptation mechanism forms §5.3. Finally, the chapter is summarised in §5.4.

5.1 Mesh Network Topology Design

Network topology design, briefly introduced in §3.1, determines which node pairs in the network are to be directly connected given the node locations and static traffic requirements. For mesh network topology design, the choice of node pairs to interconnect is unconstrained, resulting in an arbitrary topology. In this thesis, the metric used for this is Network Cost Model A, outlined in §4.1.1 on p. 68. To summarise, this metric, given a network topology, routes all the primary and protection

GA	ML	SPCO	LGCO	DEG2	DEGD
1	0.35	0.45			
2	0.35		0.45		
3	0.25		0.35	0.10	0.10

Table 5.1: Initial operator probabilities – topology

paths, determines the link capacities, and then uses link and node cost models to estimate an overall cost for the network. It incorporates, where appropriate, penalties for either failing to route, or to provide a protection path, for each individual traffic requirement. However, while it does take into account the high relative cost of optical switching in selecting paths, it does not include fibre choice or wavelength allocation. Clearly, the overall design objective is to minimise this network cost.

5.1.1 Initial Topology Experiments

The twelve operators developed by the author for mesh network topology design within NOMaD were described in §4.2.2 on p. 80. In devising experiments based on these, it was decided initially to focus on comparing a more straightforward algorithm, using just single-link mutation (ML) and single-point crossover (SPCO)[1], first with one incorporating a problem-specific crossover instead, link-group crossover (LGCO), and then adding in two problem-specific heuristics, the degree-two (DEG2) and degree-decrement (DEGD) operators. A subset of these experiments was first published by the author at the 3rd Intl. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97) [34] (reprinted in Appendix A).

The three GAs employed each had a population size of 200, used tournament selection (of size 4), and were limited to a maximum of 5,000 trials (*i.e.* somewhat more than 25 generations). The operators used and their initial probabilities are given in Table 5.1. The latter were selected by a few test runs, aiming for settings that would remain approximately constant for the first few generations across all the network design problems tackled. It could be argued that such starting values best promote high-quality solutions by approximately matching operator probabilities to their initial productivities.

It should be emphasised that for all three GAs the operator-probability adaptation mechanism described in §4.2.4 on p. 87 was employed. Default values for this were used throughout *i.e.* credit was awarded for improvement over the best fitness found at the end of the previous generation; the credit constant, κ, was 0.5; adaptation occurred every $\epsilon = 4$ generations; adaptation weight was

[1]However, recall that this is not a traditional bit-string GA, as both an object-oriented representation and operator-probability adaptation are being employed.

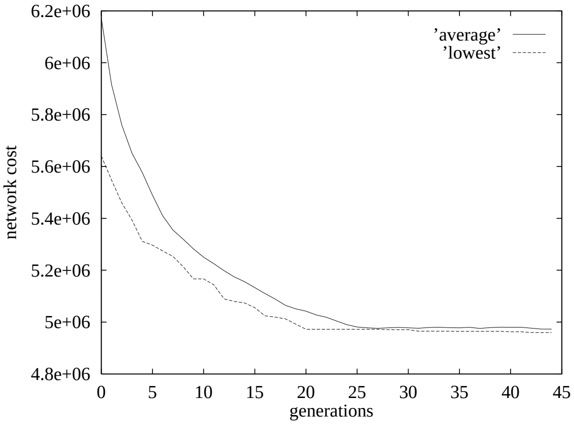
calculated *wrt.* children generated; the minimum probability level, $p_{\min}$, was set to 0.05; and the adaptation proportion, ψ, was taken to be 0.15.

The three GAs were each applied ten times with different pseudo-random number generator seeds to each of the five network design problems listed in §F.2. However, in assessing the designs, non-default values were used for certain parameters of the Network Cost Model A. The default values were originally selected by the author for design of the COST 239 European Optical Network [166], but this has both a larger number of nodes and covers a greater geographical area than the five problems considered here. Consequently, more appropriate parameter values needed to be selected. These, and the other non-default parameter values common to the three GAs are listed in Table F.11 in Appendix F.

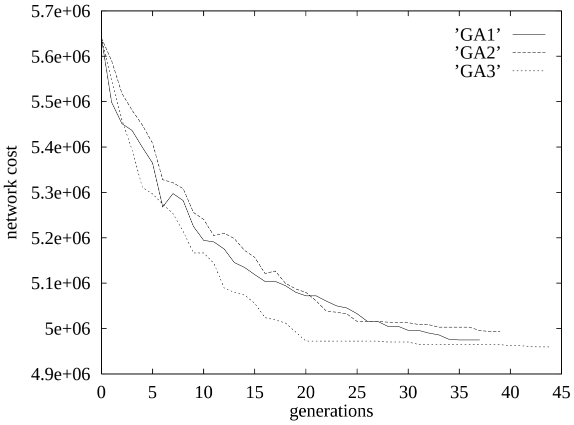
The progress of a single typical run of GA3 is given in Fig. 5.1(a), showing both the average network cost of the population and the lowest (*i.e.* best in population) cost, generation by generation. As can be seen from the average cost curve, after Generation 25 the population had substantially converged to the lowest network cost. The evolution of the best individual from the same run is illustrated in Fig. 5.2, which shows first the best individual in the initial random population (Generation 0), then at four subsequent points in the run, and finally the best overall (Generation 43). Considerable cost reductions were achieved in the first 13 generations, followed by more gradual refinement over the next 30. The final network obtained is close (at 4.960×10^6) to both the best obtained over ten runs each by the three GAs (4.953×10^6; not illustrated) and over five runs each (4.954×10^6), illustrated in Fig. 5.3(a) on p. 99. Comparing Figures 5.2(f) and 5.3(a), it can be seen that the two networks differ in only a few links, resulting in the small cost difference.

To compare the progress of typical runs from all three GAs, their lowest costs from individual runs all employing the same random seed, and thus starting with the same initial population, are given in Fig. 5.1(b). All three GAs employed the same maximum number of trials[2], but due to the different operators employed and their varying probabilities, the number of generations used necessarily differed. Clearly, at least for these three runs, GA3, with both problem-specific crossover and heuristics (in addition to mutation), made the most rapid progress and achieved the lowest cost. However, in this instance, the next lowest cost was achieved by GA1, just using mutation and single-point crossover. An additional point of interest in Fig. 5.1(b), is that the curve for GA1 clearly illustrates the non-elitist strategy employed by NOMaD: the lowest network cost *increases* from

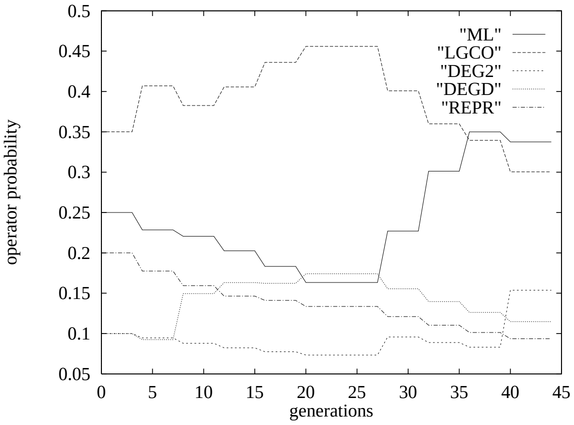
[2] Actual trials used will differ slightly, as here a GA is always allowed to complete the processing of its final generation, even if its maximum number of trials has been exceeded—see §3.9 and the description of the `GeneticAlgorithm` class on p. 74.



(a) Progress over a single GA3 run

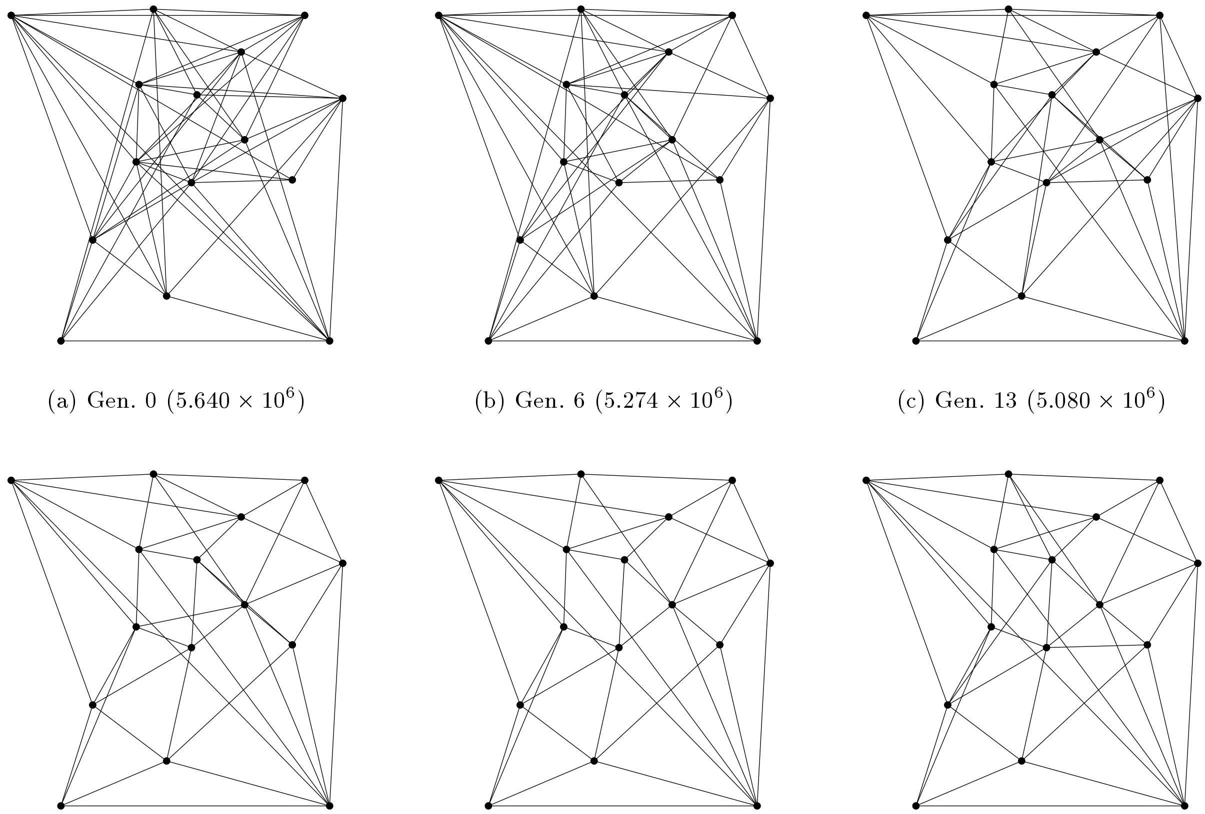


(b) Comparing lowest costs



(c) Operator-probability adaptation for GA3

Figure 5.1: Typical initial topology results (Problem 1, Run 1)



(a) Gen. 0 (5.640×10^6)

(b) Gen. 6 (5.274×10^6)

(c) Gen. 13 (5.080×10^6)

(d) Gen. 20 (4.972×10^6)

(e) Gen. 31 (4.965×10^6)

(f) Gen. 43 (4.960×10^6)

Figure 5.2: Best topologies over a single GA3 run (Problem 1, Run 1)

Problem	GA1 Best ($\times 10^6$)	GA1 Median ($\times 10^6$)	GA2 Best ($\times 10^6$)	GA2 Median ($\times 10^6$)	GA3 Best ($\times 10^6$)	GA3 Median ($\times 10^6$)	Significance Level (%) $s_{1,2}$	$s_{1,3}$	$s_{2,3}$
1	4.975	4.994	**4.953**	**4.975**	4.954	4.977	5.5	5.5	—
2	4.772	4.789	**4.751**	**4.764**	4.755	4.767	1.1	0.1	—
3	4.446	4.486	4.415	**4.421**	**4.404**	4.430	0.1	0.1	—
4	4.618	4.641	4.593	**4.612**	**4.591**	4.614	0.1	1.1	—
5	4.461	4.484	**4.424**	**4.450**	4.438	4.454	5.5	1.1	—

Table 5.2: Cost results for initial topology experiments

Generation 5 to 6. The lowest cost in the population is thus clearly distinct from the best so far.

The operation of the operator-probability adaptation mechanism for the same run of GA3 can be seen in Fig. 5.1(c). For the first 25 generations, while the population is converging—see Fig. 5.1(a), the operator probabilities remain substantially unchanged, with the majority of the work being undertaken by link-group crossover (LGCO). However, after this point, single-link mutation (ML) makes an increasing contribution, and link-group crossover less so, perhaps reflecting the need to introduce greater population diversity. It may be further speculated that the increase in probability of the degree-two operator (DEG2) at the end of the run results from an increased need to repair nodes almost or completely disconnected by destructive mutations.

In addition to the results provided in Fig. 5.1, the interested reader may like to note that further topology results, showing typical progress and operator-probability adaptation over single runs of both GA1 and GA2 on Problem 1, are illustrated in Fig. F.4.

Statistical comparison of the quality of results from the three GAs is given in Table 5.2. This records the best and median results from each group of ten runs, with both the best overall result and best median for each problem displayed in bold. In addition, as the same random seeds were used for each set of runs on a problem, a non-parametric sign test could be applied to matched pairs [307] from different GAs to establish if there were significant differences in their medians (for details, see §F.3.1 in Appendix F). The resulting significance levels are included in the table, with *e.g.* $s_{1,2}$ representing the significance level of the comparison of GA1 with GA2. For completeness, the full set of results from applying the GAs to all five of the network design problems, summarised here, is recorded in Tables F.12–F.14.

Surprisingly, the best medians for all five problems and the best overall result for three out of five were obtained by GA2, whereas GA3 only achieved the best result for two out of five problems. Clearly (for the problem sizes and types examined), adding two problem-specific heuristics to GA2, the degree-two and degree-decrement operators, so as to create GA3 has not resulted in the hoped for improvement in the final result quality. If anything, the reverse, with the necessary reduction

Network	1	2	3	4	5	6	7
Average node degree	5.33	5.07	5.07	4.93	4.80	4.00	3.33

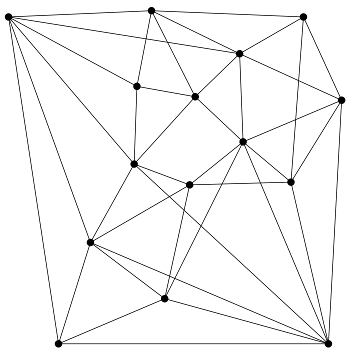
Table 5.3: Test network average node degrees

in link-group crossover probability to accommodate the two additional operators (see Table 5.1 on p. 93) perhaps responsible for the reduction in solution quality. However, it should be noted that these apparent differences between GA2 and GA3 are not reflected in a significant difference in their medians. In contrast, though, both GA2 and GA3 are significantly, and in some cases very highly significantly, better than GA1 in terms of median differences. Thus, as anticipated, employing link-group crossover provides a consistent improvement in solution quality over the use of single-point crossover.

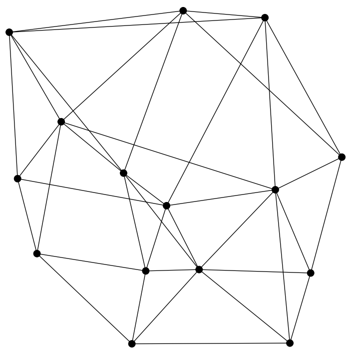
5.1.2 Test Networks

In addition to comparing different GAs in the initial experimental work, the opportunity was taken to select several topologies for use in subsequent wavelength allocation work. The initial choice of five networks was made when only five runs had been completed by each of the three GAs on the five network design problems introduced in §F.2. Consequently, it was the lowest-cost topologies at this stage, rather than the best of ten runs each, recorded in Table 5.2, that were selected. These five networks are illustrated in Fig. 5.3, and details of both their selection and their links are recorded in Tables F.20 and F.21, respectively, in Appendix F.

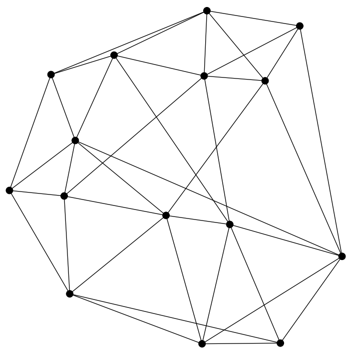
Inspection of the five test networks revealed, however, that they were all fairly similar and had relatively high average node degree, so to provide more diverse topologies, two further networks were generated from Problem 1. These have increasingly lower average node degrees, achieved by suitably modifying the parameters of Network Cost Model A. The actual parameter sets used are recorded in Tables F.16 and F.17. The first introduces a non-linear capacity dependency by lowering the value of α to 0.8; the second also increases the relative cost of network nodes by increasing K_0 and K_n. The results of applying GA2 and GA3 five times each, using the two parameter sets in turn, are given in Tables F.18 and F.19, respectively. The two networks selected (the lowest cost for each of the two parameter sets) are illustrated in Fig. 5.4, and then the average node degrees for all seven test networks are recorded in Table 5.3.



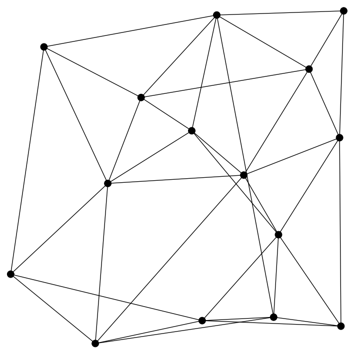
(a) Network 1



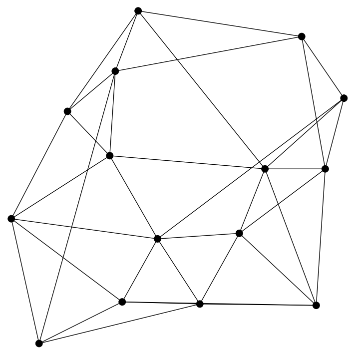
(b) Network 2



(c) Network 3

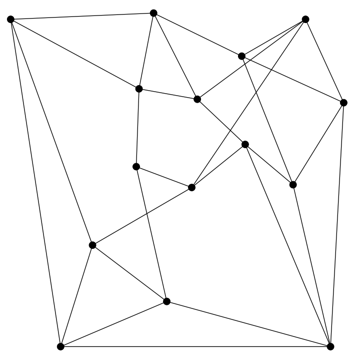


(d) Network 4

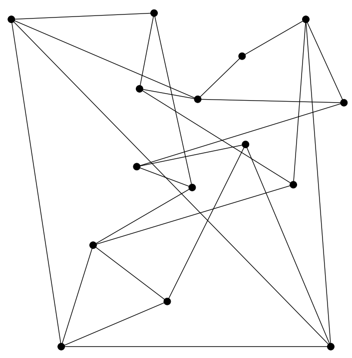


(e) Network 5

Figure 5.3: Test networks



(a) Network 6



(b) Network 7

Figure 5.4: Low-average-node-degree test networks

5.1.3 Local Search Operators

The second set of topology experiments focuses on a comparison between a straightforward GA employing just single-link mutation (ML) and link-group crossover (LGCO), and GAs which also include one or more local search heuristics. These are random add link (RAL), random remove link (RRL) and random change link (RCL; all three described in §4.2.2 on p. 81). These operators make a series of random link additions, removals or changes, respectively, until they either make an improvement or detect they are trapped in a local optimum.

In addition, as each of these operators can potentially use many trials whilst performing local search, it was felt appropriate to investigate the per-trial variant of the basic operator-probability adaptation mechanism (see §4.2.4 on p. 90). In the basic approach, operator credit is divided by the number of children each operator has generated before being used to reassign operator probabilities. However, if an operator uses more than one trial per child generated, this may unfairly favour operators that are expensive in their use of trials. An alternative is to use the number of trials itself as the divisor before probability reassignment. Thus an operator will have to provide benefit *proportional to the number of trials used* to gain probability share.

The nine GAs each had a population size of 200, used tournament selection (of size 4), and were limited to a maximum of 15,000 trials. The larger number of trials (*cf.* GA1–GA3) was discovered to be necessary (using a few test runs) to allow those GAs including local search operators to complete a reasonable number of generations (more than 25)[3]. The operators used and their initial probabilities are given in Table 5.4. Other non-default parameter values common to the nine GAs are listed in Table F.22 in Appendix F. Essentially, these GAs differ from those used in the initial topology experiments only in respect to: choice of operators and initial probabilities, increased run length, and whether per-child or per-trial operator-probability reassignment is used. In respect of the latter parameter, GA4, GA5, GA7, GA9 and GA11 all use child-based, whereas GA6, GA8, GA10 and GA12 use trial-based reassignment. In particular, GA4 is identical to the successful GA2 from the initial experiments, except for its larger maximum number of trials.

The nine GAs were each applied ten times with different random seeds to each of the five network design problems listed in §F.2. The progress of a single typical run of GA4 on Problem 1 is given in Fig. 5.5(a), showing both the average network cost of the population and the lowest (*i.e.* best in population) cost, generation by generation. The operator-probability adaptation curves for

[3]It could be argued that this consequently gives an unfair advantage to those GAs incorporating local search operators.

GA	ML	LGCO	RAL	RRL	RCL
4	0.35	0.45			
5/6	0.15	0.40	0.30		
7/8	0.15	0.40		0.30	
9/10	0.15	0.40			0.30
11/12	0.15	0.40	0.15	0.15	

Table 5.4: Initial operator probabilities – topology using local search operators

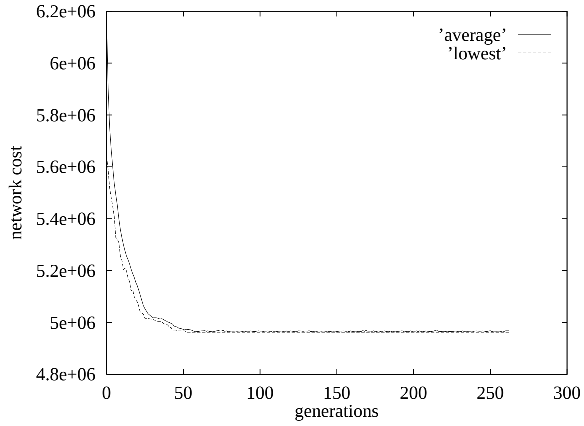
the same run are shown in Fig. 5.5(b). The adaptation curves can clearly be seen to have 'flat lined' after about Generation 50. The reason for this is that the adaptation mechanism is using the default credit-assignment approach, which rewards operators for improvements over the best so far individual. If no further improvement is made, then the operator probabilities are left unchanged, giving this 'flat line' appearance[4]. Thus, GA4 is hardly benefiting from the increased run length over the earlier GA2, which ran for nearly 40 generations (see Fig. F.4(c) on p. 310), with most evolution occurring in the first 30.

To provide a comparison between two otherwise identical GAs using either per-child or per-trial probability reassignment, Figures 5.5(c) and (d) show the adaptation curves for typical runs of GA9 and GA10, respectively, on Problem 1. Both GAs use the random change link operator (RCL), and in Fig. 5.5(c), with per-child reassignment, the random change link operator can be seen gaining probability largely at the expense of link-group crossover (LGCO). However, in Fig. 5.5(d), with per-trial allocation, a fairer assessment of the contribution of this multi-trial local search heuristic results in its probability declining, and link-group crossover earning an increasing share. Similar changes, resulting from using per-trial allocation, can be seen in comparing adaptation curves for typical runs of GA5 and GA6 (employing the random add link operator, RAL) and between GA7 and GA8 (with random remove link, RRL)—see Figures F.5 and F.6.

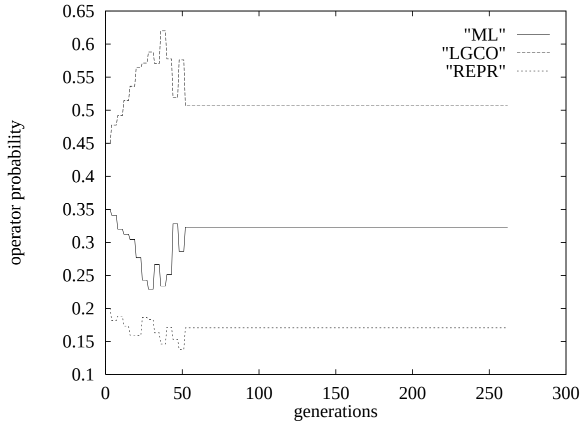
The lowest cost curves for five GAs, all but GA4 employing trial-based reassignment[5], and all using the same random seed and thus starting with the same initial population, are given in Figures 5.5(e) and (f). The former is plotted on a generation-by-generation basis, whereas the latter shows network cost against trials. As was noted *wrt.* the initial experiments, although all five GAs

[4]An investigation into *why* no further improvement occurred indicated that the population had almost completely converged, at least in fitness terms, and that single-link mutation was unable to discover further improvements over the best-so-far fitness. This was confirmed by increasing the level of mutation to 0.8, both with and without operator probability adaptation; in both cases a similar final outcome was observed. Thus, the use of a more disruptive mutation operator (changing more than one link at a time) may be beneficial, or perhaps the use of local search operators, the approach adopted here.

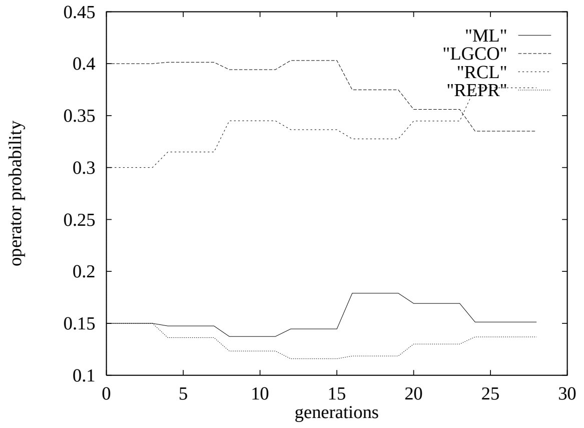
[5]However, as GA4 does not include any multi-trial operators, the run would have been (almost) identical using trial-based reassignment. The only differences would arise because of the different handling of the reproduction operator—see §4.2.4 on p. 90.



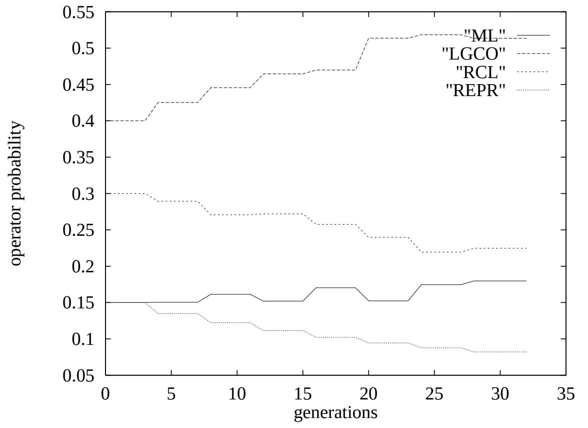
(a) Progress over a single GA4 run



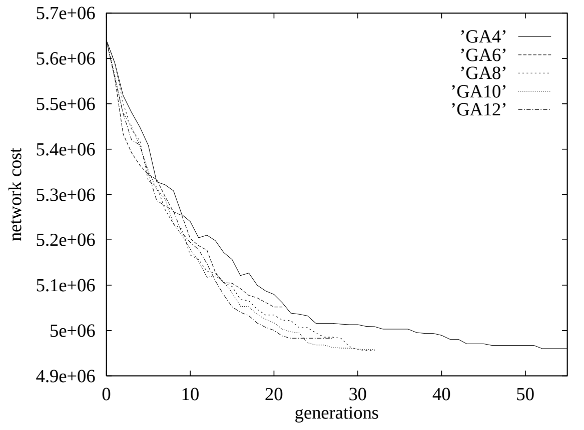
(b) Operator-probability adaptation for GA4



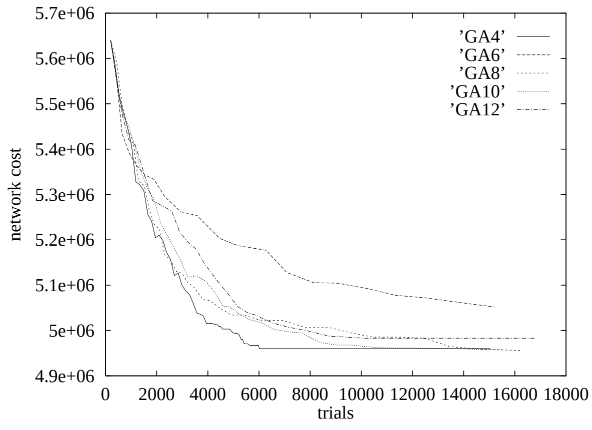
(c) Operator-probability adaptation for GA9



(d) Operator-probability adaptation for GA10



(e) Comparing lowest costs (*vs.* generations)



(f) Comparing lowest costs (*vs.* trials)

Figure 5.5: Typical topology results using local search operators (Problem 1, Run 1)

Problem	GA4 ($\times 10^6$)	GA5 ($\times 10^6$)	GA6 ($\times 10^6$)	GA7 ($\times 10^6$)	GA8 ($\times 10^6$)	GA9 ($\times 10^6$)	GA10 ($\times 10^6$)	GA11 ($\times 10^6$)	GA12 ($\times 10^6$)
1	4.947	4.988	4.987	4.953	4.956	4.947	**4.946**	4.957	4.951
2	4.746	4.838	4.809	4.747	**4.745**	4.747	4.748	4.750	4.753
3	4.409	4.474	4.466	4.418	4.407	**4.400**	4.408	4.447	4.415
4	4.591	4.692	4.616	4.592	4.598	4.594	**4.589**	4.619	4.593
5	4.424	4.513	4.452	4.435	4.434	**4.417**	4.426	4.439	4.423

Table 5.5: Best cost results for topology using local search operators

Problem	GA4 ($\times 10^6$)	GA5 ($\times 10^6$)	GA6 ($\times 10^6$)	GA7 ($\times 10^6$)	GA8 ($\times 10^6$)	GA9 ($\times 10^6$)	GA10 ($\times 10^6$)	GA11 ($\times 10^6$)	GA12 ($\times 10^6$)
1	4.958	5.062	5.021	4.980	4.969	4.962	**4.957**	4.986	4.975
2	**4.753**	4.871	4.840	4.766	4.763	4.758	4.755	4.774	4.759
3	**4.412**	4.539	4.496	4.425	4.420	4.421	4.414	4.463	4.430
4	**4.601**	4.718	4.672	4.607	4.608	4.604	4.602	4.639	4.616
5	**4.430**	4.569	4.527	4.452	4.452	4.440	4.438	4.473	4.459

Table 5.6: Median cost results for topology using local search operators

employed the same maximum number of trials, due to the different operators employed and their varying probabilities, the number of generations used necessarily differed. This effect is particularly marked here as, since GA4 does not employ local search operators, it uses considerably fewer trials per generation, and thus continues for more than 250 generations. Consequently, Fig. 5.5(f) provides the better basis for comparison. Overall, the best result for these runs is obtained by GA8, closely followed by GA10, however GA4 not only showed the most rapid progress, but also provided the best overall result up to 10,000 trials (although it seemed to make no further improvement after about the first 6,000 trials). Only by extending their runs beyond this point were two of the GAs incorporating local search operators able to produce (small) improvements in final solution quality over GA4.

Supplementing the results provided in Fig. 5.5, additional topology results, showing typical progress and operator-probability adaptation over single runs of GA5–GA12 on Problem 1, are provided in Figures F.5–F.8.

A summary of the results of applying the GAs is recorded in Tables 5.5 and 5.6, which list the best and median network costs, respectively, for each group of ten runs. In each table, either the best overall result or the best median for each problem are displayed in bold. Moreover, the full set of results from applying the GAs to all five of the network design problems is recorded in Tables F.23–F.31.

Of the best overall results, two were obtained by GA10 (random change link; trial-based), two by GA9 (random change link; child-based) and one by GA8 (random remove link; trial-based). However, four of the five best medians are due to GA4 (the straightforward GA), with only one

Problem	Significance level (%)							
	$s_{5,4}$	$s_{6,4}$	$s_{7,4}$	$s_{8,4}$	$s_{9,4}$	$s_{10,4}$	$s_{11,4}$	$s_{12,4}$
1	0.1	0.1	1.1	5.5	—	—	5.5	5.5
2	0.1	0.1	—	—	—	—	0.1	5.5
3	0.1	0.1	—	—	5.5	—	0.1	5.5
4	0.1	0.1	5.5	5.5	5.5	—	0.1	5.5
5	0.1	0.1	5.5	1.1	—	—	0.1	5.5

Table 5.7: Comparing GA4 with GAs using local search operators

best median obtained by GA10 (random change link; trial-based). Clearly, while some of the best results were obtained by those algorithms enhanced by local-search operators, the straightforward algorithm, with only single-link mutation and problem-specific crossover, performed well especially in median terms.

To assess the statistical significance of these differences, a non-parametric sign test was again applied to matched pairs from different GAs [307]. First, Table 5.7 shows comparisons between all those GAs using local search operators and GA4. Clearly, this more straightforward algorithm is significantly better for many of the problems, in terms of median difference, than all of the other GAs, with the single exception of GA10 (random change link; child-based), where the difference is not significant. These median differences are particularly marked for the two algorithms including the random add link operator, GA5 and GA6, all being very highly significant. Given that the best initial random networks are often more highly connected than those eventually evolved (*cf.* Figures 5.2(a) and (f) on p. 96), it is likely that using this local search operator, which can only *add* links, is counter-productive, serving only to limit the effectiveness of link-group crossover. The next two GAs, GA7 and GA8, use the random remove link operator, which by *removing* unwanted links has a greater chance of improving the initially over-connected networks. This is reflected in a less conclusive victory by GA4, with no significant median difference for two problems, and indeed a best overall result in one case for GA8 (trial-based). However, it would appear that the greater freedom from *both adding and removing* links, resulting from the use of the random change link operator in GA9 and GA10, leads to almost no significant median differences with GA4, and indeed the best overall results in four out of five problems. Interestingly, providing two separate operators to both add and remove links, as is done in GA11 and GA12, does not seem as successful as the use of the combined random change link operator, with significantly better medians obtained by GA4 over both GA11 and GA12 in all cases.

Next, Table 5.8 compares the otherwise identical child and trial-based credit division GAs directly. While for both GA6 over GA5 (random add link), and GA12 over GA11 (both random add

Problem	Significance level (%)			
	$s_{5,6}$	$s_{7,8}$	$s_{9,10}$	$s_{11,12}$
1	0.1	—	—	5.5
2	5.5	—	—	1.1
3	1.1	—	1.1	1.1
4	0.1	—	—	5.5
5	1.1	—	—	1.1

Table 5.8: Comparing child and trial-based credit division for GA5–GA12

Problem	Significance level (%)		
	$s_{6,10}$	$s_{8,10}$	$s_{12,10}$
1	0.1	—	5.5
2	0.1	5.5	5.5
3	0.1	—	1.1
4	0.1	5.5	1.1
5	0.1	—	5.5

Table 5.9: Comparing GA10 with other trial-based GAs using local search operators

and remove link), the trial-based algorithms are better than the child-based, only one significant median difference was found between the other two pairs of GAs. Nevertheless, given earlier reasoning, those median differences found, and that the best results were discovered by trial-based GAs for three out of the five problems, it seems reasonable to adopt trial-based probability reassignment for GAs incorporating such multi-trial operators.

Finally, Table 5.9 compares the other three trial-based credit division GAs with GA10 (random change link), which appears the strongest of those GAs incorporating local search operators. In all three cases, significant median differences were found, confirming this view, although this was perhaps less marked in the case of GA8 (random remove link), whose potential has already been noted above.

Overall, it would seem that GA4 (for the problem sizes and types examined), with just single-link mutation and link-group crossover, is a consistent algorithm producing good results. Only when given sufficient runs of adequate length might a better overall result be achieved with a GA incorporating a flexible local search operator, random-change link, as well as applying trial-based probability reassignment. This then confirms the strength of the basic GA4 algorithm, and in particular the usefulness of link-group crossover.

5.1.4 Iterative Heuristic Operators

The final set of topology experiments explores the use of the three iterative heuristic operators for mesh network topology design. These are random add links (RALS), random remove links (RRLS) and random change links (RCLS; all three described in §4.2.2 on p. 82). All of them are extensions

GA	ML	LGCO	RALS	RRLS	RCLS
4	0.35	0.45			
13	0.20	0.50	0.10		
14	0.20	0.50		0.10	
15	0.20	0.50			0.10

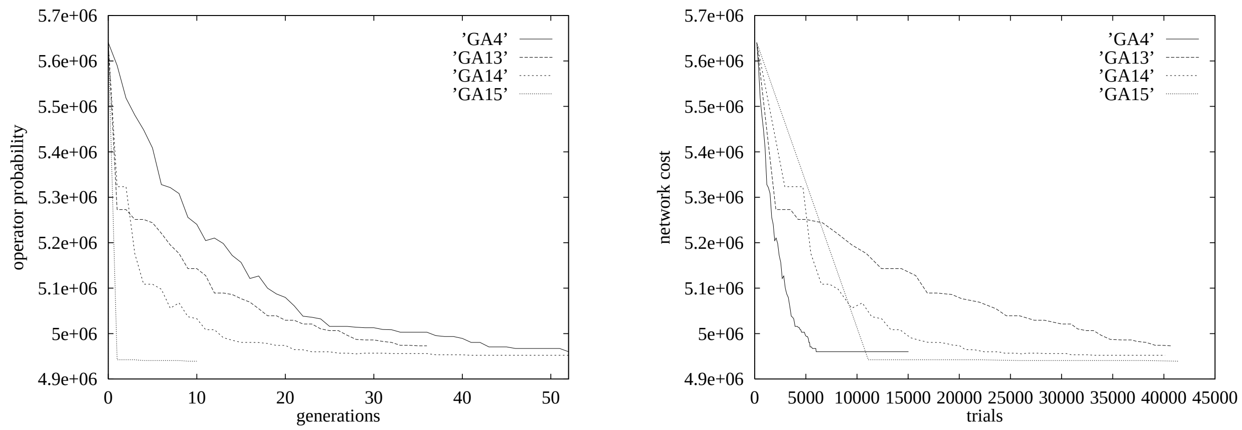
Table 5.10: Initial operator probabilities – topology using iterative heuristic operators

of the corresponding local search operators used in the previous section. However, rather than stopping after the first improvement is found, they each continue until a local optimum is detected. Thus each one is a complete iterative stochastic hill climber [29, 30]. Once again they will be compared against a more straightforward algorithm employing only single-link mutation (ML) and link-group crossover (LGCO).

The three GAs incorporating iterative heuristic operators (GA13, GA14 and GA15) each had a population size of 200, used tournament selection (of size 4), trial-based credit division, and were limited to a maximum of 40,000 trials. The further increase in the number of trials (*cf.* GA4–GA12) was again discovered to be necessary (using a few test runs) to allow the GAs, particularly GA15 (random change links), to complete a reasonable number of generations[6]. The operators used and their initial probabilities are given in Table 5.10. Other non-default parameter values common to the three GAs are listed in Table F.32 in Appendix F. Essentially, these GAs differ from those used in the local search experiments only in respect to: choice of operators and initial probabilities, increased run length, and that all three use per-trial operator-probability reassignment. However, GA4 is here identical to that used in the last section. As the algorithm makes no further progress after about 6,000 trials, and was in any case being run for 15,000, it seemed unnecessary to increase this to 40,000 trials and repeat the earlier runs.

The three GAs using iterative heuristic operators were each applied ten times with different random seeds to each of the five network design problems listed in §F.2. The lowest cost curves for these three GAs, as well as GA4, all using the same random seed and thus starting with the same initial population, are given in Figures 5.6(a) and (b); the former generation by generation, the latter *vs.* trials. Overall, the best result for these runs is obtained by GA15 (random change links), followed by GA14 (random remove links), however GA4 once again showed the most rapid progress, producing the best overall result up to 10,000 trials. Only beyond this point were two of the GAs incorporating iterative heuristic operators eventually able to overtake it. This again demonstrates the utility of the basic GA4 algorithm, incorporating just single-link mutation and

[6]Once again, this arguably gives an unfair advantage to those GAs incorporating iterative heuristic operators.



(a) Comparing lowest costs (*vs.* generations)

(b) Comparing lowest costs (*vs.* trials)

Figure 5.6: Typical lowest costs using iterative heuristic operators (Problem 1, Run 1)

Problem	GA4 Best ($\times 10^6$)	GA4 Median ($\times 10^6$)	GA13 Best ($\times 10^6$)	GA13 Median ($\times 10^6$)	GA14 Best ($\times 10^6$)	GA14 Median ($\times 10^6$)	GA15 Best ($\times 10^6$)	GA15 Median ($\times 10^6$)
1	4.947	4.958	4.936	4.951	4.936	4.948	**4.935**	**4.937**
2	4.746	4.753	4.747	4.755	4.739	4.756	**4.736**	**4.742**
3	4.409	4.412	4.408	4.426	**4.400**	4.411	**4.400**	**4.402**
4	4.591	4.601	4.591	4.603	4.590	4.594	**4.585**	**4.587**
5	4.424	4.430	4.419	4.441	4.419	4.430	**4.416**	**4.418**

Table 5.11: Cost results for topology with iterative heuristic operators

link-group crossover.

Expanding on the results provided in Fig. 5.6, extra topology results, showing typical progress and operator-probability adaptation over single runs of GA13–GA15 on Problem 1, are given in Fig. F.9.

A summary of the results of applying the three GAs, as well as GA4, is recorded in Table 5.11, which lists the best and median network costs for each group of ten runs. As before, the best overall result and the best median for each problem are displayed in bold. In addition, the full set of results from applying the three GAs to all five of the network design problems is recorded in Tables F.33–F.35; the earlier GA4 results are listed in Table F.23.

Clearly, GA15 (random change links) is the best of the four algorithms, having obtained the best overall result and best medians for all five problems, although admittedly GA14 (random remove links) tied for best overall in one case. To assess the statistical significance of the median differences, a non-parametric sign test [307] was applied to compare matched pairs from the other algorithms with GA15. The results are listed in Table 5.12, which records that while there are almost no significant median differences between either GA13 or GA14 and GA4, GA15 is very

Problem	Significance level (%)		
	$s_{4,13}$	$s_{4,14}$	$s_{4,15}$
1	—	—	0.1
2	—	—	0.1
3	—	—	0.1
4	—	5.5	0.1
5	—	—	0.1

Table 5.12: Comparing GA4 with GAs using iterative heuristic operators

highly significantly better than GA4.

Overall, it would seem that for mesh network topology design (and the problem sizes and types examined) combining an iterative heuristic operator with the solid GA framework provided by GA4, including its use of single-link mutation and link-group crossover, consistently produces better results, although at the cost of extended run lengths (in terms of trials, rather than generations).

5.2 Routing, Fibre & Wavelength Allocation

Once a topology has been determined, the next problem is the network routing, fibre choice and wavelength allocation. The networks considered in the thesis are flat, all-optical, with arbitrary-mesh multiple-fibre-link topologies, and employ WP routing. Consequently, as was mentioned in §3.1, first the static traffic requirement between each node pair in the network has to be partitioned into an appropriate number of separate wavelength channels. Then, for each of these, a route through the network must be found, including which fibre and wavelength will be used on each link (cable) along the path. As WP routing is employed, each channel must use the same wavelength end-to-end, since no wavelength conversion is available, as well as obeying the usual constraint that on any one fibre, no wavelength is used by more than one channel. However, although restoration paths are supported during network topology design, NOMaD's network model provides no direct support for the fibre and wavelength allocation of restoration paths, and so these are not included in the allocation.

For the experiments in this section, the test networks described in §5.1.2 on p. 98 were used. Each link was taken to be a cable of two fibres, and the capacity of the individual wavelength channels set to 10 Gbit/s. The total wavelength channel requirements for the networks[7], calculated from the traffic matrices in §F.2, are given in Table 5.13.

[7]Recall that Networks 1, 6 and 7 were all designed for Problem 1, and so have identical traffic requirements.

Network	1/6/7	2	3	4	5
Channel requirement	268	248	258	262	268

Table 5.13: Total wavelength channel requirements of the test networks

GA	WKM	WKCO	WWRR	WWSO
16	0.25	0.50		
17	0.05	0.15	0.30	0.35

Table 5.14: Initial operator probabilities – minimum NWR

5.2.1 Minimum Network Wavelength Requirement

The first set of experiments employ the network wavelength requirement (NWR) metric outlined in §4.1.2 on p. 71. This modifies the actual NWR by adding a penalty for each unsatisfied traffic requirement (*i.e.* wavelength channel that remains unrouted) to avoid the false minimum of carrying no traffic at all, which would otherwise have an NWR of zero[8].

The four operators developed by the author for routing, fibre and wavelength allocation within NOMaD were described in §4.2.3 on p. 82. In devising experiments based on these, it was decided to compare, for two different GAs, the best of five runs with results obtained from three recent wavelength-allocation heuristics published in the literature[9]. The first GA is a more straightforward algorithm[10], which uses just wavelength-allocation k-shortest path mutation (WKM) and wavelength-allocation k-shortest path single-point crossover (WKCO), while the second GA also includes the two problem-specific heuristics, wavelength-allocation path reroute (WWRR) and wavelength-allocation path shift-out (WWSO). These experiments were first published by the author in *Electronic Letters* [35] (reprinted in Appendix A).

Both GAs had a population size of 500 and used tournament selection (of size 4); their initial operator probabilities are given in Table 5.14. Other non-default parameter values for GA16 and GA17 are listed in Tables F.36 and F.37 in Appendix F. It should be emphasised that they both used the operator-probability adaptation mechanism with default parameters throughout.

The two GAs were both applied five times with different random seeds to Networks 1–5. The progress of single typical runs of GA16 and GA17 on Network 1 are given in Figures 5.7(a) and (c), showing both the average NWR of the population and the lowest (*i.e.* best in population) NWR,

[8]The additional per-fibre penalty, described in §4.1.2, was not used in these experiments

[9]There are relatively few heuristics that tackle the specific allocation problem addressed here, and so those selected were simply those that the author was able to locate at the time. Additional heuristics discovered since will be included in the suggestions for further work (Ch. 6)

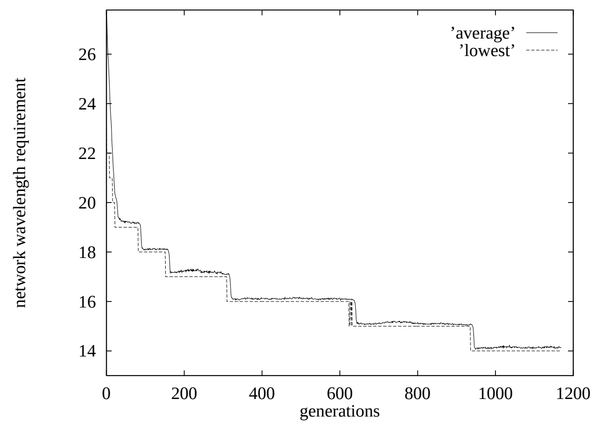
[10]As with GA1 for mesh network topology design, this is not a traditional bit-string GA, as both an object-oriented representation and operator-probability adaptation are being employed.

generation by generation. The operator-probability adaptation curves for the same runs are shown in Figures 5.7(b) and (d).

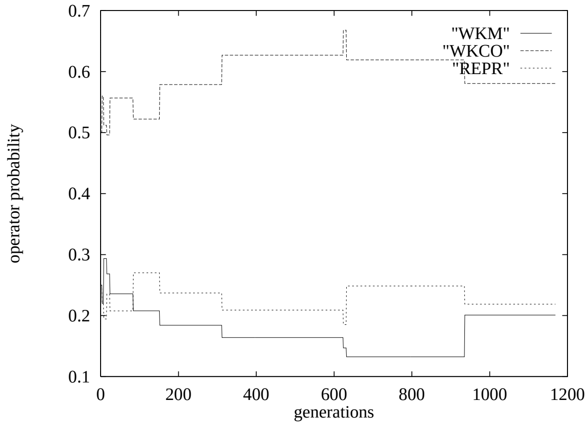
In contrast to most of the topology curves, all four figures show abrupt changes in value. This results from the integer-valued nature of NWR: whenever there is a new best individual in the population, its NWR is always at least a full unit lower than the previous best. In both Figures 5.7(a) and (c), this improvement appears to rapidly propagate through the population, as the average NWR swiftly drops to close to the best. In addition, as the operator-probability adaptation mechanism is driven by credit assigned when a new best individual is found, the integer-valued nature of NWR also results in fairly abrupt changes in operator probabilities, as can be observed in both Figures 5.7(b) and (d).

Turning to the NWR results, there is clearly a considerable difference in the progress of the two GAs. Whilst GA17 takes about 80 generations to obtain an NWR of 9, GA16 is only able to achieve an NWR of 14 using nearly 1200 generations (500,000 trials). This difference is even more apparent in Fig. 5.7(e) which directly compares the lowest NWR curves for the two GAs. Both these runs employed the same random seed, and thus started from the same initial population. From the operator-probability adaptation curves it appears that, at least for these runs, GA16 chiefly depends on the crossover operator (WKCO), with a modest contribution from mutation (WKM), whereas GA17 makes full use of both its problem-specific heuristic operators, shift-out (WWSO) and reroute (WWRR), with only minimal contributions from crossover and mutation.

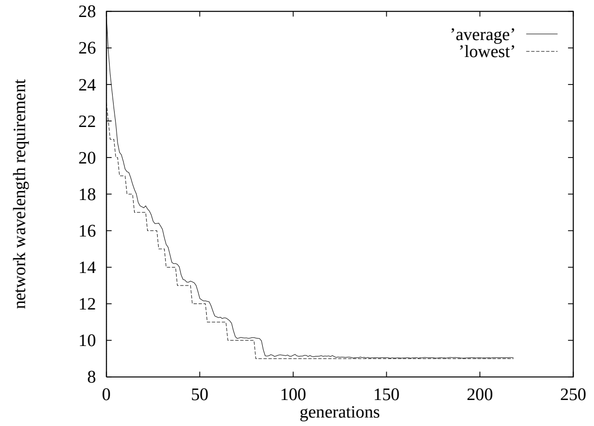
In addition to the two GAs, a wavelength-allocation heuristic by Wuttisittikulkij & O'Mahony and two by Wauters & Demeester were also applied to Networks 1–5. Wuttisittikulkij & O'Mahony's heuristic [298] (W&O) is simple, fast and non-iterative, working with k-lowest hop count paths (and multiple fibres) to obtain a low NWR. The first heuristic by Wauters & Demeester [306] (W&D1) is based on a version of Dijkstra's algorithm modified such that in "extend[ing] an already found part of a route, only those [cables] are considered which have a wavelength channel free which is available on all [cables] of the already found part of the route"; this algorithm produces a low-cost allocation, whilst also keeping NWR low. Their second algorithm (W&D2), is based on iterative application of a modified version of Yen's k-shortest path algorithm, and aims to produce the lowest cost allocation that still achieves minimum NWR. Description of all three heuristics is given in Appendix C, and their non-default parameters listed in Table F.40. Of the three algorithms, Wuttisittikulkij & O'Mahony's is non-deterministic and so, like the GAs, it was allowed the best of



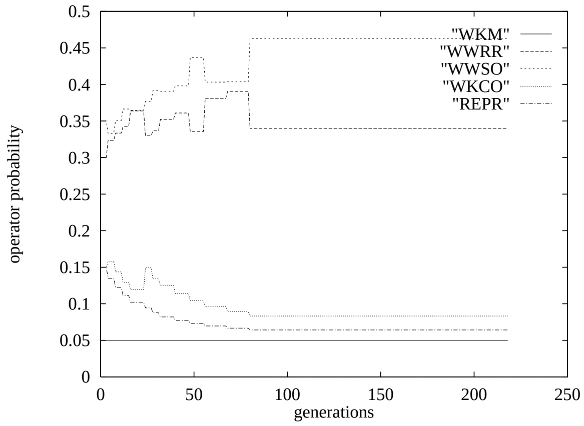
(a) Progress over a single GA16 run



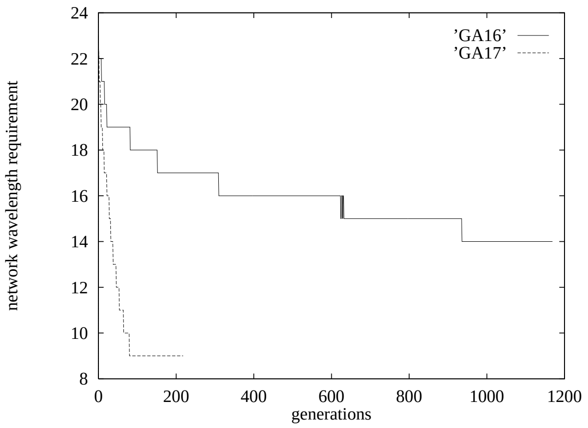
(b) Operator-probability adaptation for GA16



(c) Progress over a single GA17 run



(d) Operator-probability adaptation for GA17



(e) Comparing lowest NWR

Figure 5.7: Typical minimum NWR results (Network 1, Run 1)

Network	W&O	W&D1	W&D2	GA16		GA17	
	NWR	NWR	NWR	NWR	Trials	NWR	Trials
1	**8**	15	10	13	351,264	9	31,874
2	**7**	11	**7**	10	401,919	**7**	38,291
3	8	19	9	10	448,235	**7**	44,467
4	**8**	15	9	11	429,485	**8**	37,997
5	9	18	9	12	428,962	**8**	48,448

Table 5.15: Results for minimum NWR

five runs[11].

A summary of the results of applying the two GAs and three heuristics is recorded in Table 5.15, which lists the NWR, or the best NWR for five runs as appropriate, as well as the number of trials used to find the recorded NWR in the best GA run in each case[12]. As before, the best overall result for each network is displayed in bold. In addition, the full set of results from applying the two GAs and W&0 to all five of the networks is recorded in Tables F.38, F.39 and F.41, respectively.

Of the three heuristic algorithms, the poorest in this context is W&D1, which in aiming for low network cost, has obtained relatively high NWRs. Although the more recent W&D2 has clearly improved on this in NWR terms, the best heuristic overall is W&O, which despite its simplicity, achieves good NWR results. Although over the five test networks, GA16 consistently obtained better NWRs than W&D1, it was no match for the other two heuristics. However, GA17 equalled or exceeded the results of all three heuristics, with one exception, and using considerably fewer trials than GA16. Clearly, therefore, the addition of the two heuristic operators (reroute and shift-out), has led to a marked improvement of GA17 over GA16. In addition, despite these two additional operators being based on steps of the W&D2 heuristic, GA17 has in many cases exceeded the results of W&D2, managing to match the best heuristic, W&O.

Overall, these experimental results show that, at least for the five test networks, the GA/heuristic hybrid approach is as promising as has been hoped, as the blend of a GA with steps from Wauters & Demeester's HRWA heuristic [306] resulted in both improved performance (over the GA alone) and improved results (over the heuristic alone). In addition, the GA/heuristic hybrid results matched those of the best wavelength-allocation heuristic examined.

[11]The option to allocate the paths in order of hop count, largest first, was exercised, as this was both recommended by the heuristic's authors and found to obtain lower NWR results in trial runs.

[12]Note that as none of the heuristics make use of trials, as such, their comparative effort is zero!

5.2.2 Minimum Cost Allocation

The second set of experiments on routing, fibre choice and wavelength allocation used Network Cost Model B, outlined in §4.1.3 on p. 72. This changes the assessment from just looking at the NWR to an objective that consists of a simplified assessment of the cost of the whole network including, at an appropriate level, the cost of exceeding the minimum NWR. The cost model is parameterised, amongst others, by three constants, (α, β, γ). The first two, α and β, adjust the model of link cost in the network to allow for different dependencies on capacity. With $\alpha = \beta = 1$, the dependency is linear—doubling the capacity doubles the cost—whereas with $\alpha = \beta < 1$, the cost increases more slowly with capacity[13]. The remaining parameter, γ, allows the cost model to reflect the relative contributions to link and node costs of the capacity and link wavelength requirements. With $\gamma = 1$, the cost model only takes account of the actual capacity used; whereas, with $\gamma = 0$, the capacity is ignored, and the focus is entirely on the costs resulting from wavelength requirements. An intermediate value of γ, such as 0.5, is regarded as being more representative of the actual costs involved.

For these experiments, the the same four routing, fibre and wavelength allocation operators used in GA17, and described in §4.2.3 on p. 82, were used. Given the outcome of the experiments on minimum NWR design, it was decided to immediately employ all four operators within a GA, rather than further explore ineffective algorithms that did not include the problem-specific heuristic operators. Instead, the emphasis was on competing, on a network cost basis, with the same three overall wavelength-allocation heuristics used for the minimum NWR study. These experiments were first published at the 6th IEE Conf. on Telecommunications [36] and subsequently in *IEE Proc. Communications* [37] (both reprinted in Appendix A).

The three GAs used were identical except that they had different initial operator probabilities (given in Table 5.16). These were selected for problems with differing values of γ by using the usual approach of a few test runs, aiming for settings that would remain approximately constant for the first few generations across all the networks examined. All three GAs had a population size of 500, a maximum of 100,000 trials, and used tournament selection (of size 4). Other non-default parameter values for GA18–20 are listed in Table F.42 in Appendix F. It should be emphasised that they all used the operator-probability adaptation mechanism with default parameters throughout.

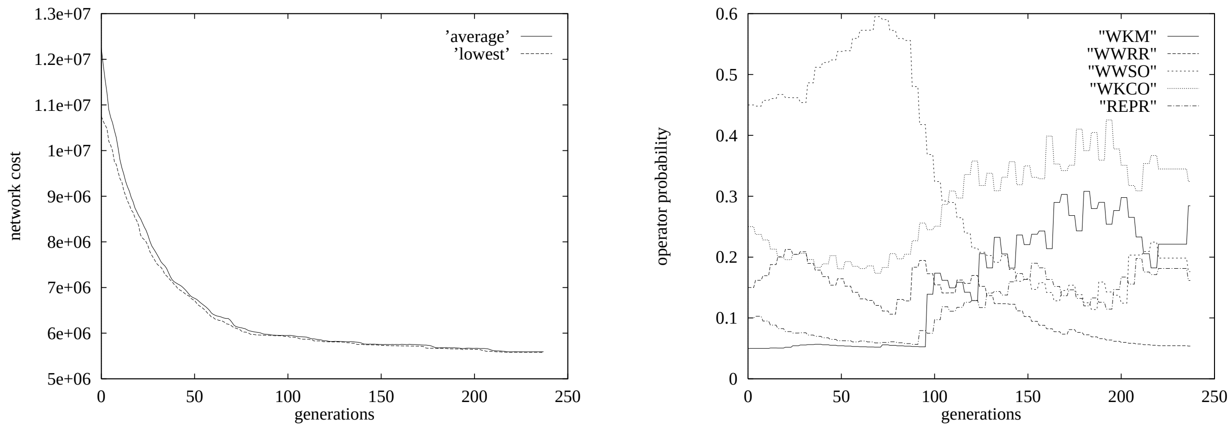
All three GAs were applied five times with different random seeds (*i.e.* best of five runs) to

[13] In these experiments, α and β were always given the same value, however, β is included in the model to fine-tune the dependency of cost on link wavelength requirement rather than just capacity alone

GA	WKM	WKCO	WWRR	WWSO
18	0.05	0.25	0.15	0.45
19	0.05	0.30	0.10	0.40
20	0.05	0.20	0.25	0.40

Table 5.16: Initial operator probabilities – minimum cost allocation

Networks 1–7, with γ set to 0.5, 1 and 0 for GA18, GA19 and GA20, respectively; first for $\alpha = \beta = 1$, then for $\alpha = \beta = 0.8$. For all three GAs (and the heuristics–see below), k was set to 8 for the Networks 1-5, and 4 for the remaining two, reflecting their lower average node degrees. The progress of single typical runs of GA18 on Network 1 with $(\alpha, \beta, \gamma) = (1, 1, 0.5)$ and $(0.8, 0.8, 0.5)$ are given in Figures 5.8(a) and 5.9(a), respectively, showing both the average network cost of the population and the lowest (*i.e.* best in population) cost, generation by generation. The operator-probability adaptation curves for the same runs are shown in Figures 5.8(b) and 5.9(b).

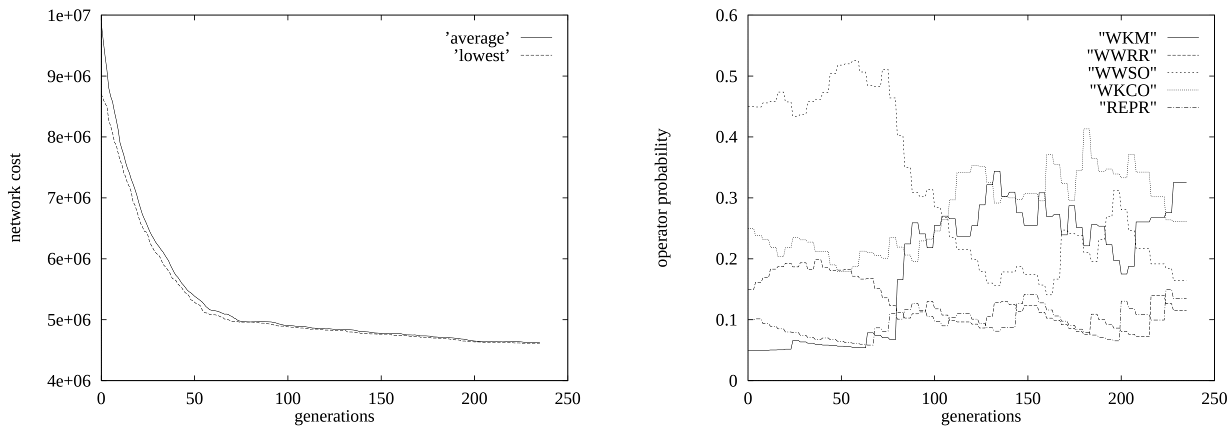


(a) Progress over a single run

(b) Operator-probability adaptation

Figure 5.8: Typical results for GA18, $\alpha = \beta = 1$ and $\gamma = 0.5$ (Network 1, Run 1)

Both progress curves show a similar pattern of rapid initial reduction in network cost up to about generation 100, followed by more gradual improvement for the balance of the run. Examination of the operator probability adaptation curves indicates that, during this first phase of the run, the dominant operator is shift-out (WWSO), although this proves much less useful towards the end of the run. In contrast, however, after about generation 100, mutation (WKM), which started with a very low initial probability, becomes much more productive, as does crossover (WKCO). These changes probably reflect the far greater opportunity for local improvements by shift-out (and to a lesser extent, reroute) in the initial random individuals, whereas later in the run, crossover and mutation are needed to introduce diversity into the highly-evolved population, allowing further cost



(a) Progress over a single run

(b) Operator-probability adaptation

Figure 5.9: Typical results for GA18, $\alpha = \beta = 0.8$ and $\gamma = 0.5$ (Network 1, Run 1)

Network	W&O ($\times 10^6$)	W&D1 ($\times 10^6$)	W&D2 ($\times 10^6$)	GA18 ($\times 10^6$)
1	5.909	6.007	6.358	**5.579**
2	5.151	5.241	5.278	**4.886**
3	5.153	5.707	5.637	**4.976**
4	5.413	5.528	5.765	**5.117**
5	5.424	5.773	5.658	**5.108**
6	5.662	6.260	5.908	**5.344**
7	6.077	6.545	5.962	**5.651**

Table 5.17: Minimum cost allocation results for $\alpha = \beta = 1$ and $\gamma = 0.5$

reductions.

Supplementing the results provided in Figures 5.8 and 5.9, further minimum cost allocation results, showing typical progress and operator-probability adaptation over single runs of both GA19 and GA20 on Network 1 for both $(\alpha, \beta, \gamma) = (1, 1, 0.5)$ and $(0.8, 0.8, 0.5)$, are illustrated in Figures F.10–F.13.

In addition to the GAs, the three wavelength allocation heuristics by Wuttisittikulkij & O'Mahony (W&O) and Wauters & Demeester (W&D1 and W&D2) were also applied to Networks 1–7 for the same six combinations of (α, β, γ). As before, Wuttisittikulkij & O'Mahony's non-deterministic algorithm was allowed the best of five runs.

A summary of the results of applying all six GAs and heuristics is recorded in Tables 5.17–5.22, which lists the network cost, or the best cost for five runs, as appropriate. As before, the best overall result for each network, for each combination of (α, β, γ), is displayed in bold. In addition, the full set of results from applying the three GAs and W&0 to all seven of the networks is recorded in Tables F.43–F.48 and F.50–F.55.

Network	W&O ($\times 10^6$)	W&D1 ($\times 10^6$)	W&D2 ($\times 10^6$)	GA19 ($\times 10^6$)
1	5.704	5.172	5.896	**4.994**
2	5.081	4.589	5.046	**4.436**
3	5.051	4.625	5.446	**4.394**
4	5.313	4.808	5.388	**4.645**
5	5.302	4.806	5.326	**4.640**
6	5.516	5.010	5.282	**4.753**
7	5.907	5.380	5.606	**5.228**

Table 5.18: Minimum cost allocation results for $\alpha = \beta = 1$ and $\gamma = 1$

Network	W&O ($\times 10^6$)	W&D1 ($\times 10^6$)	W&D2 ($\times 10^6$)	GA20 ($\times 10^6$)
1	**6.036**	6.843	6.820	6.116
2	**5.213**	5.893	5.510	5.228
3	**5.229**	6.789	5.828	5.362
4	5.503	6.247	6.142	**5.499**
5	5.544	6.740	5.991	**5.523**
6	5.790	7.510	6.535	**5.781**
7	6.220	7.709	6.319	**6.001**

Table 5.19: Minimum cost allocation results for $\alpha = \beta = 1$ and $\gamma = 0$

Network	W&O ($\times 10^6$)	W&D1 ($\times 10^6$)	W&D2 ($\times 10^6$)	GA18 ($\times 10^6$)
1	4.653	4.890	5.103	**4.450**
2	4.089	4.238	4.212	**3.909**
3	4.138	4.656	4.559	**3.866**
4	4.312	4.474	4.608	**4.062**
5	4.313	4.684	4.518	**4.040**
6	4.246	4.842	4.482	**4.065**
7	4.203	4.674	4.178	**3.965**

Table 5.20: Minimum cost allocation results for $\alpha = \beta = 0.8$ and $\gamma = 0.5$

Network	W&O ($\times 10^6$)	W&D1 ($\times 10^6$)	W&D2 ($\times 10^6$)	GA19 ($\times 10^6$)
1	4.489	4.233	4.757	**4.014**
2	4.036	3.731	4.039	**3.531**
3	4.056	3.802	4.414	**3.541**
4	4.230	3.910	4.320	**3.716**
5	4.216	3.913	4.267	**3.716**
6	4.123	3.921	4.038	**3.633**
7	4.082	3.883	3.952	**3.686**

Table 5.21: Minimum cost allocation results for $\alpha = \beta = 0.8$ and $\gamma = 1$

Network	W&O ($\times 10^6$)	W&D1 ($\times 10^6$)	W&D2 ($\times 10^6$)	GA20 ($\times 10^6$)
1	**4.755**	5.546	5.448	4.895
2	**4.128**	4.745	4.384	4.249
3	**4.200**	5.510	4.704	4.271
4	**4.382**	5.039	4.896	4.444
5	4.409	5.455	4.769	**4.363**
6	**4.343**	5.762	4.926	4.344
7	4.306	5.464	4.404	**4.228**

Table 5.22: Minimum cost allocation results for $\alpha = \beta = 0.8$ and $\gamma = 0$

For $\gamma = 0.5$, a value regarded as being reasonably representative of actual costs, the best heuristic for both values of α,β was found to be W&O (except for Network 7), which is thus providing a good trade-off between capacity cost and link wavelength requirements. Nevertheless, for all seven networks the lowest network costs were actually obtained by the hybrid algorithm, GA18 (Tables 5.17 and 5.20).

With $\gamma = 1$, focusing the network cost on capacity rather than wavelength requirement, the best heuristic for both values of α,β was W&D1, and by a wide margin; this is perhaps to be expected, as W&D1 uses shortest paths (rather than k-lowest hop count or k-shortest paths, as do W&O and W&D2), thereby placing the emphasis on cost instead of wavelength requirement. However, once again, the hybrid algorithm, GA19, provided the best results for all seven networks (Tables 5.18 and 5.21).

Finally, setting $\gamma = 0$, so that the network cost depends entirely on wavelength requirement rather than capacity, the best heuristic for both values of α,β was W&O; this is no doubt due to the superiority of W&O, with respect to NWR, compared to the other two heuristics (see §5.2.1). Although for minimum NWR, GA17 matched the results of W&O, here, disappointingly, the GA20 hybrid algorithm was only able to obtain the best overall cost for four networks with $\alpha = \beta = 1$ and two networks with $\alpha = \beta = 0.8$, although its results were close to those of W&O on the other eight (Tables 5.19 and 5.22).

Overall, the experimental results show that, at least for the seven test networks, the GA/ heuristic hybrid approach has again shown itself to be as promising as has been hoped, as in competition with three recent heuristics, it was able to obtain the lowest network costs for γ set to both 0.5 and 1; and in the case of $\gamma = 0$, was able to approach the results of the best of the allocation heuristics.

5.3 Operator-probability Adaptation

Complementing the earlier experimental work applying NOMaD's GA/heuristic hybrid approach to both mesh network topology design and routing, fibre & wavelength allocation, this section presents an investigation into NOMaD's operator-probability adaptation mechanism. Both the basic adaptation algorithm and its variants were described in §4.2.4, and some insight into the basic algorithm has already been provided from examining the operator-probability adaptation curves for the earlier experimental work.

In this study, it was decided to focus on the most difficult of the design problems examined so far, minimum cost allocation, which was nevertheless one in which the GA/heuristic hybrid approach performed well. To limit the amount of computation time required, only three of the test networks from §5.1.2 on p. 98 were used: Networks 1, 3 & 5. As before, each link was taken to be a cable of two fibres, and the capacity of the individual wavelength channels set to 10 Gbit/s.

For these experiments, all thirteen GAs employed the same four routing, fibre and wavelength allocation operators used in GA17–20, and described in §4.2.3 on p. 82. The metric for network assessment was Network Cost Model B, outlined in §4.1.3 on p. 72, with the default parameter values, including $(\alpha, \beta, \gamma) = (1, 1, 0.5)$. All the GAs were identical except for different settings of the adaptation algorithm parameters (see below). They had a population size of 500, a maximum of 100,000 trials, and used tournament selection (of size 4). Other non-default parameter values for GA21–25 and GA26–33 are listed in Tables F.56 and F.62 in Appendix F.

In applying NOMaD to specific design problems, initial operator probabilities have so far been selected by a few trial runs, aiming for settings that would remain approximately constant for the first few generations across all the networks examined. However, to avoid the potential bias of pre-tuned values, here the initial probabilities of all four operators plus the reproduction (*i.e.* null) operator, REPR, have been set to 0.2 in all the GAs.

5.3.1 Basic Adaptation Algorithm

The first set of experiments uses the basic adaptation algorithm, where operators are rewarded as a result of improvement over the fitness of the best individual found up to the end of the previous generation. These experimental results were first published by the author at the IEEE Intl. Conf. on Evolutionary Computation [33] (reprinted in Appendix A).

To explore the influence of the adaptation proportion, ψ, three values were selected, 0.15, 0.00 and 0.30, and combined with a credit constant, $\kappa = 0.5$, these are denoted GA21, GA22 and GA23 in Table 5.23. The levels in GA21 are those that have been used on the specific design problems; in GA22, the adaptation algorithm is completely deactivated, as with $\psi = 0.00$, the initial operator probabilities will remain unchanged for the entire run; GA23 represents a higher than usual rate of adaptation.

In examining the influence of the credit constant, the adaptation proportion was fixed to its usual value ($\psi = 0.15$), and κ set to 0.5, 0.0 and 1.0 in GA21, GA24 and GA25 respectively (Table 5.23). While the usual value of $\kappa = 0.5$ allows reduced credit to parent (0.5) and grandparent (0.25)

GA	ψ	κ
21	0.15	0.5
22	0.00	0.5
23	0.30	0.5
24	0.15	0.0
25	0.15	1.0

Table 5.23: Adaptation parameters for GAs using the basic adaptation algorithm

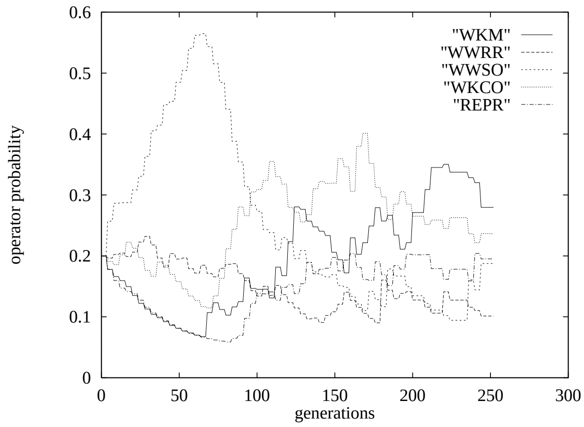
operators; in GA24, credit is only awarded to the actual operator responsible for an improvement; whereas in GA25, the operator, parent and grandparent operators are all rewarded equally.

All five GAs were then applied ten times with different random seeds to Networks 1, 3 & 5. The operator-probability adaptation curves for single typical runs of GA21, GA23, GA24 and GA25 are given in Fig. 5.10. As before, the graphs are for Network 1 with the same random seed, and hence have identical initial populations. However, the equivalent figure for GA22 is not shown, as all five curves remain unchanged at 0.2 throughout the run, since the adaptation mechanism is deactivated with $\psi = 0.00$. Corresponding graphs illustrating the progress of the runs for GA21–25, showing both the average network cost of the population and the lowest (*i.e.* best in population) cost, generation by generation, are given in Fig. F.14.

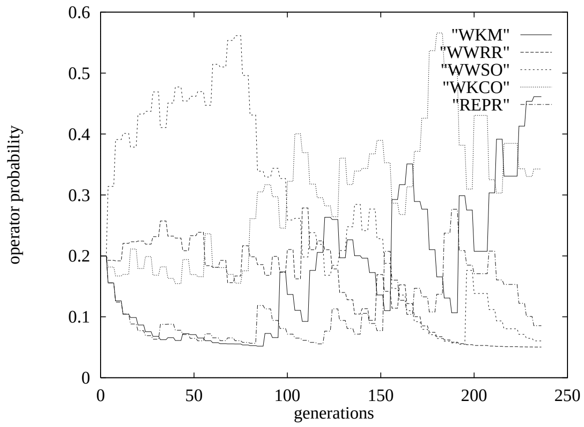
Inspecting GA21 in Fig. 5.10(a), the effect of the adaptation algorithm can be clearly seen. For example, the early productivity of the heuristic path shift-out operator, WWSO, results in its probability rising from 0.2 to nearly 0.6 in the first 50 generations, although it subsequently reduces to below 0.2. In contrast, the far simpler path mutation, WKM, falls from 0.2 to below 0.1 before generation 50, only then to rise to over 0.3 due to higher productivity in the latter part of the run.

Comparing GA21 and GA23, the impact of the higher adaptation proportion (ψ) in the latter is obvious in the greater variability of the probability curves (Fig. 5.10(b)). A similar effect is also apparent in Fig. 5.10(c), where setting the credit constant (κ) to 0.0 in GA24 resulted in wide fluctuations of operator probability, due to the hybrid's exclusive focus on operators producing direct improvements in fitness. In contrast, for GA25 (Fig. 5.10(d)), although there is some initial variation of probabilities, particularly for the productive path shift-out operator (WWSO), after 120 generations all the operators settle to an approximately equal share of probability. This would seem to indicate that with grandparent and parent operators receiving equal credit ($\kappa = 1.0$), the basic algorithm is unable to respond to all but the widest differences in relative productivity.

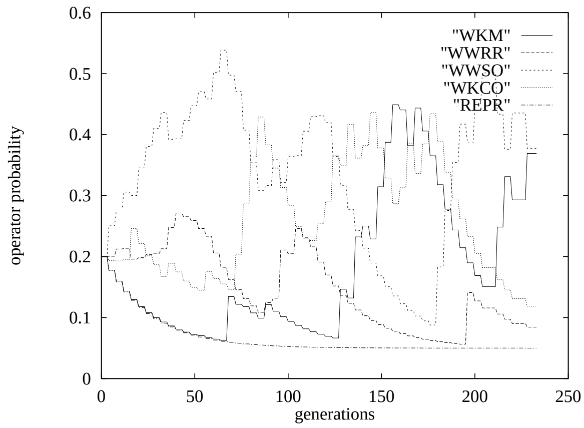
A summary of the results from applying the five GAs is given in Tables 5.24 and 5.25 (with GA21 included in both tables for ease of reference). These record the best and median results at both the end (100,000 trials) and the early part (25,000 trials) of ten runs on all three networks. In



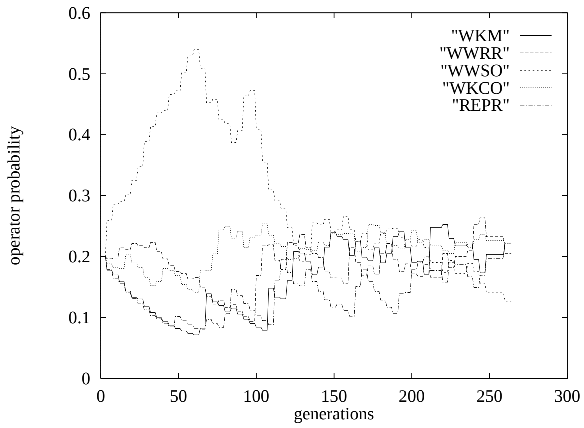
(a) GA21, $(\psi, \kappa) = (0.15, 0.5)$



(b) GA23, $(\psi, \kappa) = (0.30, 0.5)$



(c) GA24, $(\psi, \kappa) = (0.15, 0.0)$



(d) GA25, $(\psi, \kappa) = (0.15, 1.0)$

Figure 5.10: Typical operator-probability adaptation, using the basic adaptation algorithm, for GA21, GA23–GA25 (Network 1, Run 1)

Network	GA22 $\psi = 0.00$ Best ($\times 10^6$)	GA22 Median ($\times 10^6$)	GA21 $\psi = 0.15$ Best ($\times 10^6$)	GA21 Median ($\times 10^6$)	GA23 $\psi = 0.30$ Best ($\times 10^6$)	GA23 Median ($\times 10^6$)	Significance Level (%) $s_{22,21}$	$s_{23,21}$	$s_{22,23}$
25,000 trials									
1	6.527	6.680	**6.222**	**6.517**	6.306	6.558	1.1	—	5.5
3	5.642	5.722	**5.468**	**5.604**	5.535	5.739	0.1	5.5	—
5	5.922	6.070	5.869	**5.967**	**5.784**	6.038	1.1	—	5.5
100,000 trials									
1	5.621	5.681	**5.582**	5.659	5.609	**5.658**	—	—	—
3	4.861	**4.917**	4.879	4.945	**4.833**	4.924	—	—	—
5	**5.063**	5.171	5.081	5.144	5.090	**5.140**	—	—	—

Table 5.24: Basic adaptation algorithm results with adaptation proportion $\psi = 0.00$, 0.15 and 0.30

Network	GA24 $\kappa = 0.0$ Best ($\times 10^6$)	GA24 Median ($\times 10^6$)	GA21 $\kappa = 0.5$ Best ($\times 10^6$)	GA21 Median ($\times 10^6$)	GA25 $\kappa = 1.0$ Best ($\times 10^6$)	GA25 Median ($\times 10^6$)	Significance Level (%) $s_{24,21}$	$s_{25,21}$	$s_{24,25}$
25,000 trials									
1	6.331	6.520	**6.222**	6.517	**6.222**	**6.476**	—	—	—
3	**5.460**	5.610	5.468	**5.604**	5.545	5.631	—	—	—
5	5.875	6.026	5.869	**5.967**	**5.819**	5.998	—	—	—
100,000 trials									
1	5.613	5.659	5.582	5.659	**5.445**	**5.635**	—	—	5.5
3	4.887	4.932	4.879	4.945	**4.875**	**4.928**	—	—	—
5	**5.079**	5.158	5.081	**5.144**	**5.079**	5.145	—	—	—

Table 5.25: Basic adaptation algorithm results with credit constant $\kappa = 0.0$, 0.5 and 1.0

each table, both the best overall result and best median for a given network and number of trials are displayed in bold. Also, as the same random seeds were used for each set of runs on a network, a non-parametric sign test could be applied to matched pairs [307] from different GAs to establish if there were significant differences in their medians. The resulting significance levels are included in the tables, with *e.g.* $s_{22,21}$ representing the significance level of the comparison of GA22 with GA21. In addition, the full set of results from applying the five GAs to all three networks is recorded in Tables F.57–F.61.

Surprisingly, Table 5.24 at 100,000 trials shows that despite the wide differences in dynamic behaviour observed on individual runs, and discussed above, the final quality of results is unaffected by the choice of adaptation proportion (ψ). However, this can perhaps best be explained by also considering the relative performance in the early part of the same runs. The best results at this stage were achieved, as expected, by GA21 ($\psi = 0.15$), both in comparison with GA22 ($\psi = 0.00$; two highly and one very highly significant median differences) and GA23 ($\psi = 0.30$; a significant median difference). It would appear, therefore, that although at 25,000 trials, GA21 is out-performing the other two, by 100,000 trials the productivity of the combination of problem-specific operators used is such as to allow both GA22 (fixed probabilities) and GA23 (high adaptation rate) to obtain final results of equal quality to GA21.

Different values of credit constant (κ) also produced much less dramatic variations in solution quality than expected. Table 5.25 records that at the end of the runs, the best results were obtained by GA25 ($\kappa = 1.0$). However, there was little significant difference in medians, with only some indication that GA25 ($\kappa = 1.0$) was better than GA24 ($\kappa = 0.0$). Evidence from the early part of the runs is inconclusive, indicating little effect by the credit constant on the initial quality of results. Taking account of the final results though, there is nevertheless some indication both that crediting parent and grandparent operators is to be preferred over simply rewarding the operator responsible for an improvement, and also that a higher credit constant than the usual $\kappa = 0.5$ may be beneficial.

Overall, these experiments on the basic adaptation algorithm indicate that, at least for this application, with powerful problem-specific operators, the main benefits of operator-probability adaptation are in relieving the experimenter of the burden of setting initial probabilities and in the early performance of the GA/heuristic hybrid, rather than in improvements of the final solution quality obtained.

5.3.2 Variants on the Basic Algorithm

Complementing the basic adaptation algorithm, §4.2.4 describes three variants that assign credit to operators for improvements over: best-in-window fitness; the median fitness of the population at the end of the previous generation; or the fitness of the parent individual(s). This subsection explores these variants by repeating the experiments previously applied to the basic algorithm.

However, it would be unproductive, for this application, to use best-of-window fitness, as the (near) monotonic improvement in population fitness results means there is little difference between the fitness of the best individual found up to the end of the previous generation, as used by the basic algorithm, and best-in-window fitness. Consequently, only parental fitness and median fitness credit assignment will be considered.

In addition, there is no need to repeat the results for GA22, as with $\psi = 0.00$ the adaptation algorithm is completely deactivated, including credit assignment of whatever type. As a consequence, the GA22 results can be reused here.

The adaptation parameters (ψ, κ) used by the eight GAs are listed in Table 5.26. GA26-29 use parental credit assignment, and GA30–33 median. By comparing this with Table 5.23 on p. 119,

GA	ψ	κ
26/30	0.15	0.5
27/31	0.30	0.5
28/32	0.15	0.0
29/33	0.15	1.0

Table 5.26: Adaptation parameters for GAs using variant adaptation algorithms

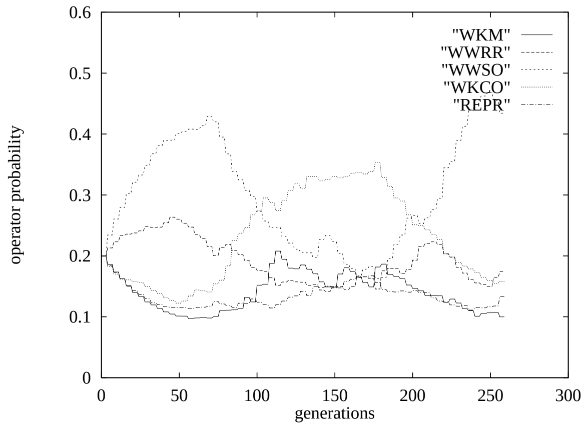
and recalling that there is no need to repeat the runs for GA22, the correspondence with GAs using the basic adaptation algorithm should be clear.

The four GAs using parental credit assignment were each applied ten times with different random seeds to Networks 1, 3 & 5. The operator-probability adaptation curves for single typical runs of GA26-29 are given in Fig. 5.11. In addition, corresponding graphs illustrating the progress of the runs for the four GAs are provided in Fig. F.15.

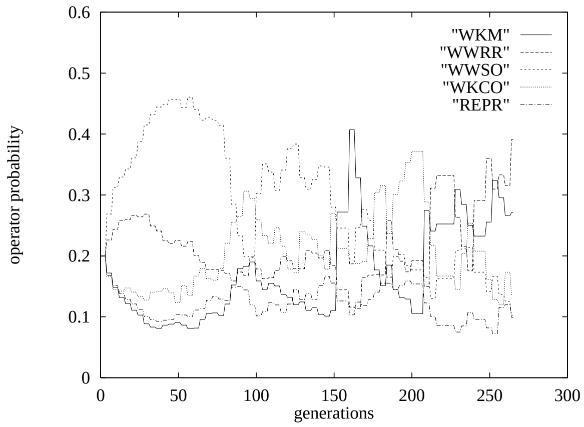
For GA26 in Fig. 5.11(a), the path shift-out operator (WWSO) is the most productive in the first 100 generations, then crossover (WKCO) for the next 100 generations, before a strong resurgence by shift-out at the end of the run. Compare this with GA21 in Fig. 5.10(a), whose adaptation parameters are identical to GA26, except that the former uses best-so-far, rather than parental, credit assignment. In that case, while first shift-out and then crossover were the most productive operators, it was mutation (WKM) that had highest probability at the end of the run. It would appear that, while shift-out struggles to make further improvements in best-so-far fitness in GA21, it is still able to improve the fitness of children *wrt.* their parents in GA26, even in the less genetically diverse conditions at the end of the run.

Although employing parental credit assignment here, altering the adaptation parameters showed similar effects on the operator-probability curves for the other three GAs as were previously observed for the basic adaptation algorithm. In Figures 5.11(b) and (c), both GA27 and GA28 show increased variability compared with GA26, doubtless for similar reasons as suggested above. However, this is particularly marked in Fig. 5.11(c) only for those operators, shift-out and crossover, that appear to perform well under parental credit assignment. This is to be expected, as with $\kappa = 0.0$, it is only the operator directly responsible for an improvement that is rewarded. For GA29 in Fig. 5.11(d), the same tendency to settle on equal operator probabilities of 0.2 can be seen as was noted for the equivalent GA using best-so-far credit assignment (GA25), although this is less marked towards the end of the run.

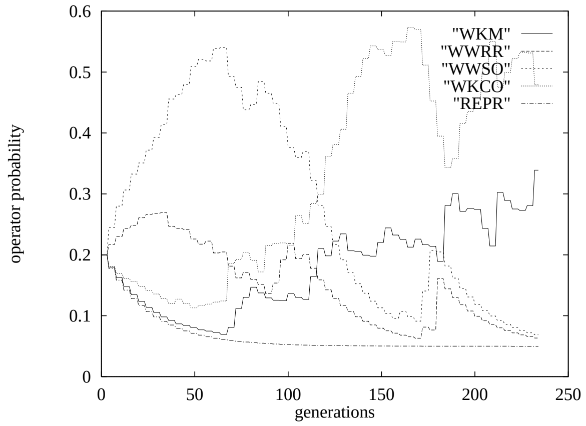
A summary of the results from applying the four GAs (plus GA22) is given in Tables 5.27 and 5.28 (with GA26 included in both tables for ease of reference). These record the best and



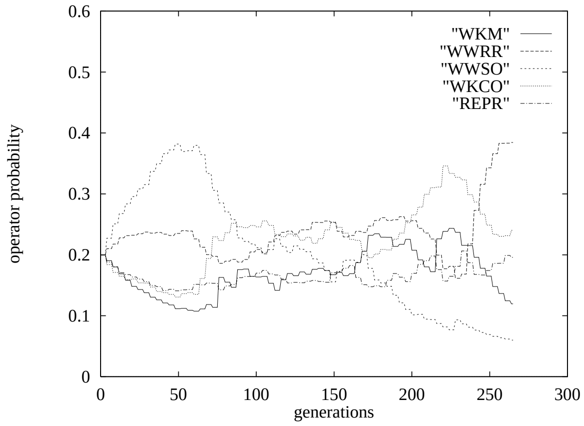
(a) GA26, $(\psi, \kappa) = (0.15, 0.5)$



(b) GA27, $(\psi, \kappa) = (0.30, 0.5)$



(c) GA28, $(\psi, \kappa) = (0.15, 0.0)$



(d) GA29, $(\psi, \kappa) = (0.15, 1.0)$

Figure 5.11: Typical operator-probability adaptation, using parental credit assignment, for GA26–GA29 (Network 1, Run 1)

Network	GA22 $\psi = 0.00$ Best ($\times 10^6$)	GA22 Median ($\times 10^6$)	GA26 $\psi = 0.15$ Best ($\times 10^6$)	GA26 Median ($\times 10^6$)	GA27 $\psi = 0.30$ Best ($\times 10^6$)	GA27 Median ($\times 10^6$)	Significance Level (%) $s_{22,26}$	$s_{27,26}$	$s_{22,27}$
25,000 trials									
1	6.527	6.680	**6.371**	6.536	6.402	**6.474**	5.5	—	0.1
3	5.642	5.722	5.523	**5.602**	**5.470**	5.614	0.1	—	0.1
5	5.922	6.070	**5.707**	5.986	5.751	**5.948**	1.1	—	1.1
100,000 trials									
1	5.621	5.681	5.584	5.666	**5.576**	**5.646**	—	—	1.1
3	4.861	4.917	4.874	**4.907**	**4.840**	4.932	—	—	—
5	**5.063**	5.171	5.095	**5.150**	5.138	5.166	—	—	—

Table 5.27: Parental credit assignment results with adaptation proportion $\psi = 0.00$, 0.15 and 0.30

Network	GA28 $\kappa = 0.0$ Best ($\times 10^6$)	GA28 Median ($\times 10^6$)	GA26 $\kappa = 0.5$ Best ($\times 10^6$)	GA26 Median ($\times 10^6$)	GA29 $\kappa = 1.0$ Best ($\times 10^6$)	GA29 Median ($\times 10^6$)	Significance Level (%) $s_{28,26}$	$s_{29,26}$	$s_{28,29}$
25,000 trials									
1	6.332	6.491	6.371	6.536	**6.219**	**6.487**	—	—	—
3	5.536	5.621	5.523	5.602	**5.490**	**5.577**	—	—	5.5
5	5.760	**5.969**	**5.707**	5.986	5.765	6.013	—	—	—
100,000 trials									
1	**5.563**	**5.622**	5.584	5.666	5.568	5.634	—	—	—
3	4.884	4.941	**4.874**	**4.907**	4.893	4.968	—	5.5	—
5	5.091	**5.134**	5.095	5.150	**5.042**	5.164	—	—	—

Table 5.28: Parental credit assignment results with credit constant $\kappa = 0.0$, 0.5 and 1.0

median results at both the end (100,000 trials) and the early part (25,000 trials) of ten runs on all three networks. As before, in each table, both the best overall result and best median for a given network and number of trials are displayed in bold. The significance levels from a non-parametric sign test on median differences [307] is also included. The full set of results from the runs is recorded in Tables F.63–F.66.

Once again, after 100,000 trials (Table 5.27) there is little to choose between the final results, despite the different levels of adaptation proportion (ψ). However, at 25,000 trials, while both GA26 and GA27 produced very similar results, they were both quite significantly better than GA22, which with $\psi = 0.00$ had the adaptation algorithm deactivated.

Changing the value of the credit constant (κ) appeared to have little effect on either the early part of the run or the final results (Table 5.28). Only two significant median values were found, and the overall pattern of results did not seem to favour any particular κ setting.

Turning now to median credit assignment, the next four GAs (GA30–33) were each applied ten times with different random seeds to Networks 1, 3 & 5. The operator-probability adaptation curves for single typical runs of these are shown in Fig. 5.12. The corresponding graphs illustrating

Network	GA22 $\psi = 0.00$ Best ($\times 10^6$)	GA22 $\psi = 0.00$ Median ($\times 10^6$)	GA30 $\psi = 0.15$ Best ($\times 10^6$)	GA30 $\psi = 0.15$ Median ($\times 10^6$)	GA31 $\psi = 0.30$ Best ($\times 10^6$)	GA31 $\psi = 0.30$ Median ($\times 10^6$)	Significance Level (%) $s_{22,30}$	$s_{30,31}$	$s_{22,31}$
25,000 trials									
1	6.527	6.680	6.396	6.442	**6.249**	**6.425**	0.1	5.5	0.1
3	5.642	5.722	**5.395**	5.590	5.420	**5.569**	0.1	—	1.1
5	5.922	6.070	5.871	5.946	**5.733**	**5.904**	5.5	—	1.1
100,000 trials									
1	5.621	**5.681**	**5.595**	5.706	5.637	5.712	—	—	—
3	**4.861**	**4.917**	4.919	4.971	4.890	4.925	(1.1)	—	—
5	**5.063**	5.171	5.113	**5.161**	5.107	5.179	—	—	—

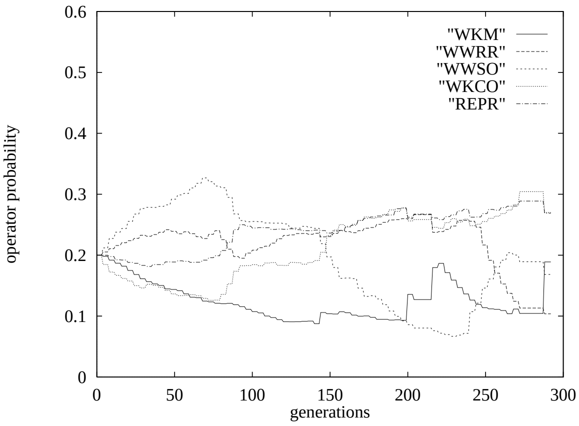
Table 5.29: Median credit assignment results with adaptation proportion $\psi = 0.00$, 0.15 and 0.30

the progress of the runs are given in Fig. F.16.

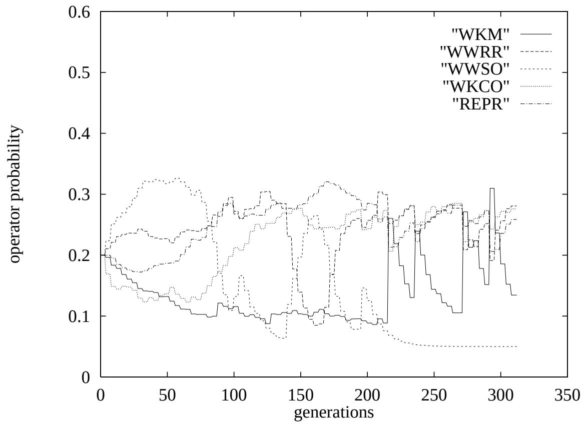
All four GAs seem to produce approximately the same overall pattern of probability curves. After some initial divergence from the common starting point of 0.2, the curves occupy the narrow band between 0.1 and 0.3. The shift-out operator (WKSO) perfoms well initially in all four GAs, but seems less productive later on, particularly for GA31 with its high adaptation proportion ($\psi = 0.30$). In contrast, mutation (WKM) appears to perfom poorly throughout in all four GAs, although this is less marked again in GA31. Overall, it may be that rewarding operators for improvements over median fitness is a relatively poor indicator of their productivity, as the result of a GA run is judged on the best individual found rather than the population median achieved. Consequently, basing credit assignment on median fitness is likely to give little guidance to the adaptation algorithm, resulting in the approximately equal apportionment of probability observed over the different adaptation parameter settings.

A summary of the results from applying the four GAs (plus GA22) is given in Tables 5.29 and 5.30 (with GA30 included in both tables). These record the best and median results at both the end (100,000 trials) and the early part (25,000 trials) of ten runs on all three networks. As usual, in each table, both the best overall result and best median for a given network and number of trials are displayed in bold. The significance levels from a non-parametric sign test on median differences [307] is also included. The full set of results from the runs is recorded in Tables F.67–F.70.

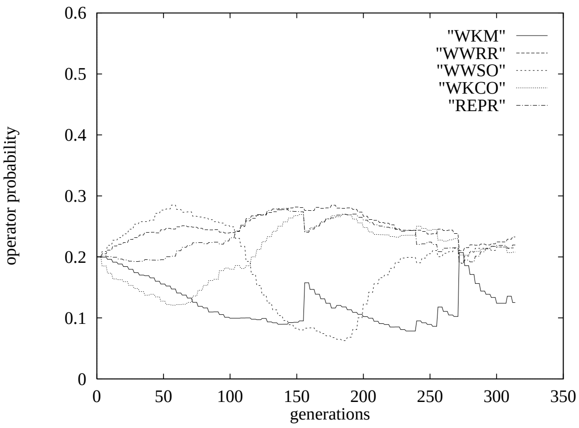
Considering first variations in the adaptation proportion (ψ) summarised in Table 5.29, the results at 100,000 trials surprisingly show GA22, with the adaptation algorithm deactivated, actually performing better than the other two GAs, although this is not a significant median difference except for one network. At 25,000 trials, however, it was GA31, with a high adaptation proportion ($\psi = 0.30$), which obtained the best results compared to GA22 (adaptation deactivated), although



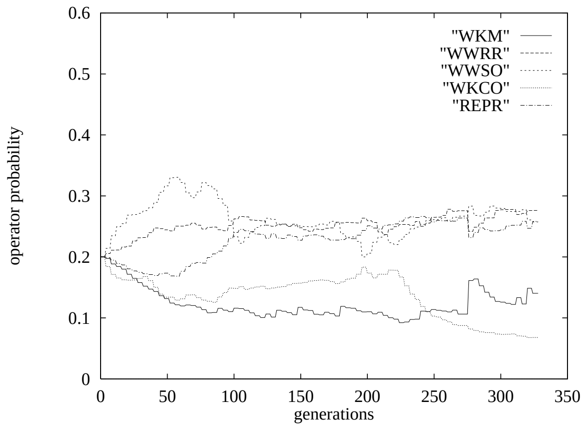
(a) GA30, $(\psi, \kappa) = (0.15, 0.5)$



(b) GA31, $(\psi, \kappa) = (0.30, 0.5)$



(c) GA32, $(\psi, \kappa) = (0.15, 0.0)$



(d) GA33, $(\psi, \kappa) = (0.15, 1.0)$

Figure 5.12: Typical operator-probability adaptation, using median credit assignment, for GA30–GA33 (Network 1, Run 1)

Network	GA32 $\kappa = 0.0$		GA30 $\kappa = 0.5$		GA33 $\kappa = 1.0$		Significance Level (%)		
	Best ($\times 10^6$)	Median ($\times 10^6$)	Best ($\times 10^6$)	Median ($\times 10^6$)	Best ($\times 10^6$)	Median ($\times 10^6$)	$s_{30,32}$	$s_{33,30}$	$s_{32,33}$
25,000 trials									
1	**6.305**	6.446	6.396	**6.442**	6.401	6.517	—	5.5	—
3	5.450	5.607	**5.395**	5.590	5.475	**5.575**	—	—	—
5	**5.791**	**5.933**	5.871	5.946	5.827	5.943	—	—	—
100,000 trials									
1	5.623	5.686	5.595	5.706	**5.547**	**5.669**	5.5	—	—
3	4.890	**4.956**	4.919	4.971	**4.840**	4.962	—	—	—
5	5.099	**5.134**	5.113	5.161	**5.075**	5.175	5.5	—	—

Table 5.30: Median credit assignment results with credit constant $\kappa = 0.0$, 0.5 and 1.0

GA30 was also better than GA22 with good significance levels. As usual, variations in credit constant (κ) had little effect, either in the early part or at the end of the run (Table 5.30).

Overall, it was not felt appropriate to judge which of the three variants of the adaptation algorithm examined was 'best' for minimum cost allocation, primarily because the initial operator probabilities were all artificially set to 0.2 in this study, rather than being tuned to improve algorithm performance. However, comparing GA21, GA26 and GA30, which all used the default adaptation parameter settings of $(\psi, \kappa) = (0.15, 0.5)$, there would appear to be little to choose between best-so-far and parental credit assignment, but that both would appear to be better than employing median fitness. In addition, the conclusions from the study of the basic adaptation algorithm also seem to apply here, namely that the main benefits of operator-probability adaptation are in relieving the experimenter of the burden of setting initial probabilities and in the early performance of the GA/heuristic hybrid, rather than in improvements of the final solution quality obtained.

5.4 Summary

This chapter has provided a number of experiments exploring the suitability of the author's GA/heuristic hybrid approach for multi-wavelength all-optical transport network design. Three studies exploring different approaches to mesh network topology design were reported: initial experiments established the superiority of a problem-specific form of crossover; work on local search operators showed the limited merits of incorporating a flexible local search operator combined with trial-based probability reassignment; finally, the power of iterative heuristic operators was demonstrated, although not without cost in computational terms. In addition, the initial topology approach was used to design the test networks that were to be used in the next group of experiments.

GA/heuristic hybrids for routing, fibre choice and wavelength allocation were then investigated. Two metrics for network assessment were used, one based on NWR, and the other on a simplified model of network cost. In both cases, the GAs were compared with recent wavelength-allocation heuristics published in the literature. For minimum NWR, the GA/heuristic hybrid approach was shown to be as promising as had been hoped, as the blend of a GA with steps from one of the heuristics resulted in both improved performance (over the GA alone) and improved results (over the heuristic alone). In addition, the GA results matched those of the best wavelength-allocation heuristic examined. Similar success was achieved with the minimum cost allocation, as in competition with three recent heuristics, the GA/heuristic hybrid approach was able to obtain the lowest network costs for two settings of the cost parameters (including the one judged most realistic), and in a third case, was able to approach the results of the best of the allocation heuristics.

The final investigation was of NOMaD's operator-probability adaptation mechanism, both in its basic form, where credit is assigned for improvements over the best-so-far individual, as well as two variants that reward improvements over parental and median fitness. At least in the application examined, minimum cost allocation which uses powerful problem-specific operators, the main benefits of operator-probability adaptation were found to be in relieving the experimenter of the burden of setting initial probabilities and in the early performance of the GA/heuristic hybrid, rather than in improvements of the final solution quality obtained.

In the next chapter, there will be more detailed reflection on the GA/heuristic hybrid approach, including the design of NOMaD, further discussion of the experimental results obtained, and consideration given to directions for further work.

Chapter 6

Discussion & Conclusions

Now all has been heard; here is the conclusion of the matter: Fear God and keep his commandments, for this is the whole duty of man.

Ecclesiastes 12:13, NIV

This chapter reflects on the limitations and, to some extent, the successes of the research presented in the thesis, makes suggestions for further work[1], records the author's conclusions, and lists the contributions of the thesis.

The chapter is divided into six sections. The genetic-algorithm (GA)/heuristic hybrid approach in general is discussed in §6.1, while §6.2 focuses particularly on operator-probability adaptation. The following two sections examine NOMaD's two main application areas: mesh network topology design (§6.3) and routing, fibre choice and wavelength allocation (§6.4). The key conclusions reached in both the earlier chapters and this one, as well as the author's overall conclusions, are recorded in §6.5 and, finally, §6.6 repeats the list of contributions of the thesis first provided in §1.7.

6.1 GA/Heuristic Hybrid Approach

Although the GA/heuristic hybrid approach implemented in NOMaD and presented in this thesis has been reasonably successful for the particular design problems and network sizes considered, there nevertheless remains considerable potential for further work exploring variations to the algorithms and operators examined.

One area is creation of the initial populations. For mesh network topology design, initialisation has been with the random links operator (see p. 80), which inserts a link between each node pair

[1] Additional discussion and suggestions for further work *wrt.* the NOMaD software have been recorded in §E.8

in the network with probability 0.5. Now, with a network of n nodes, the initial expected number of links is $0.5 \times n(n-1)/2$ links, *i.e.* for a 15-node network, 52.5 links. However, Networks 1–5 (§5.1.2 and Fig. 5.3), designed during the initial topology experiments, evolved to have between 36 and 40 links. This reduction in the number of links in a 15-node network during a GA run is further demonstrated in Fig 5.2 on p. 96. Consequently, it may be that initialising the networks with a smaller number of links, similar to the number actually required in the final design, would accelerate the early evolution of the topologies. This could be achieved by adding a link-probability parameter to the random links operator (and for 15-node networks setting it to, say, 0.36). Such a parameter might become even more necessary in larger networks, where otherwise the expected number of links in the initial random population will grow $O(n^2)$, whereas the final designs might be expected to have a number of links determined more by node degree, *i.e.* $O(n)$.

Similar considerations apply to routing, fibre and wavelength allocation, where the initial population is currently created using the random k-shortest paths operator (p. 82). Other possibilities might include seeding the initial population with the results from an heuristic allocation algorithm; or using a random k-lowest hop count routing operator.

Another area meriting examination is the choice and parameter(s) of the selection function. Throughout this thesis, as was explained in §4.2.1, deterministic tournament selection has been used[2], with a tournament size of 4, as there is strong support for tournament selection in the literature [304,305]. However, in future work, initially the influence of tournament size, and subsequently the use of other selection functions, could be explored to identify any improvements over the results reported here.

An alternative to the generational model currently used by NOMaD would be the adoption of *steady-state reproduction* [9, 308]. In this approach, rather than the old population being entirely replaced by the new one, instead only a small number of individuals are replaced each iteration of the GA. Typically, only the one or two worst individuals are replaced and, in addition, offspring are often checked before insertion and, if they are identical to existing members of the population, discarded (*steady-state without duplicates*). The main advantage of steady-state reproduction is that superior individuals in the population are retained, and can continue to contribute to evolution, until better individuals replace them. However, whether this potential would be realised in the context of the design problems examined in this thesis has not been established. An additional benefit, given the large memory requirements of the GA/heuristic hybrid apporoach, is that only a single population

[2]In deterministic tournament selection, the best individual in the tournament always wins.

must be held in memory at once, rather than both an old and a new. Consequently, either larger populations or larger networks could be addressed within the same memory constraints.

A novel population dynamic, the *stud GA*, was recently introduced by Khatib & Fleming [309]. They propose allowing only the best individual to mate (*the stud*), applying crossover between the stud and *all* members of the population, and when the population diversity starts to drop, using mutation. While such an approach could not be applied unmodified to a GA/heuristic hybrid, with its many operators, the idea of only mating the best individual may have advantages in a context where powerful local-search/heuristic operators are being utilised. Explicitly recombining the fittest individual with each member of the population in turn before randomly choosing an heuristic operator to apply to the offspring may result in more rapid evolution, and this merits investigation.

Whilst the development of a distributed multi-population approach has already been mentioned, one additional possibility would be to employ different genetic operator combinations in the different sub-populations (*heterogeneous distributed GAs*). This might lead to further improvements beyond those resulting from both the inherent parallelism of the distributed approach and the possible niche/island effect of separate evolution in the different sub-populations. Schnecke & Vornberger [257] successfully used such an approach in their hybrid GA for constrained placement problems. Also, a recent paper by Herrera & Lozano [310] examines several such heterogeneous distributed approaches for real-number representations, establishing their superiority over sequential and homogeneous distributed GAs in this context.

Another area for future work lies in recognising the potential for multi-objective optimisation in the design problems addressed in the thesis. All of the metrics for design assessment in §4.1 combine different elements into a single overall objective, although their relative impact can be adjusted by parameters. Instead, NOMaD could be extended to allow metrics with multiple objectives to be defined and used to guide the evolutionary process [275–277]. In practical network design, it is usually neither possible nor desirable to combine all the objectives and constraints into a single overall metric [10], as the final choice of design may be negotiated and involve factors that cannot easily be included. Consequently, developing a multi-objective approach would enhance the applicability of the work to practical optical network design.

6.2 Operator-probability Adaptation

In §5.3 both the basic operator-probability adaptation algorithm and two of the relevant variants from §4.2.4 were examined in the context of minimum-cost wavelength allocation. Overall, it was concluded that, at least for this design problem and the network sizes examined, the main benefits of operator-probability adaptation are in relieving the experimenter of the burden of setting initial probabilities and in the early performance of the GA/heuristic hybrid, rather than in improvements of the final solution quality obtained. Clearly, however, there are a number of avenues of further work that could be explored.

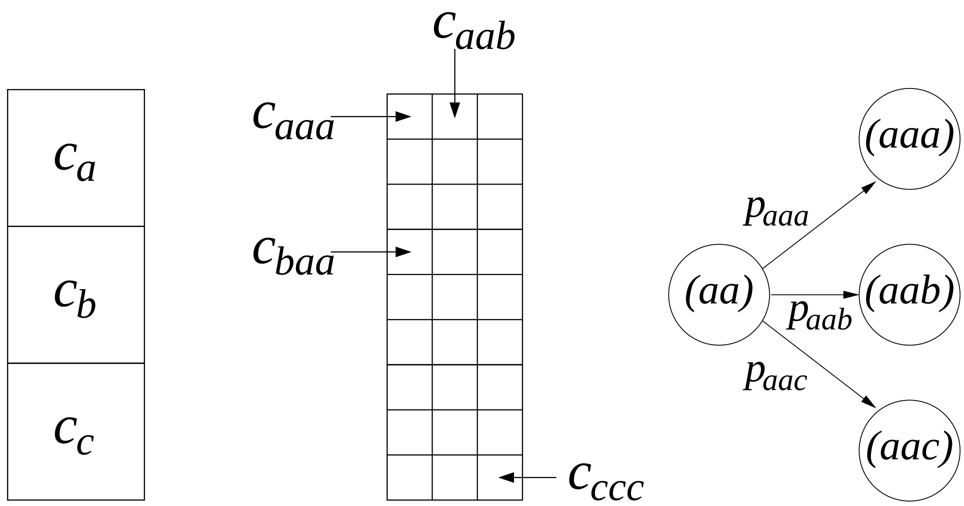
Both the basic and variant adaptation algorithms could be tested under a wider variety of conditions: with varying network sizes and for different design problems, starting with minimum NWR design and mesh network topology optimisation. Indeed, whilst there is merit in experimenting with the adaptation mechanism in the context of engineering design problems, additional insight might also be gained by using some of the 'standard' evolutionary algorithm test functions [2, 309] or a test-case generator [311] for a systematic study.

A different class of credit-assignment algorithm could also be investigated. Instead of awarding credit equal to the actual improvement in fitness (over the best-so-far, best-in-window, parental or median fitness), a constant amount of credit, say 1 unit, could be awarded each time there was an improvement[3]. This approach would allow operators that were making small improvements in fitness to receive equal reward to those making large improvements. While this may seem counter-productive, it may well be beneficial where fitness improvement events are comparatively rare, and a single large improvement in the credit period (ε) could cause one operator to dominate another one that was nevertheless contributing many small improvements.

Currently, all the credit-assignment algorithms examined reward operators when they produce improved children. However, a future possibility would be to also penalise them when they fail to generate improvements, particularly for those algorithm variants which depend on parental fitness [261].

Further, although the present credit-assignment mechanism rewards parent and grandparent operators, thereby recognising that several operators in a chain may be responsible for an eventual improvement in fitness, these chains of operators are not explicitly tracked by the algorithm. Consider, for example, a GA/heuristic hybrid with three operators, a, b and c. With the present

[3]Suggested to the author by George D. Smith of the University of East Anglia.



(a) Current algorithm (b) Chain credit (c) Chain probability

Figure 6.1: Operator-chain credit assignment

credit-assignment algorithm, three variables are maintained (Fig. 6.1(a)) to accumulate the credit awarded to the three operators, c_a, c_b and c_c. With the proposed operator-chain credit assignment[4], 27 variables would be used (Fig. 6.1(b)), ranging from c_{aaa} to c_{ccc}. If an operator chain (abc), *i.e.* consisting of grandparent operator a, parent b and then operator c, produced an improvement, the credit would be accumulated in variable c_{abc}. Distinct conditional operator-chain probabilities, derived in the usual way from the accumulated credit, would then be used to choose the operators applied. For example in Fig. 6.1(c), when selecting an operator to apply to an individual whose grandparent and parent operators were (aa), the three conditional probabilities p_{aaa}, p_{aab} and p_{aac} would be employed[5]. While complex, this approach may allow more accurate tracking of successful operator chains, leading to improved performance by a GA/heuristic hybrid utilising this operator-probability adaptation mechanism.

One of the difficulties in designing an effective conventional heuristic is not only must the appropriate operators (steps, perturbations, *etc.*) to incorporate be selected, but also their sequence of application must be determined. One possible approach is to exhaustively try all possible combinations of operators and sequences on some trial design problems. This was attempted by one of the author's M.Sc. students [312, 313] for optical mesh network topology design. Clearly though, such an approach is necessarily limited due to the enormous number of potential operators and se-

[4]This approach was originally inspired by a question from Riccardo Poli of the University of Birmingham.

[5]This treatment assumes a, b and c are unary operators; for binary, the average of the conditional probabilities applying to the two individuals could be substituted.

quences for realistic problems. However, the operator-probability adaptation mechanism described in this thesis may be able to provide insight into both the relative effectiveness and also, if the operator-chain approach was adopted, sequencing of heuristic operators. Consequently, a GA/heuristic hybrid could be developed side-by-side with a conventional heuristic. The operator probability results would allow steps in the heuristic to be evaluated as operators within the GA. This could then lead to the development of a more advanced overall heuristic, using the operators and sequencing identified. Further experimentation with this advanced overall heuristic, which would necessarily be more straightforward to trace and analyse than a GA, might provide inspiration for the development of further operators. These could in turn be assessed in the GA, and so on. It is likely, given the memory and computer time requirements of a population-based approach, that while a conventional heuristic developed in such a context would probably not produce results of the same quality as the GA for small problems, nevertheless it might well be superior to the GA for larger problems, where memory constraints or computation time would disqualify the GA.

6.3 Mesh Network Topology Design

Three studies exploring different approaches to mesh network topology design were reported in §5.1: initial experiments established the superiority of a problem-specific form of crossover; work on local search operators showed the merits of incorporating a flexible local search operator combined with trial-based probability reassignment; finally, the power of iterative heuristic operators was demonstrated, although not without cost in computational terms.

Clearly, future work could examine a wider range of network design problems, perhaps exploring additional heuristic operators for optical mesh network topology design, or exploiting some of the potential improvements in the NOMaD framework itself. Another possibility would be to try and gain additional insight into the design problem itself by characterising the fitness landscape under the different operators [314]. However, a more productive avenue for research may be to apply the GA/heuristic hybrid approach to a different range of network topology design problems.

In those GAs used in §5.1 for mesh network topology design, while the individual links included in the network are selected by the GA, their capacity is only indirectly determined by it. Instead, during assessment by Network Cost Model A (§4.1.1), the traffic requirements are first routed across the network, and then the link capacities set to whatever level is needed to provide reasonable occupancies. In addition, the capacity of the links is costed by steps fitting a continuous function, rather than by arbitrary costs for a fixed set of capacities. However, it would be relatively straightforward to extend NOMaD to include such explicit capacity assignment and arbitrary costs in the evolutionary process (*cf.* Deeter & Smith [148], who used arbitrary costs for a fixed set of link reliabilities).

Although the topology designs in this thesis have been flat mesh networks, nevertheless more constrained topology classes may provide benefits in lower provisioning and maintenance costs, or in simplified network management [19]. One possibility is hierarchical networks, where nodes and links are each associated with one or more discrete levels. NOMaD's network model already provides flexible support for hierarchical networks with up to 32 levels (see §E.4), although this potential has not been used in the author's own work[6]. By developing suitable operators and metrics, the GA/heuristic hybrid approach could also be applied to the topology design and allocation of hierarchical networks, and the results compared to those here.

A more focused topology class again is that of dual-homing multi-ring hierarchical networks. For example, Fig. 6.2 shows a two-level network of 16 nodes. The nodes have been grouped into four rings at level 1 (R_{11}, R_{12}, R_{13} and R_{14}). From each of the level 1 rings, two nodes have then been selected to form the higher level interconnecting rings (hence *dual-homing*). There are two of these at level 2 (R_{21} and R_{22}), with one of the common nodes from each level 1 ring included in a different higher-level ring, an interconnection strategy referred to as *different common ring* (DCR). One of the author's former Ph.D. students, Katerina Proestaki, has developed several topology design, routing, fibre and wavelength-allocation heuristics for this class of networks, all implemented on NOMaD [20–24]. An interesting area of future work would be to employ the steps in these heuristics as operators within GA/heuristic hybrid algorithms, hopefully providing both improved design results, at least for small networks, as well as additional insight into this class of problems.

6.4 Routing, Fibre & Wavelength Allocation

Routing, fibre choice and wavelength allocation were studied in §5.2. Two metrics for network assessment were used, one based on NWR, and the other on a simplified model of network cost. In both cases, the GAs were compared with recent wavelength-allocation heuristics. For minimum NWR, the GA/heuristic hybrid approach fulfilled its promise, as the blend of a GA with steps from

[6] Although it has been used by Noel Parnis.

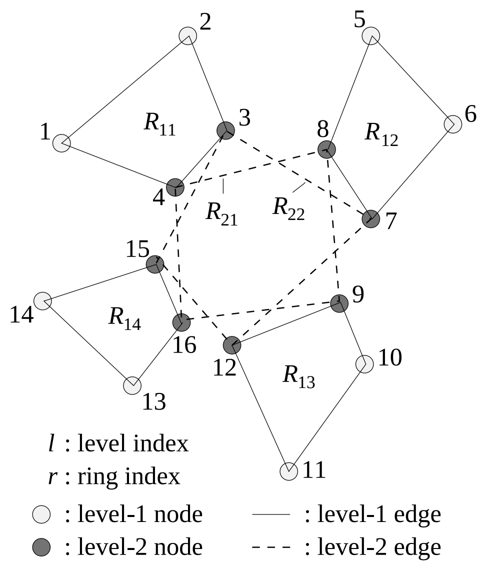


Figure 6.2: Dual-homing multi-ring network with DCR interconnection [20]

one of the heuristics resulted in both improved performance (over the GA alone) and improved results (over the heuristic alone). In addition, the GA results matched those of the best wavelength-allocation heuristic examined. Similar success was achieved with the minimum cost allocation, as in competition with three recent heuristics, the GA/heuristic hybrid approach was able to obtain the lowest network costs for two settings of the cost parameters (including the one judged most realistic), and in a third case, was able to approach the results of the best of the allocation heuristics.

In future, as well as exploring a wider range of network sizes, comparisons could be made with more of the allocation heuristics from the literature including, for example, Nagatsu *et al.*'s wavelength-path (WP) allocation heuristic [296], which is already implemented in NOMaD (see Appendix C), an interesting one by Baroni & Bayvel [300], and a recent one by Alanyali & Ayanoglu [315] that can be adapted to take account of network restoration.

The extension of NOMaD to allow the representation of restoration paths for fibre and wavelength allocation is suggested in §E.8.3. Clearly further experiments could then be performed in this area, and comparisons made with allocation heuristics that also include restoration paths, *e.g.* Nagatsu *et al.* [297, 302]. Likewise, the modelling of restrictions on *e.g.* the number of wavelengths on the fibres could lead in turn to comparisons with similarly designed heuristics, such as others by Nagatsu *et al.* [302, 303].

While the focus in the thesis has been on WP routing, without considering the more straightforward problems of virtual-wavelength-path routing (VWP; see §1.2), a further possibility remains. This is sparse-wavelength-path (SWP) routing, where although wavelength conversion is available in the network, it is only available at some nodes, for example, or where the number of conversions allowed in the network is constrained in some other way [316]. This approach confers some of the flexibility of VWP, particularly during failure conditions, without incurring the high cost and potential transmission limitations of the unrestricted use of wavelength conversion. Appropriate development of NOMaD would allow the GA/heuristic hybrid approach to be applied to this area as well.

The support of NOMaD's network model for both network hierarchy and node clustering is outlined in §E.4, and the potential application of these to topology design of hierarchical networks in general, and dual-homing multi-ring networks in particular, was suggested above. Clearly, it would also be worthwhile to apply NOMaD to the routing, fibre and wavelength allocation of such networks.

In this thesis, network topology design and routing, fibre and wavelength allocation have largely been considered separately, with the network topology being established before the fibre and wavelength allocation is performed. However, it may well be beneficial to combine both into a single integrated design problem (*cf.* for example work by Schnecke & Vornberge [257], who found notable benefits from such an integrated approach in the context of constrained placement problems). This would allow the topology designed to be directly influenced by the allocations, which may well result in a better overall network assessment (*e.g.* lower cost) than tackling the two areas sequentially. For example, in Tables 5.17–5.22 on p. 115, the network costs obtained for Network 6 are always consistently below those for Network 1. However, while both networks have the same node distribution and experience the same traffic demand (see §5.1.2 on p. 98), the topology of Network 1 was obtained with $\alpha = 1$ in Network Cost Model A and that of Network 6 using $\alpha = 0.8$. Consequently, after wavelength allocation, while it might have been expected that Network 6 would have the lower cost with $\alpha = \beta = 0.8$ in Network Cost Model B (Tables 5.20–5.22), Network 1 should have been better for $\alpha = \beta = 1$ (Tables 5.17–5.19). This was not the case, however, and so there is clearly 'room for improvement' that could be exploited by an integrated design approach[7].

6.5 Conclusions

The introduction (Ch. 1) explained the overall direction of the thesis: the application of the genetic-algorithm (GA)/heuristic hybrid approach to two of the main problems in the design of multi-

[7]The author is grateful to Alan Hill of Ilotron Ltd. for a useful discussion on this topic.

wavelength all-optical transport networks: mesh network topology design, and wavelength-path (WP) routing, fibre choice and wavelength allocation. It established the level of background knowledge expected of the reader and briefly discussed the choice of a flat, all-optical, arbitrary-mesh multiple-fibre-link topology with WP routing as the problem domain selected. The background and motivation behind the solution approach of evolutionary algorithms (EAs) in general, and GA/heuristic hybrids in particular, was given, before the overall aims and objectives of the thesis were elaborated.

The chapter on evolutionary algorithms (Ch. 2) included a brief overview and historical perspective on EAs, a tutorial introduction to GAs, as well as three literature surveys. The overview considered early work in the field, GAs, evolutionary programming, evolution strategies, genetic programming, classifier systems and evolutionary hardware, as well as conferences and books on the area. The tutorial provided background on both GAs in general and Goldberg's Simple Genetic Algorithm in particular before introducing in more detail a central theme of the thesis, GA/heuristic hybrids. The first survey gave near-exhaustive coverage of evolutionary telecommunications, *i.e.* the application of EAs to the telecommunications problem domain, up to December 1998. It concluded with some comments on promising directions for future research and encouragement to future authors in regard to both citation of earlier work and the reproducibility of their research. The two remaining sections gave more limited surveys of prior work on GA/heuristic hybrids and operator-probability adaptation, respectively, in areas other than evolutionary telecommunications. Overall, the chapter gave the context for the evolutionary aspects of the author's research, as well as establishing the foundation on which it was built.

The chapter on NOMaD (Ch. 3) provided an overview of the software toolset, including its application areas, design metrics and supported solution approaches. The overall NOMaD architecture, its object-oriented network toolset, interactive scripting capabilities, graphical user interface, testing & verification strategy and an example compiled application were all discussed.

The chapter on GA/heuristic hybrid design (Ch. 4) presented greater detail of NOMaD's design metrics and evolutionary algorithm framework. It is these elements, implemented in the NOMaD software toolset, that encapsulate the author's GA/heuristic hybrid approach to multi-wavelength all-optical transport network design. Details of the design metrics used by the author for network assessment were presented: one for minimum-cost mesh network topology design, another for network wavelength requirement (NWR), and the final one for minimum-cost wavelength allocation. Moreover, the evolutionary algorithm framework was described, including the overall structure of the GA/heuristic hybrid; operators for network topology design; operators for routing, fibre and wavelength allocation; and the operator-probability adaptation mechanism.

The experimental results chapter (Ch. 5) described a number of experiments exploring the suitability of the author's GA/heuristic hybrid approach for multi-wavelength all-optical transport network design. Three studies exploring different approaches to mesh network topology design were reported: initial experiments established the superiority of a problem-specific form of crossover; work on local search operators showed the limited merits of incorporating a flexible local search operator combined with trial-based probability reassignment; finally, the power of iterative heuristic operators was demonstrated, although not without cost in computational terms. In addition, the initial topology approach was used to design the test networks that were to be used in the next group of experiments.

GA/heuristic hybrids for routing, fibre choice and wavelength allocation were then investigated. Two metrics for network assessment were used, one based on NWR, and the other on a simplified model of network cost. In both cases, the GAs were compared with recent wavelength-allocation heuristics published in the literature. For minimum NWR, the GA/heuristic hybrid approach was shown to be as promising as had been hoped, as the blend of a GA with steps from one of the heuristics resulted in both improved performance (over the GA alone) and improved results (over the heuristic alone). In addition, the GA results matched those of the best wavelength-allocation heuristic examined. Similar success was achieved with the minimum cost allocation, as in competition with three recent heuristics, the GA/heuristic hybrid approach was able to obtain the lowest network costs for two settings of the cost parameters (including the one judged most realistic), and in a third case, was able to approach the results of the best of the allocation heuristics.

The final investigation was of NOMaD's operator-probability adaptation mechanism, both in its basic form, where credit is assigned for improvements over the best-so-far individual, as well as two variants that reward improvements over parental and median fitness. At least in the application examined, minimum cost allocation which uses powerful problem-specific operators, the main benefits of operator-probability adaptation were found to be in relieving the experimenter of the burden of setting initial probabilities and in the early performance of the GA/heuristic hybrid, rather than in improvements of the final solution quality obtained.

Earlier sections in this chapter reflected on both the successes and, to a greater extent, the limitations of the work described in this thesis, outlining from these the potential for future research. In §6.1 and §6.2, the GA/heuristic hybrid approach in general was discussed, with operator-probability adaptation considered separately. Future work could enhance initialisation, selection and population dynamics, implement a distributed multi-population GA, or employ multi-objective optimisation. The credit-assignment mechanism could be extended in a variety of ways, including explicit reward of operator chains. In addition, the potential of the GA/heuristic hybrid approach as a testbed for both operators and their sequences within conventional heuristics was also recognised.

In §6.3 and 6.4, NOMaD's two main application areas, mesh network topology design, as well as routing, fibre choice and wavelength allocation, were explored. Whilst recognising the success of the GA/heuristic hybrid approach in the experimental results, a number of directions for future work were nevertheless identified. Addressing hierarchical networks in general, and dual-homing multi-ring networks in particular, was recommended for both topology and wavelength allocation. Moreover, for the latter area, restoration-path allocation, sparse-wavelength-path routing, and the integration of both topology and wavelength allocation into a single design process are all future possibilities.

Overall, the application of the GA/heuristic hybrid approach to optical network design can be considered a success. In particular, problem-specific representation and the incorporation of problem-specific heuristics proved successful for both mesh network topology design, as well as routing, fibre and wavelength allocation in multi-wavelength all-optical transport networks. In many cases, a hybrid GA provided not only better performance than a GA alone, but also better results than conventional heuristics. However, some limitations of the approach were identified, especially the large memory and processing requirements. In addition, the operator-probability adaptation mechanism developed did not provide the expected improvements in result quality, although it still conferred some notable benefits. Finally, several of the questions posed in §1.5 remain open, and in particular whether object-oriented representation is really to be preferred to binary (or other) encoding, but these, and many other questions raised by the research, must remain topics for further work.

6.6 Contributions of the Thesis

The key contributions of the thesis (repeated from §1.7) are:

- A tutorial introduction to GAs and GA/heuristic hybrids (§2.2).
- A near-exhaustive survey of evolutionary telecommunications, *i.e.* the application of EAs to the telecommunications problem domain, up to December 1998 (§2.3) [8].

- The design of the network model and GA/heuristic hybrid framework incorporated in NOMaD (§E.3 and §4.2) [31, 32].
- NOMaD's operator-probability adaptation mechanism (§4.2.4 and §5.3) [33].
- The GA/heuristic hybrid algorithms for optical mesh network topology design, particularly the new link-group crossover operator (§5.1) [34].
- The GA/heuristic hybrid algorithms for minimum network wavelength requirement routing, fibre-choice and wavelength allocation, particularly the new problem-specific operators: wavelength-allocation random k-shortest paths, k-shortest path mutation, path re-route, path shift-out and k-shortest path single-point crossover (§5.2.1) [35].
- The GA/heuristic hybrid algorithms for minimum cost routing, fibre choice and wavelength allocation, using the same new problem-specific operators (§5.2.2) [36, 37].

Acknowledgements

Give thanks to the LORD*, for he is good;*
his love endures for ever.

Psalm 107:1, NIV

All stories have to start somewhere, and my thanks go first to Alan Hind, former British Telecom Research Fellow at the School of Engineering and Computer Science, University of Durham, who in 1992 first encouraged me to consider the use of genetic algorithms (GAs) for network design problems.

I am grateful to those who helped in locating papers for the survey on evolutionary telecommunications in Ch. 2, including George D. Smith, Steve Hurley, W. Don Potter and Roger L. Wainwright. A special mention is also due to Paul Naish of the Albert Sloman Library, University of Essex, for tracking down many difficult inter-library loans.

My thanks go to Katerina Proestaki and Noel Parnis (both formerly research assistants in the Department of Electronic Systems Engineering, University of Essex) for the many interesting and stimulating discussions during the development of NOMaD. They also both helped me by reading through early drafts of Ch. 3, Ch. 4 and Appendix E. Katerina was initially supported by the European Commission on ACTS project AC029 (WOTAN), and subsequently was the Fujitsu Research Fellow; she is now with Nortel. Noel was supported by ACTS project AC066 (OPEN), and is now with Lucent.

I am also grateful to Paul Griffith, formerly a research assistant in the Department of Electronic Systems Engineering, University of Essex, supported by the European Commission on ACTS project AC029 (WOTAN), and now with GPT Ltd., for helpful discussions during the development of the traffic distribution approach (§F.1.2), which was first used in his M.Sc. project.

A thank you is due to Carlos Fonseca, formerly of the Department of Automatic Control and Systems Engineering, University of Sheffield, and now with University of Algarve, for first suggesting I adopt the use of non-parametric hypothesis testing.

I am grateful to George D. Smith of the School of Information Systems, University of East Anglia on at least two counts: for suggesting constant-value credit assignment (see §6.2) and for being such an excellent host during my Visiting Fellowship to the School of Information Systems in academic year 1998/9. Many thanks for all those conversations over lunch and coffee at UEA!

Ricardo Poli of the School of Computer Science, University of Birmingham has my thanks for first suggesting operator-chain credit assignment in a question responding to my paper [33] at the Intl. Conf. on Evolutionary Computation in 1998.

Alan Hill of Ilotron Ltd. has my gratitude for a useful recent discussion on integrating network topology design with routing, fibre and wavelength allocation. This resulted in the illustrative example included in §6.4.

My thanks go to the authors of `xfig` [288], `gnuplot` [317] and LaTeX [318,319], which all proved invaluable in the preparation of this thesis.

My examiners, Peter Fleming of Department of Automatic Control and Systems Engineering, University of Sheffield and Andy Keane of the School of Engineering Sciences, University of Southampton, deserve a thank you for their patience. While my thesis was originally scheduled for submission in December 1998, this did not actually take place until April 2001.

My thanks also go to all those people—too many to mention—who kept asking "Have you finished your thesis yet?" While I found this a constant irritation, it was also a spur to progress!

I am very grateful to Rob Massara of the Department of Electronic Systems Engineering, University of Essex, who acted as *de facto* supervisor. Many thanks for all those encouraging conversations in the Blues Cafe over cappuchino, and for reading (almost) all the thesis (more than once)!

Finally, very many thanks to Shirley, my wife, who had to put up with a lot from me over the many years it has taken to put this thesis together. Thank you for not giving up!

References

[1] Mukherjee, B., *Optical Communication Networks*, McGraw-Hill, 1997

[2] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989

[3] Mitchell, M., *An Introduction to Genetic Algorithms*, MIT Press, 1996

[4] Fogel, D.B. (Ed.), *Evolutionary Computation: The Fossil Record*, IEEE Press, 1998

[5] Kirkpatrick, S., Gelatt, C.D., Jr., & Vecchi, M.P., "Optimization by simulated annealing", *Science* v220 n4598, May 1983, pp. 671–680

[6] Glover, F., "Tabu search—part I", *ORSA Journal on Computing* v1 n3, Summer 1989, pp. 190–206

[7] Glover, F., "Tabu search—part II", *ORSA Journal on Computing* v2 n1, Winter 1990, pp. 4–32

[8] Sinclair, M.C., "Evolutionary telecommunications: A summary", Proc. GECCO'99 Workshop on Evolutionary Telecommunications: Past, Present and Future, Orlando, Florida, USA, July 1999, pp. 209–212

[9] Davis, L. (Ed.), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991

[10] Cahn, R.S., *Wide Area network Design: Concepts and Tools for Optimization*, Morgan Kaufmann, 1998

[11] Kershenbaum, A., *Telecommunications Network Design Algorithms*, MacGraw-Hill, 1993

[12] Kershenbaum, A., *Instructor's Manual to Accompany Telecommunications Network Design Algorithms*, MacGraw-Hill, 1993

[13] Ramaswami, R. & Sivaraja, K.N., *Optical Networks: A Practical Perspective*, Morgan Kaufmann, 1998

[14] Boorstyn, R.R. & Frank, H., "Large-scale network topological optimization", *IEEE Transactions on Communications*, v25 n1, January 1977, pp. 29–47

[15] Booch, G., *Object-Oriented Analysis and Design with Applications (2nd Ed.)*, Benjamin-Cummings, 1994

[16] Stroustrup, B., *The C++ Programming Language (2nd Ed.)*, Addison-Wesley, 1991

[17] Lippman, S.B., *C++ Primer (2nd Ed.)*, Addison-Wesley, 1991

[18] Chbat, M.W., Grard, E., *et al.*, "Towards wide-scale all-optical transparent networking: The ACTS Optical Pan-European Network (OPEN) project", *IEEE Journal on Selected Areas in Communications* v16 n7, September 1998, pp. 1226–1244

[19] Parnis, N., Jones, E.V., Sinclair, M.C. & O'Mahony, M.J., "Hierarchical network design for ACTS OPEN: Initial considerations", Proc. 12th UK Performance Engineering Workshop, Edinburgh, September 1996, pp. 141–155

[20] Proestaki, A. & Sinclair, M.C., "Design and dimensioning of dual-homing hierarchical multi-ring networks", *IEE Proc. Communications* v147 n2 April 2000 pp. 96–104

[21] Proestaki, A. & Sinclair, M.C., "Interconnection strategies for dual-homing multi-ring networks", Proc. 16th International Teletraffic Conference (ITC-16), Edinburgh, UK, June 1999, pp. 169–181

[22] Proestaki, A. & Sinclair, M.C., "Wavelength routing in all-optical dual-homing hierarchical multi-ring networks", Proc. European Conference on Networks and Optical Communications — Core Networks and Network Management (NOC'99), Delft, The Netherlands, June 1999, pp. 52–59

[23] Proestaki, A. & Sinclair, M.C., "Impact of topology on wavelength and switch-port requirements in all-optical hierarchical multi-ring networks", Proc. IEEE Global Telecom. Conf. (GLOBECOM'99), Rio de Janeiro, Brazil, December, 1999

[24] Proestaki, A. & Sinclair, M.C., “Dimensioning all-optical dual-homing hierarchical multi-ring networks with various connectivities”, Proc. Second Intl. Workshop on the Design of Reliable Communication Networks (DRCN2000), Munich, Germany, April 2000

[25] Ahuja, R.K., Magnanti, T.L. & Orlin, J.B., *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, 1993

[26] Schwefel, H.-P., “On natural life’s tricks to survive and evolve”, Plenary Session, IEEE Intl. Conf. on Evolutionary Computation (ICEC’98), Anchorage, Alaska, USA, May 1998

[27] Sinclair, M.C., “The application of a genetic algorithm to trunk network routing table optimisation”, Proc. 10th UK Teletraffic Symposium, Martlesham Heath, UK, April 1993, pp. 2/1–2/6

[28] Sinclair, M.C., “Trunk network fault-tolerance through routing table updating in the event of trunk failure”, Proc. 9th UK Teletraffic Symposium, Guildford, April 1992, pp. 17/1–17/9

[29] Tan, L.G. & Sinclair, M.C., “Wavelength assignment between the central nodes of the COST 239 European optical network”, Proc. 11th UK Performance Engineering Workshop, Liverpool, UK, September 1995, pp. 235–247

[30] Moore, S.J. & Sinclair, M.C., “Design of routing tables for a survivable military communications network using genetic algorithms”, Proc. Congress on Evolutionary Computation (CEC’99), Washington DC, USA, July 1999, pp. 1788–1795

[31] Sinclair, M.C., “NOMaD: Initial architecture of an optical network optimisation, modelling and design tool”, Proc. 12th UK Performance Engineering Workshop, Edinburgh, UK, September 1996, pp. 157–167

[32] Sinclair, M.C., “NOMaD: An optical network optimisation, modelling and design toolset”, Proc. IEE Colloquium on Multiwavelength Optical Networks: Devices, Systems and Network Implementations, Digest No. 1998/296, London, UK, June 1998, pp. 6/1–6/6

[33] Sinclair, M.C., “Operator-probability adaptation in a genetic-algorithm/heuristic hybrid for optical network wavelength allocation”, Proc. IEEE Intl. Conf. on Evolutionary Computation (ICEC’98), Anchorage, Alaska, USA, May 1998, pp. 840–845

[34] Sinclair, M.C., "NOMaD: Applying a genetic-algorithm/heuristic hybrid approach to optical network topology design", Proc. 3rd Intl. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97), University of East Anglia, Norwich, UK, April 1997, pp. 299–303

[35] Sinclair, M.C., "Minimum network wavelength requirement design using a genetic-algorithm/heuristic hybrid", *Electronics Letters* v34 n4 February 1998 pp. 388–389

[36] Sinclair, M.C., "Minimum cost routing and wavelength allocation using a genetic-algorithm/heuristic hybrid approach", Proc. 6th IEE Conf. on Telecommunications, Edinburgh, UK, March/April 1998, pp. 67–71

[37] Sinclair, M.C., "Minimum cost wavelength-path routing and wavelength allocation using a genetic-algorithm/heuristic hybrid approach", *IEE Proc. Communications* v146 n1 February 1999 pp. 1–7

[38] Bäck, T., Hammel, U. & Schwefel, H.-P., "Evolutionary computation: Comments on the history and current state", *IEEE Transactions on Evolutionary Computation* v1 n1, April 1997, pp. 3–17 (reprinted in [4])

[39] Friedman, G.J., *Selective Feedback Computers for Engineering Synthesis and Nervous System Analogy*, master's thesis, University of California, Los Angeles, USA, February 1956 (reprinted in [4])

[40] Fraser, A.S., "Simulation of genetic systems by automatic digital computers: I. Introduction", *Australian J. Biological Sciences* v10, 1957, pp.484–491 (reprinted in [4])

[41] Box, G.E.P., "Evolutionary operation: A method for increasing industrial productivity", *Applied Statistics* v6 n2, 1957, pp. 81–101

[42] Friedberg, R.M., "A learning machine: Part I", *IBM J. Research and Development* v2 n1, 1958, pp. 2–13

[43] Friedberg, R.M., Dunham, B. & North, J.H., "A learning machine: Part II", *IBM J. Research and Development* v3, 1959, pp. 282-287

[44] Holland, J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975

[45] Holland, J.H., *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence (2nd Ed.)*, MIT Press, 1992

[46] Heitkötter, J. & Beasley, D., *The Hitch-Hiker's Guide to Evolutionary Computation, Issue 8.1*, March 2000 (`ftp://rtfm.mit.edu/pub/usenet/news.answers/ai-faq/genetic/`)

[47] Fogel, D.B., personal communication, 1998

[48] Fogel, L.J., Owens, A.J. & Walsh, M.J., "Artificial intelligence through simulated evolution", Proc. 2nd Cybernetic Sciences Symp., 1965, pp. 131–155 (reprinted in [4])

[49] Fogel, L.J., Owens, A.J. & Walsh, M.J., *Artificial Intelligence through Simulated Evolution* Wiley, 1966

[50] Fogel, L.J., *Intelligence through Simulated Evolution*, Wiley, 1999

[51] Rechenberg, I., "Cybernetic solution path of an experimental problem", Royal Aircraft Establishment, Library Translation 1122, 1965 (reprinted in [4])

[52] Schwefel, H.-P., *Evolution and Optimum Seeking*, Wiley, 1995

[53] Koza, J.R., "Hierarchical genetic algorithms operating on populations of computer programs", Proc. 11th Intl. Joint Conf. on Artificial Intelligence, 1989, pp. 768–774 (reprinted in [4])

[54] Koza, J.R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992

[55] Koza, J.R., *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, 1994

[56] Koza, J.R., Bennett, F.H., III, Andre, D. & Keane, M.A., *Genetic Programming III: Darwinian Invention and Problem Solving*, Morgan Kaufmann Publishers, 1999

[57] Kinnear, K.E., Jr. (Ed), *Advances in Genetic Programming*, MIT Press, 1994

[58] Angeline, P.J. & Kinnear, K.E., Jr. (Eds), *Advances in Genetic Programming, Volume II*, MIT Press, 1996

[59] Spector, L., Langdon, W.B., O'Reilly, U.-M. & Angeline, P.J. (Eds), *Advances in Genetic Programming, Volume III*, MIT Press, 1999

[60] Banzhaf, W., Nordin, P., Keller, R.E. & Francone, F.D., *Genetic Programming — An Introduction*, Morgan Kaufmann Publishers, 1998

[61] Nordin, P., *Evolutionary Program Induction of Binary Machine Code and its Application*, Krehl Verlag, 1997

[62] Holland, J.H. & Reitman, J.S., "Cognitive systems based on adaptive algorithms" in Waterman, D.A. & Hayes-Roth, F. (Eds), *Pattern-Directed Inference Systems*, Academic Press, 1978, pp. 313–329 (reprinted in [4])

[63] Sipper, M., *Evolution of Parallel Cellular Machines*, Lecture Notes in Computer Science, v1194, Springer-Verlag, 1997

[64] Thompson, A., *Hardware Evolution: Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution*, Springer-Verlag, 1998

[65] Mange, D. & Tomassini, M. (Eds), *Bio-Inspired Computing Machines*, Presses Polytechniques et Universitaries Romandes, 1998

[66] Koza, J.R., "59 authored and 30 edited books on evolutionary computation", January 2000 (`http://www.genetic-programming.org/books.html`)

[67] Conrad, M., *Adaptability*, Plenum, 1983

[68] Holland, J.H, Holyoak, K.J., Nisbett, R.E. & Thagard, P.A., Induction: Processes of Inference, Learning, and Discovery, MIT Press, 1986

[69] Davis, L. (Ed.), *Genetic Algorithms and Simulated Annealing*, Pittman, l987

[70] Voigt, H.-M., *Evolution and Optimization: An Introduction to Solving Complex Problems by Replicator Networks*, Akademie-Verlag, 1989

[71] Davidor, Y. *Genetic Algorithms and Robotics: A Heuristic Strategy for Optimization*, World Scientific, 1990

[72] Fogel, D.B., *System Identification through Simulated Evolution*, Ginn Press, 1991

[73] Forrest, S., *Parallelism and Programming in Classifier Systems*, Pittman, 1991

[74] Buckles, B.P. & Petry, F.E., *Genetic Algorithms*, IEEE, 1992

[75] Stender, J. (Ed.), *Parallel Genetic Algorithms*, IOS Publishing, 1993

[76] Bauer, R.J., Jr., *Genetic Algorithms and Investment Strategies*, Wiley, 1994

[77] Bhanu, B. & Lee, S., *Genetic Learning for Adaptive Image Segmentation*, Kluwer Academic Publishers, 1994

[78] Stender, J., Hillebrand, E. & Kingdon, J. (Eds), *Genetic Algorithms in Optimization, Simulation, and Modeling*, IOS Publishing, 1994

[79] Banzhaf, W. & Eeckman, F.H. (Eds), *Evolution and Biocomputation: Computational Models of Evolution*, Lecture Notes in Computer Science, v899, Springer-Verlag, 1995

[80] Biethahn, J. & Nissen, V. (Eds), *Evolutionary Algorithms in Management Applications*, Springer-Verlag, 1995

[81] Chambers, L. (Ed.), *Practical Handbook of Genetic Algorithms: Applications: Volume I*, CRC Press, 1995

[82] Chambers, L. (Ed.), *Practical Handbook of Genetic Algorithms: New Frontiers: Volume II*, CRC Press, 1995

[83] Bäck, T., *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, 1996

[84] Dawid, H., *Adaptive Learning by Genetic Algorithms: Analytical Results and Applications to Economic Models*, Lecture Notes in Economics and Mathematical Systems, v441, Springer-Verlag, 1996

[85] Devillers, J. (Ed.), Genetic Algorithms in Molecular Modeling, Academic Press, 1996

[86] Herrera, F. & Verdegay, J.L. (Eds), *Genetic Algorithms and Soft Computing*, Physica Verlag, 1996

[87] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.)*, Springer-Verlag, 1996

[88] Mattfield, D.C., *Evolutionary Search and the Job Shop*, Physica-Verlag, 1996

[89] Pal, S.K. & Wang, P.P., *Genetic Algorithms and Pattern Recognition*, CRC Press, 1996

[90] van Rooij, A.J.F., Jain, L.C. & Johnson, R.P., *Neural Network Training Using Genetic Algorithms*, World Scientific, 1996

[91] Babovic, V., *Emergence, Evolution, Intelligence: Hydroinformatics*, Balkema Publishers, 1996

[92] Bäck, T., Fogel, D.B. & Michalewicz, Z. (Eds), *Handbook of Evolutionary Computation*, Institute of Physics Publishing, 1997

[93] Dasgupta, D. & Michalewicz, Z. (Eds), *Evolutionary Algorithms in Engineering Applications*, Springer-Verlag, 1997

[94] Pedrycz, W., *Fuzzy Evolutionary Computation*, Kluwer, 1997

[95] Rudolph, G., *Convergence Properties of Evolutionary Algorithms*, Verlag Kovac, 1997

[96] Vonk, E., Jain, L.C. & Johnson, R.P., *Automatic Generation of Neural Network Architecture Using Evolutionary Computation*, World Scientific, 1997

[97] Dracopoulos, D.C., *Evolutionary Learning Algorithms for Neural Adaptive Control*, Springer Verlag, l997

[98] Falkenauer, E., *Genetic Algorithms and Grouping Problems*, Wiley, 1997

[99] Gen, M. & Cheng, R., *Genetic Algorithms and Engineering Design*, Wiley, 1997

[100] Geyer-Schulz, A., *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning (2nd Rev. Ed.)*, Physica-Verlag, 1997

[101] Man, K.F., Tang, K.S., Kwong, S. & Halang, W.A., *Genetic Algorithms for Control and Signal Processing*, Springer-Verlag, 1997

[102] Sanchez, E., Shibata, T. & Zadeh, L.A. (Eds), *Genetic Algorithms and Fuzzy Logic: Soft Computing Perspectives*, World Scientific, 1997

[103] Zalzala, A.M.S. & Fleming, P.J. (Eds), *Genetic Algorithms in Engineering Systems*, IEE, 1997

[104] Blickle, T., *Theory of Evolutionary Algorithms and Application to System Synthesis*, vdf Hochschul Verlag AG, 1997

[105] Drechsler, R., *Evolutionary Algorithms for VLSI CAD*, Kluwer Academic Publishers, 1998

[106] Gill, M.A.C. & Zomaya, A.Y., *Obstacle Avoidance in Multi-Robot Systems: Experiments in Parallel Genetic Algorithms*, World Scientific, 1998

[107] Haupt, R.L. & Haupt, S.E., *Practical Genetic Algorithms*, Wiley, 1998

[108] Langdon, W.B., *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*, Kluwer Academic Publishers, 1998

[109] Lai, L.L., *Intelligent System Applications in Power Engineering: Evolutionary Programming and Neural Networks*, Wiley, 1998

[110] Dorigo, M. & Colombetti, M., *Robot Shaping: An Experiment in Behavior Engineering*, MIT Press, 1998

[111] Bentley, P.J. (Ed.), *Evolutionary Design by Computers* Wiley, 1999

[112] , Eiben, A.E. & Michalewicz, Z. (Eds), *Evolutionary Computation* IOS Press, 1999

[113] Fogel, D.B., *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence (2nd Ed.)*, IEEE, 1999

[114] Chambers, Lance (Ed.), *Practical Handbook of Genetic Algorithms: Complex Coding Systems: Volume III*, CRC Press, 1999

[115] Coley, D.A., *An Introduction to Genetic Algorithms for Scientists and Engineers*, World Scientific, 1999

[116] Dawid, H., *Adaptive Learning by Genetic Algorithms: Analytical Results and Applications to Economic Models (2nd Rev. Ed.)*, Springer Verlag, 1999

[117] Karr, C.L. & Freeman, K.M. (Eds), *Industrial Applications of Genetic Algorithms*, CRC Press, 1999

[118] Man, K.F., Tang, K.S., Kwong, S. & Halang, W.A., *Genetic Algorithms: Concepts and Designs*, Springer-Verlag, 1999

[119] Mazumder, P. & Rudnick, E.M. (Eds), *Genetic Algorithms for VLSI Design, Layout and Test Automation*, Prentice Hall, 1999

[120] Rahmat-Samii, Y. & Michielssen, E. (Eds), *Electromagnetic Optimization by Genetic Algorithms*, Wiley, 1999

[121] Vose, M., *The Simple Genetic Algorithm: Foundations and Theory*, MIT Press, 1999

[122] Ryan, C., *Automatic Re-engineering of Software Using Genetic Programming*, Kluwer Academic Publishers, 1999

[123] Wong, M.L. & Leung, K.S., *Data Mining Using Grammar Based Genetic Programming and Applications*, Kluwer Academic Publishers, 2000

[124] Wasserman, P.D., *Advanced Methods in Neural Computing*, Van Nostrand Reinhold, 1993

[125] Proestaki, A. & Sinclair, M.C., "Initial survey of heuristics for optical core network design in ACTS WOTAN", Proc. 13th UK Teletraffic Symposium, Glasgow, March 1996, pp. 20/1–20/8

[126] Smith, G.D., *ECTELNET — Application of Evolutionary Algorithms in the Telecommunications Domain: A Preliminary Report*, The ECTELNET Working Group, 1998

[127] Smith, G.D., Debuse, J.C.W. & Ryan, M., *The Fixed Channel Assignment Problem*, School of Information Systems, University of East Anglia, 1998

[128] Dorigo, M., Maniezzo, V. & Colorni, A., "The ant system: optimization by a colony of cooperating agents", *IEEE Transactions on Systems, Man and Cybernetics — Part B*, v26 n1, 1996, pp. 1–13

[129] Schoonderwoerd, R., Holland, O.E., Brutten, J.L. & Rothkrantz, L.J.M., "Ant-based load balancing in telecommunication networks", *Adaptive Behaviour*, v5 n2, 1997, pp. 169–207

[130] Di Caro, G. & Dorigo, M., "Mobile agents for adaptive routing", Proc. 31st Hawaii Intl. Conf. Systems Sciences (HICSS-31), Kohala Coast, Hawaii, January 1998, pp. 74–83

[131] Balestrino, A., Bini Verona, F., Caiti, A. & Landi, R., "A hybrid genetic algorithm for binary channel equalization", Proc. 1st Intl. ICSC Symp. on Intelligent Industrial Automation and Soft Computing (IIA'96/SOCO'96), University of Reading, UK, March 1996, pp. B 166–B 170

[132] Comellas, F. & Roca, R., "Using genetic algorithms to design weight codes", Proc. Intl. Workshop on Applications of Neural Networks to Telecommunications (IWANNT'93), Princetown, NJ, USA, 1993, pp. 119–124

[133] Lee, Y.M. & Ellis, J.H., "Comparison of algorithms for nonlinear integer optimization: application to monitoring network design", *Journal of Environmental Engineering* v122 n6, 1996, pp. 524–531

[134] Ramachandran, V., Shanmuganathan, V.P., Jayabarathi, T., "Optimal load balancing in tree networks using genetic algorithms", Proc. 4th World Congress of Expert Systems: Application of Advanced Information Technologies, Mexico City, Mexico, March 1998, Ch. 139, pp. 893–897

[135] Sannier, A.V., II & Goodman, E.D., "Midgard: A genetic approach to adaptive load balancing for distributed systems", Proc. 5th Intl. Conf. on Machine Learning (ML-88), Ann Arbor, Michegan, USA, June 1988, pp. 174–180

[136] Rho, S. & March, S.T., "A nested genetic algorithm for distributed database design", Proc. 27th Annual Hawaii Intl. Conf. on System Sciences, Hawaii, USA, 1994, pp. 33–42

[137] Davis, L. & Coombs, S., "Optimizing network link sizes with genetic algorithms" in Elzas, M.S., Oren, T.I. & Zeigler, B.P. (Eds), *Modelling and Simulation Methodology*, Elsevier Science Publishers B.V. (North-Holland), 1989, Ch. IV.3, pp. 317–331

[138] Davis, L. & Coombs, S., "Genetic algorithms and communication link speed design: Theoretical considerations", Proc. 2nd Intl. Conf. on Genetic Algorithms (ICGA'87), Cambridge, MA, USA, July 1987, pp. 252–256

[139] Coombs, S. & Davis, L., "Genetic algorithms and communication link speed design: Constraints and operators", Proc. 2nd Intl. Conf. on Genetic Algorithms (ICGA'87), Cambridge, MA, USA, July 1987, pp. 257–260

[140] Cox, L.A., Jr., Davis, L. & Qiu, Y., "Dynamic anticipatory routing in circuit-switched telecommunications networks", in [9], Ch. 11, pp. 124–143

[141] Pan, H. & Wang, I.Y., "The bandwidth allocation of ATM through genetic algorithms", IEEE Global Telecom. Conf. (GLOBECOM'91), v1, 1991, pp. 125–129

[142] Potter, W.D., Pitts, R., Gillis, P., Young, J. & Caramadre, J., "IDA-NET: An intelligent decision aid for battlefield communications network configuration", Proc. 8th Conf. Artificial Intelligence for Applications (CAIA'92), Monterey, California, USA, March 1992, pp. 247–253

[143] Michalewicz, Z., "A step towards optimal topology of communications networks", Proc. Conf. on Data Structures and Target Classification, Orlando, Florida, USA, April 1991, pp. 112–122

[144] Hesser, J., Männer, R. & Stucky, O., "Optimization of Steiner trees using genetic algorithms", Proc. 3rd Intl. Conf. on Genetic Algorithms (ICGA'89), George Mason University, Arlington, Virginia, USA, June 1989, pp. 231–236

[145] Kumar, A., Pathak, R.M., Gupta, M.C. & Gupta, Y.P., "Genetic algorithm based approach for designing computer network topology", Proc. 21st ACM Annual Comput. Sci. Conf. Indianapolis, USA, 1992, pp. 358–365

[146] Sobotka, M., "Network design: An application of genetic algorithms", Proc. 1992 Applied Defense Simulation Conference, 1992, pp. 63–66

[147] Aiyarak, P., Saket, A.S. & Sinclair, M.C., "Genetic programming approaches for minimum cost topology optimisation of optical telecommunication networks", Proc. 2nd IEE/IEEE Intl. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'97), Glasgow, September 1997, pp. 415–420

[148] Deeter, D.L. & Smith, A.E., "Heuristic optimization of network design considering all-terminal reliability", Proc. Annual Reliability and Maintainability Symposium, Philadelphia, Pennsylvania, USA, 1997, pp. 194–199

[149] He, C., "Route selection and capacity assignment in computer communication networks based on genetic algorithm", Proc. 1997 IEEE Intl. Conf. on Intelligent Processing Systems, Beijing, China, October 1997, pp. 548–552

[150] Hewitt, J., Soper, A. & McKenzie, S., "CHARLEY: A genetic algorithm for the design of mesh networks", Proc. 1st IEE/IEEE Intl. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'95), University of Sheffield, UK, September 1995, pp. 118–122

[151] Huang, R., Ma, J. & Tsuboi, E., "Communication network design via a genetic algorithm based learning algorithm", Proc. IASTED Intl. Conf. on Artificial Intelligence, Expert Systems, and Neural Networks, Honolulu, Hawaii, USA, August 1996, pp. 15–18

[152] Huang, R. & Ma, J., "A distributed genetic algorithm over a transputer based parallel machine for survivable network designs", Proc. Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'96), Sunnyvale, California, USA, August 1996, pp. 1202–1211

[153] Huang, R., Ma, J., Kunii, T.L. & Tsuboi, E., "Parallel genetic algorithms for communication network design", Proc. Second Aizu Intl. Symposium on Parallel Algorithms/Architecture Synthesis, 1997, pp. 370–377

[154] Ko, K.-T., Tang, K.-S., Chan, C.-Y., Man, K.-F. & Kwong, S., "Using genetic algorithms to design mesh networks", *Computer* v30 n8, August 1997, pp. 56–61

[155] Kozdrowski, S., Pióro, M., Arabas, J. & Szcześniak, M., "Robust design of multicommodity integral flow networks", Proc. 7th Intl. Conf. on Genetic Algorithms (ICGA'97), July 1997, Michigan State University, East Lansing, Michigan, USA, pp. 607–614

[156] Mann, J.W. & Smith, G.D., "The ring-loading and ring-sizing problem", Proc. 3rd Intl. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97), University of East Anglia, Norwich, UK, April 1997, pp. 289–293

[157] Mikac, B. & Inkret, R., "Application of a genetic algorithm to the availability-cost optimization of a transmission network topology", Proc. 3rd Intl. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97), University of East Anglia, Norwich, UK, April 1997, pp. 304–307

[158] Munakata, T. & Hashier, D.J., "A genetic algorithm applied to the maximum flow problem", Proc. 5th Intl. Conf. on Genetic Algorithms (ICGA'93), University of Illinois at Urbana-Champaign, USA, July 1993, pp. 488–493

[159] Palmer, C.C. & Kershenbaum, A., "Representing trees in genetic algorithms", Proc. IEEE Intl. Conf. on Evolutionary Computation (ICEC'94), Orlando, Florida, USA, June 1994, pp. 379–384

[160] Palmer, C.C. & Kershenbaum, A., "An approach to a problem in network design using genetic algorithms", *Networks* v26 n3, 1995, pp. 151–163

[161] Paul, H & Tindle, J., "Passive optical network planning in local access networks — an optimisation approach utilising genetic algorithms", *BT Technology Journal* v14 n2, April 1996, pp. 110–115

[162] Paul, H., Tindle, J. & Ryan, H.M., "Experiences with a genetic algorithm-based optimization system for passive optical network planning in the local access network", Proc. Broadband Superhighway (NOC'96-I), Heidelberg, Germany, June 1996, Ch. 55, pp. 105–112

[163] Pierre, S. & Legault, G., "An evolutionary approach for configuring economical packet-switched computer networks", *Artificial Intelligence in Engineering* v10 n2, 1996, pp. 127–134

[164] Pierre, S.& Legault, G., "A genetic algorithm for designing distributed computer network topologies", *IEEE Transactions on Systems, Man and Cybernetics—Part B: Cybernetics* v28 n2, April 1998, pp. 249–258

[165] Qin, Z., Wu, F.F. & Law, N., "Designing B-ISDN network topologies using the genetic algorithm", Proc. Fifth Intl. Symp. on Modelling, Analysis & Simulation of Computer & Telecommunication Systems (MASCOTS'97), Ch. 36, 1997, pp. 140–145

[166] Sinclair, M.C., "Minimum cost topology optimisation of the COST 239 European optical network", Proc. 2nd Intl. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA'95), Alès, France, April 1995, pp. 26–29

[167] Tanaka, Y. & Berlage, O., "Application of genetic algorithms to VOD network topology optimization", *IEICE Transactions on Communications* vE79-B n8, Aug 1996, pp. 1046–1053

[168] Walters, G.A. & Smith, D.K., "Evolutionary design algorithm for optimal layout of tree networks", *Engineering Optimization* v24, 1995, pp. 261–281

[169] Chang, F.L., Potter, W.D., Gillis, P. & Sanders, M., "Genetic algorithms in battlefield communications network configuration", Proc. ACM 31st Annual Southeast Conference, Birmingham, Alabama, USA, April 1993, pp. 96–102

[170] Potter, W.D., Chang, F.L., Gillis, P. & Sanders, M., "An ITS expert system module to aid battlefield communications network configuration", Proc. 6th Intl. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE'93), Edinburgh, UK, June 1993, pp. 165–174

[171] Chang, T.F. & Potter, W.D., "A genetic algorithm approach to solving the battlefield communication network configuration problem", Proc. 3rd Golden West Intl. Conf. on Intelligent Systems (1994 GWIC), Las Vegas, Nevada, USA, June 1994, pp. 483–496

[172] Davis, L., Orvosh, D., Cox, A. & Qiu, Y., "A genetic algorithm for survivable network design", Proc. 5th Intl. Conf. on Genetic Algorithms (ICGA'93), University of Illinois at Urbana-Champaign, USA, July 1993, pp. 408–415

[173] Garcia, B.-L., Mahey, P., LeBlanc, L.J., "Iterative improvement methods for a multiperiod network design problem", *European Journal of Operational Research* v110 n1, 1998, pp. 150–165

[174] Brittain, D., Williams, J.S. & McMahon, C., "A genetic algorithm approach to planning the telecommunications access network", Proc. 7th Intl. Conf. on Genetic Algorithms (ICGA'97), July 1997, Michigan State University, East Lansing, Michigan, USA, pp. 623–628

[175] Routen, T., "Genetic algorithm and neural network approaches to local access network design", Proc. 2nd Intl. Symp. on Modelling, Analysis & Simulation of Computer & Telecommunication Systems (MASCOTS'94), Durham, N. Carolina, USA, 1994, pp. 239–243

[176] Kapsalis, A., Rayward-Smith, V.J. & Smith, G.D., "Solving the graphical Steiner tree problem using genetic algorithms", *Journal of the Operational Research Society* v44 n4, April 1993, pp. 397–406

[177] Kapsalis, A., Rayward-Smith, V.J. & Smith, G.D., "Using genetic algorithms to solve the radio link frequency assignment problem", Proc. 2nd Intl. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA'95), Alès, France, April 1995, pp. 37–40

[178] Chardaire, P., Kapsalis, A., Mann, J.W., Rayward-Smith, V.J. & Smith, G.D., "Applications of genetic algorithms in telecommunications", Proc. 2nd Intl. Workshop on Applications of Neural Networks to Telecommunications (IWANNT'95), Stockholm, Sweden, May 1995, pp. 290–299

[179] Celli, G., Costamagna, E. & Fanni, A., "Genetic algorithms for telecommunication network optimization", Proc. IEEE Intl. Conf. on Systems, Man and Cybernetics, 1995, v2, pp. 1227–1232

[180] Kumar, A., Pathak, R.M & Gupta, Y.P., "Genetic-algorithm-based reliability optimization for computer network expansion", *IEEE Transactions on Reliability* v44 n1, March 1995, pp. 63–72

[181] Kumar, A., Pathak, R.M., Gupta, Y.P. & Parsaei, H.R., "A genetic algorithm for distributed system topology design", *Computers & Industrial Engineering* v28 n3, 1995, pp. 659–670

[182] Srivastava, A., Kumar, A. & Pathak, R., "Generalized distributed genetic algorithm for optimization problems", *Integrated Computer-aided Engineering* v4 n4, 1997, pp. 276–289

[183] Webb, A., Turton, B.C.H. & Brown, J.M., "Application of a genetic algorithm to the design and optimisation of a self-healing ring network", Proc. 4th Communication Networks Symposium, Manchester, UK, July 1997, pp. 2–5

[184] Webb, A., Turton, B.C.H. & Brown, J.M., "Application of genetic algorithm to a network optimisation problem", Proc. 6th IEE Conf. on Telecommunications, Edinburgh, UK, March/April 1998, pp. 62–66

[185] Gondim, P.R.L., "Genetic algorithms and the location area partitioning problem in cellular networks", Proc. IEEE 46th Vehicular Technology Conf., Atlanta, Georgia, USA, April/May 1996, v3, pp. 1835–1838

[186] Calégari, P., Guidec, F., Kuonen, P. & Kobler, D., "Parallel island-based genetic algorithm for radio network design", *Journal of Parallel and Distributed Computing* v47 n1, November 1997, pp. 86–90

[187] Tang, K.S., Man, K.F. & Ko, K.T., "Wireless LAN design using hierarchical genetic algorithm", Proc. 7th Intl. Conf. on Genetic Algorithms (ICGA'97), July 1997, Michigan State University, East Lansing, Michigan, USA, pp. 629–635

[188] Dengiz, B., Altiparmak, F. & Smith, A.E., "A genetic algorithm approach to optimal topological design of all terminal networks", *Intelligent Engineering Systems Through Artificial Neural Network* v5, 1995, pp. 405–410

[189] Dengiz, B., Altiparmak, F. & Smith, A.E., "Genetic algorithm design of networks considering all-terminal reliability", Proc. 6th Industrial Engineering Research Conference (IERC6), Miami Beach, Florida, USA, May 1997, Ch. 168, pp. 30–35

[190] Dengiz, B., Altiparmak, F. & Smith, A.E., "Efficient optimization of all-terminal reliable networks, using an evolutionary approach", *IEEE Transactions on Reliability* v46 n1, March 1997, pp. 18–26

[191] Dengiz, B., Altiparmak, F. & Smith, A.E., "Local search genetic algorithm for optimization of highly reliable communications networks", Proc. 7th Intl. Conf. on Genetic Algorithms (ICGA'97), July 1997, Michigan State University, East Lansing, Michigan, USA, pp. 650–657

[192] Dengiz, B., Altiparmak, F. & Smith, A.E., "Local search genetic algorithm for optimal design of reliable networks", *IEEE Transactions on Evolutionary Computation* v1 n3, September 1997, pp. 179–188

[193] Abuali, F.N., Schoenefeld, D.A. & Wainwright, R.L., "Terminal assignment in a communications network using genetic algorithms", Proc. 22nd Annual ACM Computer Science Conf. (CSC'94), March 1994, Phoenix, Arizona, USA, pp. 74–81

[194] AbuAli, F.N., Wainwright, R.L. & Schoenefeld, D.A., "Solving the subset interconnection design problem using genetic algorithms", Proc. 1996 ACM Symp. on Applied Computing (SAC'96), Philadelphia, PA, USA, February 1996, pp. 299–304

[195] Saha, D. & Chakraborty, U.K., "An efficient link enhancement strategy for computer networks using genetic algorithm", *Computer Communications* v20 n9, 1997, pp. 798–803

[196] Nakamura, H. & Oda, T., "Optimization of facility planning and circuit routing for survivable transport networks—an approach based on genetic algorithm and incremental assignment", *IEICE Transactions on Communications* vE80-B n2, February 1997, pp. 240–251

[197] Christensen, H.L., Wainwright, R.L. & Schoenefeld, D.A.,"A hybrid algorithm for the point to multipoint routing problem", Proc. 1997 ACM Symp. on Applied Computing (SAC'97), February/March 1997, San Jose, California, USA, pp. 263–268

[198] Zhu, L., Wainwright, R.L. & Schoenefeld, D.A., "A genetic algorithm for the point to multipoint routing problem with varying number of requests", Proc. 1998 IEEE Intl. Conf. on Evolutionary Computation (ICEC'98), Anchorage, Alaska, USA, May 1998, pp. 171–176

[199] Julstrom, B.A., "A genetic algorithm for the rectilinear Steiner problem", Proc. 5th Intl. Conf. on Genetic Algorithms (ICGA'93), University of Illinois at Urbana-Champaign, USA, July 1993, pp. 474–480

[200] Esbensen, H. & Mazumder, P., "A genetic algorithm for the Steiner problem in a graph", European Conf. on Design Automation (EDAC'94), Paris, France, February 1994, pp. 402–406

[201] Esbensen, H., "Computing near-optimal solutions to the Steiner problem in a graph using a genetic algorithm", *Networks* v26 n4, 1995, pp. 173–185

[202] Esbensen, H., "Finding (near-)optimal Steiner trees in large graphs", Proc. 6th Intl. Conf. on Genetic Algorithms (ICGA'95), University of Pittsburgh, USA, July 1995, pp. 485–491

[203] Abuali, F.N., Schoenefeld, D.A. & Wainwright, R.L., "The design of a multipoint line topology for a communication network using genetic algorithms", Proc. 7th Oklahoma Conf. on Artificial Intelligence, Stillwater, Oklahoma, November 1993, pp. 101–110

[204] Abuali, F.N., Schoenefeld, D.A. & Wainwright, R.L., "Designing telecommunications networks using genetic algorithms and probabilistic minimum spanning trees", Proc. 1994 ACM Symp. Applied Computing (SAC'94), Phoenix, Arizona, USA, March 1994, pp. 242–246

[205] Abuali, F.N., Wainwright, R.L. & Schoenefeld, D.A., "Determinant factorization and cycle basis: Encoding schemes for the representation of spanning trees on incomplete graphs", Proc. 1995 ACM Symp. on Applied Computing, Nashville, TN, USA, February 1995, pp. 305–312

[206] Abuali, F.N., Wainwright, R.L. & Schoenefeld, D.A., "Solving the three-star tree isomorphism problem using genetic algorithms", Proc. 1995 ACM Symp. on Applied Computing (SAC'95), Nashville, TN, USA, February 1995, pp. 337–343

[207] Abuali, F.N., Wainwright, R.L. & Schoenefeld, D.A., "Determinant factorization: A new encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem", Proc. 6th Intl. Conf. on Genetic Algorithms (ICGA'95), University of Pittsburgh, USA, July 1995, pp. 470–477

[208] Zhao, J., McKenzie, S. & Soper, A.J., "A genetic algorithm for the design of multipoint connections in a local access network", Proc. 1st IEE/IEEE Intl. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'95), University of Sheffield, UK, September 1995, pp. 30–33

[209] Berry, L., Murtagh, B., Sugden, S. & McMahon, G., "Application of a genetic-based algorithm for optimal design of tree-structured communication networks", Proc. Regional Teletraffic Engineering Conference of the International Teletraffic Congress, Pretoria, South Africa, September 1995, pp. 361–370

[210] Elbaum, R. & Sidi, M., "Topological design of local area networks using genetic algorithms", Proc. IEEE INFOCOM'95, Boston, Massachusettes, USA, April 1995, v1, pp. 64–71

[211] Elbaum, R. & Sidi, M., "Topological design of local-area networks using genetic algorithms", *IEEE/ACM Transactions on Networking* v4 n5, October 1996, pp. 766–778

[212] Shimamoto, N., Hiramatsu, A. & Yamasaki, K., "A dynamic routing control based on a genetic algorithm", Proc. IEEE Intl. Conf. on Neural Networks (ICNN'93), San Francisco, California, USA, March/April 1993, v2, pp. 1123–1128

[213] Mann, J.W., Rayward-Smith, V.J. & Smith, G.D., "Telecommunication traffic routing: A case study in the use of genetic algorithms", Proc. Applied Decision Technologies (ADT'95), London, April 1995, pp. 315–325

[214] Mann, J.W. & Smith, G.D., "A comparison of heuristics for telecommunication traffic routing", in Rayward-Smith, V.J., Osman, I.H., Reeves, C.R. & Smith, G.D. (Eds), *Modern Heuristic Search Methods*, Wiley, 1996, Ch. 14, pp. 235–254

[215] Hoelting, C.J., Schoenefeld, D.A. & Wainwright, R.L., "Finding investigator tours in telecommunication networks using genetic algorithms", Proc. 1996 ACM Symp. on Applied Computing (SAC'96), Philadelphia, PA, USA, February 1996, pp. 82–87

[216] Hoelting, C.J., Schoenefeld, D.A. & Wainwright, R.L., "A genetic algorithm for the minimum broadcast time problem using a global precendence vector", Proc. 1996 ACM Symp. on Applied Computing (SAC'96), Philadelphia, PA, USA, February 1996, pp. 258–262

[217] Servet, I., Travé-Massuyès, L. & Stern, D., "Evolutionary computation techniques for telephone networks traffic supervision based on a qualitative stream propogation model", Proc. 3rd Intl. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97), University of East Anglia, Norwich, UK, April 1997, pp. 294–298

[218] Bentall, M., Turton, B.C.H. & Hobbs, C.W.L., "Benchmarking the restoration of heavily loaded networks using a two dimensional order-based genetic algorithm", Proc. 2nd IEE/IEEE Intl. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'97), Glasgow, September 1997, pp. 151–156

[219] Kirkwood, I.M.A., Shami, S.H. & Sinclair, M.C., "Discovering simple fault-tolerant routing rules using genetic programming", Proc. 3rd Intl. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97), University of East Anglia, Norwich, UK, April 1997, pp. 285–288

[220] Shami, S.H., Kirkwood, I.M.A., & Sinclair, M.C., "Evolving simple fault-tolerant routing rules using genetic programming", *Electronics Letters* v33 n17 August 1997 pp. 1440–1441

[221] Carse, B., Fogarty, T.C. & Munro, A., "Adaptive distributed routing using evolutionary fuzzy control", Proc. 6th Intl. Conf. on Genetic Algorithms (ICGA'95), University of Pittsburgh, USA, July 1995, pp. 389–396

[222] Fogarty, T.C., Bull, L. & Carse, B., "Evolving multi-agent systems", in Winter, G., Periaux, J., Galán, M. & Cuesta, P. (Eds), *Genetic Algorithms in Engineering and Computer Science*, Wiley, 1995, Ch. 1, pp. 3–22

[223] Munetomo, M., Takai, Y. & Sato, Y., "An adaptive network routing algorithm employing genetic operators", Proc. 7th Intl. Conf. on Genetic Algorithms (ICGA'97), July 1997, Michigan State University, East Lansing, Michigan, USA, pp. 643–649

[224] Tanterdtid, S., Steanputtanagul, W. & Benjapolakul, W., "Optimum virtual paths system based in ATM network using genetic algorithm", Proc. Intl. Conf. on Information, Communications and Signal Processing (ICICS'97), Singapore, September 1997, pp. 596–601

[225] Tanterdtid, S., Steanputtanagul, W. & Benjapolakul, W., "Optimizing ATM network throughput based on virtual path concept by using genetic algorithm", Proc. 1997 IEEE Intl. Conf. on Intelligent Processing Systems, Beijing, China, October 1997, pp. 1634–1639

[226] Markaki, M. Vasilakos, A. & Kassotakis, I., "A hybrid genetic algorithm for the provision of isochronous service in high speed networks", Proc. 1997 IEEE Intl. Conf. on Evolutionary Computation (ICEC'97), Indianapolis, Indiana, USA, April 1997, pp. 133–137

[227] Yener, A. & Rose, C., "Near-optimal admission policies for cellular networks using genetic algorithms", Proc. 6th Intl. Conf. on Wireless Comm. (Wireless'94), Calgary, Canada, July 1994

[228] Yener, A. & Rose, C., "Local call admission policies for cellular networks using genetic algorithms", Proc. 29th Annual Conf. on Information Sciences and Systems (CISS'95), Baltimore, USA, March 1995

[229] Yener, A. & Rose, C., "Genetic algorithms applied to cellular call admission: Local policies", *IEEE Transactions on Vehicular Technology* v46 n1, February 1997, pp. 72–79

[230] Abed, M.A.A. & Ghanta, S., "Optimizing logical topology of lightwave network architecture (LNA) using genetic algorithms", Proc. 1996 IEEE 15th Annual Intl. Phoenix Conf. on Computers and Communications, Scottsdale, Arizona, USA, March 1996, pp. 501–507

[231] Crompton, W., Hurley, S. & Stephens, N.M., "Frequency assignment using a parallel genetic algorithm", Proc. 2nd IEE/IEEE Workshop on Natural Algorithms in Signal Processing (NASP'93), Chelmsford, UK, November 1993, pp. 26/1–26/8

[232] Crompton, W., Hurley, S. & Stephens, N.M., "A parallel genetic algorithm for frequency assignment problems", Proc. IMACS/IEEE Intl. Symp. on Signal Processing, Robotics and Neural Networks (SPRANN'94), Lille, France, April 1994, pp. 81–84

[233] Crompton, W., Hurley, S. & Stephens, N.M., "Applying genetic algorithms to frequency assignment problems", Proc. SPIE Conf. Neural and Stochastic Methods in Image and Signal Processing, San Diego, USA, July 1994, v2304, pp. 76–84

[234] Hurley, S. & Smith, D.H., "Fixed spectrum frequency assignment using natural algorithms", Proc. 1st IEE/IEEE Intl. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'95), University of Sheffield, UK, September 1995, pp. 373–378

[235] Hurley, S., Thiel, S.U. & Smith, D.H., "A comparison of local search algorithms for radio link frequency assignment problems", Proc. 1996 ACM Symp. on Applied Computing (SAC'96), Philadelphia, PA, USA, February 1996, pp. 251–257

[236] Valenzuela, C.L., Jones, A. & Hurley, S., "Breeding permutations for minimum span frequency assignment", Proc. 3rd Intl. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97), University of East Anglia, Norwich, UK, April 1997, pp. 308–311

[237] Valenzuela, C., Hurley, S. & Smith, D., "A permutation based genetic algorithm for minimum span frequency assignment", Proc. 5th Intl. Conf. on Parallel Problem Solving from Nature (PPSN V), Amsterdam, The Netherlands, September 1998, pp. 907–916

[238] Cuppini, M., "A genetic algorithm for channel assignment problems", *European Transactions on Telecommunications* v5 n2, March/April 1994, pp. 285–294

[239] Kapsalis, A. & Smith, G.D.,"A meta-genetic algorithm for the radio link frequency assignment problem", Proc. 1st Intl. ICSC Symp. on Intelligent Industrial Automation and Soft Computing (IIA'96/SOCO'96), University of Reading, UK, March 1996, pp. B 318–B 324

[240] Crisan, C. & Mühlenbein, H., "The breeder genetic algorithm for frequency assignment", Proc. 5th Intl. Conf. on Parallel Problem Solving from Nature (PPSN V), Amsterdam, The Netherlands, September 1998, pp. 897–906

[241] Ngo, C.Y. & Li, V.O.K., "Fixed channel assignment in cellular radio networks using a modified genetic algorithm", *IEEE Transactions on Vehicular Technology* v47 n1, February 1998, pp. 163–171

[242] Kaminsky, A.R., "A fuzzy genetic algorithm for automatic channel assignment for tactical HF radio networks", Proc. 7th Intl. Conf. on HF Radio Systems and Techniques, Nottingham, UK, July 1997, pp. 236–241

[243] Sandalidis, H.G., Stavroulakis, P.P. & Rodriguez-Tellez, J., "Combinatorial evolution strategy-based implementation of dynamic channel assignment in cellular communications", Proc. 6th IEE Conf. on Telecommunications, Edinburgh, UK, March/April 1998, pp. 170–174

[244] Jog, P., Suh, J.Y. & Van Gucht, D., "The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the travelling salesman problem", Proc. 3rd Intl. Conf. on Genetic Algorithms (ICGA'89), George Mason University, Arlington, Virginia, USA, June 1989, pp. 110–115

[245] Bramlette, M.F. & Bouchard, E.E., "Genetic algorithms in parametric design of aircraft", in [9], Ch. 10, pp. 109–123

[246] Davidor, Y., "A genetic algorithm applied to robot trajectory generation", in [9], Ch. 12, pp. 144–165

[247] Forrest, S. & Mayer-Kress, G., "Genetic algorithms, nonlinear dynamic systems, and models of international security", in [9], Ch. 13, pp. 166–185

[248] Grefenstette, J.J., "Strategy acquisition with genetic algorithms", in [9], Ch. 14, pp. 186–201

[249] Harp, S.A., "Genetic sysnthesis of neural network architecture", in [9], Ch. 15, pp. 202–221

[250] Karr, C.L., "Air-injected hydrocyclone optimization via genetic algorithm", in [9], Ch. 16, pp. 222–236

[251] Liepins, G.E. & Potter, W.D., "A genetic algorithm approach to multiple-fault diagnosis", in [9], Ch. 17, pp. 237–250

[252] Lucasius, C.B., Blommers, M.J.J., Buydens, L.M.C & Kateman, G., "A genetic algorithm for conformational analysis of DNA", in [9], Ch. 18, pp. 251–281

[253] Montana, D.J., "Automated parameter tuning for interpretation of synthetic images", in [9], Ch. 19, pp. 282–311

[254] Powell, D.J., Skolnick, M.M. & Tong, S.S., "Interdigitation: a hybrid technique for engineering design optimisation employing genetic algorithms, expert systems, and numerical optimization", in [9], Ch. 20, pp. 312–331

[255] Syswerda, G., "Schedule optimization using genetic algorithms", in [9], Ch. 21, pp. 332–349

[256] Whitley, D., Starkweather, T. & Shaner, D., "The travelling salesman and sequence scheduling: quality solutions using genetic edge recombination", in [9], Ch. 22, pp. 350–372

[257] Schnecke, V. & Vornberger, O., "Hybrid genetic algorithms for constrained placement problems", *IEEE Transactions on Evolutionary Computation* v1 n4, November 1997, pp. 266–277

[258] Zhang, F., Zhang, Y.F. & Nee, A.Y.C., "Using genetic algorithms in process planning for job shop machining", *IEEE Transactions on Evolutionary Computation* v1 n4, November 1997, pp. 278–289

[259] Xiao, J., Michalewicz, Z., Zhang, L. & Trojanowski, K., "Adaptive evolutionary planner/ navigator for mobile robots", *IEEE Transactions on Evolutionary Computation* v1 n1, April 1997, pp. 18–28

[260] Śmierzchalski, R. & Michalewicz, Z., "Modeling of ship trajectory in collision situations by an evolutionary algorithm", *IEEE Transactions on Evolutionary Computation* v4 n3, September 2000, pp. 227–241

[261] Magyar, G., Johnsson, M. & Nevalainen, O., "An adaptive hybrid genetic algorithm for the three-matching problem", *IEEE Transactions on Evolutionary Computation* v4 n2, July 2000, pp. 135–146

[262] Bhandarkar, S.M. & Zhang, H., "Image segmentation using evolutionary computation", *IEEE Transactions on Evolutionary Computation* v3 n1, April 1999, pp. 1–21

[263] Eiben, Á.E., Hinterding, R. & Michalewicz, Z., "Parameter control in evolutionary algorithms", *IEEE Transactions on Evolutionary Computation* v3 n2, July 1999, pp. 124–141

[264] Hinterding, R., Michalewicz, Z. & Eiben, A.E., "Adaptation in evolutionary computation: A survey", Proc. 1997 IEEE Intl. Conf. on Evolutionary Computation (ICEC'97), Indianapolis, Indiana, USA, April 1997, pp. 65–69

[265] Tuson, A.L., *Adapting Operator Probabilities in Genetic Algorithms*, M.Sc. thesis, Dept. of Artificial Intelligence, University of Edinburgh, UK, 1995

[266] Tuson, A. & Ross, P., "Adapting operator settings in genetic algorithms", *Evolutionary Computation* v6 n2, 1998, pp. 161–184

[267] Fogarty, T.C., "Varying the probability of mutation in the genetic algorithm", Proc. 3rd Intl. Conf. on Genetic Algorithms (ICGA'89), George Mason University, Arlington, Virginia, USA, June 1989, pp. 104–109

[268] Davis, L., "Adapting operator probabilities in genetic algorithms", Proc. 3rd Intl. Conf. on Genetic Algorithms (ICGA'89), George Mason University, Arlington, Virginia, USA, June 1989, pp. 61–69

[269] Srinivas, M. & Patnaik, L.M., "Adaptive probabilities of crossover and mutation in genetic algorithms", *IEEE Transactions on Systems, Man and Cybernetics* v24 n4, April 1994, pp. 656–667

[270] Julstrom, B.A., "What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm", Proc. 6th Intl. Conf. on Genetic Algorithms (ICGA'95), University of Pittsburgh, USA, July 1995, pp. 81–87

[271] Tuson, A. & Ross, P., "Cost based operator rate adaptation: An investigation", Proc. 4th Intl. Conf. on Parallel Problem Solving From Nature (PPSN IV), Berlin, Germany, September 1996

[272] Hill, A.M. & Houghton, A.J.N., "Optical networking in the European ACTS programme", OFC'96, San Jose, USA, 1996

[273] O'Mahony, M.J., "An overview of WDM research in Europe", Proc. IEE Colloquium on Multiwavelength Optical Networks: Devices, Systems and Network Implementations, Digest No. 1998/296, London, UK, June 1998, pp. 1/1–1/3

[274] Berger, M., Chbat, M., Jourdan, A., Sotom, M., Demeester, P., Van Caenegem, B., Godsvang, P., Hein, B., Huber, M., Marz, R., Leclert, A., Olsen, T., Tobolka, G. & Vanden Broeck, T., "Pan-European optical networking using wavelength division multiplexing", *IEEE Communications Magazine* v35 n4, 1997, pp. 82–88

[275] Fonseca, C.M. & Fleming, P.J., "Multiobjective genetic algorithms made easy: Selection, sharing and mating restriction", Proc. 1st IEE/IEEE Intl. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'95), University of Sheffield, UK, September 1995, pp. 45–52

[276] Fonseca, C.M. & Fleming, P.J., "An overview of evolutionary algorithms in multiobjective optimization", *Evolutionary Computation* v3 n1, 1995, pp. 1–16

[277] Zitzler, E. & Thiele, L., "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach", *IEEE Transactions on Evolutionary Computation* v3 n4, November 1999, pp. 257–271

[278] Orvosh, D. & Davis, L., "Shall we repair? Genetic algorithms, combinatorial optimization and feasibility constraints", Proc. 5th Intl. Conf. on Genetic Algorithms (ICGA'93), University of Illinois at Urbana-Champaign, USA, July 1993, p. 650

[279] Richardson, J.T., Palmer, M.R., Liepins, G. & Hilliard, M., "Some guidelines for genetic algorithms with penalty functions", Proc. 3rd Intl. Conf. on Genetic Algorithms (ICGA'89), George Mason University, Arlington, Virginia, USA, June 1989, pp. 191–197

[280] Smith, A.E. & Tate, D.M., "Genetic optimization using a penalty function", Proc. 5th Intl. Conf. on Genetic Algorithms (ICGA'93), University of Illinois at Urbana-Champaign, USA, July 1993, pp. 499–505

[281] Coplein, J.O., *Advanced C++ Programming Styles and Idioms*, Addison-Wesley, 1992

[282] Stroustrup, B., *The Design and Evolution of C++*, Addison-Wesley, 1994

[283] Rational Software Corporation, *Using Rational Rose/C++ (Revision 2.7)*, 1995

[284] Ousterhout, J,K., *Tcl and the Tk Toolkit*, Addison-Wesley, 1994

[285] Welch, B.B., *Practical Programming in Tcl and Tk*, Prentice-Hall, 1995

[286] Lin, P.M., Leon, B.J. & Stewart, C.R., "Analysis of circuit-switched networks employing originating-office control with spill-forward", *IEEE Transactions on Communications* v26 n6, June 1978, pp. 754–765

[287] Sedgewick, R., *Algorithms (2nd Ed.)*, Addison-Wesley, 1988

[288] `xfig`—*Facility for Interactive Generation of Figures under X11, v2.1.8*, 1993

[289] Meyer, B., *Object-oriented Software Construction*, Prentice Hall, 1988

[290] Merilatt, R., "C dynamic memory use: Finding and exterminating C's dynamic memory errors", *Dr. Dobb's Journal*, August 1989, pp. 62,64,66,67,125–127

[291] O'Mahony, M., Sinclair, M.C. & Mikac, B., "Ultra-high capacity optical transmission network: European research project COST 239", *Informacija Telekomunikacije Automati* v12 n1–3, 1993, pp. 33–45

[292] Chen, W.-K., *Theory of Nets: Flows in Networks*, Wiley, 1990

[293] Brown, G.N., Donachie, S.J., Beggs, S.L., Johnson, D. & Botham, C.P., "TENDRA — a simulation tool for the analysis of transport networks employing distributed restoration algorithms", Proc. IEE Colloquium on Resilience in Optical Networks, Digest No. 1992/189, London, October 1992, pp. 4/1–4/6

[294] Sato, K.I., Okamoto, S. and Hadama, H., "Network performance and integrity enhancement with optical path layer technologies", *IEEE Journal on Selected Areas in Communications* v12 n1, 1994, pp. 159–170

[295] Ishio, H., "Next generation lightwave communications technologies — towards communications networks evolution", *NTT Review* v4 n6 1992 pp. 62–68

[296] Nagatsu, N., Hamazumi, Y. & Sato, K.I., "Number of wavelengths required for constructing large-scale optical path networks", *Electronics and Communications in Japan, Part I — Communications* v78 n9, 1995, pp. 1–11

[297] Nagatsu, N., Hamazumi, Y. & Sato, K.I., "Optical path accommodation designs applicable to large scale networks", *IEICE Transactions on Communications* vE78-B n4, 1995, pp. 597–607

[298] Wuttisittikulkij, L. & O'Mahony, M.J. "A simple algorithm for wavelength assignment in optical networks", Proc. 21st Eur. Conf. on Opt. Comm. (ECOC'95), Brussels, 1995, pp. 859–862

[299] Banerjee, D. & Mukherjee, B., "A practical approach for routing and wavelength assignment in large wavelength-routed optical networks", *IEEE Journal on Selected Areas in Communications* v14 n5, 1996, pp. 903–908

[300] Baroni, S. & Bayvel P., "Wavelength requirements in arbitrary connected wavelength-routed optical networks", *Journal of Lightwave Technology* v15 n2, 1997, pp. 242–251

[301] O'Mahony, M.J., Simeonidou, D., Yu, A. & Zhou, J., "The design of a European optical network", *Journal of Lightwave Technology* v13 n5, 1995, pp. 817–828

[302] Nagatsu, N., Okamoto, S. & Sato, K.I., "Optical path cross-connect system scale evaluation using path accommodation design for restricted wavelength multiplexing", *IEEE Journal on Selected Areas in Communications* v14 n5, 1996, pp. 893–902

[303] Nagatsu, N. & Sato, K.I., "Optical path accommodation design enabling cross-connect system scale evaluation", *IEICE Transactions on Communications* vE78-B n9, 1995, pp. 1339–1343

[304] Blickle, T. & Thiele, L.,"A mathematical analysis of tournament selection", Proc. 6th Intl. Conf. on Genetic Algorithms (ICGA'95), University of Pittsburgh, USA, July 1995, pp. 9–16

[305] Blickle, T. & Thiele, L., *A Comparision of Selection Schemes used in Genetic Algorithms*, Technical Report 11 Version 2, Computer Engineering and Communications Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zurich, December 1995

[306] Wauters, N. & Demeester, P., "Design of the optical-path layer in multiwavelength cross-connected networks", *IEEE Journal on Selected Areas in Communications* v14 n5 1996 pp. 881–892

[307] Sprent, P., *Applied Nonparametric Statistical Methods (2nd Ed.)*, Chapman & Hall, 1993

[308] Syswerda, G., "Uniform crossover in genetic algorithms", Proc. 3rd Intl. Conf. on Genetic Algorithms (ICGA'89), George Mason University, Arlington, Virginia, USA, June 1989, pp. 2–9

[309] Khatib, W. & Fleming, P.J., "The stud GA: A mini revolution?", Proc. 5th Intl. Conf. on Parallel Problem Solving from Nature (PPSN V), Amsterdam, The Netherlands, September 1998, pp. 683–691

[310] Herrera, F. & Lozano, M., "Gradual distributed real-coded genetic algorithms", *IEEE Transactions on Evolutionary Computation* v4 n1, April 2000, pp. 43–63

[311] Michalewicz, Z., Deb, K., Schmidt, M. & Stidsen, T., "Test-case generator for nonlinear continuous parameter optimization techniques", *IEEE Transactions on Evolutionary Computation* v4 n3, September 2000, pp. 197–215

[312] Griffith, P.S., *Heuristic Topological Design of Low-Cost Optical Telecommunication Networks*, M.Sc. project report, Dept. of Electronic Sysems Engineering, University of Essex, UK, May 1996

[313] Griffith, P.S., Proestaki, A. & Sinclair, M.C. , "Heuristic topological design of low-cost optical telecommunication networks", Proc. 12th UK Performance Engineering Workshop, Edinburgh, UK, September 1996, pp. 129–140

[314] Naudts, B. & Kallel, L., "A comparison of predictive measures of problem difficulty in evolutionary algorithms", *IEEE Transactions on Evolutionary Computation* v4 n1, April 2000, pp. 1–15

[315] Alanyali, M. & Ayanoglu, E., "Provisioning algorithms for WDM optical networks", *IEEE/ACM Transactions on Networking* v7 n5, October 1999, pp. 767–778

[316] Ramaswami, R. & Sasaki, G., "Multiwavelength optical networks with limited wavelength conversion", *IEEE/ACM Transactions on Networking* v6 n6, December 1998, pp. 744–754

[317] Williams, T. & Kelly, C., *GNUPLOT: An Interactive Plotting Program, v3.5*, 1993

[318] Lamport, L., *LaTeX: A Document Preparation System (2nd Ed.)*, Addison-Wesley, 1994

[319] Goossens, M., Mittelbach, F. & Samarin, A., *The LaTeX Companion*, Addison-Wesley, 1994

[320] Yen, J.Y., "Finding the k shortest loopless paths in a network", *Management Science* v17 n11, July 1971, pp. 712–716

[321] Zongker, D. & Punch, B.: 'lil-gp 1.0 user's manual' (`http://isl.cps.msu.edu/GA/software/lil-gp/docs/lilgp.docs.html`)

[322] Gondran, M. & Minoux, M., *Graphs and Algorithms*, Wiley, 1984

[323] Sinclair, M.C., *Strategy for a Fault Tolerant Trunk Network*, M.Sc. project report, Dept. of Electronic Sysems Engineering, University of Essex, UK, March 1990

[324] Sinclair, M.C., "Single-moment analysis of unreliable trunk networks: Two methods for the network grade-of-service", Proc. 8th UK Teletraffic Symposium, Beeston, April 1991, pp. 8/1–8/11

[325] Sinclair, M.C., "Single-moment analysis of unreliable trunk networks employing k-shortest-path routing", Proc. IEE Colloquium on Resilience in Optical Networks, Digest No. 1992/189, London, October 1992, pp. 3/1–3/6

[326] Pothering, G.J. & Naps, T.L., *Introduction to Data Structures and Algorithm Analysis with C++*, West, 1995

[327] Ghanbari, M., Hughes, C.J., Sinclair, M.C. & Eade, J.P., *Principles of Performance Engineering for Telecommunication and Information Systems*, IEE, 1997

[328] Fishman, G.S., *Principles of Discrete-Event Digital Simulation*, Wiley, 1978

[329] Meyer, B., *Eiffel: The Language*, Prentice Hall, 1992

[330] Switzer, R., *Eiffel: An Introduction*, Prentice Hall, 1993

[331] Waldén, K. & Nerson, J.-M., *Seamless Object-Oriented Software Architecture: Analysis and Design of Reliable Systems*, Prentice Hall, 1995

[332] *The Virtual Reality Modeling Language*, International Standard ISO/IEC 14772-1:1997 (`http://www.vrml.org/fs_specifications.htm`)

[333] Horstmann, C.S. & Cornell, G, *Core Java 2 Volume 1: Fundamentals (5th Ed.)*, Prentice Hall, 2001

[334] Horstmann, C.S. & Cornell, G, *Core Java 2 Volume 2: Advanced Features (4th Ed.)*, Prentice Hall, 2000

[335] Lawn, J., *Optimisation of Survivability in a Military Communications Network using a Hybrid Genetic Algorithm*, M.Sc. project report, Dept. of Electronic Sysems Engineering, University of Essex, UK, May 2000

[336] Tan, C.H., *Uncertain Multiperiod Telecommunications Network Design using Evolutionary Algorithms*, M.Sc. project report, Dept. of Electronic Sysems Engineering, University of Essex, UK, May 2000

[337] Brittain, D., "Optimisation of the Telecommunications Access Network", Ph.D. thesis, Dept. of Engineering Mathematics, University of Bristol, UK, 1999

[338] Tsutsui, S. & Ghosh, A., "Genetic algorithms with a robust solution searching scheme", *IEEE Transactions on Evolutionary Computation* v1 n3, September 1997, pp. 201–208

[339] Ismail, A., *Traffic Estimation for Future Broadband Networks*, M.Sc. project report, Dept. of Electronic Sysems Engineering, University of Essex, UK, September 1997

[340] Toh, H.H., *Node Placement for the Optical Core Network*, M.Sc. project report, Dept. of Electronic Sysems Engineering, University of Essex, UK, May 1997

[341] Sinclair, M.C. & Shami, S.H., "Evolving simple software agents: Comparing genetic algorithm and genetic programming performance", Proc. IEE/IEEE Intl. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'97), Glasgow, September 1997, pp. 421–426

[342] Chan, C.S., *Development of NOMaD Network Editor in Java*, B.Eng. project report, Dept. of Electronic Sysems Engineering, University of Essex, UK, April 1998

[343] Maseko, T.J., *Development of NOMaD Network Editor in Java*, B.Eng. project report, Dept. of Electronic Sysems Engineering, University of Essex, UK, April 1998

[344] Berkelaar, M., ftp site for `lp_solve` (`ftp://ftp.es.ele.tue.nl/pub/lp_solve`)

[345] Gavalas, D., *Optical Network Design: Virtual Topology and Wavelength Assignment Using Linear Programming*, M.Sc. project report, Dept. of Electronic Sysems Engineering, University of Essex, UK, May 1997

[346] Sinclair, M.C., "Improved model for European international telephony traffic", *Electronics Letters* v30 n18 September 1994 pp. 1468–1470

[347] Hopkins, M., Louth, G., Bailey, H., Yellon, R., Ajibulu, A. & Niva , M., "A multi-faceted approach to forecasting broadband demand and traffic", *IEEE Communications Magazine* v33 n2 1995 pp. 36–42

[348] Klingman, D., Napier, A. & Stutz, J., "NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems", *Management Science* v20 n5, January 1974, pp. 814–821

[349] McKay, M.D., Beckman, R.J. & Conover, W.J., "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code", *Technometrics* v21 n2 May 1979 pp. 239–245

[350] Welch, W.J., Buck, R.J., Sacks, J., Wynn, H.P., Mitchell, T.J. & Morris, M.D., "Screening, predicting, and computer experiments", *Technometrics* v34 n1 February 1992 pp. 15–25

[351] Bates, R.A., Buck, R.J., Riccomagno, E. & Wynn, H.P., "Experimental design and observation for large systems", *Journal of the Royal Statistical Society Series B—Methodological* v58 n1 1996 pp. 77–94

Appendix A

Supporting Publications

At submission, the author had published eight papers that directly support the thesis. These are listed below, and copies of the papers themselves have been included in the following pages.

1. Sinclair, M.C., "NOMaD: Initial architecture of an optical network optimisation, modelling and design tool", Proc. 12th UK Performance Engineering Workshop, Edinburgh, UK, September 1996, pp. 157–167

2. Sinclair, M.C., "NOMaD: Applying a genetic-algorithm/heuristic hybrid approach to optical network topology design", Proc. Intl. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97), Norwich, UK, April 1997, pp. 299–303

3. Sinclair, M.C., "Minimum network wavelength requirement design using a genetic-algorithm/heuristic hybrid", *Electronics Letters* v34 n4 February 1998 pp. 388–389

4. Sinclair, M.C., "Minimum cost routing and wavelength allocation using a genetic-algorithm/heuristic hybrid approach", Proc. Sixth IEE Conf. on Telecommunications, Edinburgh, UK, March/April 1998, pp. 67–71

5. Sinclair, M.C., "Operator-probability adaptation in a genetic-algorithm/heuristic hybrid for optical network wavelength allocation", Proc. IEEE Intl. Conf. on Evolutionary Computation (ICEC'98), Anchorage, Alaska, USA, May 1998, pp. 840–845

6. Sinclair, M.C., "NOMaD: An optical network optimisation, modelling and design toolset", Proc. IEE Colloquium on Multiwavelength Optical Networks: Devices, Systems and Network Implementations, Digest No. 1998/296, London, UK, June 1998, pp. 6/1–6/6

7. Sinclair, M.C., "Minimum cost wavelength-path routing and wavelength allocation using a genetic-algorithm/heuristic hybrid approach", *IEE Proc. Communications* v146 n1 February 1999 pp. 1–7

8. Sinclair, M.C., "Evolutionary telecommunications: A summary", Proc. GECCO'99 Workshop on Evolutionary Telecommunications: Past, Present and Future, Orlando, Florida, USA, July 1999, pp. 209–212

Appendix B

Personal Publications

A complete list of the author's publications, at submission, is given below (including those directly supporting the thesis, already listed in Appendix A).

1. Sinclair, M.C., "Single-moment analysis of unreliable trunk networks: Two methods for the network grade-of-service", Proc. 8th UK Teletraffic Symposium, Beeston, April 1991, pp. 8/1–8/11

2. Sinclair, M.C., "Trunk network fault-tolerance through routing table updating in the event of trunk failure", Proc. 9th UK Teletraffic Symposium, Guildford, April 1992, pp. 17/1–17/9

3. Sinclair, M.C., "Trunk network fault-tolerance through network augmentation", Proc. 4th Bangor Symposium on Communications, University of Wales — Bangor, May 1992, pp. 252–256

4. Sinclair, M.C., "Single-moment analysis of unreliable trunk networks employing k-shortest-path routing", Proc. IEE Colloquium on Resilience in Optical Networks, Digest No. 1992/189, London, October 1992, pp. 3/1–3/6

5. Sinclair, M.C., "The application of a genetic algorithm to trunk network routing table optimisation", Proc. 10th UK Teletraffic Symposium, Martlesham Heath, April 1993, pp. 2/1–2/6

6. Chng, R.S.K., Sinclair, M.C., Donachie, S.J. & O'Mahony, M.J., "Distributed restoration algorithm for multiple failures in a reconfigurable network", Proc. 5th Bangor Symposium on Communications, University of Wales — Bangor, June 1993, pp. 203–206

7. Sinclair, M.C. & O'Mahony, M.J., "COST239: Initial network design and analysis", Proc. 36th RACE Concertation, Brussels, July 1993, pp. 89–94

8. O'Mahony, M., Sinclair, M.C. & Mikac, B., "Ultra-high capacity optical transmission network: European research project COST 239", *Informacija Telekomunikacije Automati* v12 n1-3 1993 pp. 33–45

9. Chng, R.S.K., Sinclair, M.C., Botham, C.P. & O'Mahony, M.J., "Multi-layer real-time restoration for survivable networks", Proc. IEE Colloquium on Safety Critical Distributed Systems, Digest No. 1993/189, London, October 1993, pp. 8/1–8/5

10. Stevens, R.D., Sinclair, M.C. & Dilworth, I.J., "Finite source modelling of private mobile radio systems", Proc. 11th UK Teletraffic Symposium, Cambridge, March 1994, pp. 12B/1–12B/7

11. Johnson, D., Brown, G.N., Beggs, S.L., Botham, C.P., Hawker, I., Chng, R.S.K., Sinclair, M.C. & O'Mahony, M.J., "Distributed restoration strategies in telecommunications networks", Proc. IEEE Intl. Conf. Commun. (ICC'94), New Orleans, USA, May 1994, pp. 483–488

12. Stevens, R.D., Dilworth, I.J. & Sinclair, M.C., "Fading and its effects on the planning of private mobile radio systems", Proc. 11th National Radio Science Colloquium, Liverpool, July 1994

13. Stevens, R., Dilworth, I. & Sinclair, M., "Fading and its effects on private mobile radio systems", *Philips Telecom Review* v52 n2 August 1994 pp. 24–27

14. Sinclair, M.C., "Improved model for European international telephony traffic", *Electronics Letters* v30 n18 September 1994 pp. 1468–1470

15. Stevens, R.D., Sinclair, M.C. & Dilworth, I.J., "Finite source analysis and simulation of private mobile radio systems", Proc. 10th UK Performance Engineering Workshop, Edinburgh, September 1994, pp. 39–45

16. Chng, R.S.K., Botham, C.P., Johnson, D., Brown, G.N., Sinclair, M.C., O'Mahony, M.J., & Hawker, I., "A multi-layer restoration strategy for reconfigurable networks", Proc. IEEE Global Telecom. Conf. (GLOBECOM'94), San Francisco, USA, November 1994, pp. 1872–1878

17. Robson, S.R. & Sinclair, M.C., "An improved channel resource queueing algorithm for trunked mobile radio", Proc. 12th UK Teletraffic Symposium, Old Windsor, March 1995, pp. 17/1–17/10

18. Segkhoonthod, S. & Sinclair, M.C., "Analysis of a fault-tolerant ATM switch based on a parallel architecture", Proc. 12th UK Teletraffic Symposium, Old Windsor, March 1995, pp. 22/1–22/9

19. Sinclair, M.C., "Minimum cost topology optimisation of the COST 239 European optical network", Proc. Intl. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA'95), Alès, France, April 1995, pp. 26–29

20. Segkhoonthod, S. & Sinclair, M.C., "Re-routing analysis of a fault-tolerant ATM switch based on a parallel architecture", Proc. 3rd Workshop on Performance Modelling and Evaluation of ATM Networks, Ilkley, July 1995, pp. 73/1–73/12

21. Robson, S.R. & Sinclair, M.C., "Reduction of channel idle time in a trunked mobile radio network", Proc. 11th UK Performance Engineering Workshop, Liverpool, September 1995, pp. 355–369

22. Brander, A.W. & Sinclair, M.C., "A comparative study of k-shortest path algorithms", Proc. 11th UK Performance Engineering Workshop, Liverpool, September 1995, pp. 370–379

23. Tan, L.G. & Sinclair, M.C., "Wavelength assignment between the central nodes of the COST 239 European optical network", Proc. 11th UK Performance Engineering Workshop, Liverpool, September 1995, pp. 235–247

24. Segkhoonthod, S. & Sinclair, M.C., "Performance analysis of a fault-tolerant ATM switch in single-fault conditions", Proc. 11th UK Performance Engineering Workshop, Liverpool, September 1995, pp. 48–63

25. Proestaki, A. & Sinclair, M.C., "Initial survey of heuristics for optical core network design in ACTS WOTAN", Proc. 13th UK Teletraffic Symposium, Glasgow, March 1996, pp. 20/1–20/8

26. Griffith, P.S., Proestaki, A. & Sinclair, M.C. , "Heuristic topological design of low-cost optical telecommunication networks", Proc. 12th UK Performance Engineering Workshop, Edinburgh, September 1996, pp. 129–140

27. Parnis, N., Jones, E.V., Sinclair, M.C. & O'Mahony, M.J., "Hierarchical network design for ACTS OPEN: Initial considerations", Proc. 12th UK Performance Engineering Workshop, Edinburgh, September 1996, pp. 141–155

28. Sinclair, M.C., "NOMaD: Initial architecture of an optical network optimisation, modelling and design tool", Proc. 12th UK Performance Engineering Workshop, Edinburgh, UK, September 1996, pp. 157–167

29. Kirkwood, I.M.A., Shami, S.H. & Sinclair, M.C., "Discovering simple fault-tolerant routing rules using genetic programming", Proc. Intl. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97), Norwich, UK, April 1997, pp. 285–288

30. Sinclair, M.C., "NOMaD: Applying a genetic-algorithm/heuristic hybrid approach to optical network topology design", Proc. Intl. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97), Norwich, UK, April 1997, pp. 299–303

31. Ghanbari, M., Hughes, C.J., Sinclair, M.C. & Eade, J.P., *Principles of Performance Engineering for Telecommunication and Information Systems*, IEE, 1997

32. Jackson, E., Spindley, R., Sinclair, M. & Reader, S., "SS7 network management using link monitors", Proc. IEEE Intl. Conf. Commun. (ICC'97), Montréal, Canada, June 1997, pp. 883–888

33. Segkhoonthod, S. & Sinclair, M.C., "Design of fault-tolerant ATM switch based on parallel architecture", *Electronics Letters* v33 n15 July 1997 pp. 1289–1290

34. Stevens, R.D. & Sinclair, M.C., "Finite-source analysis of traffic on private mobile radio systems", *Electronics Letters* v33 n15 July 1997 pp. 1292–1293

35. Shami, S.H., Kirkwood, I.M.A., & Sinclair, M.C., "Evolving simple fault-tolerant routing rules using genetic programming", *Electronics Letters* v33 n17 August 1997 pp. 1440–1441

36. Aiyarak, P., Saket, A.S. & Sinclair, M.C., "Genetic programming approaches for minimum cost topology optimisation of optical telecommunication networks", Proc. IEE/IEEE Intl. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'97), Glasgow, September 1997, pp. 415–420

37. Sinclair, M.C. & Shami, S.H., "Evolving simple software agents: Comparing genetic algorithm and genetic programming performance", Proc. IEE/IEEE Intl. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'97), Glasgow, September 1997, pp. 421–426

38. Sinclair, M.C., "Minimum network wavelength requirement design using a genetic-algorithm/heuristic hybrid", *Electronics Letters* v34 n4 February 1998 pp. 388–389

39. Sinclair, M.C., "Minimum cost routing and wavelength allocation using a genetic-algorithm/heuristic hybrid approach", Proc. Sixth IEE Conf. on Telecommunications, Edinburgh, UK, March/April 1998, pp. 67–71

40. Sinclair, M.C., "Operator-probability adaptation in a genetic-algorithm/heuristic hybrid for optical network wavelength allocation", Proc. IEEE Intl. Conf. on Evolutionary Computation (ICEC'98), Anchorage, Alaska, USA, May 1998, pp. 840–845

41. Sinclair, M.C., "NOMaD: An optical network optimisation, modelling and design toolset", Proc. IEE Colloquium on Multiwavelength Optical Networks: Devices, Systems and Network Implementations, Digest No. 1998/296, London, UK, June 1998, pp. 6/1–6/6

42. Sinclair, M.C., "Minimum cost wavelength-path routing and wavelength allocation using a genetic-algorithm/heuristic hybrid approach", *IEE Proc. Communications* v146 n1 February 1999 pp. 1–7

43. Proestaki, A. & Sinclair, M.C., "Interconnection strategies for dual-homing multi-ring networks", Proc. 16th International Teletraffic Conference (ITC-16), Edinburgh, UK, June 1999, pp. 169–181

44. Proestaki, A. & Sinclair, M.C., "Wavelength routing in all-optical dual-homing hierarchical multi-ring networks", Proc. European Conference on Networks and Optical Communications — Core Networks and Network Management (NOC'99), Delft, The Netherlands, June 1999, pp. 52–59

45. Shami, S.H. & Sinclair, M.C., "Co-evolutionary agents for telecommunication network restoration", Proc. Recent Advances in Soft Computing'99, Leicester, UK, July 1999, pp. 146–151

46. Navarro Varela, G. & Sinclair, M.C., "Ant colony optimisation for virtual-wavelength-path routing and wavelength allocation", Proc. Congress on Evolutionary Computation (CEC'99), Washington DC, USA, July 1999, pp. 1809–1816

47. Moore, S.J. & Sinclair, M.C., "Design of routing tables for a survivable military communications network using genetic algorithms", Proc. Congress on Evolutionary Computation (CEC'99), Washington DC, USA, July 1999, pp. 1788–1795

48. Sinclair, M.C., Corne, D. & Smith, G.D., "Evolutionary telecommunications: Past, present and future", Proc. GECCO'99 Workshop on Evolutionary Telecommunications: Past, Present and Future, Orlando, Florida, USA, July 1999, p. 208

49. Sinclair, M.C., "Evolutionary telecommunications: A summary", Proc. GECCO'99 Workshop on Evolutionary Telecommunications: Past, Present and Future, Orlando, Florida, USA, July 1999, pp. 209–212

50. Sinclair, M.C. & Clark, A.F., "Evolving an artificial vision system: Initial considerations", Proc. GECCO'99 Workshop on Evolution of Sensors in Nature, Hardware and Simulation, Orlando, Florida, USA, July 1999, pp. 191–195

51. Sinclair, M.C., "Optical mesh topology design using node-pair encoding genetic programming", Proc. Genetic and Evolutionary Computation Conference (GECCO-99), Orlando, Florida, USA, July 1999, pp. 1192–1197

52. Pagonis, J. & Sinclair, M.C., "Evolving personal agent environments to reduce internet information overload: Initial considerations", Proc. IEE Colloquium on Lost in the Web: Navigation on the Internet, Digest No. 1999/169, London, UK, November 1999, pp. 2/1–2/10

53. Proestaki, A. & Sinclair, M.C., "Impact of topology on wavelength and switch-port requirements in all-optical hierarchical multi-ring networks", Proc. IEEE Global Telecom. Conf. (GLOBECOM'99), Rio de Janeiro, Brazil, December 1999, pp. 1035–1041

54. Proestaki, A. & Sinclair, M.C., "Dimensioning all-optical dual-homing hierarchical multi-ring networks with various connectivities", Proc. Second Intl. Workshop on the Design of Reliable Communication Networks (DRCN2000), Munich, Germany, April 2000

55. Proestaki, A. & Sinclair, M.C., "Design and dimensioning of dual-homing hierarchical multi-ring networks", *IEE Proc. Communications* v147 n2 April 2000 pp. 96–104

56. Pagonis, J. & Sinclair, M.C., "Evolving user profiles to reduce internet information overload", Proc. Intl. Conf. on Recent Advances in Soft Computing 2000, Leicester, UK, June 2000

57. Sinclair, M.C., "Node-pair encoding genetic programming for optical mesh network topology design" in Corne, D.W., Oates, M.J. & Smith, G.D. (Ed.), *Telecommunications Optimization: Heuristic and Adaptive Techniques*, Wiley, 2000, Ch.6, pp. 99–114

Appendix C

Wavelength Allocation Heuristics

This appendix provides details of four overall heuristics for wavelength-path (WP) routing, fibre choice and wavelength allocation, plus one for virtual-wavelength-path (VWP). They have all been implemented in NOMaD, and the corresponding classes are listed in Table 4.2 and illustrated in Fig. 4.6 on p. 83.

The appendix is divided into three sections: §C.1 describes a WP allocation heuristic by Wuttisittikulkij & O'Mahony [298]; §C.2 details two WP heuristics by Wauters & Demeester [306]; and finally, §C.3 presents a VWP allocation heuristic by Nagatsu *et al.*, which forms the basis for a further WP one by the same authors [296].

C.1 Wuttisittikulkij & O'Mahony

Wuttisittikulkij and O'Mahony [298] have developed a simple, fast and non-iterative WP allocation heuristic, which works with k-lowest hop count paths[1] (and multiple fibres) to obtain a low network wavelength requirement (NWR).

First, for all source-destination node pairs, the k-lowest hop count paths between them are determined. Then, in the basic (unsorted) variant of the heuristic, the node pairs are considered in order of their end-node ids. For each wavelength channel required, the path currently with the lowest free wavelength end-to-end is selected from the choice of k paths. If there is more than one path with the same lowest wavelength free then, of these, the one with the lowest hop count is chosen. However, if there is still not a unique choice, then a random selection is made between all those with the lowest wavelength and hop count.

[1] In [298], the expression 'k-shortest path' is used rather than 'k-lowest hop count path'. However, this and other potential ambiguities in the paper were clarified in discussion between the author and Wuttisittikulkij.

Parameter	Default	Description
`ww95.theK`	4	Number of alternative paths, k
`ww95.sortPaths`	`false`	Allocate the paths in hop-count order?
`ww95.firstFibreOnly`	`true`	Use only first fibre on each cable?

Table C.1: `WaveWutti95` parameters

For the sorted variant, rather than simply allocating in order of their end-node ids, the node pairs are instead pre-sorted in decreasing order of the minimum number of hops between them. It was hoped by the paper's authors that this might improve the resulting NWR, as paths with a greater number of hops are often found to be more difficult to allocate.

Their heuristic has been implemented in NOMaD as class `WaveWutti95`, the parameters for which are given in Table C.1. The first of these (`ww95.theK`) is simply the constant k. The third, `ww95.firstFibreOnly`, occasionally avoids the need to edit the NOMaD network file from which the wavelength-routed network is built: if `true`, the router uses only the *first* fibre on each cable in the network. This option can be useful if a single-fibre network design is under consideration. The remaining parameter, `ww95.sortPaths`, determines whether the paths are allocated in hop-count order or not.

C.2 Wauters & Demeester

In [306], Wauters and Demeester presented two WP allocation heuristics. The first of these is a non-iterative algorithm based on a version of Dijkstra's algorithm modified such that in "extend[ing] an already found part of a route, only those [cables] are considered which have a wavelength channel free which is available on all [cables] of the already found part of the route". Although not stated in [306], the allocation presumably proceeds in arbitrary order of source-destination node pairs, such as according to end-node ids, and one wavelength channel at a time. This heuristic results in a low-cost allocation, whilst also keeping NWR low.

Their second algorithm (HRWA), is based on iterative application of a modified version of Yen's k-shortest path algorithm [320], and aims to produce the lowest cost allocation that still achieves minimum NWR. The modification to Yen's algorithm required can be achieved by simply incorporating as a sub-routine the modified version of Dijkstra's algorithm already referred to above. An overall flowchart for the HRWA heuristic[2] is given in Fig. C.1.

[2]The flowchart and description differ slightly from that given in [306] due to some ambiguities/inconsistencies in the original paper. Although Wauters was contacted to obtain clarification of these, no response was received.

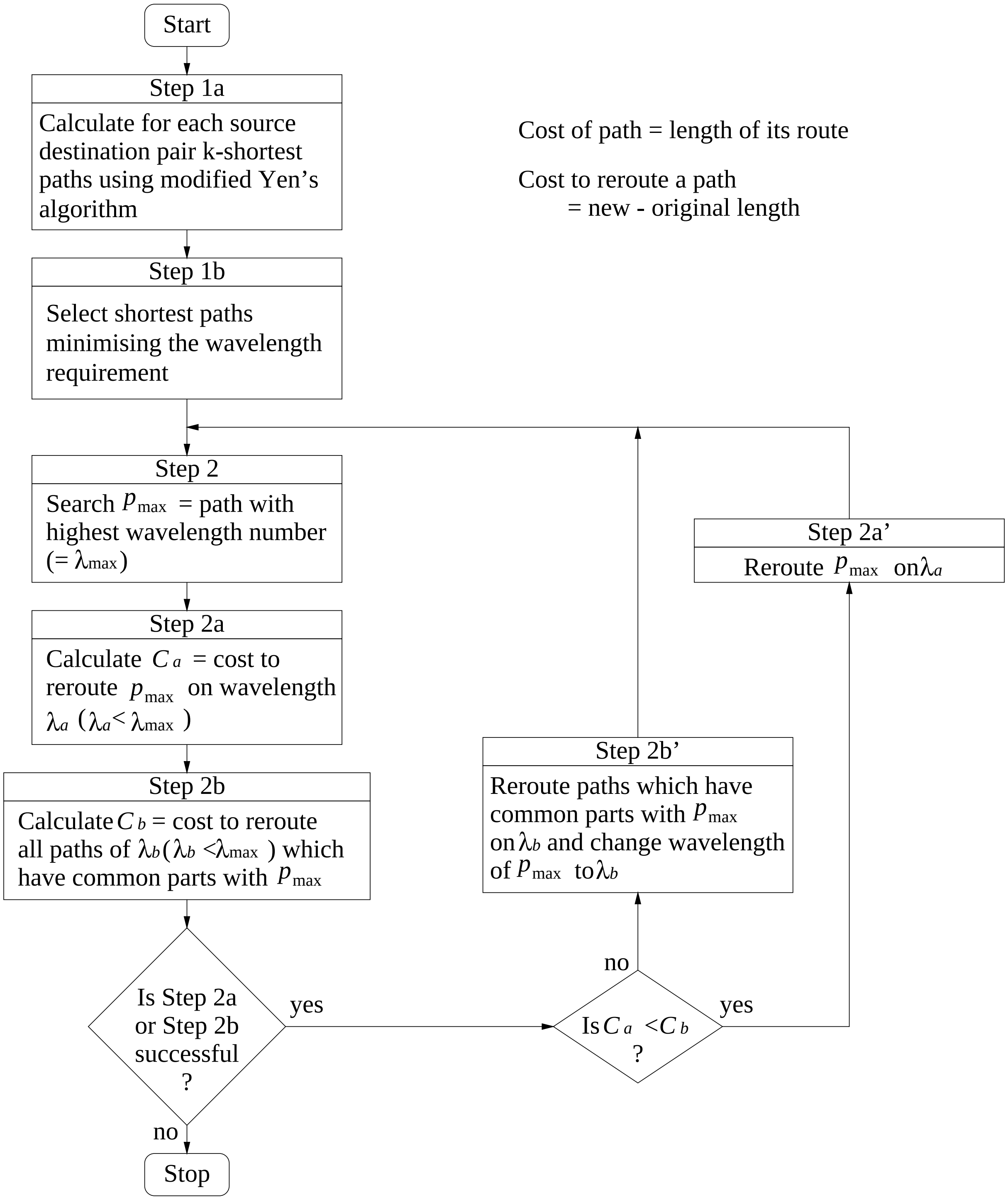


Figure C.1: Flowchart of HRWA (after [306])

Parameter	Default	Description
`wadi.firstFibreOnly`	`true`	Use only first fibre on each cable?

Table C.2: `WaveWauDijk` parameter

Parameter	Default	Description
`wahr.theK`	4	Number of alternative paths, k
`wahr.firstFibreOnly`	`true`	Use only first fibre on each cable?

Table C.3: `WaveWauHRWA` parameters

The HRWA heuristic starts (Step 1a) by determining the k-shortest paths for each node pair using the modified version of Yen's algorithm. For each wavelength channel, in arbitrary node-pair order, that path is selected with the lowest free wavelength end-to-end; if there is more than one, the shortest amongst these is chosen (Step 1b). Then, the first wavelength channel (*i.e.* wavelength-routed path; $p_{\max}$) in the network that uses the highest wavelength number ($\lambda_{\max}$), found by searching the cables in order of their end-node ids, is located (Step 2). Two different attempts are then made to move this wavelength channel to a lower wavelength number. The first (Step 2a) attempts to reroute it on a lower wavelength number (and the lowest available end-to-end) using one of the k-shortest paths (and the shortest path is selected if there is more than one at the lowest available wavelength number). The second (Step 2b) attempts to shift it to a lower wavelength number by shifting out (rerouting) all the paths blocking that particular wavelength number end-to-end. The wavelength chosen is the one that results in the smallest increase in overall path length, given the constraint that all rerouting must be to lower wavelength numbers than the original wavelength of the channel being shifted down. Both these steps have been selected as wavelength allocation operators for NOMaD, and they are described on p. 84 and illustrated by examples in Fig. 4.8. If both Step 2a and 2b are feasible, then the one that causes the least growth in overall network path length is selected. Having changed the network allocation, the heuristic then proceeds to another iteration in the hope of further improvements. It should be noted however that the modified k-shortest paths have to be recalculated, when required, each iteration due to the effect of the reallocations. If at any point no improvements can be made, the algorithm terminates.

Both Wauters & Demeester's heuristics have been implemented in NOMaD as classes `WaveWauDijk` and `WaveWauHRWA` respectively, the parameters for which are given in Tables C.2 and C.3. These parameters are directly equivalent to those used by `WaveWutti95`, and similar considerations apply.

C.3 Nagatsu *et al.*

In contrast, an overall allocation heuristic for VWP networks is described by Nagatsu *et al.* [296]. This is a two-stage algorithm, which first establishes reasonable initial routes, and then improves these in a second iterative rerouting stage, aiming for minimum NWR[3].

The initial routing (Step 1) is achieved by first determining both the minimum number of hops and the number of wavelength channels required between each source-destination node pair. Individual channels are then routed in priority order, with the highest priority given to the node pair with the greatest product of minimum hop count and channels remaining to be routed (*cf.* Wuttisittikulkij & O'Mahony's sorted variant). The routing itself is achieved by maintaining a weight associated with each cable; this is simply the number of channels (paths) using it. A channel is actually routed by assigning it to the path with the lowest sum of link weights using Dijkstra's algorithm[4]; any affected weights are then updated.

Once all the channels have been given an initial route, rerouting can begin (Step 2). For each iteration, two different attempts are made to improve the eventual NWR. Both involve identifying those cables (called *maximum-accommodating cables*) with a fibre carrying the highest number of channels, and which will therefore have to use the highest wavelength number after allocation. First, those paths which use the largest number of maximum-accommodating cables are identified as candidates, and an attempt is made to reroute them using a minimum-link-weight path. If the rerouted path would be on a lower wavelength than the original, then it is adopted, otherwise the original path is restored. If, after trying all candidates paths, no improvement is achieved, the second attempt is made. This time, any path which uses one or more maximum-accommodating cables is a candidate, and each one is rerouted onto a new path that contains as few as possible of the maximum-accommodating cables it was originally using (and not including, of course, any maximum-accommodating cables that were not part of the original path). Throughout, Nagatsu *et al.* impose an upper limit on path length: no path is ever adopted which is more than, say, 2 hops longer than the minimum-hop-count path between that node pair. In addition, they impose a limit, say 50, on the number of iterations of the second stage.

Finally, wavelength assignment is simply achieved by a greedy method, which assigns channels to the lowest available wavelength on any fibre of a cable.

[3] As with previous algorithms, some reasonable assumptions were taken to overcome ambiguities/inconsistencies in the original paper.

[4] The initial weights, which would otherwise be zero, are instead initialised to a small constant causing lowest-hop-count routing.

Parameter	Default	Description
`wnvw.firstFibreOnly`	`true`	Use only first fibre on each cable?
`wnvw.step2Count`	`50`	Maximum iterations of Step 2
`wnvw.hopIncrMax`	`2`	Maximum allowed increase in path hop count over minimum achievable

Table C.4: `WaveNagVWP` parameters

In the same paper [296], Nagatsu *et al.* also described a WP allocation heuristic derived straightforwardly from their earlier VWP one. They argue that the routes used by the VWP paths they obtain would also make excellent choices for the equivalent WP-routed network. Consequently, after their VWP algorithm has terminated, all the routes used by the network paths are adopted unchanged, but their associated fibre and wavelength allocations discarded. Then, starting from the lowest wavelength number, each path in turn is checked to see whether it can be allocated end-to-end on that wavelength. After each iteration, the wavelength number is incremented, and all the remaining paths checked again. The algorithm terminates when the last path has been successfully re-allocated.

Both of Nagatsu *et al.*'s heuristics have been implemented in NOMaD as classes `WaveNagVWP` and `WaveNagWP` respectively. The latter has no parameters of its own, being based on the VWP heuristic, so both share the parameters given in Table C.4. The first of these, as before, is used to limit the heuristic to just the first fibre of each cable. The other two correspond to the two parameters of the VWP heuristic discussed above.

Appendix D

Evolutionary Telecommunications Tables

This appendix provides detailed tables that were judged inappropriate for direct inclusion in Ch. 2. Those papers included within the scope of the evolutionary telecommunications survey of §2.3 are listed (in alphabetical order of first author surname) in Table D.1. For each paper, as well as its own reference number in the thesis bibliography, those papers it cites and which in turn cite it, from within the survey, are also recorded. In addition, classification of each paper in terms of application area (and problem element), solution representation and genetic operators (including population dynamics) are provided[1]. Next, Table D.2 shows the chronological distribution of the survey papers. Finally, Table D.3 records their classification into application groups. While the papers have usually only been included in one column (application area), they have been counted in as many rows (problem elements) as necessary.

[1] It has usually been assumed in determining classifications that authors have followed their earlier work in subsequent papers, unless they indicate otherwise.

Table D.1: Evolutionary telecommunications papers – alphabetical order

Author(s) (Year)	Ref.	Appl.	Repr.	Ops.	Cites	Cited by
Abed & Ghanta (1996)	[230]	O, w	b	E, f, x, m	—	—
Abuali *et al.* (1993)	[203]	G, x	i, p	G, S, u, x, p	—	[193], [204], [205], [206], [207]
Abuali *et al.* (1994)	[193]	G, t	p	G, f, p, m, a	[203]	[204], [205], [206], [207], [192]
Abuali *et al.* (1994)	[204]	G, x	i	x, u, h, m	[203], [193]	[205], [206], [207], [192]
Abuali *et al.* (1995)	[205]	G, x	i	u, m	[203], [204], [193]	[206]
Abuali *et al.* (1995)	[206]	G, x	i, p	G, f, h, p, m, a	[203], [204], [193], [205]	—
Abuali *et al.* (1995)	[207]	G, x	i	u, h, m, a, r	[203], [204], [193]	—
AbuAli *et al.* (1996)	[194]	G, t	b	G, f, x, u, m, a, r	[172]	—
Aiyarak *et al.* (1997)	[147]	O, t, d	g	G, t, g	[166]	—
Bentall *et al.* (1997)	[218]	G, p	h:p	G, f, p, m	[178]	—
Berry *et al.* (1995)	[209]	G, x	i	E, h, m	—	—
Brittain *et al.* (1997)	[174]	O, n, t	h:b,p	S, x, p, m	[161]	—
Calégari *et al.* (1997)	[186]	R, n	b	G, M, m	—	—
Carse *et al.* (1995)	[221]	C, r	h:r	S, s, h, m	—	[222], [223]
Celli *et al.* (1995)	[179]	G, n	b	G, f, u, m	[138]	—
Chang *et al.* (1993)	[169]	G, R, d, n	i, b	E, f, u, x, m, r	[138], [137], [142]	[171], [170]
Chang & Potter (1994)	[171]	G, R, d, n	i, b	E, f, u, x, m, r	[169], [138], [137], [142], [170]	—
Chardaire *et al.* (1995)	[178]	G, R, C, x, f, n	b	G, E, f, s, x, u, h, m, r	[177], [176]	[218]
Christensen *et al.* (1997)	[197]	G, r, x	p	G, f, p	[140], [202]	[198]

Applications: general networks (G); optical networks (O); radio networks (R); computer networks (C); dimensioning (d); node location (n); topology (t); trees (x); routing (r); restoration (p); call admission (a); wavelength allocation (w); frequency assignment (f)

Representation: binary (b); integer (i); real (r); permutation (p); tree (g); hybrid (h)

Operators: generational (G); elitist (E); steady-state (S); multi-population (M); breeder (B); fit. prop. selection (f); tournament selection (t); ranked selection (s); single- or multi-point crossover (x); uniform crossover (u); permutation crossover (p); heuristic recombination (h); tree crossover (g); mutation (m); inversion (i); adaptation (a); repair or local search (r)

continued on next page

continued from previous page

Author(s) (Year)	Ref.	Appl.	Repr.	Ops.	Cites	Cited by
Coombs & Davis (1987)	[139]	C, d	i	E, s, x, m, a, r	[137], [138]	[138], [213], [214], [163], [164]
Cox *et al.* (1991)	[140]	G, r	p	p	—	[197], [172], [185], [214], [223], [164], [198]
Crisan & Mühlenbein (1998)	[240]	R, f	i	B, h, m, r	[232]	—
Crompton *et al.* (1993)	[231]	R, f	i	E, M, f, x, u, m, r	—	[234], [235], [177], [239]
Crompton *et al.* (1994)	[232]	R, f	i, h:i	E, M, f, x, u, h, m	—	[240], [234], [235], [236], [237]
Crompton *et al.* (1994)	[233]	R, f	i, h:i	E, M, f, x, u, h, m, r	—	—
Cuppini (1994)	[238]	R, f	b	E, f, m	—	[241]
Davis & Coombs (1987)	[138]	C, d	i	E, s, x, m, a	[139], [137]	[179], [169], [171], [139], [137], [213], [214]
Davis & Coombs (1989)	[137]	C, d	i	E, s, x, m, a	[138]	[169], [171], [139], [138], [163], [164]
Davis *et al.* (1993)	[172]	G, d, r	h:i,p	S, s, h, m, r	[140]	[194], [191], [192], [151], [152], [153], [214]
Deeter & Smith (1997)	[148]	G, t, d	i	E, s, u, m	[188], [181], [168], [180]	[190], [191]
Dengiz *et al.* (1995)	[188]	G, t	b	E, f, x, m	—	[148]
Dengiz *et al.* (1997)	[189]	G, t	b	E, f, x, m	[181], [180]	—
Dengiz *et al.* (1997)	[190]	G, t	b	E, f, x, m	[181], [180], [148]	[191], [192]
Dengiz *et al.* (1997)	[191]	G, t	i	E, s, h, m, r	[172], [148], [190], [181], [180]	[192]

Applications: general networks (G); optical networks (O); radio networks (R); computer networks (C); dimensioning (d); node location (n); topology (t); trees (x); routing (r); restoration (p); call admission (a); wavelength allocation (w); frequency assignment (f)

Representation: binary (b); integer (i); real (r); permutation (p); tree (g); hybrid (h)

Operators: generational (G); elitist (E); steady-state (S); multi-population (M); breeder (B); fit. prop. selection (f); tournament selection (t); ranked selection (s); single- or multi-point crossover (x); uniform crossover (u); permutation crossover (p); heuristic recombination (h); tree crossover (g); mutation (m); inversion (i); adaptation (a); repair or local search (r)

continued on next page

continued from previous page

Author(s) (Year)	Ref.	Appl.	Repr.	Ops.	Cites	Cited by
Dengiz *et al.* (1997)	[192]	G, t	i	E, s, h, m, r	[181], [180], [172], [193], [204], [168], [190], [191]	—
Elbaum & Sidi (1995)	[210]	C, x	h:i,p,i	G, f, x, m, r	[175]	[154]
Elbaum & Sidi (1996)	[211]	C, x	h:i,p,i	G, f, x, m, r	[175], [27], [142]	—
Esbensen & Mazumder (1994)	[200]	G, x	h:b	E, s, x, m, i, r	[176]	[201]
Esbensen (1995)	[201]	G, x	h:b	E, s, x, m, i, r	[200], [144], [199], [176]	[202]
Esbensen (1995)	[202]	G, x	b	S, s, u, m, a, r	[201], [176]	[197], [198]
Fogarty *et al.* (1995)	[222]	C, r	h:r	S, s, h, m	[221]	—
Garcia *et al.* (1998)	[173]	G, d	h:i,r	S, x, m, r	—	—
Gondim (1996)	[185]	R, n	i	E, h	[140], [227]	—
He (1997)	[149]	C, d, r	i	E, f, h, m	—	—
Hesser *et al.* (1989)	[144]	G, x	b	G, f, x, m	—	[201], [199]
Hewitt *et al.* (1995)	[150]	C, t, d	h	G, s, h, m	[143], [175], [27]	[34]
Hoelting *et al.* (1996)	[215]	G, r	p	G, p, m	—	—
Hoelting *et al.* (1996)	[216]	G, r	p	p, m	—	—
Huang *et al.* (1996)	[151]	G, r, t, d	i	S, f, h, m	[172], [160]	[153]
Huang & Ma (1996)	[152]	G, r, t, d	i	M, f, h, m	[172], [160]	[153]
Huang *et al.* (1997)	[153]	G, r, t, d	i	M, f, h, m	[172], [151], [152], [160]	—
Hurley & Smith (1995)	[234]	R, f	h:i	E, M, f, h, m	[231], [232]	[236]
Hurley *et al.* (1996)	[235]	R, f	h:i	E, M, f, h, m	[231], [232]	—
Julstrom (1993)	[199]	G, x	h:i,b	S, s, x, h, m	[144]	[201], [160]
Kaminsky (1997)	[242]	R, f	h:i	S, s, u, m	—	—

Applications: general networks (G); optical networks (O); radio networks (R); computer networks (C); dimensioning (d); node location (n); topology (t); trees (x); routing (r); restoration (p); call admission (a); wavelength allocation (w); frequency assignment (f)

Representation: binary (b); integer (i); real (r); permutation (p); tree (g); hybrid (h)

Operators: generational (G); elitist (E); steady-state (S); multi-population (M); breeder (B); fit. prop. selection (f); tournament selection (t); ranked selection (s); single- or multi-point crossover (x); uniform crossover (u); permutation crossover (p); heuristic recombination (h); tree crossover (g); mutation (m); inversion (i); adaptation (a); repair or local search (r)

continued on next page

continued from previous page

Author(s) (Year)	Ref.	Appl.	Repr.	Ops.	Cites	Cited by
Kapsalis *et al.* (1993)	[176]	G, x	b	E, f, s, x, m, r	[144]	[178], [200], [201], [202], [177], [239], [213], [214]
Kapsalis *et al.* (1995)	[177]	R, f	b	G, x, u, h, m, a	[231], [176]	[178], [213]
Kapsalis & Smith (1996)	[239]	R, f	b	G, x, u, h, m, a, r	[176], [231]	—
Kirkwood *et al.* (1997)	[219]	G, r	g	M, f, g	—	[220]
Ko *et al.* (1997)	[154]	C, t, r, d	b, i, h:i	S, f, x, h, m, r	[210], [160], [163]	—
Kozdrowski *et al.* (1997)	[155]	G, r, d	h:i	S, s, u, m	—	—
Kumar *et al.* (1992)	[145]	C, t	b	E, t, h, m, r	—	[182]
Kumar *et al.* (1995)	[181]	C, t	b	E, t, h, m, r	—	[148], [189], [190], [191], [192]
Kumar *et al.* (1995)	[180]	C, n, t	b	E, t, h, m, r	—	[148], [189], [190], [191], [192], [182]
Mann *et al.* (1995)	[213]	G, r	b	—	[139], [138], [176], [177], [212]	[156]
Mann & Smith (1996)	[214]	G, r	b	S, f, t, x, u, m	[139], [140], [138], [172], [176], [141], [212]	[156]
Mann & Smith (1997)	[156]	O, r, d	b	G, E, S, f, t, x, u, m	[213], [214]	—
Markaki *et al.* (1997)	[226]	C, r	h:b	S, f, h, m, r	—	—
Michalewicz (1991)	[143]	C, t	h:b	G, h, m	—	[150], [158], [208]
Mikac & Inkret (1997)	[157]	O, t, d	b	E, f, x, m	[166]	—
Munakata & Hashier (1993)	[158]	G, r, d	i	G, f, h, m, a	[143]	[223]
Munetomo *et al.* (1997)	[223]	C, r	h:i	S, s, h, m	[140], [158], [221]	—
Nakamura & Oda (1997)	[196]	G, t	i	S, x, m	—	—

Applications: general networks (G); optical networks (O); radio networks (R); computer networks (C); dimensioning (d); node location (n); topology (t); trees (x); routing (r); restoration (p); call admission (a); wavelength allocation (w); frequency assignment (f)

Representation: binary (b); integer (i); real (r); permutation (p); tree (g); hybrid (h)

Operators: generational (G); elitist (E); steady-state (S); multi-population (M); breeder (B); fit. prop. selection (f); tournament selection (t); ranked selection (s); single- or multi-point crossover (x); uniform crossover (u); permutation crossover (p); heuristic recombination (h); tree crossover (g); mutation (m); inversion (i); adaptation (a); repair or local search (r)

continued on next page

continued from previous page

Author(s) (Year)	Ref.	Appl.	Repr.	Ops.	Cites	Cited by
Ngo & Li (1998)	[241]	R, f	b	G, f, h, m, r	[238]	—
Palmer & Kershenbaum (1994)	[159]	G, x, d	b, i	E, f, x, m	—	—
Palmer & Kershenbaum (1995)	[160]	G, x, d	b, i	E, f, x, m	[199]	[151], [152], [153], [154]
Pan & Wang (1991)	[141]	C, r, d	i	G, f, h, m	—	[214], [212], [224], [225]
Paul & Tindle (1996)	[161]	O, n, x, d	h	h	—	[174], [162]
Paul *et al.* (1996)	[162]	O, n, x, d	h	h	[161]	—
Pierre & Legault (1996)	[163]	C, t, d	b	G, f, x, m	[139], [137]	[154]
Pierre & Legault (1998)	[164]	C, t, d	b	G, f, x, m	[140], [139], [137]	—
Potter *et al.* (1992)	[142]	G, R, d, n	i, b	E, f, u, h, x, m	—	[169], [171], [211], [170]
Potter *et al.* (1993)	[170]	G, R, d, n	i, b	E, f, u, x, m, r	[169], [142]	[171]
Qin *et al.* (1997)	[165]	C, t, d	b	E, f, x, m, r	—	—
Routen (1994)	[175]	G, t, x, n	i	G, f, x, m	—	[210], [211], [150], [208]
Saha & Chakraborty (1997)	[195]	G, t	b	G, f, x, m	—	—
Sandalidis *et al.* (1998)	[243]	R, f	b	G, s, m	—	—
Servet *et al.* (1997)	[217]	G, r	b, r	E, f, x, h, m	—	—
Shami *et al.* (1997)	[220]	G, r	g	M, f, g	[219]	—
Shimamoto *et al.* (1993)	[212]	G, r	i	S, f, x, m	[141]	[213], [214], [224], [225]
Sinclair (1993)	[27]	G, r	b	G, f, x, m	—	[211], [150], [224], [225]
Sinclair (1995)	[166]	O, t, d	b	E, f, x, m	—	[147], [157], [34], [36], [29], [184]
Sinclair (1997)	[34]	O, t, d	h	G, t, x, h, m, a, r	[166], [150]	[35], [36]
Sinclair (1998)	[35]	O, r, w	h	G, t, h, m, a, r	[34], [29]	—

Applications: general networks (G); optical networks (O); radio networks (R); computer networks (C); dimensioning (d); node location (n); topology (t); trees (x); routing (r); restoration (p); call admission (a); wavelength allocation (w); frequency assignment (f)

Representation: binary (b); integer (i); real (r); permutation (p); tree (g); hybrid (h)

Operators: generational (G); elitist (E); steady-state (S); multi-population (M); breeder (B); fit. prop. selection (f); tournament selection (t); ranked selection (s); single- or multi-point crossover (x); uniform crossover (u); permutation crossover (p); heuristic recombination (h); tree crossover (g); mutation (m); inversion (i); adaptation (a); repair or local search (r)

continued on next page

continued from previous page

Author(s) (Year)	Ref.	Appl.	Repr.	Ops.	Cites	Cited by
Sinclair (1998)	[36]	O, r, w	h	G, t, h, m, a, r	[34], [166]	—
Sobotka (1992)	[146]	G, t	?	E, h	—	—
Srivastava *et al.* (1997)	[182]	G, t, n	b	E, t, h, m, r	[145], [180]	—
Tan & Sinclair (1995)	[29]	O, r, w	b	G, f, x, m	[166]	[35]
Tanaka & Berlage (1996)	[167]	G, t, d	h:b	G, f, h, r	—	—
Tang *et al.* (1997)	[187]	R, n	h:b,r	S, s, x, h, m	—	—
Tanterdtid *et al.* (1997)	[224]	C, r	i	S, f, x, m	[141], [27], [212]	—
Tanterdtid *et al.* (1997)	[225]	C, r	i	S, f, x, m	[141], [27], [212]	—
Valenzuela *et al.* (1997)	[236]	R, f	p	S, s, p, m	[232], [234]	—
Valenzuela *et al.* (1998)	[237]	R, f	p	S, s, p, m	[232]	—
Walters & Smith (1995)	[168]	G, x, d	i	E, s, h, m	—	[148], [192]
Webb *et al.* (1997)	[183]	G, n, t	b	G, f, x, m, r	—	—
Webb *et al.* (1998)	[184]	G, n, t	b	G, f, x, m, r	[166]	—
Yener & Rose (1994)	[227]	R, a	b	G, f, x, m	—	[185], [228], [229]
Yener & Rose (1995)	[228]	R, a	b	G, f, x, m	[227]	[229]
Yener & Rose (1997)	[229]	R, a	b	G, f, x, m	[227], [228]	—
Zhao *et al.* (1995)	[208]	G, x	h	E, s, h, m, r	[143], [175]	—
Zhu *et al.* (1998)	[198]	G, r, x	p	G, f, p	[197], [140], [202]	—

Applications: general networks (G); optical networks (O); radio networks (R); computer networks (C); dimensioning (d); node location (n); topology (t); trees (x); routing (r); restoration (p); call admission (a); wavelength allocation (w); frequency assignment (f)

Representation: binary (b); integer (i); real (r); permutation (p); tree (g); hybrid (h)

Operators: generational (G); elitist (E); steady-state (S); multi-population (M); breeder (B); fit. prop. selection (f); tournament selection (t); ranked selection (s); single- or multi-point crossover (x); uniform crossover (u); permutation crossover (p); heuristic recombination (h); tree crossover (g); mutation (m); inversion (i); adaptation (a); repair or local search (r)

Year	Count	Papers
1987	2	[139], [138]
1988	0	
1989	2	[137], [144]
1990	0	
1991	3	[140], [143], [141]
1992	3	[145], [142], [146]
1993	10	[203], [169], [231], [172], [199], [176], [158], [170], [212], [27]
1994	10	[193], [204], [171], [232], [233], [238], [200], [159], [175], [227]
1995	25	[205], [206], [207], [209], [221], [179], [178], [188], [210], [201], [202], [222], [150], [234], [235], [177], [181], [180], [213], [160], [166], [29], [168], [228], [208]
1996	14	[230], [194], [211], [185], [215], [216], [151], [152], [239], [214], [161], [162], [163], [167]
1997	33	[147], [218], [174], [186], [197], [148], [189], [190], [191], [192], [149], [153], [242], [219], [154], [155], [156], [226], [157], [223], [196], [165], [195], [217], [220], [34], [182], [187], [224], [225], [236], [183], [229]
1998	10	[240], [173], [241], [164], [243], [35], [36], [237], [184], [198]
Total	112	

Table D.2: Evolutionary telecommunications papers – chronological order

Problem element	Application area			
	General networks (G)	Optical networks (O)	Radio networks (R)	Computer networks (C)
Dimensioning (d)	[169], [171], [172], [148], [173], [151], [152], [153], [155], [158], [159], [160], [142], [170], [167], [168]	[147], [156], [157], [161], [162], [166], [34],	[169], [171], [142], [170]	[139], [138], [137], [149], [150], [154], [141], [163], [164], [165]
Node location (n)	[179], [169], [171], [142], [170], [175], [182], [183], [184]	[174], [161], [162]	[186], [169], [171], [185], [142], [170], [187]	[178], [180]
Topology (t)	[193], [194], [148], [188], [189], [190], [191], [192], [151], [152], [153], [196], [175], [195], [146], [182], [167], [183], [184]	[147], [174], [157], [166], [34]	—	[150], [154], [145], [181], [180], [143], [163], [164], [165]
Trees (x)	[203], [204], [205], [206], [207], [209], [178], [197], [200], [201], [202], [144], [199], [176], [159], [160], [175], [168], [208], [198]	[161], [162]	—	[210], [211]
Routing, restoration & call admission (r/p/a)	[218], [197], [140], [172], [215], [216], [151], [152], [153], [219], [155], [213], [214], [158], [217], [220], [212], [27], [198]	[156], [35], [36], [29]	[227], [228], [229]	[221], [222], [149], [154], [226], [223], [141], [224], [225]
Wavelength allocation & frequency assignment (w/f)	—	[230], [35], [36]. [29]	[178], [240], [231], [232], [233], [238], [234], [235], [242], [177], [239], [241], [243], [236], [237]	—

Table D.3: Evolutionary telecommunications papers – application groups

Appendix E

Detail of NOMaD

This appendix provides details of NOMaD that were judged inappropriate for Chapters 3, 4 and 6. It is divided into eight sections: §E.1 gives details of the NOMaD file formats, and §E.2 provides cross reference tables for NOMaD's classes. The central features of NOMaD's network model are presented in §E.3. NOMaD's support for hierarchical networks and node clustering is discussed in §E.4, and §E.5 presents details of various iterator classes. Then in §E.6, class diagrams and descriptions of classes from the `Utilities` class category are provided. Greater detail on NOMaD's testing and verification strategy, initially reviewed in §3.8, is given in §E.7. The final section (§E.8) considers the design and implementation of the NOMaD software and makes suggestions for further work.

E.1 NOMaD File Formats

To simplify the input of both network specifications (`.net`) and algorithm parameters (`.nmd`), as well as the output of results (`.out`), a standard file format was adopted (after [321]). Each line in a NOMaD file is blank or a comment (and hence ignored) or a variable-value pair (*i.e.* a *parameter*). Comment lines simply begin with a `#`. Variable-value pairs have the following form:

```
variable = value
```

where `variable` is a string of non-white characters, and `value` is an arbitrary string. The variable-value pairs in a file can be included in any order, although clearly they are more readable to a user if some structure is imposed. However, if a variable appears more than once in a file, the last value is the one retained.

E.1.1 NOMaD Network (`.net`) Files

Network specification files are usually designated by a `.net` suffix, and provide either the node locations (plus other details) and traffic distribution required for network topology design, or the more complete network descriptions necessary for routing, fibre choice and wavelength allocation.

The variable-value pairs used for network topology design are listed in Table E.1, where the 'M/O' column indicates whether a pair is mandatory (M) or optional (O). The additional pairs required to specify a topology are given in the first part of Table E.2; the second part lists those necessary for routing, fibre and wavelength allocation. The designation of M or O is fairly arbitrary for some pairs in the latter table as, for example[1], `links[?].nodeIdS` is only mandatory if the optional `numLinks` has been specified with a value greater than zero!

E.1.2 NOMaD Parameter (`.nmd`) Files

NOMaD parameter files, designated by a `.nmd` suffix, provide all those parameters (*i.e.* variable-value pairs) that do not directly specify either a network design problem or a network description. The parameters are tabulated here in the context of the classes which make use of them. However, the class descriptions themselves can be found in Ch. 4, for the majority of class categories, Appendix C for the wavelength-allocation heuristics, and later in this Appendix for the `Utilities` class category (§E.6).

The parameters for class `Network` are given in Table E.3, and discussed on p. 256 in the main body of the thesis.

Next, the parameters for the design assessment metrics of §4.1 are tabulated. Those for classes `AlesCost` and `AlesRouter` are given in Tables E.4 and E.5, respectively. Together they implement Network Cost Model A, discussed in §4.1.1 on p. 68. Next, those for class `NetWaveReq` are detailed in Table E.6. This class implements the network wavelength requirement metric, described in §4.1.2 on p. 71. Finally, the parameters for class `PhyNetCost` are listed in Table E.7. This class implements Network Cost Model B, discussed in §4.1.3 on p. 72.

The next four tables give the parameters for the `GeneticAlgorithm`, `Population`, `Individual` and `OperatorPool` classes *i.e.* Tables E.8, E.9, E.10 and E.11, respectively. They are all part of the implementation of the author's genetic-algorithm (GA)/heuristic hybrid approach, discussed in §4.2 on p. 74.

[1] In this appendix, '`?`' has been used as a wildcard.

Variable	Example Value	M/O	Description
`numnodes`	`15`	M	Number of nodes; must be ≥ 2
`kmPerUnit`	`1`	O	Multiplier for node `x` and `y` locations; defaults to 200.0, must be > 0.0
`nodes[0].name`	`Paris`	O	Node 0 name; defaults to `Node0`, `Node1`, ...
`nodes[0].x`	`184.9`	M	Node 0 `x` location; must be ≥ 0.0
`nodes[0].y`	`970.1`	M	Node 0 `y` location; must be ≥ 0.0
`nodes[0].levels`	`{0,1}`	O	Node 0 levels; defaults to `{0}`, each entry in the list of levels must be in the range `0`–`31`
`nodes[0].clusters`	`{0,1}`	O	Node 0 clusters; defaults to `{0}`, each entry in the list of clusters must be in the range `0`–`31`
`traMatrix[0][1]`	`3.480`	M	Traffic from node 0 to node 1; must be ≥ 0.0, except self traffic (*e.g.* node 0 to node 0) which must be 0.0
`lenMatrix[0][1]`	`742.5`	O	Length of (potential) link between node 0 and node 1; defaults to the Euclidean distance between end nodes, must be > 0.0, and only lengths between node `x` and node `y` where `x` < `y` are recognised

Table E.1: Network file variable-value pairs

Variable	Example Value	M/O	Description
`numLinks`	`42`	O	Number of links; must be ≥ 0
`links[0].nodeIdS`	`1`	M	End node of link 0; must be included for all links in the range `0`–`(numLinks-1)`, value must be in the range `0`–`(numNodes-1)`
`links[0].nodeIdT`	`2`	M	Other end node of link 0; must be included for all links in the range `0`–`(numLinks-1)`, value must be in the range `0`–`(numNodes-1)`, no self loops allowed, and all links must be between different pairs of end nodes
`links[0].levelS`	`1`	O	Connect level of link 0 at end node `S`; defaults to `0`, must be in the range `0`–`31`, and node `S` must be on this level
`links[0].levelT`	`1`	O	Connect level of link 0 at end node `T`; defaults to `0`, must be in the range `0`–`31`, and node `T` must be on this level
`cabMatrix[0][1].numFibres`	`2`	M	Number of fibres in cable between node 0 and node 1; required for all links; must be > 0
`net.defaultNumFibres`	`2`	O	Default number of fibres; equivalent to specifying `cabMatrix[?][?].numFibres` for all possible links, although overridden by any specific `cabMatrix[?][?].numFibres` pairs included; must be > 0
`net.lambdaCap`	`10.0`	O	Capacity of a single wavelength channel; defaults to `2.5`; must be ≥ 0.0

Table E.2: Additional network file variable-value pairs

Parameter	Default	Description
`net.metric`	`ALCT`	Metric used for network self-assessment; default is `AlesCost` (§4.1.1)
`net.router`	`ALES`	Router used to route the network prior to self-assessment; default is `AlesRouter` (§4.1.1)

Table E.3: `Network` parameters

The parameters for the wavelength-allocation random k-shortest paths router class, `WaveRandKSP` (see p. 82), are given in Table E.12. The first of these (`wrks.theK`) is simply the constant k. The second, `wrks.firstFibreOnly`, occasionally avoids the need to edit the `.net` file from which the `PhyNetwork` was built: if `true`, the router uses only the *first* fibre on each cable in the network. This option can be useful if a single-fibre network design is under consideration. The final parameter (`wrks.cacheKSP`) is only appropriate if the `WaveRandKSP` object is to be applied to multiple `PhyNetwork`s all with identical topologies (as happens in a GA/heuristic hybrid!). The k-shortest paths used are generated by `WaveRandKSP`'s `Router` base class (using Yen's algorithm [320]): for identical topologies, an identical set of paths is always generated. Consequently, if the k-shortest paths are obtained the first time the `WaveRandKSP` router is applied, and then cached, they can be re-used on subsequent applications with a considerable saving in execution time.

Those for the wavelength-allocation k-shortest path mutation operator class, `WaveKSPMutate` (see p. 84), are detailed in Table E.13. They are directly equivalent to those used by `WaveRandKSP` above, and similar considerations apply.

Next, the parameters for the wavelength-allocation path reroute operator class, `WaveWauReroute` (see p. 84), are tabulated in Table E.14. Both of these are equivalent to the similarly named parameters of `WaveRandKSP` above, but unfortunately path cacheing cannot be applied, as the results of Wauters and Demeester's modified Yen's algorithm depend not only on the network topology, but also on the wavelength occupancies of the cables. Consequently, it must be run afresh whenever a k-shortest path decision is made.

Those for the wavelength-allocation path shift-out operator class, `WaveWauShiftOut` (see p. 85), are listed in Table E.15. These parameters are directly equivalent to those used by `WaveWauReroute` above, and again similar considerations apply.

Finally, the parameters for the wavelength-allocation k-shortest path single-point crossover operator class, `WaveKSPCO` (see p. 85), are given in Table E.16. They are directly equivalent to those used by `WaveRandKSP` above (*i.e.* path cacheing can be used here), and once again, similar considerations apply.

Parameter	Default	Description
`ales.eMin`	`200.0`	Constant K_0, shared with `AlesRouter`
`ales.eLink`	`100.0`	Constant K_n, shared with `AlesRouter`
`ales.traToCapK`	`1.4`	Constant K_T
`ales.capUnit`	`2.5`	Constant K_G
`ales.nnPenalty`	`500,000.0`	Penalty P_N
`ales.apPenalty`	`250,000.0`	Penalty P_R
`ales.alpha`	`1.0`	Constant α

Table E.4: `AlesCost` parameters

Parameter	Default	Description
`ales.eMin`	`200.0`	Constant K_0, shared with `AlesCost`
`ales.eLink`	`100.0`	Constant K_n, shared with `AlesCost`
`ales.altPathWgtK`	`1.0`	Constant K_R

Table E.5: `AlesRouter` parameters

Parameter	Default	Description
`nwr.nnPenalty`	`5.0`	Node-to-node penalty
`nwr.pfPenalty`	`0.0`	Per-fibre penalty

Table E.6: `NetWaveReq` parameters

Parameter	Default	Description
`pnct.eMin`	`200.0`	Constant K_0
`pnct.eLink`	`100.0`	Constant K_n
`pnct.nnPenalty`	`250,000.0`	Penalty P_N
`pnct.alpha`	`1.0`	Constant α
`pnct.beta`	`1.0`	Constant β
`pnct.gamma`	`0.5`	Constant γ

Table E.7: `PhyNetCost` parameters

Parameter	Default	Description
`maxTrials`	—	Maximum number of trials; must be ≥ 1

Table E.8: `GeneticAlgorithm` parameter

Parameter	Default	Description
`pop.size`	—	Population size
`pop.winSize`	`1`	Window size for best-in-window credit assignment, w
`pop.select`	—	Selection operator
`pop.tournament.size`	`4`	Tournament size for tournament selection
`pop.initOp`	`RLS`	Population initialisation operator; default is `RandomLinks` (§4.2.2, p. 80)

Table E.9: `Population` parameters

Parameter	Default	Description
`pop.phyNetwork`	`false`	Construct the individual using a `PhyNetwork` (`true`) or a `Network` (`false`)

Table E.10: `Individual` parameter

Parameter	Default	Description
`opp.numOps`	—	Number of operators (excluding `Reproduction`), s; must be ≥ 1
`opp.ops[0].name`	—	Name of operator 0
`opp.ops[0].prob`	—	Initial probability of operator 0
`opp.adaptProp`	`0.15`	Adaptation proportion, ψ; $0 \leq \psi \leq 1$
`opp.adaptNumGens`	`4`	Adaptation generation gap, ϵ; must be ≥ 1
`opp.creditK`	`0.5`	Credit constant, κ; must be ≥ 0
`opp.fitCredType`	`best`	Credit type: `best`, `winBest`, `median` or `parent`
`opp.fitCredDiv`	`children`	Credit divisor: `children` or `trials`

Table E.11: `OperatorPool` parameters

Parameter	Default	Description
`wrks.theK`	`4`	Number of alternative paths, k
`wrks.firstFibreOnly`	`true`	Use only first fibre on each cable?
`wrks.cacheKSP`	`true`	Cache k-shortest paths?

Table E.12: `WaveRandKSP` parameters

Parameter	Default	Description
`wkm.theK`	`4`	Number of alternative paths, k
`wkm.firstFibreOnly`	`true`	Use only first fibre on each cable?
`wkm.cacheKSP`	`true`	Cache k-shortest paths?

Table E.13: `WaveKSPMutate` parameters

Parameter	Default	Description
`wwrr.theK`	`4`	Number of alternative paths, k
`wwrr.firstFibreOnly`	`true`	Use only first fibre on each cable?

Table E.14: `WaveWauReroute` parameters

Parameter	Default	Description
`wwso.theK`	`4`	Number of alternative paths, k
`wwso.firstFibreOnly`	`true`	Use only first fibre on each cable?

Table E.15: `WaveWauShiftOut` parameters

Parameter	Default	Description
`wkco.theK`	`4`	Number of alternative paths, k
`wkco.firstFibreOnly`	`true`	Use only first fibre on each cable?
`wkco.cacheKSP`	`true`	Cache k-shortest paths?

Table E.16: `WaveKSPCO` parameters

Variable	Example Value	Description
`plsMatrix[0][1]`	`{P0(0[0:2],8[0:2],12[0:2],1)}`	Path-loss sequence for source node 0, destination node 1
`cabMatrix[0][2].occup[1]`	`077777`	Occupancy of fibre 1 on the cable between node 0 and node 2

Table E.17: Output file variable-value pairs

E.1.3 NOMaD Output (`.out`) Files

The format of NOMaD output files, usually designated by a `.out` suffix, varies widely between applications. However, it is usual practice to at least output the network design obtained, via the `Network` or `PhyNetwork` `print()` member function, as appropriate.

For network topology design (`Network`), this will result in variable-value pairs for all the entries in Table E.1, plus all the entries in the first part of Table E.2. In addition, the path-loss sequence matrix will be output as a series of variable-value pairs (Table E.17, `plsMatrix`).

For routing, fibre and wavelength allocation (`PhyNetwork`), as well as all the data for a `Network`, the path-loss sequences will now contain wavelength-routed paths (not simple paths). In addition, `net.lambdaCap` and `cabMatrix[?][?].numFibres` (Table E.2) entries will be provided, plus an extra (redundant) set of variable-value pairs (Table E.17, `cabMatrix[?][?].occup[?]`) recording individual fibre occupancies; these are intended for display and consistency use, as extracting the information from the `plsMatrix` entries would require considerable processing, especially at script level.

Beyond the (`Phy`)`Network` related variable-value pairs, `ga` for example (§3.9), outputs data including: (for each generation) average fitness, lowest fitness and operator probabilities; as well as number of operators, operator names, total number of generations, best fitness value for the run, and the generation in which it was achieved.

E.2 Class Cross-references

Tables E.18–E.26 list the main classes used in NOMaD and, for each class, record references to the appropriate descriptions and diagrams from Ch. 3–4, Appendix C and later in this Appendix. Where one figure is the most important for a particular class, its figure number has been shown in bold.

Class	Text (pp.)	Figure(s)
`Network`	56, 253	**3.4(a)**, 3.4(b), 3.5, E.4, E.5, E.8, 4.1, E.13
`Node`	56, 257	**3.4(a)**, E.4, E.13
`Link`	56, 257	**3.4(a)**
`PathLossSeq`	56, 263	3.4(a), 3.4(b), **E.5**
`Path`	56, 260	3.4(a), **E.4**, E.5, E.8

Table E.18: `NetworkModel` category — `Network` classes

Class	Text (pp.)	Figure(s)
`PhyNetwork`	56, 267	**3.4(b)**, E.8, 4.1
`Cable`	56, 268	**3.4(b)**, E.8
`PhyPath`	56, 272	3.4(b), **E.8**
`Join`	275	**E.8**

Table E.19: `NetworkModel` category — `PhyNetwork` classes

Class	Text (pp.)	Figure(s)
`Metric`	67	3.4(a), **4.1**
`AlesCost`	68	**4.1**
`AlesRouter`	68	**E.2(b)**
`PhyNetCost`	72	**4.1**
`NetWaveReq`	71	**4.1**

Table E.20: `NetworkModel` category — `Metric` classes (and `AlesRouter`)

Class	Text (pp.)	Figure(s)
`PlsIter`	280	3.4(b), **E.5**
`PathIter`	278	**E.4**
`PathLinkIter`	279	**E.4**
`CableIter`	280	**3.4(b)**
`PathJoinIter`	280	**E.8**
`PathHeap`	259	

Table E.21: `NetworkModel` category — minor classes

Class	Text (pp.)	Figure(s)
`GeneticAlgorithm`	56, 74	**3.5**
`Population`	56, 76	**3.5**
`Individual`	56, 77	**3.5**
`OperatorPool`	56, 78	**3.5**
`Reproduction`	79	**3.5**

Table E.22: `GA` category classes (and `Reproduction`)

Class	Text (pp.)	Figure(s)
`Operator`	56, 259, 79	3.5, E.2(a), **4.3**, 4.4(a), 4.4(b)
`UnaryOp`	56, 259, 79	3.5, E.2(a), **4.3**, 4.4(a), 4.7(a)
`BinaryOp`	56, 259, 79	E.2(a), **4.3**, 4.4(b), 4.7(b)
`Router`	56, 259	**E.2(a)**
`UnaryRouter`	259	3.4(a), **E.2(a),** E.2(b), 4.6
`BinaryRouter`	259	**E.2(a)**

Table E.23: `Operators` category — abstract classes

Class	Text (pp.)	Figure(s)
`RandomLinks`	80	**4.4(a)**
`MutateLink`	81	**4.4(a)**
`DegreeTwo`	81	**4.4(a)**
`DegreeDec`	81	**4.4(a)**
`RandomAddLink`	81	**4.4(a)**
`RandomRemLink`	81	**4.4(a)**
`RandomChgLink`	82	**4.4(a)**
`RandomAddLinks`	82	**4.4(a)**
`RandomRemLinks`	82	**4.4(a)**
`RandomChgLinks`	82	**4.4(a)**
`SPLinkCO`	82	**4.4(b)**
`LinkGroupCO`	82	**4.4(b)**

Table E.24: `Operators` category — topology classes

Class	Text (pp.)	Figure(s)
`WaveRandKSP`	82	**4.6**
`WaveNop`	84	**4.6**
`WaveWutti95`	229	**4.6**
`WaveWauDijk`	230	**4.6**
`WaveWauHRWA`	230	**4.6**
`WaveNagVWP`	233	**4.6**
`WaveNagWP`	233	**4.6**
`WaveKSPMutate`	84	**4.7(a)**
`WaveWauReroute`	84	**4.7(a)**
`WaveWauShiftOut`	85	**4.7(a)**
`WaveKSPCO`	85	**4.7(b)**

Table E.25: `Operators` category — wavelength-allocation classes

Class	Text (pp.)	Figure(s)
`ArrayList`	281	E.4, E.8
`AlIter`	281	E.4, E.8
`List`	282	E.5, **E.12(a)**
`ListNode`	282	**E.12(a)**
`ListIter`	282	E.5, **E.12(a)**
`ParamDataBase`	284	3.4(a), **E.13**
`Parameter`	284	**E.13**
`RandomGen`	286	

Table E.26: `Utilities` category classes

E.3 Network Modelling

The primary aim of NOMaD's network model was to provide a set of classes capable of directly supporting network design in the application areas outlined in §3.1 above: both network topology design, and routing, fibre choice and wavelength allocation of multi-wavelength all-optical transport networks. In addition, it was hoped that the set of classes provided might be able to form the basis of design work in other areas, such as optical access networks, through derivation [16] and (if necessary) minor modifications to the existing classes. Further, based on the lessons learned in developing the model, it was envisaged that the author, and others, would be able to design even more robust and reusable network modelling class libraries in the future.

E.3.1 Overall Structure

The network modelling classes discussed in this subsection are those of more general applicability, including `Network`, `Node`, `Link`, `Router` and `UnaryRouter`. Those classes which are more specific to particular design problems are described in §E.3.2 and §E.3.3 below.

`Network` class

Arguably the most important class in NOMaD's network model is the `Network`, illustrated in Fig. 3.4(a) on p. 58. Both its data structure and functionality are complex as, in association with other classes, `Network` (or its derived class for wavelength-routed networks, `PhyNetwork`) must be able to represent everything from, for example, a simple 4-node network design problem (simply a traffic matrix and a set of node locations) up to, say, a 20-node or larger multi-wavelength all-optical transport network. Including, in the latter case, the details of all its routing, down to individual fibre and wavelength assignments on every link of a large number of wavelength channels.

The main `Network` constructor [16] takes two parameter database objects: one built from the network specification file, and the other from the NOMaD parameter file (see §3.6 for example code). Large numbers of variable-value pairs (*i.e.* parameters) in the network parameter database are then retrieved as necessary during construction, although a `const` pointer [16] to the network parameter database (`netDB`) is retained for possible further reference (it can also be obtained externally using a member function). However, in contrast, only two parameters are actually taken by the constructor from the NOMaD parameter database (Table E.3); their application is discussed below. (For NOMaD file formats, see §E.1.)

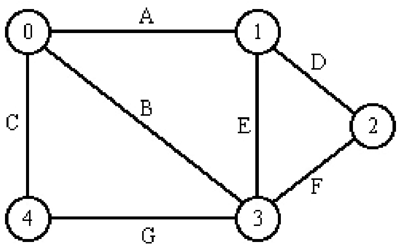
To represent the nodes of the network, such as optical cross connects, the `Network` class contains a count of the number of nodes (`numNodes`), and an array of pointers to the node objects (class `Node`) themselves (`nodes`). These are made available through respective member functions, the latter returning a `const` pointer to a node object (selected by node identification number; id), thus preventing modification by the user of the `Network`'s private node objects [16].

The network traffic matrix (`traMatrix`) is represented by a two-dimensional array of size `numNodes` $\times$ `numNodes` of double-precision floating-point numbers[2]. These represent the traffic originating in each direction between node pairs. Using a source-destination node id pair, individual traffic values can be retrieved with a member function. Example traffic matrices used in some of the network design problems studied in Ch. 5 are given in §F.2.

The network topology (if any) is represented by a count of the number of links (`numLinks`), and an adjacency matrix (`adjMatrix`) [25, 287]. This consists of a two-dimensional array of size `numNodes` $\times$ `numNodes` of pointers to link objects (class `Link`): if a link exists between two nodes, then the corresponding entry in the array will contain a valid link pointer; otherwise it is void. Fig. E.1 illustrates the concept for a small 5-node network with 7 links where, in the adjacency matrix, link letters are shown (rather than pointers!). Note that, as the links are bidirectional, they each appear in two places in the matrix (*e.g.* link D connects node 2 to 1, as well as node 1 to 2). In addition, as nodes are not permitted to link to themselves ('self loops'), the diagonal is empty. Although in the network model only one link is allowed between node pairs (*i.e.* no parallel links), support for multiple-fibre connections is provided by the `Cable` class for wavelength-routed networks: see §E.3.3 below. Within NOMaD, two adjacency matrices are actually used: `adjMatrix` is the 'true' adjacency matrix, whereas the *potential* adjacency matrix (`potentialLinks`) stores pointers to all possible links (*i.e.* all those that would be required in a full-mesh superset of the current network). By simply copying pointer values from the potential to the actual adjacency matrix when a link is added to the network topology, the time overhead of `Link` construction (and when a link is removed, destruction [16]) is eliminated. This particular use of the adjacency matrix, combined with the need to access link objects efficiently via their end-node ids, was the primary reason for the selection of a (node-node) adjacency matrix representation for the network topology, rather than the equivalent node adjacency list [25, 287] or node-link incidence matrix [25, 292].

Users can obtain the number of links using a member function. A `const` pointer to individual

[2]Since the traffic matrix is dynamically allocated, as are most objects in NOMaD, its structure is not simply that of a two-dimensional array, but a more complex pointer based one; however, the exact details need not concern us here.



(a) Network

From/To	0	1	2	3	4
0		A		B	C
1	A		D	E	
2		D		F	
3	B	E	F		G
4	C			G	

(b) Matrix

Figure E.1: Example adjacency matrix

links can be retrieved either using the link id, or the node ids of the two end nodes; the latter option also permits the adjacency matrix to be queried, as a void pointer is returned if two nodes are not connected. To allow modification of the adjacency matrix, member functions to add and remove links are provided, with links specified by their end-node ids; the former permits link connect levels to be supplied as optional parameters if an hierarchical network is being constructed (see below). Further, although a node's degree (*i.e.* number of attached links) could be determined by querying the adjacency matrix, a member function is provided to automate this for a specified node id.

NOMaD's network model directly supports the concept of network hierarchy, by enabling nodes to be on one or more levels, and links to be connected to a particular level at each end. In addition, as an aid to the design of cluster-based architectures, such as multi-ring networks, the network model provides direct support for node clusters; these are also intended to be useful in the development of any cluster-based operators or overall design heuristics. However, neither of these capabilities have been used in the experimental work reported in Ch. 5, although they have been utilised elsewhere[3]. Further details on NOMaD's support for network hierarchies and node clustering is given in §E.4.

The static routing of traffic through the network is represented using either simple paths (`Path`) or wavelength-routed paths *i.e.* paths that also record fibre choice and wavelength allocation (`PhyPath`). The former are used in `Network`s for network topology design; the latter in wavelength-routed networks (`PhyNetwork`s) for the routing, fibre and wavelength allocation problem. For each source-destination node pair in the network, the associated path(s), of the appropriate type, are listed in a path-loss sequence [286] (`PathLossSeq`): an ordered sequence of simple or wavelength-routed paths. These in turn are recorded in the network's path-loss sequence matrix (`plsMatrix`), a two-dimensional array of size `numNodes` × `numNodes` of pointers to path-loss sequence objects. Al-

[3]Specifically, by Noel Parnis for OPEN.

though not enforced by NOMaD, the author has adopted the convention that for network topology design (class `Network`), each of the path-loss sequences records just two simple paths, representing the primary and restoration routes, whereas for wavelength allocation (class `PhyNetwork`), each wavelength-routed path in a sequence represents a single wavelength channel of the primary route (and restoration routes are not included). The details of the `Path`, `PhyPath` and `PathLossSeq` classes are given in the appropriate sections below (§E.3.2 and §E.3.3), where `Network`'s member functions for the manipulations of paths, path-loss sequences and the path-loss sequence matrix will also be discussed.

Although it is possible to assess a network object against a particular metric (*e.g.* cost or network wavelength requirement) by applying the appropriate metric object (derived from the abstract `Metric` class, see §4.1), network objects also store a pointer to a metric object to allow self-assessment (`metric`). Further, as (re-)routing may need to be carried out prior to the assessment of a design (although see `WaveNop` in §4.2.3), the network object also stores a pointer to a unary router object (`router`). Thus, when the `Network`'s assessment member function is invoked, first the router's routing member function is used to reroute the network, and then the assessment itself is carried out by using the metric's assessment member function. The user's choices of which metric and router to use are made via the NOMaD parameter file: see Table E.3 on p. 247. To avoid unnecessary re-assessment however, the network object monitors internally whether any changes have been made to its topology or routing; if not, then the last assessment result, which will have been cached within the network object, is returned instead; no actual rerouting or assessment is carried out. Given that assessing a network design is often computationally intensive, this approach can therefore lead to considerable savings in processing time. Nevertheless, cacheing of network assessments in this way has implications for performance monitoring of GAs in terms of trials (*i.e.* number of assessments), and is thus also discussed in §4.2.

The output of networks can be carried out with either of two member functions: one of these writes network data in variable-value pairs on an output stream (`ostream` [16]) *i.e.* a file, whereas the other returns the data in a dynamically-allocated parameter database.

In addition to those aspects of `Network`'s data structure and functionality already presented in the preceding paragraphs, where it has been judged appropriate, description of several further elements has been included in the discussion of other classes within the network model. For example, in support of routing, the `Network` class provides several member functions to allow 'link marking'; however, rather than here, this concept will be discussed below with reference to the `Link` class.

`Node` class

The `Node` class is not a 'true' class, in the sense that it cannot exist independently of a particular network object. However, in this, NOMaD's network model reflects reality, because nodes do not exist in isolation, but only as part of a network. Although usually used to represent an optical cross connect (OXC) by the author, the `Node` class itself is not specific to any particular technology—it does not model any internal structure.

The main `Node` constructor is private, and so can only be invoked by the `Node`'s one friend class: `Network`. The constructor is supplied with a network parameter database, a pointer to the invoking network object, and the node's id: an integer unique to this node within the network. Both a `const` pointer to the node object's parent network and its own id are stored internally, and can be accessed through the appropriate member functions.

From the supplied parameter database, the `Node` extracts its (`x`, `y`) location on a Cartesian plane, its name, levels and clusters; all of which are stored internally and can be retrieved via member functions. (A description of levels and clusters is given in §E.4.) The (`x`, `y`) location is in arbitrary units and is stored as two double-precision floating-point numbers. However, the parent network object stores a variable (`kmPerUnit`) from the network parameter database, which scales node coordinates to km, and can be obtained using the appropriate `Network` member function (if not provided, it defaults to 1 *i.e.* units are simply km). In addition, given two node ids, `Network` provides a member function that will calculate the Euclidean distance in km between them. The node's name is an arbitrary string; if one is not provided in the parameter database, it defaults to `"Node<id>"`, where `<id>` is the node's id.

As with `Network`, the `Node` class provides member functions for output either to file (output stream), or to a specified parameter database; both are invoked, as appropriate, by the `Network` class to output its own nodes.

`Link` class

As was the case for nodes, link objects are entirely dependent on their parent network, and never exist in isolation. Once again, and perhaps even more than with nodes, this reflects reality!

The main `Link` constructor is private; however, it can still be invoked by the parent network, as `Network` is a friend of `Link`. The constructor takes only two arguments: a `const` pointer to the parent network object, and the length of the link in km. Both are stored by the link object, and can be accessed via member functions. However, a parameter database argument is not required, as all the detail on link connect levels this would have supplied is instead retained by the parent network. The link length, provided by the parent network, is either retrieved by it from the network parameter database, or if not present, calculated (as the Euclidean distance between the end nodes) using the `Network` class' own inter-nodal distance member function.

In addition to data supplied and stored at its construction, the `Link` class also supports link capacity, link weight and link marking. Link capacity, stored as a double-precision floating-point number, is set directly by the network object as part of its handling of network routing: the link capacity is simply the sum of the capacity of all the paths (either simple or wavelength-routed, as appropriate) that make use of it; the current value can be retrieved via a member function. Further, as the network model takes node capacity to be the sum of the capacity of all a node's attached links, `Network` provides a node capacity member function which will calculate this value for a given node id.

Link weight supports the network model's concept that routing need not necessarily be done only with respect to link length. Instead, although at construction its weight and length are equal, a link's weight can be reset to an arbitrary value using either of two `Network` member functions: one selecting the link by its id[4], the other using a node id pair. All length-based routing in NOMaD actually uses these link weights, retrieved via a member function, rather than the actual link length[5].

To support network routing, the ability to 'mark' links is provided. Each link object stores a Boolean value (`marked`), initially `false`, which can be queried by a member function. When a link is marked, NOMaD's routing functions ignore the existence of the link: it is as if it is no longer part of the network topology. This provides an invaluable aid to finding *e.g.* link-disjoint [292] or k-shortest paths through the network [322]. Link marking employs several `Network` member functions: links can be marked by choosing them using either their own id or the ids of their end nodes; nodes can be marked, selected by their id (node marking is equivalent to marking all the node's attached links); and both simple and wavelength-routed paths can be marked, equivalent to marking all but their first and last nodes (or the single link on a 2-node path). Finally, the marking of all links can be reset by calling the appropriate `Network` member function.

[4]Although link objects do not store an id, a network can determine a unique id for each one on the basis of their end-node ids; however, their ids are necessarily changed if the network topology alters.

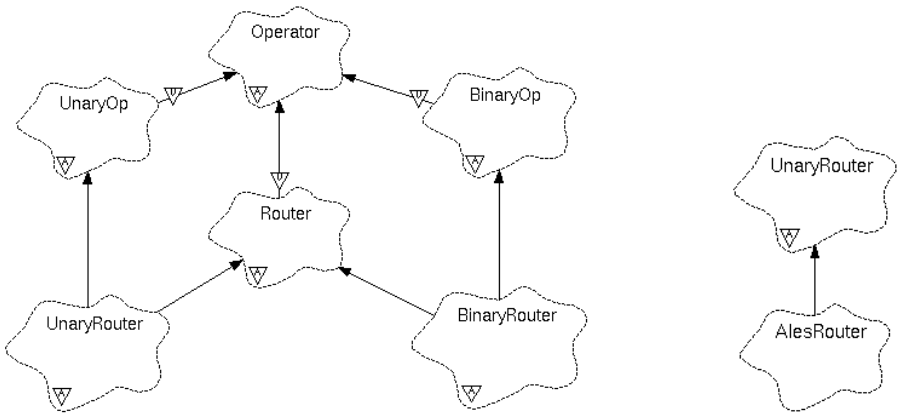
[5]Link weights are used, for example, in the `AlesRouter` class, discussed in §4.1.1.

`Router` and `UnaryRouter` classes

As was mentioned in §3.5 above, (genetic) operators in NOMaD are all derived from the abstract virtual base class `Operator` [16, 17]. To support both unary and binary[6] operators, *i.e.* operators that can be applied to either one or two network objects respectively, two further abstract classes were derived from `Operator`: `UnaryOp` and `BinaryOp` (see Fig. E.2(a) and §4.2). Although it was envisaged that operators would include both those for routing, fibre and wavelength allocation, as well as network topology design, it was felt important to distinguish the former operators from classes able to completely route a network (rather than just modifying an existing routing). These overall routing heuristics would therefore be derived from another abstract class, `Router`, but because its own derived classes were still to be considered operators, `Router` was itself derived from `Operator`. An additional complication arose because, by analogy with `BinaryOp`, it was anticipated that some routers might be developed that would route two networks, rather than just one, in a single operation. Consequently, abstract classes `UnaryRouter` and `BinaryRouter` were developed, derived from `UnaryOp` and `Router`, and from `BinaryOp` and `Router`, respectively. However, no concrete classes have been developed that derive from `BinaryRouter`, and it therefore seems likely that the development of `BinaryRouter` was an unnecessary complication. The design of `UnaryRouter` would appear to be more useful though, as classes derived from it are both unary operators (*i.e.* able to be applied to one network) and routers. Concrete classes derived from `UnaryRouter` include `AlesRouter` (see Fig. E.2(b) and §4.1.1), used in network topology design, and several wavelength-allocation routers discussed briefly in §4.2.3 and in more detail in Appendix C.

In addition to its role in the class hierarchy, the `Router` class was also used to make public several routing functions required by other classes. These include implementations of: Dijkstra's shortest-path algorithm [322] (between a pair of nodes, or between one node and all others); Yen's k-shortest-path algorithm [320] (for a pair of nodes, and which uses the `PathHeap` class); modified versions of Dijkstra and Yen that support Wauters & Demeester's HRWA algorithm [306] (§C.2); and the author's own simplistic algorithms for the k-lowest-hop-count paths and the minimum-number-of-hops paths (both for pairs of nodes).

[6] Binary operators are an important component of GAs.



(a) `Router` (b) `AlesRouter`

Figure E.2: Router class diagrams

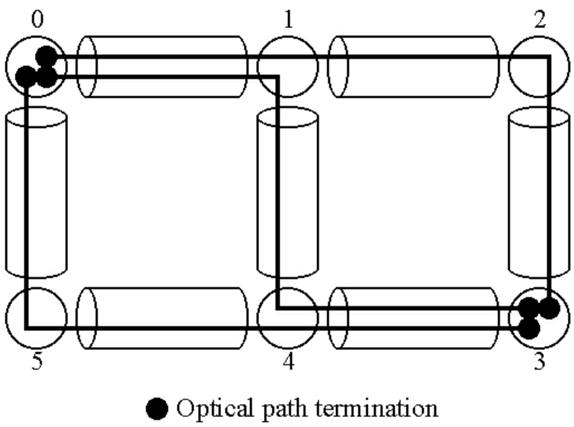
E.3.2 Mesh Network Topology Design

None of the classes in NOMaD's network model are entirely specific to network topology design. Nevertheless, two classes will be discussed in this context: `Path` and `PathLossSeq`. While `Path` is of most use in network topology design, it is also the base class of `PhyPath`, which is used for wavelength allocation. Further, given this relationship between the two path classes, `PathLossSeq` objects can be used to maintain ordered lists of either type. For descriptions of the related iterator classes, `PathIter`, `PathLinkIter` and `PlsIter`, see §E.5.

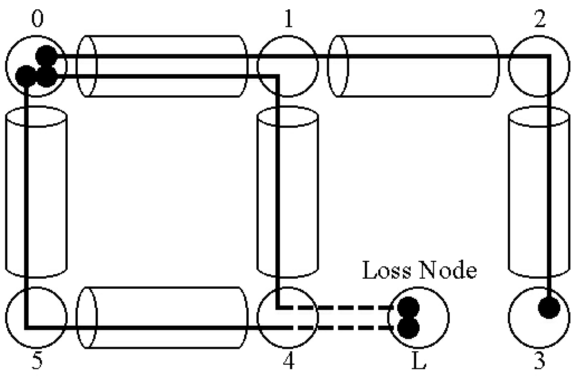
The path model used here (including the *loss node* and *path-loss sequences*) was first developed by Lin *et al.* [286], and subsequently used by the author, both in an M.Sc. project [323] and several research papers [28, 324, 325].

`Path` class

Paths are modelled in NOMaD as an ordered series of nodes, which is exactly equivalent to an ordered series of links, given the exclusion of parallel links from the network model. In addition, paths are not allowed to contain cycles *i.e.* a node may only appear *once* in a path. Fig. E.3(a) of a 6-node, 7-link network (with ids given adjacent to the nodes) shows three paths, all running between nodes 0 and 3. Taking node 0 as the source node, these can be represented as strings: `"(0,1,2,3)"`, `"(0,1,4,3)"` and `"(0,5,4,3)"`. For those cases where a path is unable to reach its intended destination (*i.e.* it is blocked), Lin *et al.* [286] introduced a fictitious loss node. This is shown, for example, in Fig. E.3(b), where link (4,3) has been removed so that two of the paths can no longer reach node 3. The three paths are now: `"(0,1,2,3)"`, `"(0,1,4,L)"` and `"(0,5,4,L)"`, where `L` is used to represent the loss node. It is thus possible to distinguish `"(0,5,4)"`, a *success path* from node 0 to node 4, from `"(0,5,4,L)"`, a *loss path*.



(a) Paths



(b) Success/loss paths

Figure E.3: Example paths and path-loss sequences

The main `Path` constructor takes only two arguments: a `const` pointer to the parent network object and the path capacity, which is passed as an optional [16] argument that defaults to zero. These are stored, although the capacity can be set to another value later; both can be accessed by member functions. Associating paths with a parent network enables considerable integrity checking, although at a cost in flexibility. For example, (apart from topology changes), NOMaD paths only consist of series of nodes that are actually connected by links in the parent network. However, a path constructed as above is *empty*, since an actual node series has not been specified. Thus, a variety of member functions are provided to extend a path, empty or otherwise (see below), but an additional constructor is also available. This takes as arguments the parent network, a string specifying the path, and (optionally) the path capacity. In the event that the string provided to this `Path` constructor also includes fibre choice and wavelength allocation (§E.3.3), a simple path will still be constructed; this also applies when a simple path object is either set from a string or input from a file.

To implement paths, the `Path` class (Fig. E.4) uses a dynamic array (an array-list, class `Array-List`, §E.6) to store an ordered set of pointers to node objects. This representation, rather than simply storing node ids, was chosen to allow more rapid access to the node objects themselves, but requires more storage space. In addition, the path also stores both its length and weight, equal to the sums of the lengths and weights, respectively, of all the links along the path. The lengths and weights are kept up-to-date by the class as the actual path changes, and can be retrieved via member functions. To enable straightforward derivation, all of the `Path` class data members are merely protected, rather than private [16]; `PhyPath`, derived from `Path`, thus has direct access to them (see §E.3.3 below).

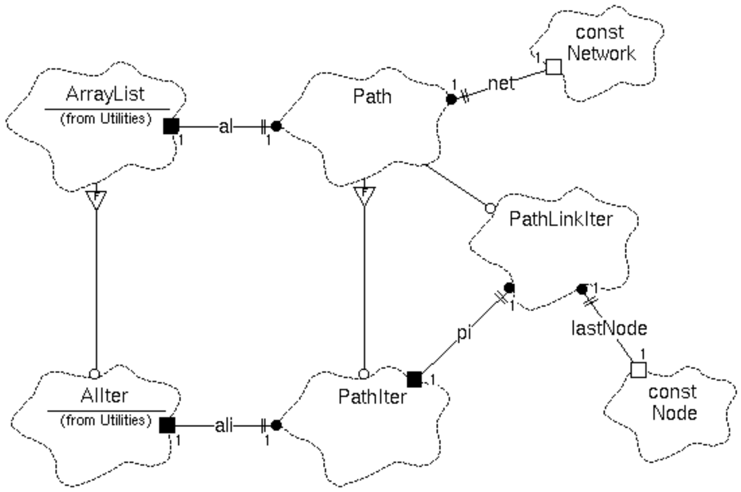


Figure E.4: `Path` class diagram

Many of the `Path` class member functions are made available in two forms: as an overloaded operator [16] or as an ordinary member function. For example, a `const` pointer to one of the node objects on a path (identified by its index number along the path) can be retrieved using either the `getNode()` member function or the overloaded '`[]`' operator (*e.g.* `pp[2]` would retrieve the third `Node`—counting from 0—along `Path pp`, by invoking the `operator[]()` member function with argument 2 [16,17]).

To simplify both the interface and implementation of paths, they can only be set (from a string), cleared (*i.e.* emptied) or extended—there is no provision for removing nodes. Hence, as long as a path is not a loss path (which can itself be tested with a member function), it can be extended by appending a single node. However, if this is not to be the first node on the path, there must be a link in the network adjacency matrix between the two nodes concerned! Examples of the addition of nodes to paths using both the `append()` member function and the overloaded '`+=`' operator are shown in §E.7. Note that to convert a success path to a loss path, the loss node itself must be appended: this is a special node object stored by each network, that can be retrieved with the appropriate member function. Further, not only paths as a whole, but also individual node objects can be queried to see if they are the loss node for their network.

As well as extending by just one node, a copy of an entire path can be appended to another, using either the `append()` member function[7] or overloaded '`+`' . Amongst the requirements imposed, the path appended to must be both a success path and must end with the same node the other path begins with. Thus, `(0,5,4) + (4,3)` would be valid in the network of Fig. E.3(a), with the result `(0,5,4,3)`, whereas `(0,5,4) + (2,3)` would not. Nevertheless, if the two paths come from different, but compatible (*i.e.* having the same number of nodes) networks, it is still possible to append one to the other: the copy would be mapped to the other network, although it (and hence the resultant path) would be truncated to a loss path if the topology could not support it[8]. Hence, for path `(1,2)`, valid in the network of Fig. E.3(b), adding `(2,3,4,5)`, although valid in network (a), would result in `(1,2,3,L)`, as link (3,4) is missing in network (b). Note that, when two path objects are combined with `append()`, the first is modified by the addition of a copy of the second; however, when '`+`' is used, neither path is actually altered, but instead a third, new path object is returned, so as to preserve the expected semantics of the '`+`' operator [16].

Several functions are provided for accessing the individual nodes along a path. As well as `getNode()` and '`[]`', mentioned above, both the first and the last nodes can be retrieved with particular member functions. A path can be queried as to whether it contains a given node object; whether it is empty or not; and what size it is (*i.e.* how many nodes it contains). In addition, two paths can be tested for equality (using either `sameAs()` member function or overloaded '`==`' operator), for both identical and compatible parent networks. A further member function tests, in a similar way, for equality up to a given number of nodes along the paths.

For file input/output (`iostream` [16]), friend functions are provided overloading '`>>`' and '`<<`', respectively. These in turn invoke virtual private member functions [16], ensuring that classes derived from `Path` (and `PhyPath` in particular) will handle input/output correctly.

`PathLossSeq` class

A path-loss sequence [286] is an ordered series of paths; however, all its paths must share the same source, and all the success paths the same destination node. Thus `"{P0(0,1,2,3), P1(0,1,4,3), P2(0,5,4,3)}"` is a valid path-loss sequence (Fig. E.3(a)), as is `"{P0(0,1,2,3), L0(0,1,4,L), L1(0,5,4,L)}"` (Fig. E.3(b)), but `"{P0(0,1,2,3), P1(3,4,1,0), P2(0,5,4,3)}"` is not. Within the string representation of a path-loss sequence, the success and loss paths are indexed separately (from `0`), and their index, along with a `P` or `L` respectively, is prepended to the path string itself.

[7] In C++, which function to invoke is determined by its arguments as well as its name; as a result, it is possible for `Path` to have two different `append()` member functions.

[8] This possibility, like most actions involving two networks, has been included to support GAs

This is mainly an aid to the user, however, and may be omitted, *e.g.*"`{(0,1,2,3), (0,1,4,3), (0,5,4,3)}`" is also valid.

Given that a path is an ordered series of nodes and a path-loss sequence is an ordered series of simple (or wavelength-routed) paths, a good measure of similarity was maintained between the two class interfaces; nevertheless, some differences between the designs did arise. As with paths, both overloaded operator and ordinary member functions were provided wherever appropriate.

One difference in particular between paths and path-loss sequences is of note. NOMaD's network model regards node objects as 'owned' by their parent network; each node object has a unique id, and nodes are never copied (they have no copy constructor[9] [16]). Consequently, a path object is seen primarily as representing a sequence of nodes: it owns the *sequence*, but not the node objects themselves. On the other hand, it was anticipated that paths and path-loss sequences would need to be able to exist 'outside' a network, although their validity is checked against a particular network. This allows the user to manipulate paths and path-loss sequences before 'giving' them to a network (see below). As a result, a path-loss sequence owns the path objects in its sequence: they are constructed by the path-loss sequence directly from a string, or as copies of 'external' path objects. In addition, the path-loss sequence's own path objects are only accessible to users via `const` pointers. These measures preserve the integrity of the path-loss sequence, as otherwise paths in its sequence might be altered without its knowledge!

As has been mentioned, path-loss sequences can contain both simple and wavelength-routed paths. However, the design decision not to introduce a separate path-loss sequence class for wavelength-routed paths has introduced some inelegance and lack of type safety. Although simple and wavelength-routed path objects should never be used together in the same sequence, the `PathLossSeq` class does not enforce this. In addition, at certain points, only the user can determine that a sequence contains objects of class `PhyPath`, and is required to cast them back from `Path`. However, the decision was taken to avoid both introducing an additional (almost) identical class to `PathLossSeq`, as well as the resulting need to develop the `Network` class' `plsMatrix` from an implementation class [283] to two full classes, so as to contain the matrix of existing and new path-loss sequence objects for networks and wavelength-routed networks, respectively.

`PathLossSeq`'s main constructor takes just a single argument: a `const` pointer to the parent network; this is stored, and can be retrieved with a member function. As with paths, such a path-loss sequence is empty, however, and so another constructor is provided which also takes a string

[9] Actually, they *do* have a copy constructor, but it is private, and fails an assertion if invoked!

argument, specifying the pass-loss sequence.

Rather than using a dynamic array, as do paths, path-loss sequences use a circularly-linked list (a `List`, §E.6) to store their sequence of pointers to path objects (Fig. E.5). Although a linked-list requires more processing than an array-list to retrieve an individual object, it is far more efficient when objects are removed: an operation which path-loss sequences, unlike paths, would need to support.

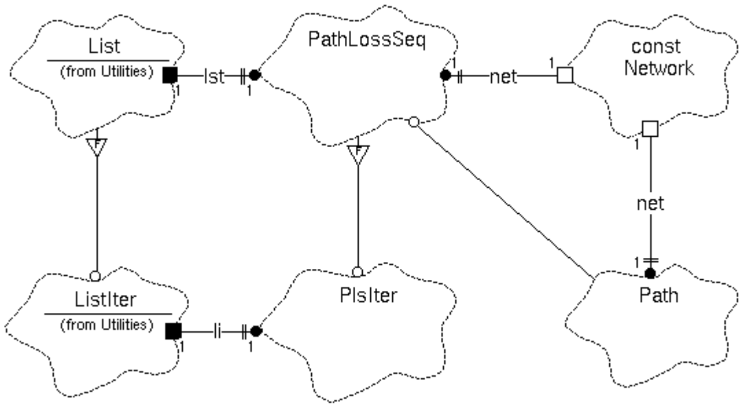


Figure E.5: `PathLossSeq` class diagram

As well as setting a path-loss sequence from a string and clearing (*i.e.* emptying) it, the sequence can be extended, by appending either a path or a whole path-loss sequence, and individual path objects can also be removed. Single paths are appended using either `append()` or '`+=`'. Each path appended must be non-empty. After the first path has been appended, all subsequent paths must be for the same source, and if not a loss path, share the same destination. They can either be from the same network as the path-loss sequence, or a compatible one *i.e.* one having the same number of nodes. (Although, in the latter case, any copy of a path that was unsupportable by its parent network topology would be truncated to a loss path.) The same set of restrictions apply when a whole path-loss sequence is appended using either the overloaded `append()` member function or '`+`' operator.

The paths in a sequence can be accessed as `const` pointers using a member function. As this is an expensive operation for the underlying linked-list, no equivalent overloaded operator function is provided (*cf.* `Path`'s '`[]`' operator). However, the first and the last paths can be retrieved using efficient special-case functions. The source and destination nodes can also be obtained (as `const` pointers) using member functions: the latter simply returns the loss node if all the paths in the sequence are loss paths. A path-loss sequence can be queried to see if it contains a given path object; whether it is empty or not; what size it is (*i.e.* how many paths it contains); and whether all its paths are loss paths or not. In addition, as with individual paths, two sequences can be tested for equality, either for identical or compatible parent networks, and equality can also be tested up to a given number of paths along the sequences.

For input/output, friend functions are provided overloading '`>>`' and '`<<`', respectively. By invoking the equivalent overloaded operators in class `Path`, which in turn invoke virtual private member functions, input/output is handled correctly whether the sequence contains either simple or wavelength-routed paths.

At this point, having reviewed some of the details of the `Path` and PathLossSeq classes, it is appropriate to examine the member functions provided by the `Network` class in support of network routing. The `Network` class' `plsMatrix` of path-loss sequences between source-destination pairs was described above, but how are these sequences set and manipulated?

One member function enables the path-loss sequence between a given source-destination node id pair to be set. Any existing sequence between the pair is deleted, and a copy of the new sequence is installed in the path-loss sequence matrix. The new sequence must be for this network, be non-empty, and apply to the correct source-destination pair. For a given pair, other member functions allow a `const` pointer to a path-loss sequence to be retrieved from the network, or the sequence explicitly removed (*i.e.* deleted). As an aid to efficiency, a member function is further provided to completely clear a network's path-loss sequence matrix (clearly of particular use in routers). In addition to whole sequences, two member functions allow individual paths to be appended to or removed from the sequence for a given source-destination pair. Where necessary, in all these functions and as was mentioned above, the capacity of the network's links is maintained to always reflect the sum of the capacities of the paths carried. However, where a simple path has a capacity of zero, its actual capacity is simply taken to be that of the traffic between the source-destination pair that form its end nodes[10]. Further, to ensure the cable, fibre and wavelength structure of wavelength-routed networks can also be appropriately adjusted during routing modifications, these member functions are both declared virtual [16] and re-implemented in the `PhyNetwork` class (see below).

[10] For a wavelength-routed path, however, its capacity is always uniquely determined by the number of wavelengths and their granularity.

E.3.3 Routing, Fibre & Wavelength Allocation

This subsection presents those classes specific to the routing, fibre choice and wavelength allocation problem: `PhyNetwork`, `Cable`, `PhyPath` and `Join`. (For descriptions of the related iterator classes, `CableIter` and `PathJoinIter`, see §E.5.) Through these classes, the NOMaD network model supports multiple-fibre cables, and both virtual-wavelength-path (VWP) and wavelength-path (WP) routing [294].

`PhyNetwork` class

The `PhyNetwork` class[11] is derived from `Network`, adding (and redefining) those elements necessary to allow the representation of the cable, fibre and wavelength-routed structure of a multi-wavelength optical network (see Fig. 3.4(b) on p. 58, and Fig. E.8 on p. 273 below).

As in `Network`, the main `PhyNetwork` class constructor takes two parameter database arguments, one built from the network specification and the other from the NOMaD parameter file. After constructing the base `Network` class, two optional parameters (the default number of fibres per cable and the wavelength channel capacity *i.e.* granularity) plus the cable matrix are retrieved from the network parameter database. The wavelength channel capacity (or its default value) is stored as a double-precision floating-point number (`lambdaCap`), and may then be accessed via a member function. It should be noted that, since there is only a single granularity value for the whole network, NOMaD's network model does not support the mixing of wavelength channels of different capacity.

The cable matrix (`cabMatrix`) consists of a two-dimensional array of size `numNodes` × `numNodes` of pointers to cable objects: if a cable exists between two nodes, then the corresponding entry in the array will be a valid cable pointer; otherwise it is void. As with links (§E.3.1 and Fig. E.1 on p. 255), cables are bidirectional, and so any cable object pointed to by the matrix appears in two entries: (i, j) and (j, i). In addition, as 'self loops' are not allowed, the diagonal is empty. The network parameter database determines whether a cable will actually be constructed for any particular node pair. If an overall default number of fibres per cable has been specified, then a cable is constructed for *every* node pair. In this case, the number of fibres used for each cable will be the default given, unless the database contains a different value for that node pair. However, if no overall default is given, then a cable will only be constructed for those node pairs where the number

[11] Originally described as a *physical* network by the author, hence `PhyNetwork`, the name wavelength-routed network is now preferred; the class name persists, however.

of fibres is specified in the database. In addition, the constructor requires that for every node pair connected in the network topology (*i.e.* for which there is a link in the adjacency matrix), there must also be a supporting cable in the cable matrix. Consequently, if a cable has been constructed for every node pair, then arbitrary topology changes can be made by adding and removing links. However, if not, then links can only be included between node pairs where the potential has been indicated by providing a cable. An entry in the cable matrix can be retrieved, as a `const` pointer to a cable object, using either a link id[12] or the end-node ids; clearly, in the former case not all cables may be able to be retrieved, but those not corresponding to links are not part of the 'real' network topology anyway.

Many of the functions in the `PhyNetwork` class' public interface are simply redefinitions of the `Network` class' virtual member functions. Thus, for example, link addition is redefined to allow for the further requirement that any new link be supported by a cable. Those functions that support routing are necessarily altered: setting or removing path-loss sequence matrix entries, plus adding or removing paths (actually wavelength-routed paths) from entries. Clearly, output to both parameter database and file also had to be changed.

The final group of member functions relates to fibres and wavelengths over the whole wavelength-routed network; these are best dealt with below, in the context of the `Cable` class.

`Cable` class

In addition to `PhyNetwork`, several classes could have been designed to represent a multi-wavelength network's cable, fibre and wavelength-routed structure: individual classes for cables, fibres and even wavelengths. However, any classes designed for fibres or wavelengths would have required only limited functionality, and for efficiency reasons, would have had to allow access to their data members by the cable class. Consequently, it was decided that a single class, `Cable`, would provide the data structure and functionality to support all three concepts.

In a wavelength-routed network, an individual cable has a number of fibres, each of which will support several wavelengths. Each potential wavelength is either occupied by a wavelength-routed path or unoccupied. For example, Fig. E.6 shows a 3-fibre cable, with three occupied wavelengths (*i.e. light* wavelengths) on fibre 0 (*i.e.* wavelengths 0, 1 and 3), five occupied on fibre 1, and four on fibre 2. If we assume each fibre can support up to 32 wavelengths, say, then there are 29 unoccupied (*i.e. dark* wavelengths) on fibre 0 (*i.e.* wavelengths 2 and 4–31), 27 on fibre 1, and 28 on fibre 2.

[12] See footnote 4 on p. 258

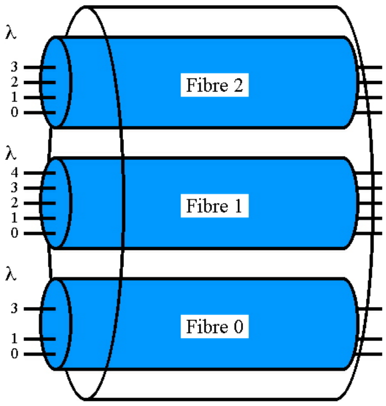


Figure E.6: Example cable of three fibres

As with nodes and links, cables are not intended to exist in isolation, but to be constructed and used by a wavelength-routed network (Fig. E.8 on p. 273). In addition, while cables necessarily store pointers to wavelength-routed paths, these are not owned by the cables themselves, but by their parent wavelength-routed network.

The main `Cable` constructor takes one argument, the number of fibres on the cable; this can subsequently be retrieved using a member function. After construction, although the number of fibres on a cable cannot be varied, this does not exclude the use of the network model to determine the number of fibres actually required by a cable. As long as a generous number of fibres has been preset, after routing, fibre and wavelength allocation is completed, dark fibres (*i.e.* those that are completely unoccupied) on each cable may be excluded from any assessment of the design.

As well as storing the number of fibres (`numFibres`), the data structure of a cable consists of an array of bit vectors (32 bits each) to store the occupancy of each fibre[13], as well as a separate occupancy matrix. The latter is a `numFibres` $\times$ 32 array of pointers to wavelength-routed paths: each row corresponding to a fibre, and each column a wavelength, allowing the individual wavelength-routed path using each occupied wavelength to be recorded. For example, the occupancy vectors (in octal) for the cable in Fig. E.6 would be `013`, `037` and `017` for fibres 0–2, respectively. The use of both these data structures is, of course, redundant: the occupancy vector for an individual fibre could be constructed simply by noting which wavelengths are occupied in a given row in the matrix (unoccupied wavelengths are indicated by a void pointer). However, reconstructing the occupancy vectors when required was judged to be more costly in processing time than maintaining them as a separate data structure (and incurs only a modest additional space requirement). The decision

[13]As with levels and clusters (§E.4), restriction to this range allows an unsigned integer [16] to be used as a bit vector (32 bits); a greater range would require the use of a true bit-vector class.

to limit the number of wavelengths per fibre to 32 was taken on the grounds of both efficiency and memory size[14]. In the event that a user wishes to enforce a lower limit, this can be achieved with appropriate constraints in the routers (or individual operators) used.

An additional restriction placed on the design of the `PhyPath` class, and which thus impacts all the other classes in this section, was that while a wavelength-routed path could make use of one or more wavelengths on each cable, these must all be on a single fibre. This decision allows the occupancy of a wavelength-routed path on a cable to be expressed as a fibre id plus an occupancy vector (32 bits) indicating the wavelengths used: a concept captured in the `Join` class (see below). Whereas, if the occupancy of a wavelength-routed path (and hence the underlying structure of the `Join` class) was to have been expressed as an arbitrary list of fibre id and wavelength number pairs, this would have had serious implications on both processing time and storage size. Limiting each wavelength-routed path to use only a single fibre on each cable was not seen as a serious limitation, in any case, as a sequence of such paths would allow any arbitrary routing to be expressed between a source-destination node pair. Indeed, in the author's own work, wavelength-routed paths were entirely restricted to just a single wavelength over a single fibre on each cable (and were thus equivalent to wavelength channels); where multiple channels were required, a sequence of separate wavelength-routed paths was used.

A wavelength-routed path can be placed on a cable using a member function; as well as the wavelength-routed path object itself, a join object must be provided to indicate the fibre id and wavelength number(s). Clearly, the new wavelength-routed path must only be added to a set of unoccupied wavelengths on the given fibre. To remove a wavelength-routed path, and free the associated wavelengths, there are two member functions. One of these, taking a pointer to a wavelength-routed path, removes it and returns a join object recording the fibre id and wavelength(s) the wavelength-routed path used to occupy. The other, given a join object, removes and returns the corresponding wavelength-routed path; however, it requires that the join object provided specify exactly the fibre and wavelength(s) occupied by a particular path. As well as the removal of individual wavelength-routed paths, the `Cable` class also has a member function to remove them all.

Several member functions allow a cable object to be queried. One checks if a given join object (*i.e.* fibre id and wavelength number(s)) is unoccupied. With another two, either the individual

[14]This is of particular importance for wavelength-routed paths, which represent a high proportion of the total space requirements for a wavelength-routed network object.

wavelength-routed path (if any) corresponding to a particular join object can be retrieved, or even a particular entry in the occupancy matrix. Further, whether a particular wavelength-routed path is carried on the cable can be checked, or the details retrieved as a join object. As well as queries relating to individual wavelength-routed paths, the occupancy of a given fibre, as a whole, can be accessed as a bit vector, or even as an array of wavelength-routed path pointers. In addition, the occupancy of the entire cable can be obtained as a bit vector, where each bit is only set if the corresponding wavelength is occupied on *all* its fibres.

A function is provided to count the (partially) occupied, completely unoccupied, or all fibres on the cable. In the same way, the wavelengths on an individual fibre can be counted; or the wavelengths on the whole cable, either counting the distinct wavelengths used between all fibres or the wavelength channels used in total. As an example, in Fig. E.6 above, for fibre 0 there are 3, 29 and 32 wavelengths, for light, dark, or in total, respectively. Now, for the cable as a whole, considering just the distinct wavelengths used, gives 5, 27 and 32; whereas we get 12, 84 and 96 by simply counting across all fibres, for light, dark, or total channels, respectively.

As an aid to assessing wavelength requirements (*i.e.* the total number of distinct wavelengths required, including dark ones), member functions are available to determine the highest wavelength number used on either a given fibre, or the cable as a whole[15]. Further, to support VWP routing [294], the `Cable` class supplies a packing member function. This re-packs the wavelength-routed paths on a cable to use an unbroken sequence of wavelength numbers, thus minimising their wavelength requirement. The actual modification to any wavelength-routed path involved is actually achieved by invoking its own packing member function, which should only be used in this context. As an example, the cable in Fig. E.6 could be re-packed so that the wavelength requirement on fibre 0 was only 3 (it is currently 4), by moving the path on wavelength 3 down to wavelength 2.

For the `PhyNetwork` class, as was mentioned above, a number of member functions are provided summarising information that could be obtained by iterating over all cables in the network topology. One function obtains, as required, a count of (partially) occupied, completely unoccupied, or all fibres in the wavelength-routed network; two more, similar counts for the network wavelengths (counting each wavelength used on more than one fibre/cable either once, or as appropriate); and yet another returns the highest occupied wavelength number in the whole network. Further functions are even defined to return the first, or all the wavelength-routed paths actually using the highest

[15]In NOMaD, the highest wavelength number is always exactly one less than the corresponding wavelength requirement, as wavelengths are numbered from 0, whereas the wavelength requirement is a *count* of the total number of distinct wavelengths needed.

wavelength number.

`PhyPath` class

In the NOMaD network model, the wavelength-routed path[16] is a straightforward extension of the path concept described in §E.3.2 above. Where a path is an ordered sequence of nodes, with the links implicit, a wavelength-routed path also specifies the fibre and wavelength(s) used on each cable (link) along the path. However, as has already been discussed, while several wavelengths may be used on each cable, they must all be on the same fibre. Also, though a different *set* of wavelengths may apply on each cable (in VWP routing), the actual *number* of wavelengths used on each cable must remain constant throughout the path. Nevertheless, if required, the user can obviously constrain the path to use the same set end-to-end *i.e.* in WP routing.

As an example, Fig. E.7 is of a 6-node, 7-cable network, with each cable consisting of two fibres, and each fibre able to carry up to two wavelengths. This network can be viewed as equivalent to that in Fig. E.3(a) on p. 261, but with the wavelength-routing structure made visible. The network carries three wavelength-routed paths, which are again equivalent to the paths in the earlier figure. Their string representation, taking node 0 as the source, is: `"(0[1:1],1[1:1],2[0:1],3)"`, `"(0[0:1],1[0:1],4[1:1],3)"` and `"(0[0:0],5[0:0],4[1:0],3)"`. Here, a *join* between nodes is represented by `"[a:b]"`, where `a` is the fibre id, and `b` a set of wavelength numbers (see class `Join`, below). Further, if the three wavelength-routed paths were placed in a path-loss sequence, then this would be represented as:

```
"{P0(0[1:1],1[1:1],2[0:1],3), P1(0[0:1],1[0:1],4[1:1],3), P2(0[0:0],5[0:0],4[1:0],3)}"
```

Reflecting the relationship between the simple and wavelength-routed path concepts, the `PhyPath` class is derived from `Path` (Fig. E.8). The main constructor takes two arguments, a `const` pointer to the parent wavelength-routed network and the (optional) capacity; these are simply passed to the `Path` base class constructor. The resulting wavelength-routed path is empty and so, as with simple paths, a second constructor is also provided that takes an additional string argument specifying the wavelength-routed path itself. Although, for compatibility with simple paths, a capacity argument is provided for both constructors, the capacity of a wavelength-routed path can always be determined by the product of the number of wavelengths used and their granularity. Consequently, a wavelength-routed path maintains its own capacity, and these constructor argu-

[16] Originally called a *physical* path by the author, hence `PhyPath`, the name wavelength-routed path is now used; however, the class name remains unchanged.

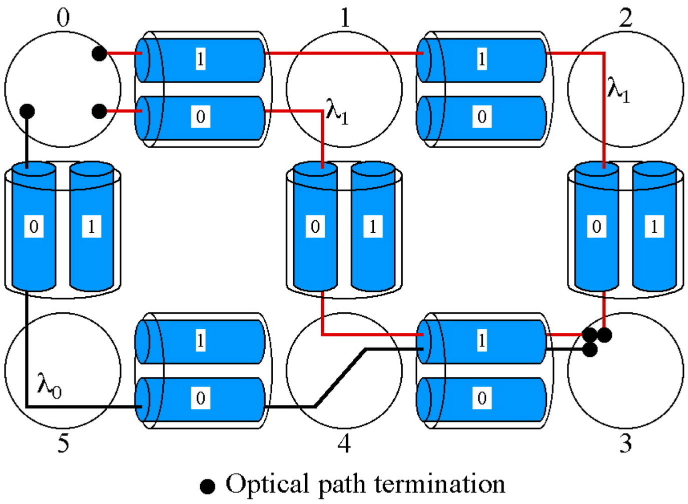


Figure E.7: Example wavelength-routed paths

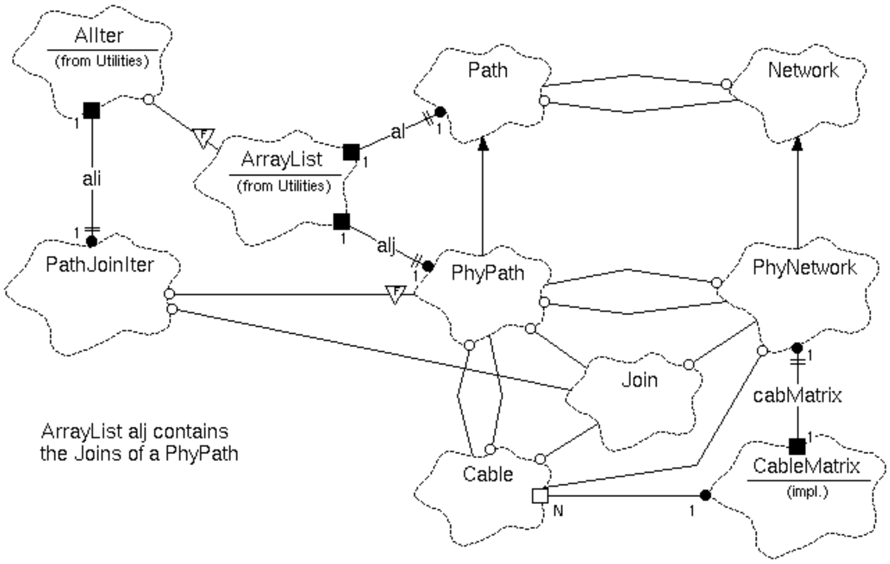


Figure E.8: `PhyPath` class diagram

ments are ignored; in addition, the member function used to set path capacity is re-implemented to fail an assertion if invoked.

To record the additional join information between each node pair along the wavelength-routed path, a dynamic array (array-list `alj`, see §E.6) of pointers to join objects is used. This structure shadows the array-list (`al`) that holds the node pointers for the underlying path, but necessarily consists of a shorter sequence of joins. For a wavelength-routed success path, exactly one less join than node will be held; for a wavelength-routed loss path, two less, as no join is required to connect the wavelength-routed path to the fictitious loss node. Unlike node objects, the joins are actually owned by the wavelength-routed path: it is responsible for their creation and destruction.

Many of the `PhyPath` class' member functions are simply necessary re-implementations of the

`Path` class'; others provide equivalent operations on the sequence of joins to those the `Path` class provides for its sequence of nodes; whereas some functions are completely new.

As well as re-implemented member functions to set (from a string) and clear a wavelength-routed path, several functions are needed to extend a wavelength-routed path by a single node or many. For extension by one node, the `append()` member function is re-implemented, but it can only be applied to an empty wavelength-routed path: in all other cases, a join must be provided as well[17]. Hence a version of `append()` is included that takes two arguments: the node to add, and the join to make the connection. The latter is passed as a `const` pointer, as only a copy will be stored by the wavelength-routed path. However, as well as the limitations imposed to extend the underlying path, several additional requirements must be met. Unless the node is the loss node, the join must be non-empty. If it is the first join on the wavelength-routed path, it sets the wavelength count; if not, it must use the same number of wavelengths as all the prior joins. Further, the join must not be blocked by the cable it will use (*i.e.* all the wavelengths and fibres needed must be free).

Directly appending an actual path object to a wavelength-routed path is impossible, due to the complete absence of wavelength-routing data in the former, so the appropriate `append()` function is re-implemented in the `PhyPath` class to fail an assertion. However, the '`+`' operator, that appends a simple path to a wavelength-routed path, *can* be re-implemented successfully: it returns the new path (*not* a wavelength-routed path) formed by first copying its wavelength-routed path argument into a new path object, and then adding a copy of the second path. Nevertheless, appending two wavelength-routed paths is supported by a more specific `append()` member function and '`+`' operator that each take two wavelength-routed path parameters. For both these functions, as well as satisfying the path requirements, the two wavelength-routed paths appended must also use the same number of wavelengths.

Access to individual joins is provided, returning a `const` pointer to maintain data integrity, by four member functions. These retrieve the first join, the last join, a join by index along the wavelength-routed path, and a join by specifying its end-node ids. The three path test functions are re-implemented to work correctly for wavelength-routed paths, as are the private member functions underlying file input/output. In addition, a member function is provided to count the total number of distinct wavelengths used on the wavelength-routed path (which may be more than the number per join if VWP routing [294] is used[18]) and another returns its highest wavelength number.

[17] This also acts as a restriction on '`+=`', which is implemented by `Path` using this version of `append()`.

[18] If the number of wavelength channels is required, then the wavelengths used on the first (or any) join can be counted.

`Join` class

As has been explained in the context of the `Cable` and `PhyPath` classes, in the network model a join is simply a fibre id and a set of wavelengths: it captures the occupancy of a wavelength-routed path on a cable. In string form, it is represented by `"[a:b]"`, where `a` is the fibre id, and `b` is a comma-separated list of either individual wavelength numbers or hyphen-separated wavelength ranges. For example, `"[2:4,6-9]"` means wavelengths 4, 6, 7, 8 and 9 on fibre 2. As elsewhere in NOMaD, wavelength numbers are restricted to the range 0–31.

Three main constructors are provided for `Join`: one takes no arguments, and constructs an empty join; another needs a string representation; the third requires both a fibre id and a bit vector (32 bits) of the wavelength numbers. The underlying data structure of a join consists of the fibre id and a bit vector (32 bits) of wavelength numbers.

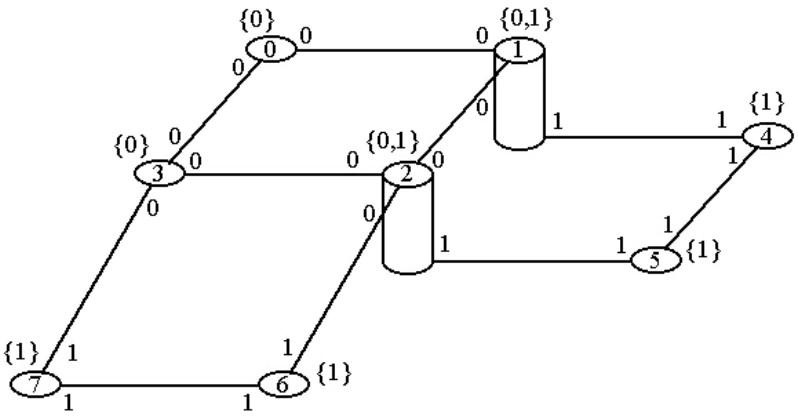
As well as member functions to clear (*i.e.* empty) the join and set it from a string, the fibre id can be set on its own, or individual wavelength numbers added or removed. Query functions allow the fibre id to be retrieved, and the wavelength numbers to be accessed as a bit vector (32 bits), or even as an individual wavelength number (if the join is on only one wavelength). The wavelengths can be counted, and the highest wavelength number obtained. The join can be tested to see if it is empty (*i.e.* on no wavelengths), or whether it is on a particular wavelength. Two joins can be tested for equality, either by an overloaded '`==`' operator, or an ordinary member function. In addition, file/input output is provided by the usual overloaded `iostream` operators.

E.4 Network Hierarchy and Node Clustering

NOMaD's network model directly supports the concept of network hierarchy, by enabling nodes to be on one or more levels, and links to be connected to a particular level at each end[19]. Fig. E.9(a) illustrates this with an 8-node, 10-link, 2-level hierarchical network. In the figure, node numbers are drawn on the nodes themselves; node levels are shown in braces adjacent to the nodes (*e.g.* the label {0,1} by node 1 records its membership of both levels 0 and 1); and link connect levels are indicated by numbers next to each end of the link. For clarity, the links themselves have not been labelled, but will be referred to by their end node ids. Nodes 0 and 3 are on level 0 (the 'top' level); nodes 4, 5, 6 and 7 on level 1; and nodes 1 and 2, drawn as cylinders, are on both levels.

[19]This section assumes the reader is familiar with §E.3.1 on p. 253, and in particular the discussion of the `Network` and `Node` classes.

Most of the links clearly belong to one level or another, but links (2,6) and (3,7) are 'sloped'—they are connected at two different levels. While NOMaD supports networks that have both nodes on multiple levels, and links that interconnect different levels, the author would expect that any given network design would use only one or other of these; this restriction is not, however, enforced by the network model.



(a) Network

From/To	0	1	2	3	4	5	6	7
0		0		0				
1	0		0		1			
2		0		0		1	0	
3	0		0					0
4		1				1		
5			1		1			
6			1					1
7				1			1	

(b) Matrix

Figure E.9: Example connect levels matrix

To represent the network hierarchy, node objects store the levels they are on (restricted by implementation to the range 0–31, with 0 the default[20]), although any change to their levels is made through the public interface of the `Network` class, to allow the overall consistency of the network hierarchy to be maintained (*e.g.* it is not permitted to remove a node from a level on which it has an attached link connected; the connection must be (re)moved first). The link connect levels are stored in the network's connect levels matrix (`conMatrix`), a two-dimensional array of size `numNodes` $\times$ `numNodes` of integers (see Fig. 3.4(a) on p. 58). It was decided to make the matrix an attribute of the `Network` class, rather than storing the individual connect levels in the links, as it was felt that these are more correctly a property of the network as a whole, as they represent connections *between* links and nodes. The connect levels matrix corresponding to the network in Fig. E.9(a) is provided in in Fig. E.9(b), where the reader should note that the matrix is not symmetrical, due to sloped links (2,6) and (3,7).

The `Network` class provides a comprehensive set of member functions for the manipulation of both levels and connect levels. For node levels, there are functions to add a level to a node[21], remove

[20] Restriction to this range allows an unsigned integer [16] to be used as a bit vector (32 bits); a greater range would require the use of a true bit-vector class.
[21] Or equivalently, to add a node to a level, depending on your point of view!

a level from a node, or change a node to a new level, along with all its connected links (provided it is only on *one* level!). Queries on the levels of an individual node are handled directly through its own member functions, described below. However, the levels can be counted or retrieved for the network as a whole, either as a string (*e.g.* `"{0,1}"`) or a bit vector (32 bits). For link connect levels, individual entries in the connect levels matrix can be set and retrieved; in addition, the connect levels for an individual node id can be counted, queried with respect to a particular level, or retrieved as a whole, either as a string or a bit vector.

As an aid to the design of cluster-based architectures, such as multi-ring networks, the network model also provides direct support for node clusters; these are also intended to be useful in the development of any cluster-based operators or overall design heuristics. Individual nodes can be included in one or more clusters, although cluster numbers are restricted to the range 0–31 for implementation reasons, with 0 the default[22]. Fig. E.10 illustrates the concept with a 9-node, 12-link, 4-ring (or 'cluster') network. Nodes 3 and 4 are in cluster 1; nodes 5 and 6 in cluster 2; and nodes 7 and 8 in cluster 3. The central three nodes, while all being in cluster 0, are each also included in two other clusters: hence node 0 is in clusters {0,1,3}, node 1 in {0,1,2} and node 2 in {0,2,3}.

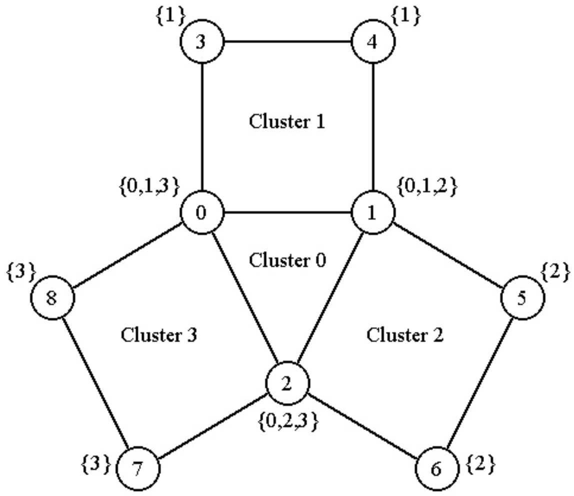


Figure E.10: Example network with node clusters

While individual nodes store the clusters they are in, any change to their clusters is made through the public interface of the `Network` class, both for consistency with the handling of network hierarchy (see above), and also to allow for some interaction between the network hierarchy and node clusters. There are `Network` class member functions to add a cluster to a node[23], remove a cluster from a

[22] As with levels, this range allows the use of an unsigned integer as a bit vector (32 bits).
[23] Or equivalently, to add a node to a cluster, depending on your point of view!

node, or change a node to a new cluster (provided it is only in *one* cluster!). Cluster queries for an individual node object are handled directly through its own member functions, described below. However, the clusters can be counted or their ids retrieved for the network as a whole, either as a string (*e.g.* `"{0,1}"`) or a bit vector (32 bits). In addition, to support the use of rings/clustering in hierarchical networks, the clusters can be counted or their ids retrieved for all nodes (as a whole) connected at a given level.

As was explained in §E.3.1 on p. 257, the main `Node` constructor can only be invoked by the `Network` class. It is supplied with a network parameter database, from which the node extracts, amongst other items, its levels and clusters. The node's levels are supplied as a string, *e.g.* `"{0,1}"`, which is mapped into a bit vector (32 bits) for storage. If absent from the parameter database, `"{0}"` is assumed. Via member functions, the levels can be counted, and retrieved either as a string or a bit vector; in addition, the node object can be queried as to whether it is on a particular level, and whether it is on only one level (if `true`, the actual level is returned as a reference parameter [16]). Clusters, apart from their meaning, are supported by the `Node` class in exactly the same way as levels.

E.5 Iterators

The descriptions of several iterator classes (useful, but less important than more central classes) are provided here, rather than in Ch. 4: `PathIter`, `PathLinkIter`, `PlsIter`, `CableIter` and `PathJoinIter`.

`PathIter` class

The *path iterator* class (`PathIter`) is provided to enable a user to access efficiently, conveniently and in series the nodes on a path object. It conforms to Stroustrup's design for a non-intrusive iterator [16].

The main constructor takes a reference to a path, and only one member function is provided, the overloaded '`()`' operator (*i.e.* function call [16]). Each time it is invoked, this returns a `const` pointer to the next node on the path, starting with the first; after these have all been returned, it simply returns a void pointer. This is perhaps better understood by examining the example code below, which implements the `Path` class' `print()` private member function (for clarity, error-handling code has been removed):

```
void Path::print(ostream&os) const {
  os << '(';
  PathIter pi(*this);
  const Node* n;
  if (n = pi())
    os << n->getId();
  while (n = pi()) {
    os << ',';
    if (n->isLoss()) {
      os << 'L';
      break;
    } else
      os << n->getId();
  }
  os << ')';
}
```

First, a bracket is output, then a path iterator (`pi`) is constructed for the current path object (`this`) and a `const` node pointer declared (`n`). The first node is assigned to `n`, and if not void, its id printed. The code then enters a loop (until the iterator is exhausted), alternatively printing commas and node ids, while ensuring the appropriate response is made if the loss node is encountered. Finally, the closing bracket is output.

Implementation of the `PathIter` class is achieved simply by using the appropriate iterator (an array-list iterator, see Fig. E.4 on p. 262) on the path's underlying array-list (see §E.6).

`PathLinkIter` class

The *path-link iterator* class [16] (`PathLinkIter`) complements the path iterator class by allowing the user sequential access to the links on a path. The main constructor has one argument, a reference to a path, and the usual overloaded function-call operator is provided. Implementation (Fig. E.4 on p. 262) is achieved using a path iterator object (`pi`) to iterate over the nodes, a data member to store a pointer to the previous node (`lastNode`), and access to the parent network (via any of the nodes) to retrieve each link in turn[24].

`PlsIter` class

The *path-loss-sequence iterator* class [16] (`PlsIter`) allows the user sequential access to the paths in a path-loss sequence. The main constructor has one argument, a reference to a path-loss sequence, and the usual function-call operator is provided. Implementation (Fig. E.5 on p. 265) is achieved using a linked-list iterator object (`li`) to iterate over the underlying linked-list object. Unfortunately, as the function-call operator returns a `const` pointer to a simple path object (class `Path`), if the sequence actually contains wavelength-routed paths (class `PhyPath`), then the appropriate cast [16] will have to be used.

`CableIter` class

The *cable iterator* class [16] (`CableIter`) allows the user sequential access to both the wavelength-routed paths on a cable and their corresponding joins. It is the iterated equivalent of the `Cable` class' `getJoin()` member function, which retrieves the join corresponding to a particular wavelength-routed path on the cable, but here the user does not need to provide the individual paths: the iterator does it all.

The main constructor has one argument, a reference to a cable, and the usual function-call operator is provided. However, this can only return one argument, a `const` pointer to the next wavelength-routed path. Consequently, a `getJoin()` function is provided which, invoked after a particular path has been retrieved, will return the corresponding (dynamically-allocated) join object.

Implementation is achieved by stepping through the fibres one at a time, retrieving each wavelength-routed path in turn. The cable iterator keeps track of which fibre it is working on and the wavelengths on that fibre for which it has already returned a path. In addition, to allow a copy to be obtained whenever requested, it also maintains a join object for the last wavelength-routed path returned.

`PathJoinIter` class

The *path-join iterator* class [16] (`PathJoinIter`) allows the user sequential access to the joins along a wavelength-routed path. The main constructor has one argument, a reference to a wavelength-

[24]For a loss path, the fictitious link to the loss node has no 'real' existence; the iterator simply returns a void pointer.

routed path, and the usual function-call operator is provided. Implementation is achieved simply by using the appropriate iterator (an array-list iterator, see Fig. E.8 on p. 273) on the wavelength-routed path's underlying array-list of joins (see §E.6).

E.6 `Utilities` Class Category

The `Utilities` class category contains a few, relatively unrelated, classes that were developed to support those in other categories. These include dynamic arrays, circularly-linked lists, file handling and pseudo-random number generation.

`ArrayList` and `AlIter` classes

The *array-list* class (`ArrayList`) is a type of dynamic array that blends some of the aspects of a linked list with the inherent capabilities of a basic C/C++ array. It maintains an indexed array of `const void*` pointers [16] *i.e.* untyped pointers that can store the address of any object, which the array-list 'grows' as and when required. Object pointers contained in the array are always stored in an unbroken range, starting from index 0. Its most important use is as a base class for the `Path` and `PhyPath` classes (see §E.3.2 and §E.3.3).

Two main constructors are provided, one without arguments which creates an empty array-list, the other with a single argument: a pointer to the first object to be contained.

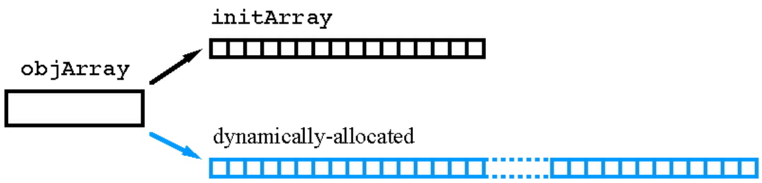


Figure E.11: `ArrayList` array implementation

An array-list is implemented as a pointer to an array (`objArray`) of `const void*` pointers. Initially, `objArray` points to another data member (`initArray`), which is simply an array of 16 `const void*` pointers (Fig. E.11). However, if the size of the array needs to be increased beyond 16, the array-list switches to using dynamically-allocated arrays, each twice the size of the one before. This approach was adopted to increase the speed of array-list allocation, because `initArray`, as a data member, is available immediately after the construction of the object. Using a dynamically-allocated array from the start would have required an additional allocation request. As the `Path` and `PhyPath` classes, both derived from `ArrayList`, use rapid class-based memory allocation, this modification resulted in considerable increase in the speed of programs that make heavy use of the two path classes (the majority of NOMaD applications), particularly where the number of nodes in the paths is 16 or less. To monitor the need to increase the size of the array, two integer data members are used: one records the number of objects contained, the other the maximum allowed in the current array. Then, whenever needed, a private member function is automatically invoked to increase (double) the array size.

The contents of the array-list can be modified in several different ways. One member function places a single object at the end of the array, whereas another places one at the start (*i.e.* index 0), moving the prior contents of the array up by one location. A complementary member function removes, and returns, the object at index 0, moving all the remaining elements down. Further, the array can be cleared (*i.e.* emptied) as a whole, or an object at an existing index replaced by a new one.

As well as modifications to the array, a number of queries have also been made available. Individual objects can be retrieved from a given location, either using a member function or an overloaded operator; special member functions also allow retrieval of the objects at either the first or the highest current index locations. The size of the array (*i.e.* a count of the number of objects contained) can be obtained, or the array checked to determine if it is empty. In addition, whether the array-list contains a particular object can be queried.

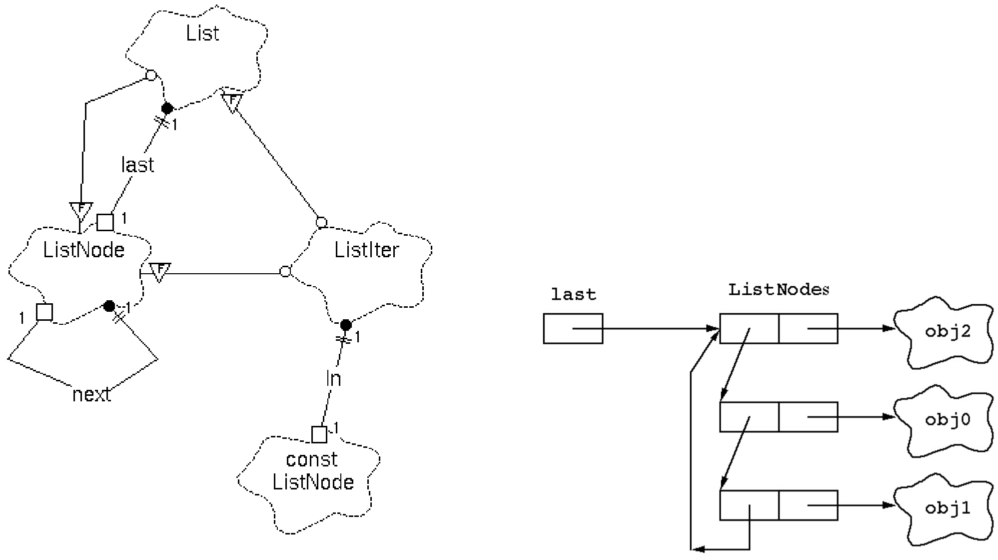
The array-list iterator class (`AlIter`) [16] allows the user sequential access to all the objects in the array[25]. Its main constructor has one argument, a reference to an array-list, and the usual overloaded function-call operator is provided. Implementation is achieved by simply maintaining an index into the array as iteration proceeds.

`List`, `ListNode` and `ListIter` classes

The *linked-list* class (`List`) is a non-intrusive circularly-linked list of objects, each stored as a `const void*` pointer [16] (see Fig. E.12(a)). While implemented very differently, its public interface has much in common with that of an array-list.

The *linked-list-nodes* themselves are objects of class `ListNode`, each of which simply consists of a `const void*` pointer to the linked object, plus a pointer to the next node in the list. For example, Fig. E.12(b) shows a list of three objects (`obj0`, `obj1` and `obj2`). The linked-list contains a single private data member, `last`, which points to the *last* node on the list: this allows efficient

[25] For a more complete description of an iterator, see the description of the path iterator on p. 278.



(a) Class diagram

(b) `List` of three objects

Figure E.12: `List` class diagram and example

access to both the beginning and end of the list, without the overheads necessary to maintain a doubly-linked list [287,326]. Thus `obj0` is the first object on the example list, `obj1` the second, and `obj2` the third and final.

The linked-list-node class (`ListNode`) is a simple one, with only one member function, its private constructor (used only by friend class `List`) which takes two arguments, a pointer to the object to be linked into the list, and a pointer to the next linked-list-node object. The two pointers are then simply copied into the appropriate data members in the newly constructed linked-list-node object.

On the other hand, the `List` class presents a more complex interface. Two main constructors are provided: one, with no arguments, creates an empty linked-list, the other constructs a linked-list containing one object, passed in as an argument.

For linked-list modification, member functions are provided to insert an object at the start, or append an object to the end of a list. Either the object at the start can be removed, or any of the objects on the linked-list, specified by pointer. Further, the list as a whole can be cleared (*i.e.* emptied) of all its objects.

Queries are supported by member functions to determine the size of the linked-list (*i.e.* how many objects it contains), test whether it is empty, or ascertain whether it contains a particular object. In addition, individual objects can be retrieved by their index in the linked-list, using either a member function or an overloaded '`[]`' operator; member functions to access the first and last

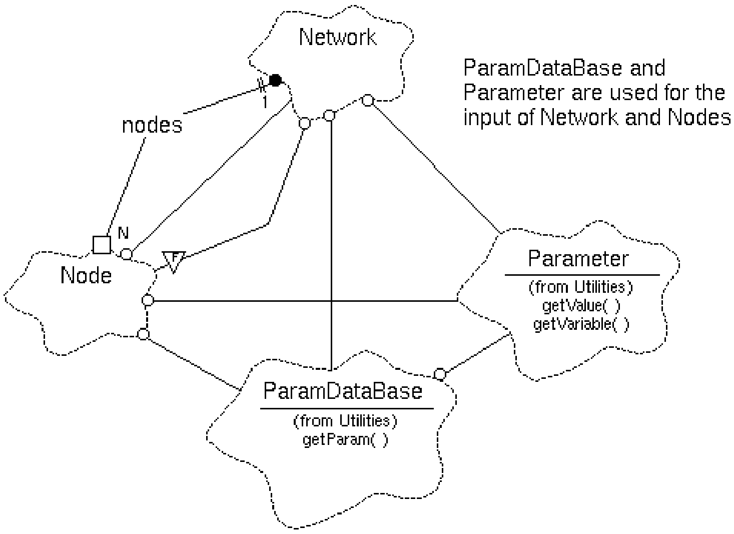


Figure E.13: `Network/Node` input/output class diagram

objects in particular are also provided.

The *linked-list iterator* class (`ListIter`) [16] allows the user sequential access to all the objects on the linked-list. Its main constructor has one argument, a reference to a linked-list, and the usual function-call operator is provided. Implementation is achieved by maintaining a pointer to the appropriate linked-list-node as iteration proceeds.

`Parameter` and `ParamDataBase` classes

The *parameter* (`Parameter`) and *parameter database* (`ParamDataBase`) classes together provide a convenient abstraction of file input/output (itself accessed indirectly through the `iostream` classes [16]). They operate on files whose formats were detailed earlier in this Appendix (§E.1), and their use by NOMaD's classes is described throughout Ch. 3 and 4, in Appendix C and elsewhere in this Appendix. For example, Fig. E.13 illustrates the relationships between the classes involved in network input/output. During network construction (§E.3.1), first a parameter database is itself constructed from an input file (both network specification and NOMaD parameter files will be required), and then the individual parameters (*i.e.* variable-value pairs) can be retrieved as parameter objects, using the `getParam()` member function (selected using the variable string argument as key). The variables or values of these can then be accessed via the `getVariable()` and `getValue()` member functions, respectively. In the same way, nodes use a parameter database during their own construction.

The `Parameter` class has two main constructors: one, without arguments, constructs an empty object; whereas the other stores the variable and value string arguments provided. Input/output is supported by overloaded `iostream` operators, with the overloaded input operator the only way to change the variable-value pair once the parameter object is constructed. As mentioned above, the stored variable and value strings can be retrieved using member functions, and in addition, whether a parameter object is empty can be tested using an overloaded `int` operator. Use of this aspect of the `Parameter` class is perhaps best illustrated by example code. The following was taken from one of the parameter database constructors; for clarity, both error checking code and some of the implementation have been deleted.

```
ParamDataBase::ParamDataBase(istream&is) {
  // ...
  is.clear();
  is.seekg(0); //Rewind input stream
  while(is) {
    //Get next Parameter
    Parameter param;
    is >> param;
    if (!param || strlen(param.getVariable()) == 0 ||
        strlen(param.getValue()) == 0)
      continue;
    //Create new entry, or modify existing one
    // ...
  }
}
```

First, the input stream (`is`) has its error status cleared, and is rewound to the beginning of the file. Then, while the input stream remains `good` (itself tested via an overloaded `int` operator thus: `while(is)`), successive lines from the file are read into parameter `param`. If it remains empty (tested via the overloaded `int` operator thus: `!param`), or either the variable or value strings are of zero length, then that line is ignored; otherwise, the parameter is stored as an entry in the parameter database (code omitted).

For efficiency, the parameter database class (`ParamDataBase`) is implemented using a tcl hash table [284, 285]. Actual parameter objects are not stored, but instead only value strings, accessed via the hash value generated from the corresponding variable string. However, this implementation detail is concealed by the public interface, which uses parameter objects.

Two main constructors are provided: one without arguments creates an empty parameter database; the other, taking an input stream argument, uses this to populate the new database with parameters (see example code above). Individual parameters can then be set, whether they are already in the database or not, using a member function that takes a parameter argument. In addition, parameters can be deleted using a function that takes a variable string argument; this returns a Boolean indicating whether the parameter was actually already in the database to be deleted. Further, given a variable string, individual parameters can be retrieved, as well as the database as a whole printed to an output stream.

`RandomGen` class

The *pseudo-random number generator* class (`RandomGen`) provides uniform, negative-exponential, geometric, Bernoulli and discrete-uniform probability distributions. It is based on the `urand5()` generator first published by the author in *Principles of Performance Engineering for Telecommunication and Information Systems* [327], which was itself an implementation of a generator described by Fishman [328]. The generator is linear congruential, of prime modulus multiplicative type, and is coded for portability to either 32- or 64-bit machine architectures.

Two main constructors are provided: one takes an `unsigned long` as the generator seed, the other assumes a seed of 1. The current number in the pseudo-random sequence can be retrieved at any point using the `getZ()` member function, but the series itself is only advanced when a number from one of the distributions is requested. These include `uniform()`, which returns a double-precision floating-point value uniformly distributed in the range 0+ to 1; and `exponential()`, which provides a double-precision floating-point value from a negative-exponential distribution, mean `1/lambda`, where `lambda` is supplied as an argument. The `geometric()` member function returns integer values, geometrically-distributed, with the probability of success given as an argument. Bernoulli-distributed numbers can be obtained via a member function `flip()`, which acts as a biased coin, returning a Boolean, where the probability of `true` is passed in as an argument (default 0.5). Finally, `intRange()` gives random integers in a range inclusive of its two arguments.

E.7 Testing & Verification

As was explained in §3.8, during the design and implementation of NOMaD's object-oriented network toolset, three different approaches were used to try and minimise errors in the software: Programming by Contract, dynamic memory tracking and regression testing. Although each of these was mentioned briefly in the main text, more detailed explanation is provided here.

Meyer's concept of Programming by Contract [289] requires that the preconditions necessary to the correct operation of a class member function be made explicit. In return, the member function will either meet certain explicit postconditions on completion, or signal failure. This contract, between the invoker and implementer of a class member function, helps to ensure software reliability. The member function should only be invoked when all the preconditions to its correct operation are present; it should then either completely satisfy all its promised postconditions, benefiting the caller, or obviously fail. In addition to member function pre- and postconditions, Meyer also introduced the idea of class invariants: these are conditions which should be met both before and after the invocation of any member function of a class (although clearly only *after* the constructor and *before* the destructor!). Effectively, the class invariants can simply be thought of as additional pre- and postconditions that are shared by all member functions.

Although pre- and postconditions, building on the concept of abstract data types, are an obvious aid to clarity in class design, they can also be used as an execution-time aid to software reliability. Meyer's Eiffel language [289] included keywords `require`, `ensure` and `invariant` to designate Boolean clauses expressing preconditions, postconditions and class invariants, respectively. If any of these conditions was violated during execution, the run-time environment would throw an exception. Various mechanisms would then permit the exception to be caught and handled, but ultimately execution would fail if a correct sequence of instructions could not be found. More details on both Programming by Contract and the Eiffel language can be found in [329, 330]. The concept of Programming by Contract has also been incorporated into various object-oriented analysis and design approaches, including BON (Business Object Notation) [331], which mainly supports Eiffel implementation, and the more general Booch Method [15].

Although C++ does not directly support pre- and postconditions on class member functions, a set of macros was written by the author, and used extensively in the implemented code. For simplicity, C++'s exception mechanism was not used, but rather a simple error message was output on condition failure, indicating source file, line number and a short textual error message (the *tag*).

For example, the source code of the `insert()` member function of class `List` is given below. This places an object, pointed to by `obj`, at the head of the circularly-linked list. For clarity, the actual implementation code has been omitted.

```
void List::insert(const void*obj) {
  require("this not void", this);
  require("obj not void", obj);
  // implementation
  ensure("not empty", !isEmpty());
  ensure("contains obj", contains(obj));
  ensure("is first", getFirst() == obj);
}
```

As can be seen above, the `require` and `ensure` macros take two arguments: the tag, *i.e.* a string containing a brief textual description of the condition, and the Boolean to be tested. The two `require` statements check that both the linked-list object (`this`) and the object to be inserted (`obj`) are non-void *i.e.* point to (hopefully) valid addresses. The three `ensure` statements test that after the insertion, the linked-list is non-empty, it contains the inserted object, and that `obj` has been placed at the front of the list.

While the use of Programming by Contract cannot eliminate all errors in a piece of software, the author found that, as well as encouraging clear thinking in both design and implementation, some subtle errors, that might otherwise have gone undetected, would often be caught by one of the conditions. However, while Programming by Contract was used throughout the development of NOMaD, it should be noted that during long program executions, the pre- and postconditions were omitted (using conditional compilation), as they would otherwise have significantly increased program execution time.

The second approach used was dynamic memory tracking: as C++ requires the programmer to directly manage the allocation and release of all dynamic memory objects [16, 282][26], this can be a significant source of error. In addition, the dynamic allocation of objects that are not subsequently released when no longer required ('memory bleed'), can lead to program failure due to exhaustion of machine memory. This was particularly important for some NOMaD runs, where a large fraction of machine memory was required to hold the evolving population of networks.

[26] "It is said that Lisp programmers know that memory management is so important that it cannot be left to the users and C programmers know that memory management is so important that it cannot be left to the system." [16]

To combat problems in this area, dynamic memory tracking code (called `xmem`) for C [290] was rewritten by the author for C++. The tracker operates by replacing the implementation of the global `new` and `delete` operators. The replacement `new` operator stores the size and location of each dynamic memory allocation in a hash table, as well as recording the current total amount of memory allocated, the largest total amount allocated so far, and the current overhead of additional allocation caused by `xmem` itself. These three values are made available to the user though global variables. In addition, every allocation request is increased by two bytes into which a known data pattern is written, in order to detect array over-run and other possible memory violations on deallocation. Clearly, the `new` operator can also detect when dynamic memory itself is exhausted. Using the hash table built by `new`, the replacement `delete` operator is able to detect deallocations that do not correspond to a currently allocated area of dynamic memory: this will catch both the use of invalid addresses, and also accidental 'double' deallocation of objects. In addition, any change to the two additional bytes after each allocated area will be reported, indicating invalid access to those memory locations.

The `xmem` dynamic memory tracker was used throughout the development of NOMaD, and several subtle memory problems were identified by its use. However, it should, once again, be noted that `xmem` was not linked into the executables used for long experimental runs, due to the increased run times produced by dynamic memory tracking.

The final approach to minimising software errors adopted was regression testing. A large test program (`testNOMaD`) was written, with a long series of assertions for many of the classes in the object-oriented network toolset. These checked the expected outcome of invoking the class member functions under a variety of different conditions. During the course of NOMaD's development, as new classes were added or existing classes extended, the regression tests would be run again. Any unexpected change in the operation of existing classes and their member functions would (hopefully) result in a failed assertion. Subsequent investigation would enable the fault to be traced to a design error, an implementation error, or a failure on the part of the author to anticipate valid changes to NOMaD's operation! Periodically, the regression test suite would itself be extended to incorporate NOMaD's growing functionality.

As an illustration of the regression tests, the code below represents a small portion of the tests applied to the `Path` class (see §E.3).

```
Path* p1 = new Path(net1);
p1->append(net1->getNode(0));
```

```
p1->append(net1->getNode(1));
p1->append(net1->getNode(2));
p1->append(net1->getNode(7));

char buf[256];
ostrstream osb(buf, 256);
osb << *p1 << '\0';
assert(strcmp("(0,1,2,7)", buf) == 0);

Path* p2 = new Path(net1);
(*p2) += (*net1->getNode(0));
(*p2) += (*net1->getNode(4));
(*p2) += (*net1->getNode(3));
(*p2) += (*net1->getNode(6));

osb.seekp(0);
osb << *p2 << '\0';
assert(strcmp("(0,4,3,6)", buf) == 0);
```

The code shows an empty path, pointed to by `p1`, being constructed for use in network `net1`. The path is then grown by the addition of nodes 0, 1, 2 and 7 using the `append()` member function. The path is then output (using its overloaded '`<<`' operator friend function [16]) into an `ostrstream` buffer, and the results compared with the expected string, `"(0,1,2,7)"`. The next few lines of code show a similar sequence for a second path, but this time the '`+=`' operator is the one under test.

E.8 NOMaD Software: Discussion & Further Work

The design and implementation of NOMaD has been considered at length in the thesis, with Ch. 3 providing an overview, Ch. 4 and §E.3 exploring much of the object-oriented toolset, and the remainder of Appendix E providing additional details.

E.8.1 Overall Architecture

The overall architecture of NOMaD is illustrated in Fig. 3.2 on p. 55, consisting on one hand of compiled applications built using the object-oriented network toolset and, on the other, of scripted applications with optional graphical user interfaces (GUIs) built using the `NOMaDsh` tcl extension library. When NOMaD was originally conceived, it was anticipated, as was explained in §3.4, that `NOMaDsh` would play an important role. However, with two research assistants[27] and the author all using the NOMaD toolset, it soon became apparent that the clear preference was for compiled applications in C++. When the object-oriented toolkit was extended by the author, the new features were available to compiled applications immediately, but there was an inevitable implementation delay in also making them available in `NOMaDsh`. Consequently, whilst working with interpreted scripts might have been preferable had NOMaD not been continually developed, in practice `NOMaDsh` fell further and further behind the object-oriented network toolset. As a result, active development of both `NOMaDsh` and its GUI script library effectively ceased after the work on mesh network topology design was completed.

An unfortunate casualty of discontinuing `NOMaDsh` development was not developing the ability to visualise the results of routing, fibre and wavelength allocations (unlike mesh network topology, which was included). This could be an important area of future work, as while it is relatively straightforward to display topology designs, routing, fibre and wavelength allocations are much more difficult to present concisely in a two-dimensional diagram, even with the use of colour. This can be eased in a design tool, to some degree, as different aspects of the allocation can be displayed in response to interactive prompts from the user (*e.g.* basic routes could be selectively elaborated to show wavelength allocations). An intriguing alternative would be to try three-dimensional representation of allocations using virtual reality modelling (*e.g.* VRML [332]). This might well provide the designer with far greater insight into the allocations than can be gleaned from, say, examining summary statistics (*e.g.* average path hop count, range of link wavelength requirements).

E.8.2 Reimplementation in Java

Given the limitations that have become apparent in combining an object-oriented network toolset in C++ with an interpreted GUI in tcl/tk, a promising future possibility would be to re-implement NOMaD in Java [333,334]. By using the AWT or Swing libraries, this would allow the development

[27] Katerina Proestaki and Noel Parnis.

of both the object-oriented network toolset and the GUI library in one programming language. As a result, GUIs could be seamlessly incorporated into applications. In addition, the multi-threading capabilities of Java would allow a greater interaction with the applications whilst background processing proceeded. Moreover, the portable nature of Java would allow applications to be run on a variety of platforms.

Indeed, partial re-implementation of NOMaD in Java was undertaken in early 2000, with many of the GA classes being ported from C++ to Java to create a simple framework for GA/heuristic hybrids (referred to as sjhga). The sjhga framework has been used successfully in two M.Sc. projects to date [335, 336], and is currently being used on a Ph.D. project by one of the author's students (John Pagonis).

One aspect of the design of NOMaD which adds notable flexibility is the use of execution-time class parameters (parameter databases), as discussed in Appendix E. However, to allow users the full choice of, say, which genetic operators to use, all the C++ class code has to be compiled into the corresponding executable. Consequently, these are often of considerable size (*e.g.* `ga` is currently 1.1 Mbytes), and are built using large and complex make files. With Java, however, only those classes actually used have to be loaded, no make files are required and they do not need to be combined into several large executables.

E.8.3 Network Modelling

NOMaD's network model has been used for optical network design in several research projects, with a variety of heuristic approaches successfully employing the model (*e.g.* [20–24]), in addition to the author's own GA/heuristic hybrid work. Nevertheless, there are a number of areas in which the object-oriented design could be improved.

The classes that support mesh network topology design, including `Network`, `Link`, `Path` and `PathLossSeq` (see Fig. 3.4(a) on p. 58) were developed before those for routing, fibre and wavelength allocation. However, given the amount of code that had already been developed, it was decided to derive `PhyNetwork` from `Network`, and `PhyPath` from `Path` (Fig. 3.4(b)). While these are valid design choices, it may be that a better choice would have been to have had abstract network and path classes from which ones specific to topology and wavelength allocation were derived. In a similar way, there are subtle problems with both path-loss sequences and the relationship between link and cable classes that might benefit from re-design.

For future work, there are a whole range of possible extensions to the network model that could be considered. At present, nodes in the NOMaD are represented by a single class. However, to support more detailed cost modelling and to allow the inclusion of appropriate design constraints, the current `Node` class could be replaced by an abstract class. From this, classes for OXCs, add-drop-multiplexers (ADMs), *etc.* could then be derived. This would allow, for example, network assessments to enumerate the ports required in a network, which could in turn be a significant factor in network cost [23].

In all the metrics for design assessment used in the thesis (§4.1) only a single design period has been used, and accurate forecasting of both costs and traffic requirements has been assumed. However, in practice networks are often deployed over several months or years, and there is often forecasting uncertainty [337]. These factors could be addressed in future work, allowing for both network roll-out and robustness to minor variations in cost or demand [338]. Indeed, the author has already supervised two M.Sc. projects in this area, one which of which studied traffic estimation for future broadband networks [339], whilst the other explored the use of the GA/heuristic hybrid approach to cable-distribution network design [336].

An interesting and difficult network design problem, not considered in this thesis, is determining the location of network nodes, perhaps choosing these from a pre-determined list of sites [10, 11]. NOMaD's network model could also be extended appropriately to include this. Again, an M.Sc. project applying a GA/heuristic hybrid approach to this difficult area has already been supervised by the author [340].

Currently, the network model allows users to set an arbitrary granularity for the capacity of a wavelength channel (`net.lambdaCap` in Table E.2 on p. 246). However, all wavelength channels must have this same granularity. The model could instead be expanded to allow a mixture of channel granularities (*e.g.* both 2.5 Gbit/s and 10 Gbit/s) in the same network.

While NOMaD supports both primary and restoration path routing when used for topology design, only primary routes can be represented in fibre and wavelength allocation. The network model could, however, be extended to include the allocation of restoration paths by allowing wavelength-routed paths to be flagged as 'reserved' for restoration purposes. An alternative, which would better allow for sharing of restoration capacity (typically employed in practical algorithms), is to only reserve the capacity, without including actual paths in the path-loss sequence matrix.

Although static traffic matrices are appropriate for the design of multi-wavelength all-optical transport networks at present, in time 'point-and-click' (dynamic) provisioning of wavelength channels may well be required. NOMaD's network model could be modified to allow for dynamic traffic, although this would not be a straightforward change, as the assumption of static traffic pervades the present design.

In this thesis, three design metrics have been used, one based on network wavelength requirement (NWR) and the other two on simplified network cost models (§4.1). However, more detailed cost models could be developed, particularly with the co-operation of network vendors or service providers. These could be based, for example, on more explicit modelling of the structure of nodes and links, including enumerating ports, amplifiers, *etc.* [23]. In addition, more physical constraints could be included either in the network model or as penalties in the design assessment. These might include restrictions on *e.g.* the number of wavelengths on the fibres [302, 303], the number of ports on the OXCs [302, 303], or the maximum transmission distance allowed to a wavelength channel [301]. In addition, the use of adaptive rather than arbitrary fixed penalties could also be investigated.

E.8.4 Memory and Execution Time Limitations

One of the potential problems with the GA/heuristic hybrid approach that has already been identified (§1.4) is its large memory requirement. The GA framework used by NOMaD is a generational one, so two complete populations—old and new—are required in memory at once. Each individual contains a complete network model, comprising a large number of separate objects. For example, GA2 for mesh network topology design (§5.1.1) requires 27.6 Mbytes of memory. This is used for two populations of 200 individuals, each of 64 kbytes, plus an overhead for the GA framework and executable code of 2.7 Mbytes. GA18 for minimum cost routing, fibre and wavelength allocation (§5.2.2), uses 91.6 Mbytes for two populations of 500 individuals, each of 90 kbytes, as well as an overhead of 3.3 Mbytes. More memory is needed in the latter case not only to represent the wavelength-routed structure of the networks, but because of the larger population size required.

While the amount of memory required may seem large, a number of design decisions were taken to minimise the memory requirements. In particular, the number of wavelengths on a fibre (`Cable` class on p. 268), and the total number of clusters and levels in a network (§E.4), are limited to 32. This allows their compact representation using unsigned integers [16] as vectors of 32 bits; a greater range would have required the use of a true bit-vector class, with a corresponding growth in memory footprint. In a similar way, the `Cable` class alone is used to represent cables, fibres and wavelengths, rather than individual separate classes.

Nevertheless, there are additional areas where further savings in memory size could be made.

At present, for example, paths are implemented using array-lists of pointers to node objects. Each network design requires many path objects, and any reduction in their size would have a large impact on the overall memory requirement for an individual. A more compact representation would be to use arrays of node ids stored as unsigned characters. This would limit the number of nodes to 256, and require additional processing to retrieve the node objects, but would nevertheless reduce the memory requirement for simple paths by a factor of four (assuming 32-bit pointers). Additional savings could be made by other similar representation changes.

Clearly, however, there is a limit in additional memory savings that can be made, and as a result both the size of networks that can be examined and the number of individuals in the population are limited by available memory. The experiments described in Ch. 5 were carried out on an UltraSPARC 5 with 128 Mbytes of memory, which limited the size of the GA to less than, say, 100 Mbytes. Machine memory might be upgraded to 256 or 512 Mbytes, but this would allow only a modest increase in the network size, as only a few additional nodes causes a marked increase in the memory needed to represent a network design. Moreover, larger problems typically require even larger populations for 'good enough' results, further magnifying the problem.

One possible direction for further work in this area would be to investigate compact representations for the network designs, such as a binary string! However, it should be noted that considerable difficulty has been previously experienced by the author in designing a successful binary representation for routing, fibre and wavelength allocation [29]. The resulting GA could still use problem-specific genetic operators, either by designing these to work directly with the binary string representation, or by expanding the encoding to an object model representation before the operators are applied. Either approach would result in a significant memory saving, as even in the latter case only one or two networks would be 'expanded' at a time. However, they would probably both involve a marked increase in processing time over the current approach, as in the former case the operators would involve partial encoding and decoding of the binary representation, and in the latter, considerable time would be lost in memory allocation.

A complementary limitation of the GA/heuristic hybrid approach, although not one that any approach to these difficult network design problems can avoid, is the amount of computer time required. For example, while a single run of GA2 takes about 99 seconds, running GA18 once takes some 5 hours 50 minutes. Consequently, for example, the work reported on operator-probability adaptation in §5.3 (390 runs) is estimated to have taken more than 95 days non-stop computer time on an UltraSPARC 5.

Nevertheless, considerable efforts were taken during NOMaD's implementation to keep the processing time down. As well as the beneficial effects of some of the design decisions taken, such as using unsigned integers as bit vectors, employing class-specific memory management for key classes also provided a considerable improvement in execution time [16].

Overall then, taking both memory and time limitations into account, it must be concluded that there is only limited scalability in NOMaD's current implementation of the GA/heuristic hybrid approach to multi-wavelength all-optical network design. However, the design problems that have been addressed are complex and difficult ones, and a similar absence of scalability is to be expected from other approaches. However, one promising direction for future work would be to exploit the obvious parallelism of the GA framework, and develop a distributed version of NOMaD. Possibilities would include either distributed design assessment, controlled by a master GA on a single processor, or a distributed multi-population approach, with distinct small populations occasionally exchanging promising individuals [341]. In this way, both the memory and processing limitations of an individual machine might be overcome, and networks of larger size designed.

E.8.5 Additional Software

A piece of software that could be developed to complement the existing NOMaD toolset would be an interactive network editor. Although a prototype editor was produced by the author (see §3.7 and Fig. 3.8), the finished tool was never developed. However, a more complete implementation was made by two of the author's B.Eng. students working in Java [342, 343], and their efforts could in future form the basis of a fully-developed network editor.

Also, to allow for the optimal design of small networks[28], software employing NOMaD's network model could be written to generate linear programming (LP) formulations of topology and wavelength-allocation problems. These could then be tackled with an LP solver, *e.g.* `lp_solve` [344] or an equivalent commercial product. Indeed, an M.Sc. project addressing 'linearised' versions of the design problems considered in this thesis has already been supervised by the author [345].

[28]Larger problems are difficult to tackle with the exact methods of mathematical programming due to their large time and memory requirements [20, 23].

Appendix F

Detail of Experimental Results

This appendix provides detailed experimental results that were judged inappropriate for Ch. 5. It is divided into five sections: §F.1 describes how the network design problems used in the experimental work were generated. Then §F.2 tabulates the five network design problems actually used in the experimental work. The remaining three sections record detailed results on mesh network topology design (§F.3); routing, fibre choice and wavelength allocation (§F.4); and operator-probability adaptation (§F.5).

F.1 Network Design Problem Generation

In the telecommunications domain, one difficulty faced by researchers when developing new network design heuristics, is to find suitable network design problems to compare their work with that of others. Such an example problem would consist of a set of node locations, between which the network is to be formed, and a traffic (or capacity) matrix, representing the demands that the network is required to meet. The heuristics being developed (or evaluated) might, for example, include those for mesh network topology design with subsequent routing and wavelength allocation of a multi-wavelength all-optical transport network. For a network planner working in a telecommunications service provider, real node locations and measured traffic statistics for the company's own network would be available. For new networks, or extensions to existing networks, proposed locations and forecast statistics would be required. Such data are of great commercial value, and telecommunications service providers are consequently reluctant to make them available to independent researchers. In some cases proposed or hypothetical networks are available, such as the European Optical Network studied by COST 239 [291]. This used twenty European cities as node locations and appropriately scaled international telephony traffic obtained from the ITU, although the latter was still incomplete and had to be augmented by an empirical model [346]. It is the situation where appropriate example problems are unavailable, or a researcher requires a greater number or range of representative examples, that is addressed here. While there is an extensive literature on traffic estimation and forecasting (*e.g.* Hopkins *et al.* [347]), little has been published on telecommunications problem generation, with only Klingman *et al.* [348], Kershenbaum [12] and Cahn's work [10] known to the author.

F.1.1 Node Locations

There are two main possibilities for node locations in example problems: either existing geographical sites can be used, as in COST 239 [291], or locations can be selected at random within a total service area, *e.g.* Griffith *et al.* [313]. However, in the latter case, with repeated sampling from a uniform random distribution, there exists the danger of sites being selected that are unrealistically close together. This limitation can be addressed by a more controlled method of space filling. One potential approach, from the literature on statistical design of experiments, is *latin hypercubes*, first proposed in [349] and now widely used (see *e.g.* [350,351]). However, this was only recently brought to the author's attention[1], and the approach adopted here, based on the concept of a *minimum clear radius*, was developed independently.

Consider a network problem of N nodes, with a total service area A, and an expected service area per node, $a = A/N$. If, for the moment, we assume circular service areas, then the *expected service radius* is given by:

$$r = \sqrt{a/\pi} = \sqrt{A/(N\pi)} \tag{F.1}$$

This is thus slightly more than half the expected distance between each node and its nearest neighbour (ignoring the 'edge effect' caused by the finite total service area). The minimum clear radius is then taken to be:

$$r_{\min} = k_r \times r \tag{F.2}$$

where k_r is a constant, and usually $0 \leq k_r \leq 0.8$, as explained below.

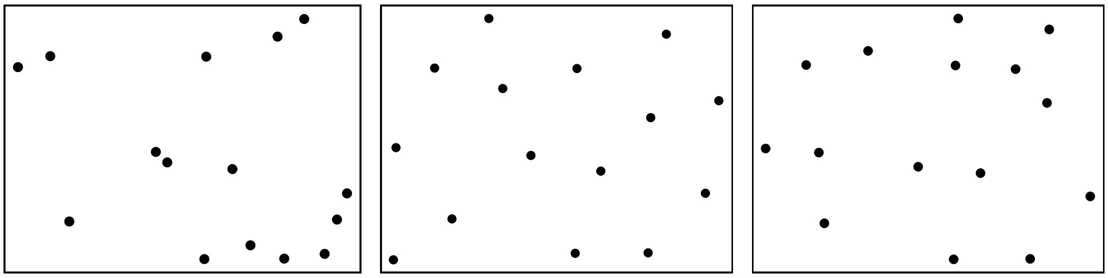
Node locations can now be be generated as follows. The location of node 0 is generated by sampling from a uniform distribution over area A. The subsequent locations $(1 \leq i \leq N-1)$ are still generated using a uniform distribution, but each node location in turn is rejected and

[1] By Andy Keane, one of the examiners.

regenerated until $r_i \geq r_{\min}$, where r_i, the *clear radius* of node i, is:

$$r_i = \min_{\forall j \neq i}(d_{ij}/2) \tag{F.3}$$

and where d_{ij} is the (Euclidean) distance between nodes i and j. With the *minimum clear radius constant* $k_r = 0$, this is simply random selection of node locations (Fig. F.1(a)), but with $k_r = 0.8$, a near grid-like arrangement of nodes is obtained (Fig. F.1(b)). A value of $k_r = 0.5$ appears to produce a more realistic distribution, and is the default for the author's own work (Fig. F.1(c)). Values of $k_r > 0.8$ are not recommended using the simple resampling method described, as they will require a large number of iterations to generate the locations; in addition, the resulting node distributions are somewhat unrealistic, being constrained to a grid.



(a) $k_r = 0$ (b) $k_r = 0.8$ (c) $k_r = 0.5$

Figure F.1: Three fifteen-node problems generated with different k_r

F.1.2 Traffic Distribution

Given a figure for the total network traffic (or capacity), a method is needed to distribute this between all node pairs (or a limited subset thereof), such that the resulting traffic matrix is a good approximation to real-world situations. A simple uniform random (or Gaussian) distribution of traffic, without regard to the location of nodes, will not create the 'communities of interest' typical of real networks. Instead, the traffic distribution approach described here is based on that developed by the author for estimating European international telephony traffic [346], and was first presented in [313].

A *weighting factor* for each node pair in the network is calculated as follows:

$$W_{ij} = (r_i^2 \times r_j^2)/d_{ij}, \qquad i \neq j \tag{F.4}$$

The amount of traffic T_{ij} assigned to a node pair $(i,\, j)$ is then:

$$T_{ij} = (W_{ij} / \sum_{\forall i \neq j} W_{ij}) \times T \tag{F.5}$$

where T is the total traffic. Hence, the amount of traffic between each node pair is approximately proportional to the number of subscribers within a service radius (represented by the clear radius) around each of the end nodes, and inversely proportional to the distance between them (Fig. F.2).

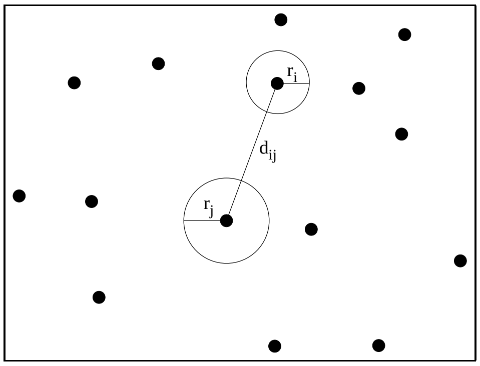
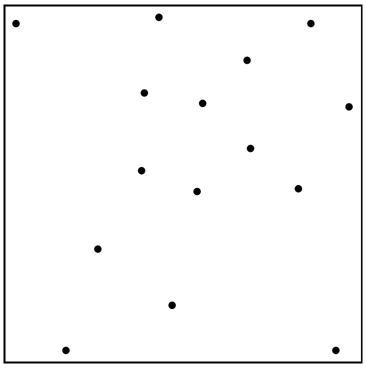


Figure F.2: Node-pair weighting factor calculation

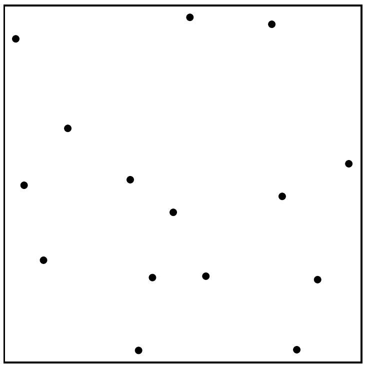
This approach assumes that the inverse distance relationship, found to be a reasonable approximation for international telephony traffic, is suitable for the traffic mix expected on the network being designed; if not, Eqn. F.4 can easily be modified to incorporate the anticipated distance dependency. In addition, another possible modification is the introduction of a small amount of (Gaussian) noise in generating either the weighting factors or the traffics themselves.

F.2 Network Design Problems

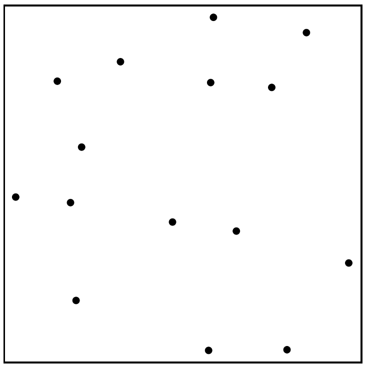
The node locations of the five fifteen-node network design problems, generated using the above method and used in §5.1, are both illustrated in Fig. F.3 and detailed in Tables F.1–F.5, with the coordinates in km from an arbitrary origin. The corresponding traffic matrices are given in Tables F.6–F.10.



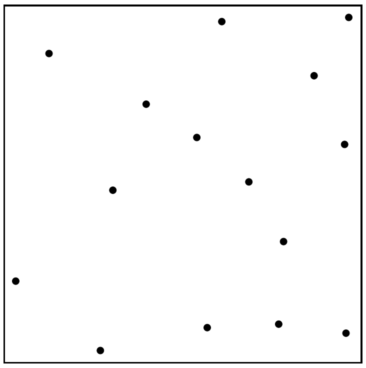
(a) Problem 1



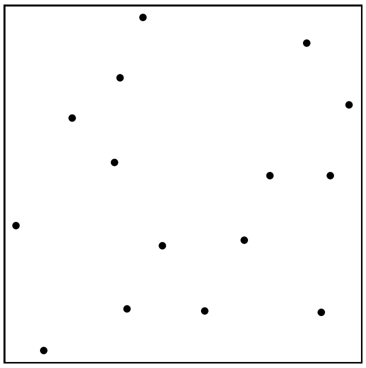
(b) Problem 2



(c) Problem 3



(d) Problem 4



(e) Problem 5

Figure F.3: Network design problems – node distributions

Node	x	y	Node	x	y
0	184.963	970.089	8	272.612	689.93
1	399.824	259.399	9	688.237	412.764
2	921.603	969.277	10	558.554	287.226
3	542.979	531.692	11	475.789	844.987
4	49.794	66.5666	12	391.042	472.429
5	819.319	523.871	13	679.526	168.088
6	853.394	67.1846	14	439.812	49.367
7	957.024	297.194			

Table F.1: Network design problem 1 – node locations

Node	x	y	Node	x	y
0	369.925	940.177	8	522.72	29.2276
1	799.649	518.797	9	571.719	736.444
2	843.205	938.555	10	998.591	429.283
3	85.9583	693.458	11	28.0581	488.135
4	159.411	333.484	12	345.844	473.75
5	768.234	48.4108	13	474.812	562.889
6	412.09	741.222	14	3.13337	87.1987
7	905.789	746.226			

Table F.2: Network design problem 2 – node locations

Node	x	y	Node	x	y
0	554.888	910.266	8	930.295	679.523
1	199.473	778.196	9	185.085	519.811
2	764.808	907.832	10	38.5772	504.265
3	628.938	595.075	11	214.456	373.646
4	149.382	199.7	12	724.512	215.422
5	457.956	571.612	13	319.437	148.101
6	560.183	201.554	14	567.147	30.0959
7	817.835	69.7889			

Table F.3: Network design problem 3 – node locations

Node	x	y	Node	x	y
0	739.85	880.355	8	903.157	379.948
1	599.297	37.5946	9	538.096	361.459
2	171.917	126.767	10	330.326	508.053
3	413.577	268.738	11	913.446	26.4141
4	828.095	188.776	12	564.17	889.717
5	90.4472	759.719	13	666.104	484.42
6	907.059	906.03	14	299.349	953.896
7	752.946	651.055			

Table F.4: Network design problem 4 – node locations

Node	x	y	Node	x	y
0	924.813	850.444	8	883.824	132.538
1	999.122	296.993	9	441.183	63.8193
2	608.013	846.387	10	378.947	224.935
3	714.896	658.459	11	172.621	951.823
4	248.97	332.833	12	948.302	486.114
5	96.5929	619.353	13	397.629	840.442
6	785.119	485.97	14	493.948	673.209
7	363.059	449.648			

Table F.5: Network design problem 5 – node locations

	To							
From	0	1	2	3	4	5	6	7
0	0	3.48041	28.2109	4.67568	16.6	3.78823	3.57941	6.18347
1	3.48041	0	7.10871	2.59262	11.4365	1.78551	2.46002	3.41731
2	28.2109	7.10871	0	11.0846	29.288	15.5817	10.7728	22.8053
3	4.67568	2.59262	11.0846	0	6.90465	3.28028	2.22152	4.1077
4	16.6	11.4365	29.288	6.90465	0	5.80517	8.85029	11.9654
5	3.78823	1.78551	15.5817	3.28028	5.80517	0	3.00873	8.18153
6	3.57941	2.46002	10.7728	2.22152	8.85029	3.00873	0	11.7723
7	6.18347	3.41731	22.8053	4.1077	11.9654	8.18153	11.7723	0
8	20.7896	4.09683	20.9339	6.01279	16.3052	3.6599	3.361	5.71205
9	3.91257	2.71065	11.8	4.83045	7.1551	5.8588	3.59739	7.41663
10	3.3198	4.83249	8.1056	3.2559	8.24158	2.5145	3.29398	4.78808
11	20.531	3.31738	34.0387	6.26122	12.9557	4.73639	3.5256	6.58389
12	4.91326	3.74075	8.82422	5.00864	8.82735	2.10229	2.0187	3.29897
13	3.04435	2.9381	8.30644	2.27951	7.95378	2.57128	6.69178	6.92201
14	4.76117	6.41144	10.616	2.84633	20.607	2.56515	5.15292	5.85751

	To							
From	8	9	10	11	12	13	14	
0	20.7896	3.91257	3.3198	20.531	4.91326	3.04435	4.76117	
1	4.09683	2.71065	20.531	3.31738	34.0387	6.26122	12.9557	
2	20.9339	11.8	8.1056	34.0387	8.82422	8.30644	10.616	
3	6.01279	4.83045	3.2559	6.26122	5.00864	2.27951	2.84633	
4	16.3052	7.1551	8.24158	12.9557	8.82735	7.95378	20.607	
5	3.6599	5.8588	2.5145	4.73639	2.10229	2.57128	2.56515	
6	3.361	3.59739	3.29398	3.5256	2.0187	6.69178	5.15292	
7	5.71205	7.41663	4.78808	6.58389	3.29897	6.92201	5.85751	
8	0	4.18596	3.72385	18.1009	7.60589	3.08525	4.89017	
9	4.18596	0	4.90572	4.62457	2.99162	4.01461	3.54067	
10	3.72385	4.90572	0	3.47403	3.19385	5.09149	5.15626	
11	18.1009	4.62457	3.47403	0	5.25084	3.07613	4.32944	
12	7.60589	2.99162	3.19385	5.25084	0	2.1113	3.29661	
13	3.08525	4.01461	5.09149	3.07613	2.1113	0	5.68844	
14	4.89017	3.54067	5.15626	4.32944	3.29661	5.68844	0	

Table F.6: Network design problem 1 – traffic (Gbit/s)

	To							
From	0	1	2	3	4	5	6	7
0	0	6.3571	6.9486	9.72611	5.15408	4.9919	10.0818	5.77075
1	6.3571	0	8.96659	5.72956	5.71384	11.8995	5.27966	15.0771
2	6.9486	8.96659	0	4.54633	3.58516	5.3979	4.277	16.0814
3	9.72611	5.72956	4.54633	0	9.9127	5.71345	6.84457	4.40464
4	5.15408	5.71384	3.58516	9.9127	0	7.21891	4.25471	3.83807
5	4.9919	11.8995	5.3979	5.71345	7.21891	0	3.85926	6.77956
6	10.0818	5.27966	4.277	6.84457	4.25471	3.85926	0	4.10727
7	5.77075	15.0771	16.0814	4.40464	3.83807	6.77956	4.10727	0
8	5.27834	9.97364	5.00121	6.74815	10.2409	29.0283	4.17233	5.93165
9	7.15029	7.4859	5.99148	4.62628	3.54007	4.20143	7.91675	6.06757
10	4.723	20.1798	7.10677	4.43091	4.50892	12.6031	3.55144	11.458
11	5.83994	4.93185	3.51491	17.0713	16.237	5.63684	4.43747	3.57801
12	4.23073	4.98555	2.87077	6.38872	8.4303	4.83321	4.42171	3.13825
13	5.0459	6.93556	3.71423	5.30038	5.04311	4.8917	6.44525	4.17262
14	7.36656	8.68726	5.6559	12.2995	23.3413	13.091	5.46776	6.05271

	To							
From	8	9	10	11	12	13	14	
0	5.27834	7.15029	4.723	5.83994	4.23073	5.0459	7.36656	
1	9.97364	7.4859	5.83994	4.93185	3.51491	17.0713	16.237	
2	5.00121	5.99148	7.10677	3.51491	2.87077	3.71423	5.6559	
3	6.74815	4.62628	4.43091	17.0713	6.38872	5.30038	12.2995	
4	10.2409	3.54007	4.50892	16.237	8.4303	5.04311	23.3413	
5	29.0283	4.20143	12.6031	5.63684	4.83321	4.8917	13.091	
6	4.17233	7.91675	3.55144	4.43747	4.42171	6.44525	5.46776	
7	5.93165	6.06757	11.458	3.57801	3.13825	4.17262	6.05271	
8	0	4.24076	9.02353	7.19227	6.0558	5.40718	19.1824	
9	4.24076	0	4.48607	3.41472	3.51687	6.12958	4.887	
10	9.02353	4.48607	0	3.91681	3.47505	4.20605	7.47683	
11	7.19227	3.41472	3.91681	0	6.18287	4.34215	16.9482	
12	6.0558	3.51687	3.47505	6.18287	0	7.48968	7.86787	
13	5.40718	6.12958	4.20605	4.34215	7.48968	0	6.06738	
14	19.1824	4.887	7.47683	16.9482	7.86787	6.06738	0	

Table F.7: Network design problem 2 – traffic (Gbit/s)

	To							
From	0	1	2	3	4	5	6	7
0	0	20.0743	23.8597	10.4555	4.38716	9.61011	4.36122	3.86114
1	20.0743	0	13.1229	11.0178	9.39288	15.5457	6.90539	5.49483
2	23.8597	13.1229	0	9.92741	3.82575	7.43681	4.20351	4.04934
3	10.4555	11.0178	9.92741	0	3.90286	13.2562	5.22912	4.11671
4	4.38716	9.39288	3.82575	3.90286	0	5.01956	5.39155	3.57813
5	9.61011	15.5457	7.43681	13.2562	5.01956	0	5.44111	3.72134
6	4.36122	6.90539	4.20351	5.22912	5.39155	5.44111	0	7.25065
7	3.86114	5.49483	4.04934	4.11671	3.57813	3.72134	7.25065	0
8	20.5076	18.6204	32.0476	19.5141	7.06505	12.605	9.22456	9.8941
9	4.5873	14.4856	3.53639	3.70343	5.48829	6.0028	3.09464	2.15682
10	3.75592	11.7999	2.96928	2.79129	5.45443	3.92522	2.52424	1.87704
11	3.97506	9.48202	3.29366	3.63302	9.74666	5.44018	4.03643	2.53838
12	4.32148	6.10245	4.45648	5.33576	3.84965	4.69546	11.5659	12.1309
13	4.49951	8.50322	4.07571	4.46178	14.4728	5.44389	8.98132	4.82953
14	3.80192	6.10077	3.71968	3.97955	5.31876	4.09433	12.0348	8.95099

	To							
From	8	9	10	11	12	13	14	
0	20.5076	4.5873	3.75592	3.97506	4.32148	4.49951	3.80192	
1	18.6204	14.4856	3.97506	9.48202	3.29366	3.63302	9.74666	
2	32.0476	3.53639	2.96928	3.29366	4.45648	4.07571	3.71968	
3	19.5141	3.70343	2.79129	3.63302	5.33576	4.46178	3.97955	
4	7.06505	5.48829	5.45443	9.74666	3.84965	14.4728	5.31876	
5	12.605	6.0028	3.92522	5.44018	4.69546	5.44389	4.09433	
6	9.22456	3.09464	2.52424	4.03643	11.5659	8.98132	12.0348	
7	9.8941	2.15682	1.87704	2.53838	12.1309	4.82953	8.95099	
8	0	5.83976	4.89743	5.85446	10.9841	7.99767	8.11453	
9	5.83976	0	8.24685	8.34516	2.4578	4.47256	2.6537	
10	4.89743	8.24685	0	5.67905	2.04539	3.89734	2.32122	
11	5.85446	8.34516	5.67905	0	2.91897	7.27611	3.42798	
12	10.9841	2.4578	2.04539	2.91897	0	5.39383	8.49426	
13	7.99767	4.47256	3.89734	7.27611	5.39383	0	8.74008	
14	8.11453	2.6537	2.32122	3.42798	8.49426	8.74008	0	

Table F.8: Network design problem 3 – traffic (Gbit/s)

	To							
From	0	1	2	3	4	5	6	7
0	0	4.27472	4.04297	1.68858	2.34375	5.981	8.21574	7.46624
1	4.27472	0	22.9635	10.3739	15.6582	11.7517	3.96408	7.1255
2	4.04297	22.9635	0	11.4638	6.80526	16.9924	3.56117	6.01457
3	1.68858	10.3739	11.4638	0	3.25985	5.66058	1.45223	2.82504
4	2.34375	15.6582	6.80526	3.25985	0	4.97925	2.2645	4.30464
5	5.981	11.7517	16.9924	5.66058	4.97925	0	4.76188	7.26023
6	8.21574	3.96408	3.56117	1.45223	2.2645	4.76188	0	5.75567
7	7.46624	7.1255	6.01457	2.82504	4.30464	7.26023	5.75567	0
8	3.89174	11.7602	7.26665	3.43644	11.7268	6.49095	3.89384	8.15475
9	2.10247	9.33258	7.38744	6.34991	4.07734	5.55292	1.77948	4.00507
10	5.63368	15.1198	20.7294	10.3637	6.19892	25.4914	4.44974	8.62251
11	1.87517	13.6601	5.99416	2.47733	10.4734	4.21356	1.85762	3.12602
12	8.54409	4.63166	4.80968	1.98122	2.35954	8.69768	4.37885	6.09473
13	3.79911	8.8997	6.88488	3.88026	5.33614	6.81571	3.15084	10.0465
14	8.07423	9.82827	11.8273	4.37208	4.55835	35.937	5.91532	8.15729

	To							
From	8	9	10	11	12	13	14	
0	3.89174	2.10247	5.63368	1.87517	8.54409	3.79911	8.07423	
1	11.7602	9.33258	1.87517	13.6601	5.99416	2.47733	10.4734	
2	7.26665	7.38744	20.7294	5.99416	4.80968	6.88488	11.8273	
3	3.43644	6.34991	10.3637	2.47733	1.98122	3.88026	4.37208	
4	11.7268	4.07734	6.19892	10.4734	2.35954	5.33614	4.55835	
5	6.49095	5.55292	25.4914	4.21356	8.69768	6.81571	35.937	
6	3.89384	1.77948	4.44974	1.85762	4.37885	3.15084	5.91532	
7	8.15475	4.00507	8.62251	3.12602	6.09473	10.0465	8.15729	
8	0	4.71993	7.82943	6.80961	3.61905	8.70561	6.3799	
9	4.71993	0	10.3271	2.73522	2.39356	7.25994	4.75457	
10	7.82943	10.3271	0	4.84699	7.53393	10.1976	18.101	
11	6.80961	2.73522	4.84699	0	1.89766	3.45591	3.81125	
12	3.61905	2.39356	7.53393	1.89766	0	3.95965	14.3123	
13	8.70561	7.25994	10.1976	3.45591	3.95965	0	6.66343	
14	6.3799	4.75457	18.101	3.81125	14.3123	6.66343	0	

Table F.9: Network design problem 4 – traffic (Gbit/s)

	To							
From	0	1	2	3	4	5	6	7
0	0	10.3285	20.4127	18.337	4.71041	17.4566	10.2602	5.81081
1	10.3285	0	4.54462	5.37594	2.53031	7.42228	6.64764	2.90508
2	20.4127	4.54462	0	12.8211	3.40048	14.2547	5.29935	4.56992
3	18.337	5.37594	12.8211	0	3.02346	10.3841	9.21669	4.20067
4	4.71041	2.53031	3.40048	3.02346	0	15.2393	2.36642	8.09108
5	17.4566	7.42228	14.2547	10.3841	15.2393	0	7.04269	15.6543
6	10.2602	6.64764	5.29935	9.21669	2.36642	7.04269	0	3.11479
7	5.81081	2.90508	4.56992	4.20067	8.09108	15.6543	3.11479	0
8	8.43706	14.3149	4.21264	4.70733	3.00266	8.08385	5.44031	3.27836
9	4.85873	3.51607	2.97956	2.93754	4.47093	8.46431	2.71127	3.75505
10	5.16948	3.25753	3.44324	3.35381	8.37056	10.9119	2.92494	6.2768
11	12.4905	4.26057	11.2453	6.59031	5.00812	34.2822	4.05383	5.81574
12	10.9696	9.69151	4.29425	5.91604	1.84304	5.72951	8.0758	2.25024
13	10.6232	3.27459	14.1423	6.56386	3.48915	18.496	3.50973	4.70411
14	12.0229	4.21434	14.3537	10.8416	4.4007	17.2282	5.3244	7.12394

	To							
From	8	9	10	11	12	13	14	
0	8.43706	4.85873	5.16948	12.4905	10.9696	10.6232	12.0229	
1	14.3149	3.51607	12.4905	4.26057	11.2453	6.59031	5.00812	
2	4.21264	2.97956	3.44324	11.2453	4.29425	14.1423	14.3537	
3	4.70733	2.93754	3.35381	6.59031	5.91604	6.56386	10.8416	
4	3.00266	4.47093	8.37056	5.00812	1.84304	3.48915	4.4007	
5	8.08385	8.46431	10.9119	34.2822	5.72951	18.496	17.2282	
6	5.44031	2.71127	2.92494	4.05383	8.0758	3.50973	5.3244	
7	3.27836	3.75505	6.2768	5.81574	2.25024	4.70411	7.12394	
8	0	4.99279	4.16813	4.35586	5.55458	3.25136	4.18887	
9	4.99279	0	9.15995	3.76703	2.23714	2.65465	3.37585	
10	4.16813	9.15995	0	4.42425	2.25449	3.20761	4.26805	
11	4.35586	3.76703	4.42425	0	3.44797	17.3785	10.2591	
12	5.55458	2.23714	2.25449	3.44797	0	2.81481	3.75115	
13	3.25136	2.65465	3.20761	17.3785	2.81481	0	13.3583	
14	4.18887	3.37585	4.26805	10.2591	3.75115	13.3583	0	

Table F.10: Network design problem 5 – traffic (Gbit/s)

F.3 Mesh Network Topology Design

F.3.1 Initial Topology Experiments

The non-default parameters values common to the three GAs discussed in §5.1.1 on p. 93 are listed in Table F.11. Typical topology results, showing progress and operator-probability adaptation over single runs of both GA1 and GA2 on one network design problem are illustrated in Fig. F.4. Equivalent traces for GA3, as well as a comparison of the lowest cost curves for all three GAs, were given in Fig. 5.1 on p. 95. The complete results from applying the GAs to all five of the network design problems are recorded in Tables F.12–F.14 (these were summarised in Table 5.2 on p. 97).

As an example of the application of the non-parametric sign test used throughout Ch. 5, consider Table F.15. In its second and third columns, this records the topology cost results from applying GA1 and GA2 to Problem 1, reproduced from the second columns of Tables F.12 and F.13, respectively. Each row in the table therefore represents two GA runs using the same random seed, and thus starting from exactly the same initial population; they differ only in the GA used. The fourth column of the table records whether, for each row, the result from GA1 was greater than that for GA2. Taking as the *null hypothesis* [307] that the two sets of results are drawn from the same distribution, and therefore that there is no median difference between the two GAs, the expected number of rows with GA1 greater than GA2 is 5 out of 10 (equivalent to tossing an unbiased coin). The actual distribution is binomial, with the probability that a GA1 result is greater, $p = 0.5$. Thus the probability of $(x_1, x_2) = (0, 10)$, where x_1 is the number of rows in which GA1 is greater and x_2 the number of rows in which GA2 is greater, is:

$$\text{prob}[(x_1, x_2) = (0, 10)] = \begin{pmatrix} 10 \\ 0 \end{pmatrix} p^0 p^{10} = p^{10} = 0.098\% \tag{F.6}$$

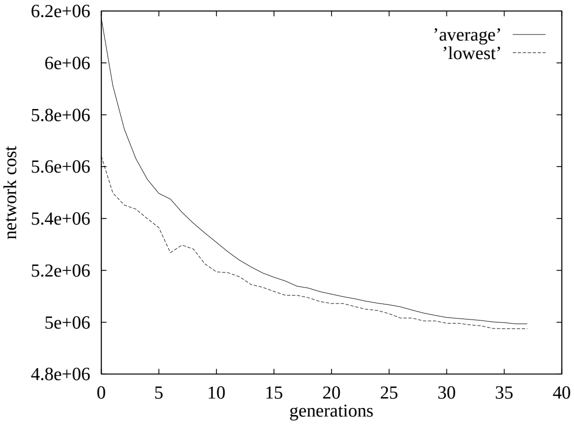
Similarly, for $(x_1, x_2) = (1, 9)$, $(2, 8)$ and $(3, 7)$:

$$\text{prob}[(x_1, x_2) = (1, 9)] = \begin{pmatrix} 10 \\ 1 \end{pmatrix} p^1 p^9 = 10p^{10} = 0.977\% \tag{F.7}$$

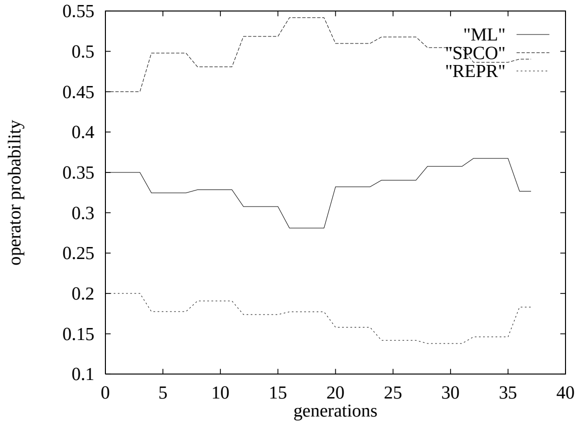
$$\text{prob}[(x_1, x_2) = (2, 8)] = \begin{pmatrix} 10 \\ 2 \end{pmatrix} p^2 p^8 = 45p^{10} = 4.39\% \tag{F.8}$$

Parameter	Value	Description
`ales.eMin`	`70.0`	Constant K_0, shared with `AlesRouter`
`ales.eLink`	`35.0`	Constant K_n, shared with `AlesRouter`
`ales.nnPenalty`	`200,000.0`	Penalty P_N
`ales.apPenalty`	`100,000.0`	Penalty P_R
`maxTrials`	`5,000`	Maximum number of trials; must be ≥ 1
`pop.size`	`200`	Population size
`pop.select`	`tournament`	Selection operator

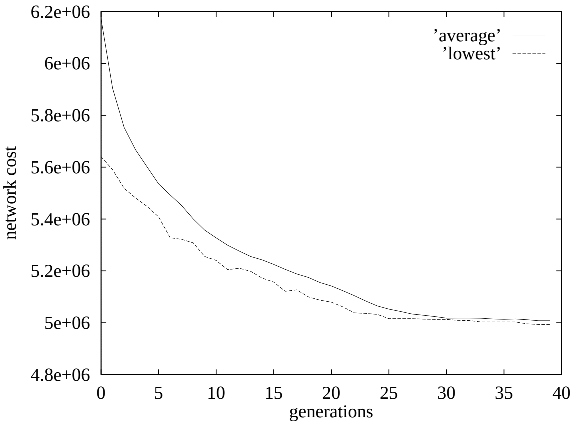
Table F.11: Non-default parameters for GA1, GA2 and GA3



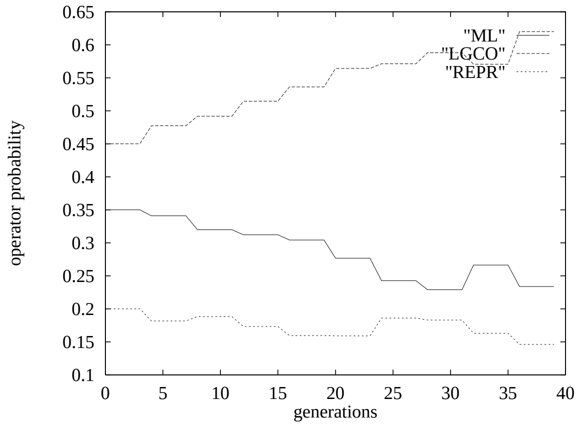
(a) Progress over a single GA1 run



(b) Operator-probability adaptation for GA1



(c) Progress over a single GA2 run



(d) Operator-probability adaptation for GA2

Figure F.4: Typical initial topology results (Problem 1, Run 1)

Run	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)
1	4.975	4.806	4.487	4.619	4.522
2	4.997	4.803	4.492	4.649	4.489
3	4.982	4.786	4.469	4.635	4.461
4	4.990	4.793	4.489	4.646	4.471
5	4.999	4.785	4.450	4.635	4.481
6	5.017	4.785	4.478	4.706	4.487
7	5.017	4.772	4.446	4.658	4.501
8	4.982	4.773	4.485	4.618	4.486
9	5.029	4.801	4.487	4.652	4.479
10	4.982	4.818	4.486	4.634	4.477
Best	4.975	4.772	4.446	4.618	4.461
Median	4.994	4.789	4.486	4.641	4.484

Table F.12: Topology cost results for GA1

Run	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)
1	4.994	4.785	4.416	4.619	4.463
2	4.954	4.774	4.443	4.610	4.449
3	4.971	4.764	4.415	4.619	4.467
4	4.978	4.751	4.425	4.607	4.427
5	4.993	4.752	4.434	4.605	4.448
6	4.953	4.751	4.423	4.616	4.448
7	4.982	4.781	4.417	4.613	4.424
8	4.969	4.765	4.426	4.602	4.450
9	4.959	4.775	4.419	4.593	4.451
10	5.005	4.761	4.418	4.618	4.496
Best	4.953	4.751	4.415	4.593	4.424
Median	4.975	4.764	4.421	4.612	4.450

Table F.13: Topology cost results for GA2

Run	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$	4 ($\times 10^6$)	5 ($\times 10^6$)
1	4.960	4.782	4.425	4.621	4.441
2	4.990	4.755	4.432	4.608	4.487
3	4.999	4.782	4.435	4.591	4.467
4	4.958	4.776	4.404	4.611	4.445
5	4.977	4.760	4.448	4.604	4.459
6	4.969	4.772	4.417	4.621	4.452
7	4.954	4.767	4.439	4.625	4.438
8	4.994	4.756	4.429	4.618	4.469
9	4.976	4.767	4.422	4.606	4.455
10	4.978	4.768	4.459	4.621	4.452
Best	4.954	4.755	4.404	4.591	4.438
Median	4.977	4.767	4.430	4.614	4.454

Table F.14: Topology cost results for GA3

Run	GA1 ($\times 10^6$)	GA2 ($\times 10^6$)	GA1 > GA2 ?
1	4.975	4.994	0
2	4.997	4.954	1
3	4.982	4.971	1
4	4.990	4.978	1
5	4.999	4.993	1
6	5.017	4.953	1
7	5.017	4.982	1
8	4.982	4.969	1
9	5.029	4.959	1
10	4.982	5.005	0
Best	4.975	4.953	
Median	4.994	4.975	

Table F.15: Applying sign test to topology cost results for GA1 and GA2 on Problem 1

$$\text{prob}[(x_1, x_2) = (3, 7)] = \begin{pmatrix} 10 \\ 3 \end{pmatrix} p^3 p^7 = 120p^{10} = 11.72\% \tag{F.9}$$

Consequently, the cumulative prob$[x_1 \leq 0] = 0.1\%$, prob$[x_1 \leq 1] = 1.1\%$ and prob$[x_1 \leq 2] = 5.5\%$. Now, in hypothesis testing, if an event occurs of sufficiently low probability, given the null hypothesis, this is taken to be significant, and the null hypothesis rejected. In this case, that would indicate there was indeed a median difference between the two GAs. Conventionally, a probability of 0.1% is taken as very highly significant, 1% as highly significant, and 5% as significant; values greater than 5% are not significant[2]. In this thesis, as comparisons were made on the basis of 10 runs, 1.1% was taken as highly significant, and 5.5% as significant. Thus it is these cumulative probabilities which are recorded in the appropriate tables in Ch. 5. For example, in Table F.15, $x_1 = 2$, and so a value of 5.5% is recorded for $s_{1,2}$, the significance level of the comparison of GA1 with GA2, in Table 5.2 on p. 97. Using a non-parametric sign test in this way allows median differences to be detected using only 10 runs where a much greater number would be needed if a Gaussian-based test were used instead. In addition, a non-parametric test does not assume an underlying Gaussian distribution in the samples (even in the limit), which in any case is unlikely here, given that they are all final results from a minimisation process, and consequently their distributions are very likely to be skewed (asymmetric), with a longer tail on the upper side.

[2]Strictly, the rejection of the null hypothesis at this level—a one-tail test—does not indicate whether it is GA1 which is better than GA2 or not. Significance levels only half those indicated would be needed for such a two-tail test; however, this technicality has not been observed in the thesis.

F.3.2 Test Networks

The non-default parameters values used with GA2 and GA3 to generate the two additional test networks described in §5.1.2 are listed in Tables F.16 and F.17. The complete results from then applying the GAs employing the two parameter sets to Problem 1 are detailed in Tables F.18 and F.19. The particular GA runs selected to provide both the original five test networks and the two additional ones (illustrated in Fig. 5.3 on p. 99 and Fig. 5.4 on p. 99, respectively) are recorded in Table F.20. In addition, a list of network links for all seven test networks (using the node ids introduced in §F.2) is given in Table F.21.

Parameter	Value	Description
`ales.eMin`	`70.0`	Constant K_0, shared with `AlesRouter`
`ales.eLink`	`35.0`	Constant K_n, shared with `AlesRouter`
`ales.nnPenalty`	`200,000.0`	Penalty P_N
`ales.apPenalty`	`100,000.0`	Penalty P_R
`ales.alpha`	`0.8`	Constant α
`maxTrials`	`5,000`	Maximum number of trials; must be ≥ 1
`pop.size`	`200`	Population size
`pop.select`	`tournament`	Selection operator

Table F.16: Reduced α parameter set for GA2 and GA3

Parameter	Value	Description
`ales.eMin`	`100.0`	Constant K_0, shared with `AlesRouter`
`ales.eLink`	`800.0`	Constant K_n, shared with `AlesRouter`
`ales.nnPenalty`	`1,000,000.0`	Penalty P_N
`ales.apPenalty`	`500,000.0`	Penalty P_R
`ales.alpha`	`0.8`	Constant α
`maxTrials`	`5,000`	Maximum number of trials; must be ≥ 1
`pop.size`	`200`	Population size
`pop.select`	`tournament`	Selection operator

Table F.17: Increased node cost parameter set for GA2 and GA3

GA	Run				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)
2	2.981	2.991	2.972	3.011	2.990
3	2.982	2.987	2.990	2.986	3.002

Table F.18: Reduced α topology results for Problem 1

GA	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)
2	30.32	30.78	30.54	30.38	30.91
3	30.71	30.58	30.31	30.48	30.25

Table F.19: Increased node cost topology results for Problem 1

Network	GA	Run	Cost ($\times 10^6$)
1	2	2	4.954
2	2	4	4.751
3	3	4	4.404
4	3	3	4.591
5	2	4	4.427
6	2	3	2.972
7	3	5	30.25

Table F.20: Test network selection

Network	Links							
1	(0,2)	(0,4)	(0,8)	(0,11)	(1,4)	(1,10)	(1,12)	(1,14)
	(2,5)	(2,7)	(2,8)	(2,9)	(2,11)	(2,12)	(3,5)	(3,8)
	(3,9)	(3,11)	(3,12)	(4,8)	(4,12)	(4,13)	(4,14)	(5,6)
	(5,7)	(5,9)	(6,7)	(6,13)	(6,14)	(7,9)	(7,13)	(8,11)
	(8,12)	(9,10)	(9,11)	(9,13)	(10,12)	(10,13)	(10,14)	(13,14)
2	(0,2)	(0,3)	(0,6)	(0,9)	(1,2)	(1,4)	(1,5)	(1,7)
	(1,9)	(1,10)	(1,13)	(2,7)	(2,9)	(3,4)	(3,6)	(3,11)
	(4,8)	(4,11)	(4,12)	(4,14)	(5,8)	(5,10)	(5,13)	(5,14)
	(6,9)	(6,12)	(6,13)	(7,9)	(7,10)	(8,10)	(8,12)	(8,14)
	(9,12)	(9,13)	(11,13)	(11,14)	(12,13)	(12,14)		
3	(0,1)	(0,2)	(0,3)	(0,5)	(0,8)	(1,2)	(1,5)	(1,9)
	(1,10)	(2,3)	(2,8)	(3,5)	(3,6)	(3,8)	(3,13)	(4,10)
	(4,11)	(4,13)	(4,14)	(5,9)	(5,11)	(5,12)	(6,7)	(6,9)
	(6,12)	(6,13)	(6,14)	(7,8)	(7,12)	(7,14)	(8,11)	(8,12)
	(9,10)	(9,11)	(10,11)	(11,13)	(12,14)	(13,14)		
4	(0,1)	(0,6)	(0,7)	(0,12)	(0,14)	(1,2)	(1,3)	(1,4)
	(1,9)	(1,11)	(2,3)	(2,5)	(2,10)	(3,4)	(3,9)	(3,10)
	(4,8)	(4,11)	(4,13)	(5,10)	(5,12)	(5,14)	(6,7)	(6,8)
	(6,12)	(7,8)	(7,9)	(7,12)	(7,13)	(8,11)	(8,13)	(9,10)
	(9,13)	(10,13)	(10,14)	(12,14)	(13,14)			
5	(0,2)	(0,3)	(0,6)	(0,12)	(0,13)	(1,6)	(1,8)	(1,12)
	(1,14)	(2,3)	(2,11)	(2,13)	(2,14)	(3,6)	(3,12)	(3,14)
	(4,5)	(4,7)	(4,9)	(4,10)	(5,7)	(5,11)	(5,13)	(5,14)
	(6,7)	(6,9)	(6,12)	(7,10)	(7,14)	(8,9)	(8,10)	(8,12)
	(9,10)	(10,11)	(11,13)	(13,14)				
6	(0,2)	(0,4)	(0,8)	(0,11)	(1,4)	(1,10)	(1,12)	(1,14)
	(2,5)	(2,7)	(2,9)	(2,11)	(3,6)	(3,8)	(3,9)	(3,12)
	(4,8)	(4,14)	(5,7)	(5,9)	(5,13)	(6,7)	(6,10)	(6,13)
	(7,13)	(8,11)	(9,10)	(10,14)	(11,12)	(13,14)		
7	(0,2)	(0,4)	(0,8)	(0,11)	(1,5)	(1,10)	(1,14)	(2,4)
	(2,6)	(2,9)	(3,8)	(3,12)	(3,14)	(4,10)	(4,14)	(5,6)
	(5,8)	(6,7)	(6,13)	(7,10)	(7,12)	(8,11)	(9,11)	(9,12)
	(10,13)							

Table F.21: Test network links

F.3.3 Local Search Operators

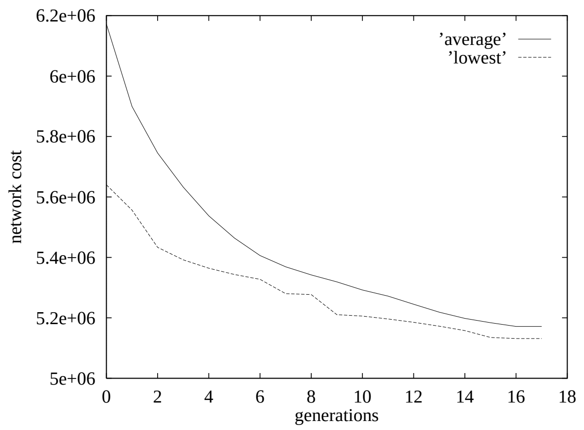
The non-default parameters values common to the nine GAs discussed in §5.1.3 on p. 100 are listed in Table F.22. Typical topology results, showing progress and operator-probability adaptation over single runs of GA5-GA12 on one network design problem are illustrated in Figures F.5–F.8. Equivalent traces for GA4, as well as a comparison of the lowest cost curves for five of the GAs, were given in Fig. 5.5 on p. 102. (It should be noted that Fig. 5.5(c) and (d) have been repeated here as Fig. F.7(b) and (d) for ease of reference.) The complete results from applying the GAs to all five of the network design problems are recorded in Tables F.23–F.31 (these were summarised in Tables 5.5 and 5.6 on p. 103).

Parameter	Value	Description
`ales.eMin`	`70.0`	Constant K_0, shared with `AlesRouter`
`ales.eLink`	`35.0`	Constant K_n, shared with `AlesRouter`
`ales.nnPenalty`	`200,000.0`	Penalty P_N
`ales.apPenalty`	`100,000.0`	Penalty P_R
`maxTrials`	`15,000`	Maximum number of trials; must be ≥ 1
`pop.size`	`200`	Population size
`pop.select`	`tournament`	Selection operator
`opp.fitCredDiv`	—	Credit divisor: `children` or `trials`

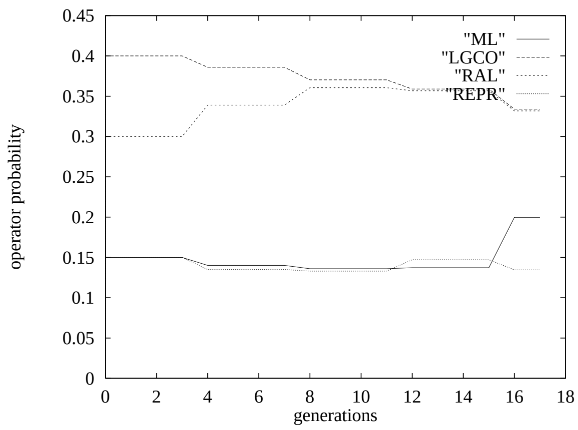
Table F.22: Non-default parameters for GA4–GA12

Run	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$	4 ($\times 10^6$)	5 ($\times 10^6$)
1	4.960	4.746	4.413	4.605	4.429
2	4.953	4.756	4.442	4.598	4.449
3	4.970	4.761	4.409	4.596	4.430
4	4.947	4.746	4.425	4.604	4.427
5	4.963	4.752	4.434	4.591	4.430
6	4.953	4.751	4.414	4.609	4.443
7	4.957	4.765	4.409	4.605	4.424
8	4.969	4.765	4.409	4.592	4.441
9	4.956	4.755	4.412	4.593	4.428
10	4.967	4.752	4.410	4.607	4.446
Best	4.947	4.746	4.409	4.591	4.424
Median	4.958	4.753	4.412	4.601	4.430

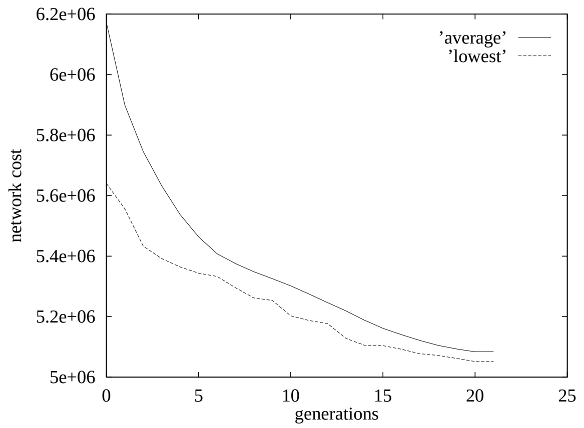
Table F.23: Topology cost results for GA4



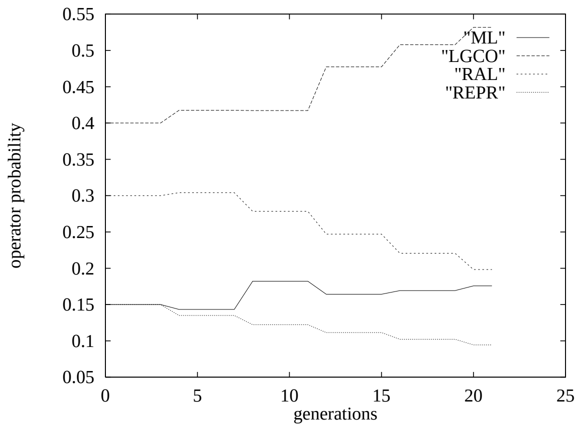
(a) Progress over a single GA5 run



(b) Operator-probability adaptation for GA5



(c) Progress over a single GA6 run

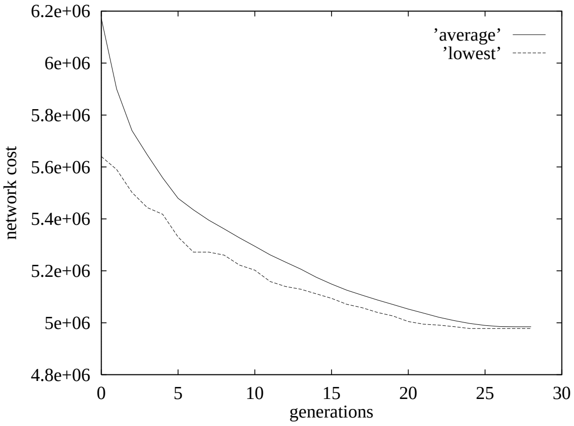


(d) Operator-probability adaptation for GA6

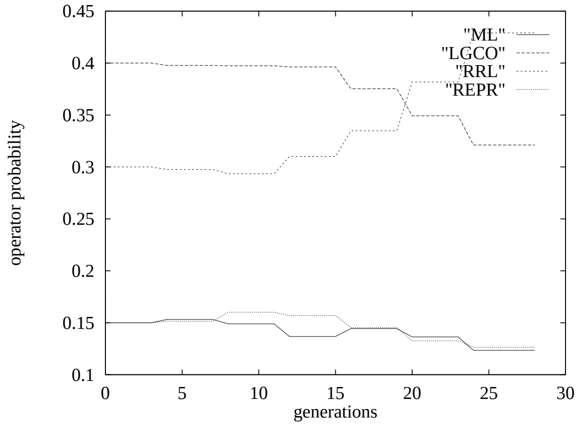
Figure F.5: Typical topology results for GA5 and GA6 (Problem 1, Run 1)

Run	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$	4 ($\times 10^6$)	5 ($\times 10^6$)
1	5.132	4.838	4.506	4.719	4.531
2	5.073	4.858	4.542	4.735	4.573
3	5.092	4.899	4.564	4.716	4.555
4	5.050	4.899	4.556	4.716	4.566
5	5.087	4.844	4.474	4.713	4.584
6	5.019	4.883	4.544	4.692	4.628
7	4.988	4.878	4.604	4.766	4.549
8	5.046	4.917	4.485	4.694	4.513
9	5.090	4.849	4.536	4.725	4.617
10	5.042	4.864	4.529	4.748	4.579
Best	4.988	4.838	4.474	4.692	4.513
Median	5.062	4.871	4.539	4.718	4.569

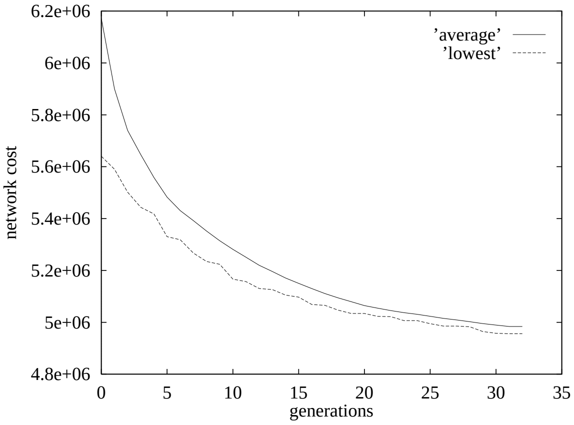
Table F.24: Topology cost results for GA5



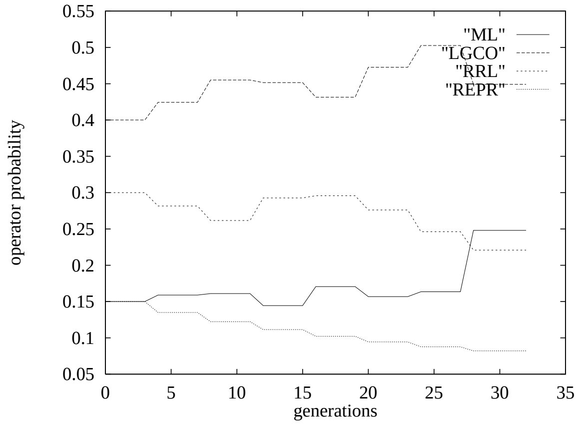
(a) Progress over a single GA7 run



(b) Operator-probability adaptation for GA7



(c) Progress over a single GA8 run

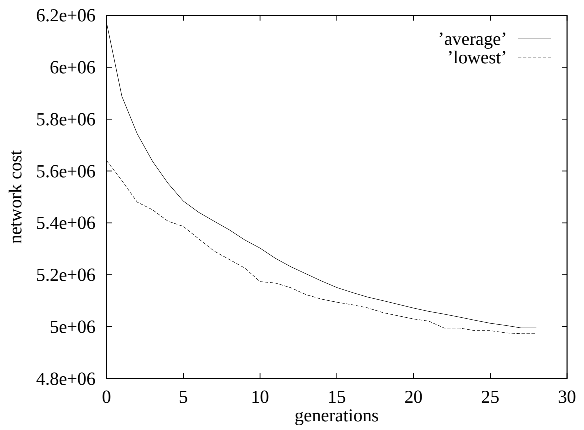


(d) Operator-probability adaptation for GA8

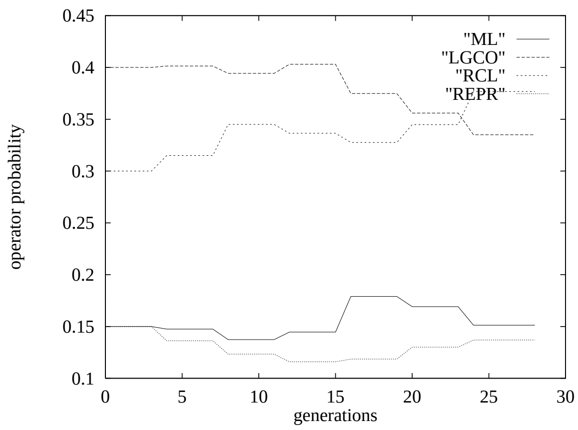
Figure F.6: Typical topology results for GA7 and GA8 (Problem 1, Run 1)

Run	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$	4 ($\times 10^6$)	5 ($\times 10^6$)
1	5.052	4.857	4.480	4.702	4.452
2	5.037	4.838	4.537	4.732	4.536
3	5.087	4.850	4.510	4.670	4.558
4	5.008	4.842	4.533	4.616	4.553
5	5.034	4.855	4.466	4.674	4.524
6	5.006	4.831	4.485	4.641	4.611
7	4.987	4.816	4.502	4.693	4.479
8	5.003	4.872	4.495	4.666	4.491
9	5.059	4.812	4.497	4.719	4.530
10	4.996	4.809	4.487	4.647	4.486
Best	4.987	4.809	4.466	4.616	4.452
Median	5.021	4.840	4.496	4.672	4.527

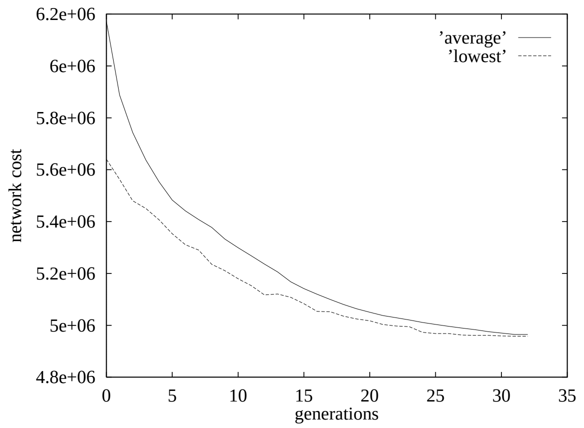
Table F.25: Topology cost results for GA6



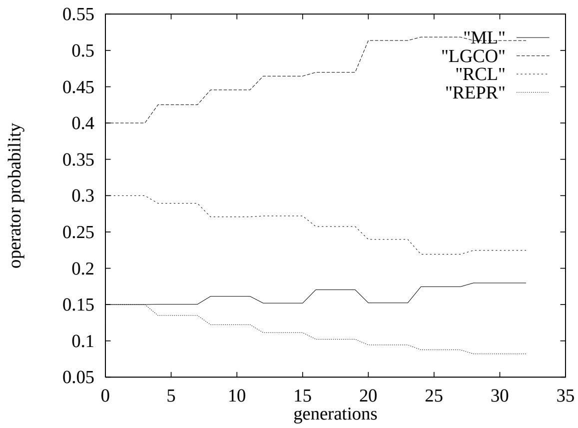
(a) Progress over a single GA9 run



(b) Operator-probability adaptation for GA9



(c) Progress over a single GA10 run

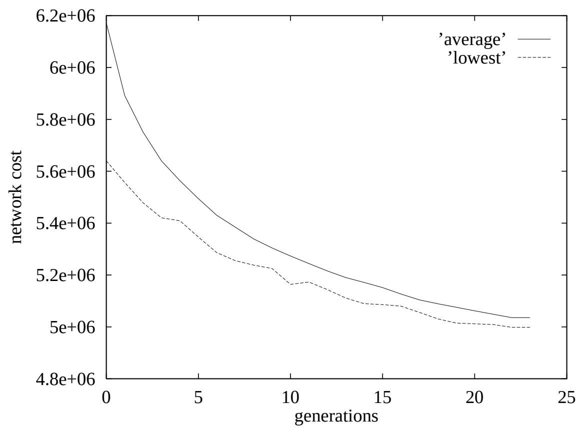


(d) Operator-probability adaptation for GA10

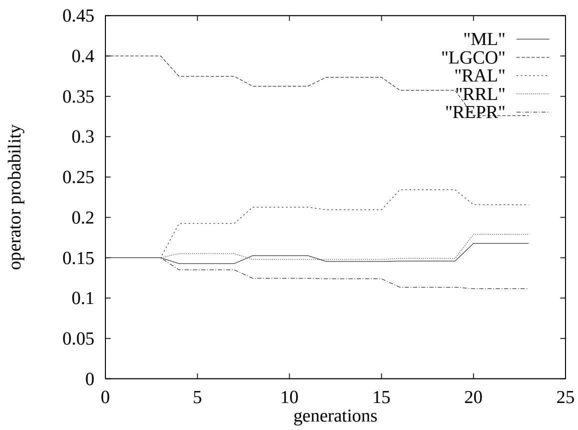
Figure F.7: Typical topology results for GA9 and GA10 (Problem 1, Run 1)

Run	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$	4 ($\times 10^6$)	5 ($\times 10^6$)
1	4.978	4.747	4.426	4.596	4.448
2	4.953	4.755	4.425	4.609	4.471
3	4.988	4.779	4.433	4.600	4.445
4	4.993	4.770	4.421	4.607	4.459
5	4.972	4.780	4.418	4.592	4.439
6	4.961	4.762	4.436	4.622	4.441
7	4.983	4.753	4.458	4.592	4.456
8	4.984	4.759	4.423	4.607	4.466
9	4.965	4.786	4.424	4.649	4.468
10	5.016	4.774	4.433	4.659	4.435
Best	4.953	4.747	4.418	4.592	4.435
Median	4.980	4.766	4.425	4.607	4.452

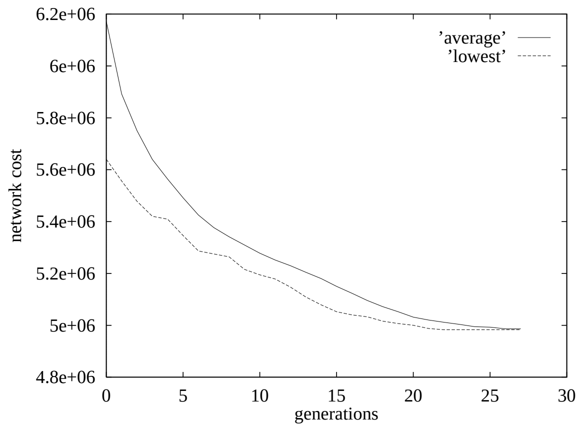
Table F.26: Topology cost results for GA7



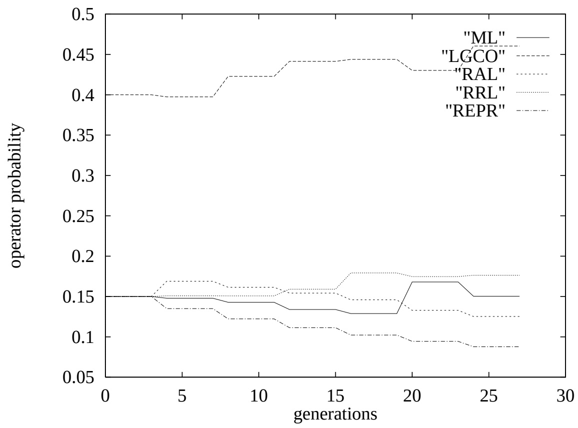
(a) Progress over a single GA11 run



(b) Operator-probability adaptation for GA11



(c) Progress over a single GA12 run



(d) Operator-probability adaptation for GA12

Figure F.8: Typical topology results for GA11 and GA12 (Problem 1, Run 1)

Run	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$	4 ($\times 10^6$)	5 ($\times 10^6$)
1	4.956	4.745	4.407	4.609	4.454
2	4.968	4.779	4.439	4.622	4.468
3	4.970	4.765	4.438	4.602	4.451
4	4.989	4.759	4.421	4.607	4.477
5	4.962	4.764	4.418	4.600	4.434
6	4.961	4.767	4.438	4.598	4.437
7	4.964	4.761	4.407	4.602	4.434
8	4.991	4.762	4.423	4.616	4.487
9	4.975	4.794	4.413	4.623	4.479
10	4.998	4.751	4.409	4.611	4.446
Best	4.956	4.745	4.407	4.598	4.434
Median	4.969	4.763	4.420	4.608	4.452

Table F.27: Topology cost results for GA8

Run	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$	4 ($\times 10^6$)	5 ($\times 10^6$)
1	4.973	4.753	4.434	4.618	4.438
2	4.980	4.756	4.414	4.607	4.439
3	4.952	4.767	4.423	4.599	4.438
4	4.962	4.757	4.400	4.594	4.454
5	4.962	4.763	4.439	4.598	4.430
6	4.952	4.759	4.419	4.607	4.447
7	4.964	4.763	4.412	4.608	4.461
8	4.973	4.761	4.418	4.601	4.459
9	4.950	4.750	4.423	4.595	4.441
10	4.947	4.747	4.452	4.612	4.417
Best	4.947	4.747	4.400	4.594	4.417
Median	4.962	4.758	4.421	4.604	4.440

Table F.28: Topology cost results for GA9

Run	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$	4 ($\times 10^6$)	5 ($\times 10^6$)
1	4.958	4.748	4.415	4.612	4.448
2	4.953	4.760	4.408	4.592	4.440
3	4.946	4.750	4.416	4.589	4.426
4	4.957	4.754	4.409	4.594	4.435
5	4.962	4.755	4.422	4.595	4.467
6	4.963	4.758	4.414	4.607	4.444
7	4.965	4.768	4.409	4.601	4.434
8	4.949	4.752	4.409	4.607	4.451
9	4.955	4.772	4.420	4.606	4.427
10	4.979	4.748	4.427	4.602	4.436
Best	4.946	4.748	4.408	4.589	4.426
Median	4.957	4.755	4.414	4.602	4.438

Table F.29: Topology cost results for GA10

Run	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$	4 ($\times 10^6$)	5 ($\times 10^6$)
1	4.998	4.750	4.450	4.686	4.439
2	4.989	4.777	4.459	4.628	4.515
3	5.073	4.772	4.447	4.710	4.474
4	4.978	4.761	4.452	4.619	4.467
5	5.001	4.875	4.466	4.625	4.455
6	4.970	4.766	4.484	4.637	4.508
7	5.015	4.792	4.484	4.634	4.470
8	4.962	4.810	4.490	4.641	4.478
9	4.983	4.847	4.462	4.676	4.536
10	4.957	4.772	4.464	4.686	4.471
Best	4.957	4.750	4.447	4.619	4.439
Median	4.986	4.774	4.463	4.639	4.473

Table F.30: Topology cost results for GA11

Run	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$	4 ($\times 10^6$)	5 ($\times 10^6$)
1	4.983	4.754	4.417	4.621	4.458
2	4.951	4.767	4.423	4.616	4.478
3	4.981	4.757	4.415	4.642	4.472
4	4.976	4.755	4.440	4.615	4.451
5	4.972	4.798	4.434	4.662	4.433
6	4.980	4.753	4.421	4.639	4.460
7	4.974	4.758	4.433	4.593	4.423
8	4.954	4.768	4.446	4.616	4.466
9	4.978	4.799	4.427	4.608	4.490
10	4.975	4.760	4.477	4.602	4.445
Best	4.951	4.753	4.415	4.593	4.423
Median	4.975	4.759	4.430	4.616	4.459

Table F.31: Topology cost results for GA12

F.3.4 Iterative Heuristic Operators

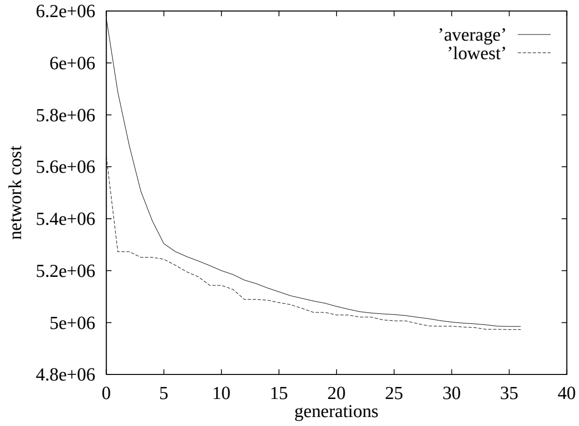
The non-default parameters values common to the three GAs discussed in §5.1.4 on p. 105 are listed in Table F.32. Typical topology results, showing progress and operator-probability adaptation over single runs of GA13-GA15 on one network design problem are illustrated in Fig. F.9. A comparison of the lowest cost curves for all three of the GAs was given in Fig. 5.6 on p. 107. In addition, the complete results from applying the GAs to all five of the network design problems are recorded in Tables F.33–F.35 (these were summarised in Table 5.11 on p. 107).

Parameter	Value	Description
`ales.eMin`	`70.0`	Constant K_0, shared with `AlesRouter`
`ales.eLink`	`35.0`	Constant K_n, shared with `AlesRouter`
`ales.nnPenalty`	`200,000.0`	Penalty P_N
`ales.apPenalty`	`100,000.0`	Penalty P_R
`maxTrials`	`40,000`	Maximum number of trials; must be ≥ 1
`pop.size`	`200`	Population size
`pop.select`	`tournament`	Selection operator
`opp.fitCredDiv`	`trials`	Credit divisor

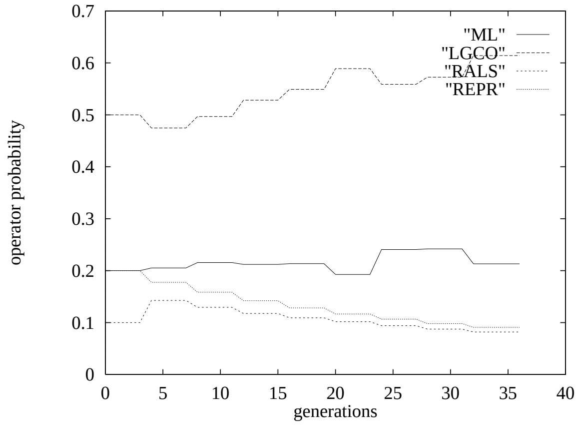
Table F.32: Non-default parameters for GA13–GA15

Run	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$	4 ($\times 10^6$)	5 ($\times 10^6$)
1	4.973	4.752	4.444	4.607	4.430
2	4.956	4.755	4.431	4.591	4.434
3	4.949	4.760	4.437	4.607	4.459
4	4.959	4.753	4.408	4.595	4.419
5	4.936	4.758	4.426	4.597	4.443
6	4.952	4.756	4.421	4.603	4.440
7	4.960	4.749	4.420	4.593	4.445
8	4.947	4.758	4.434	4.604	4.435
9	4.947	4.747	4.427	4.604	4.450
10	4.951	4.759	4.419	4.605	4.441
Best	4.936	4.747	4.408	4.591	4.419
Median	4.951	4.755	4.426	4.603	4.441

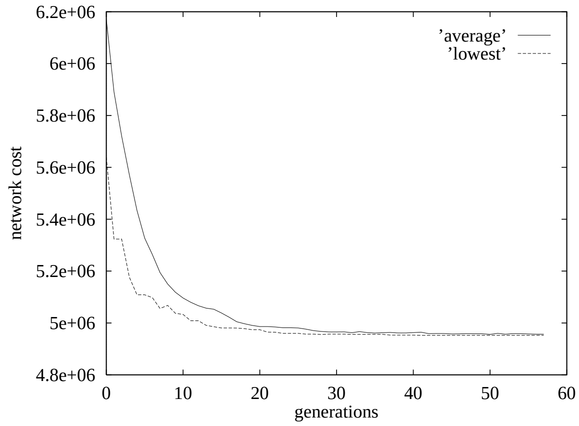
Table F.33: Topology cost results for GA13



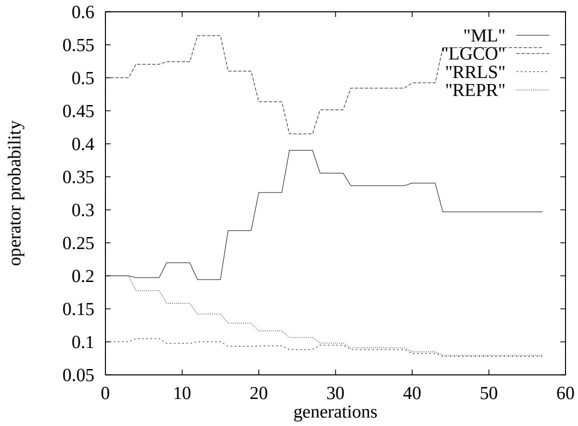
(a) Progress over a single GA13 run



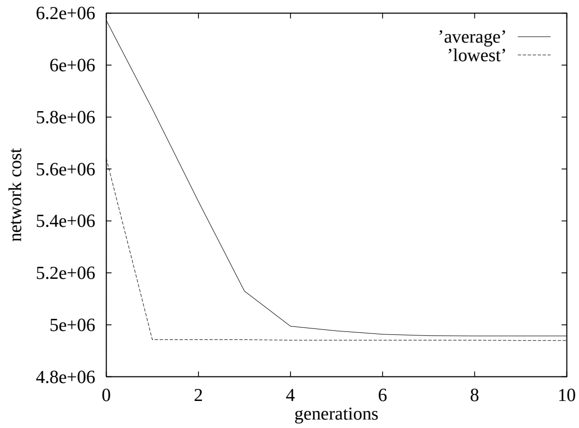
(b) Operator-probability adaptation for GA13



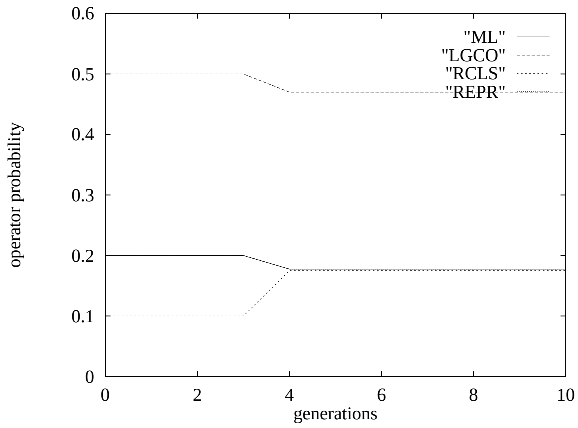
(c) Progress over a single GA14 run



(d) Operator-probability adaptation for GA14



(e) Progress over a single GA15 run



(f) Operator-probability adaptation for GA15

Figure F.9: Typical topology results for GA13, GA14 and GA15 (Problem 1, Run 1)

Run	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$	4 ($\times 10^6$)	5 ($\times 10^6$)
1	4.952	4.748	4.416	4.590	4.442
2	4.936	4.757	4.439	4.592	4.427
3	4.955	4.739	4.409	4.600	4.457
4	4.944	4.757	4.408	4.594	4.419
5	4.941	4.755	4.400	4.590	4.426
6	4.954	4.765	4.408	4.595	4.431
7	4.942	4.748	4.416	4.596	4.421
8	4.937	4.756	4.418	4.607	4.440
9	4.970	4.743	4.406	4.590	4.432
10	4.969	4.761	4.413	4.595	4.428
Best	4.936	4.739	4.400	4.590	4.419
Median	4.948	4.756	4.411	4.594	4.430

Table F.34: Topology cost results for GA14

Run	Problem				
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$	4 ($\times 10^6$)	5 ($\times 10^6$)
1	4.939	4.744	4.400	4.588	4.419
2	4.935	4.736	4.404	4.587	4.418
3	4.944	4.739	4.404	4.587	4.418
4	4.936	4.744	4.404	4.588	4.418
5	4.936	4.744	4.404	4.587	4.420
6	4.942	4.744	4.404	4.588	4.417
7	4.937	4.743	4.400	4.589	4.421
8	4.938	4.741	4.400	4.587	4.418
9	4.935	4.741	4.400	4.585	4.416
10	4.941	4.741	4.400	4.587	4.419
Best	4.935	4.736	4.400	4.585	4.416
Median	4.937	4.742	4.402	4.587	4.418

Table F.35: Topology cost results for GA15

F.4 Routing, Fibre & Wavelength Allocation

F.4.1 Minimum Network Wavelength Requirement

The non-default parameters values for the two GAs discussed in §5.2.1 on p. 109 are given in Tables F.36 and F.37. The complete network wavelength requirement (NWR) results from applying them to Networks 1–5 are then recorded in Tables F.38 and F.39. The best of each of these groups of five runs was included in Table 5.15 on p. 112.

The non-default parameters values for the three heuristics discussed in §5.2.1 are given in Table F.40. The NWR results from applying one of these, Wuttisittikulkij & O'Mahony's heuristic ($k = 8$, sorted order) [298] (W&O), to Networks 1–5 are listed in Table F.41. The best of each of these groups of five runs was again included in Table 5.15.

Parameter	Value	Description
`net.metric`	`NWR`	Metric used for network self-assessment
`net.router`	`WNOP`	Router used to route the network prior to self-assessment
`maxTrials`	`500,000`	Maximum number of trials; must be ≥ 1
`pop.size`	`500`	Population size
`pop.select`	`tournament`	Selection operator
`pop.initOp`	`WRKS`	Population initialisation operator
`pop.phyNetwork`	`true`	Construct the individual using a `PhyNetwork` (`true`) or a `Network` (`false`)
`wrks.theK`	`8`	Number of alternative paths, k
`wrks.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wkm.theK`	`8`	Number of alternative paths, k
`wkm.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wkco.theK`	`8`	Number of alternative paths, k
`wkco.firstFibreOnly`	`false`	Use only first fibre on each cable?

Table F.36: Non-default parameters for GA16

Parameter	Value	Description
`net.metric`	`NWR`	Metric used for network self-assessment
`net.router`	`WNOP`	Router used to route the network prior to self-assessment
`maxTrials`	`75,000`	Maximum number of trials; must be ≥ 1
`pop.size`	`500`	Population size
`pop.select`	`tournament`	Selection operator
`pop.initOp`	`WRKS`	Population initialisation operator
`pop.phyNetwork`	`true`	Construct the individual using a `PhyNetwork` (`true`) or a `Network` (`false`)
`wrks.theK`	`8`	Number of alternative paths, k
`wrks.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wkm.theK`	`8`	Number of alternative paths, k
`wkm.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wwrr.theK`	`8`	Number of alternative paths, k
`wwrr.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wwso.theK`	`8`	Number of alternative paths, k
`wwso.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wkco.theK`	`8`	Number of alternative paths, k
`wkco.firstFibreOnly`	`false`	Use only first fibre on each cable?

Table F.37: Non-default parameters for GA17

Run	Network					
	1		2		3	
	NWR	Trials	NWR	Trials	NWR	Trials
1	14	400,181	11	299,331	10	448,235
2	13	424,437	10	469,338	11	484,945
3	13	351,264	10	401,919	10	467,209
4	14	245,101	10	496,945	11	345,812
5	14	330,278	12	345,997	11	333,983
Best	13	351,264	10	401,919	10	448,235

Run	Network					
	4		5			
	NWR	Trials	NWR	Trials		
1	12	321,042	13	469,673		
2	12	445,261	12	436,962		
3	11	429,485	12	428,962		
4	12	480,841	13	339,015		
5	12	365,958	13	314,122		
Best	11	429,485	12	428,962		

Table F.38: Minimum NWR results for GA16

Run	Network					
	1		2		3	
	NWR	Trials	NWR	Trials	NWR	Trials
1	9	31,874	7	46,952	8	29,420
2	9	35,150	8	27,508	7	44,467
3	9	39,550	7	38,291	8	36,178
4	9	33,726	7	42,283	7	53,688
5	9	34,351	7	39,823	7	62,726
Best	9	31,874	7	38,291	7	44,467
Run	Network					
	4		5			
	NWR	Trials	NWR	Trials		
1	8	48,304	9	34,523		
2	9	29,391	9	36,118		
3	8	37,997	9	33,571		
4	8	40,284	8	48,448		
5	8	38,939	9	34,393		
Best	8	37,997	8	48,448		

Table F.39: Minimum NWR results for GA17

Parameter	Value	Description
`ww95.theK`	8	Number of alternative paths, k
`ww95.sortPaths`	`true`	Allocate the paths in hop count order?
`ww95.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wadi.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wahr.theK`	8	Number of alternative paths, k
`wahr.firstFibreOnly`	`false`	Use only first fibre on each cable?

Table F.40: Non-default parameters for W&0, W&D1 and W&D2 heuristics (minimum NWR)

Run	Network				
	1	2	3	4	5
1	8	7	8	8	9
2	8	7	8	8	9
3	8	8	8	8	9
4	8	8	8	8	9
5	8	7	8	8	9
Best	8	7	8	8	9

Table F.41: Minimum NWR results for W&O

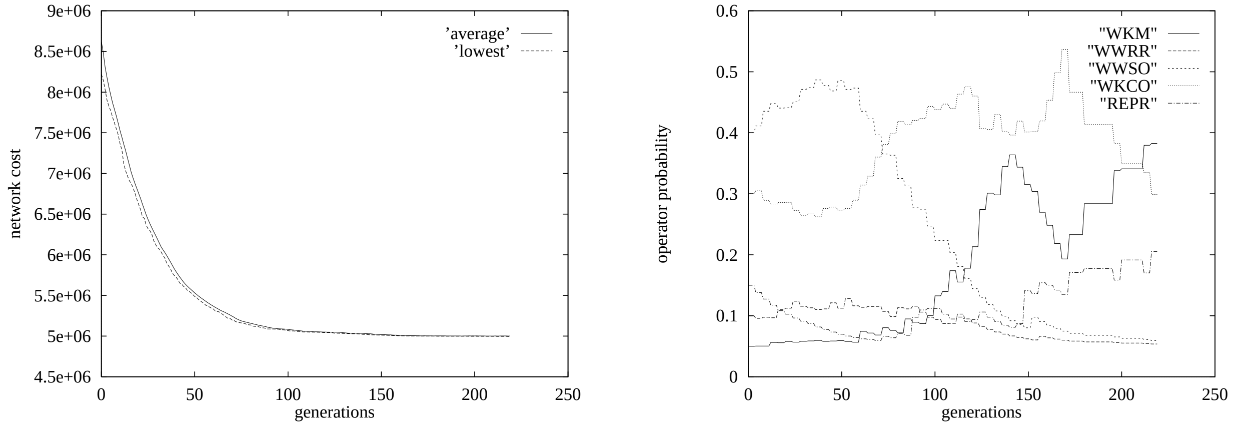
F.4.2 Minimum Cost Allocation

The non-default parameters values common to the three GAs discussed in §5.2.2 on p. 113 are given in Table F.42. Typical results, showing progress and operator-probability adaptation over single runs of GA19 and GA20 on Network 1, for different combinations of (α, β, γ), are illustrated in Figures F.10–F.13. Equivalent traces for GA18 were given in Figures 5.8 and 5.9 on p. 114. The complete results from applying all three GAs to Networks 1–7 are recorded in Tables F.43–F.48. The best of each of these groups of five runs was included in Tables 5.17–5.22 on pp. 115–116.

The non-default parameters values for the three heuristics discussed in §5.2.2 on p. 113 are given in Table F.49. The allocation results from applying one of these, Wuttisittikulkij & O'Mahony's heuristic [298], to Networks 1–7, for six different combinations of the Network Cost Model B parameters (α, β, γ), are listed in Tables F.50–F.55. The best of each of these groups of five runs was again included in Tables 5.17–5.22.

Parameter	Value	Description
`net.metric`	`PNCT`	Metric used for network self-assessment
`net.router`	`WNOP`	Router used to route the network prior to self-assessment
`maxTrials`	`100,000`	Maximum number of trials; must be ≥ 1
`pop.size`	`500`	Population size
`pop.select`	`tournament`	Selection operator
`pop.initOp`	`WRKS`	Population initialisation operator
`pop.phyNetwork`	`true`	Construct the individual using a `PhyNetwork` (`true`) or a `Network` (`false`)
`wrks.theK`	`8 or 4`	Number of alternative paths, k
`wrks.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wkm.theK`	`8 or 4`	Number of alternative paths, k
`wkm.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wwrr.theK`	`8 or 4`	Number of alternative paths, k
`wwrr.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wwso.theK`	`8 or 4`	Number of alternative paths, k
`wwso.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wkco.theK`	`8 or 4`	Number of alternative paths, k
`wkco.firstFibreOnly`	`false`	Use only first fibre on each cable?

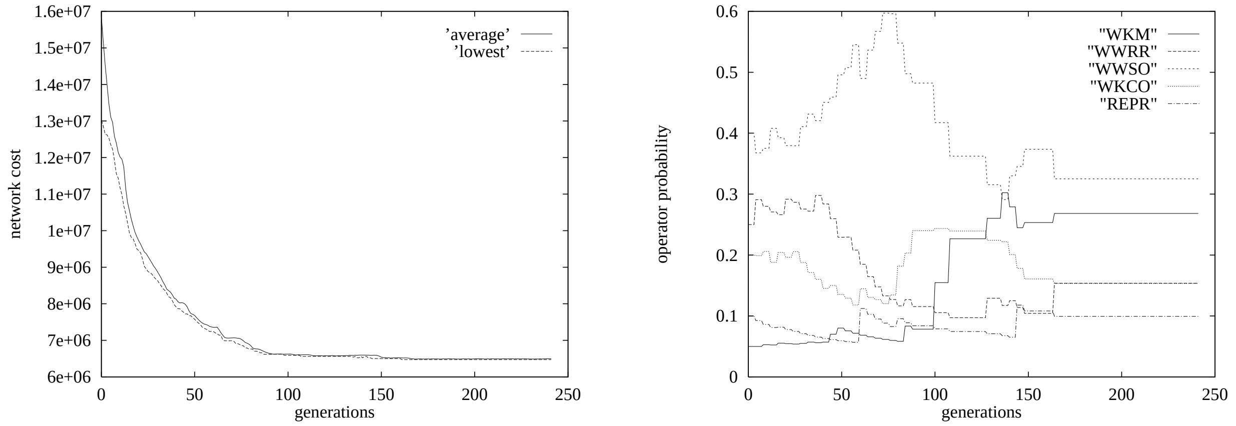
Table F.42: Non-default parameters for GA18–GA20



(a) Progress over a single run

(b) Operator-probability adaptation

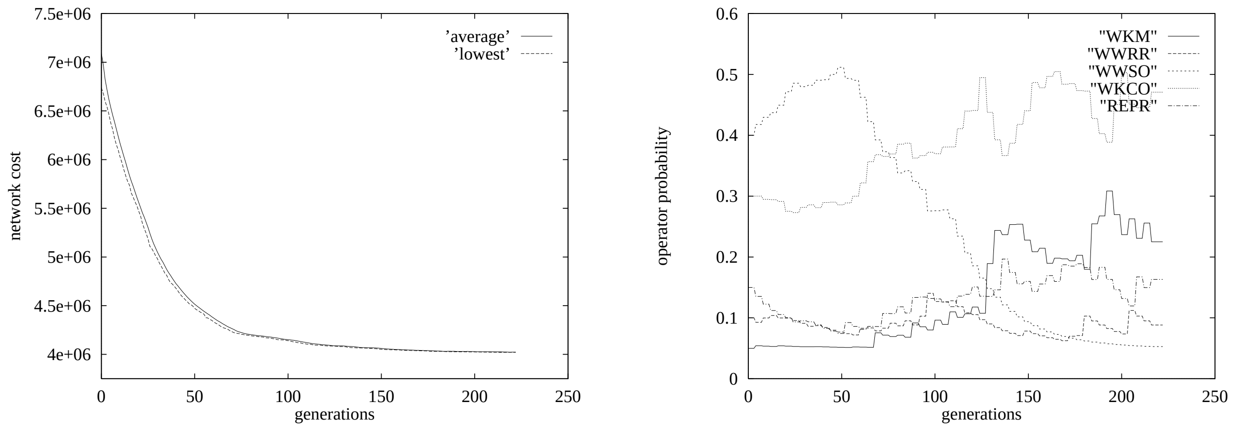
Figure F.10: Typical results for GA19, $\alpha = \beta = 1$ and $\gamma = 1$ (Network 1, Run 1)



(a) Progress over a single run

(b) Operator-probability adaptation

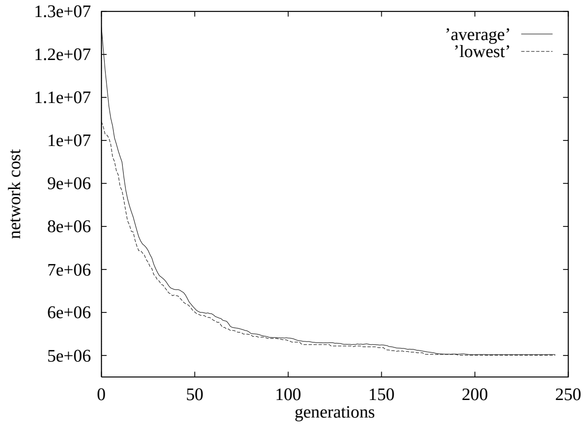
Figure F.11: Typical results for GA20, $\alpha = \beta = 1$ and $\gamma = 0$ (Network 1, Run 1)



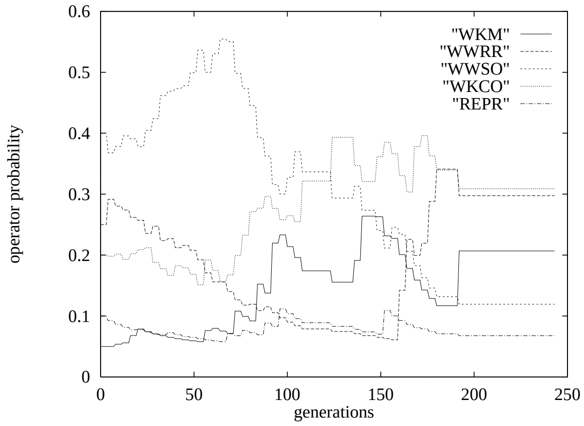
(a) Progress over a single run

(b) Operator-probability adaptation

Figure F.12: Typical results for GA19, $\alpha = \beta = 0.8$ and $\gamma = 1$ (Network 1, Run 1)



(a) Progress over a single run



(b) Operator-probability adaptation

Figure F.13: Typical results for GA20, $\alpha = \beta = 0.8$ and $\gamma = 0$ (Network 1, Run 1)

Run	Network						
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)	6 ($\times 10^6$)	7 ($\times 10^6$)
1	5.579	4.930	5.013	5.117	5.182	5.370	5.751
2	5.694	4.969	5.016	5.201	5.202	5.434	5.656
3	5.732	4.905	4.979	5.188	5.206	5.361	5.757
4	5.679	4.938	4.979	5.189	5.150	5.344	5.681
5	5.629	4.886	4.976	5.248	5.108	5.435	5.651
Best	5.579	4.886	4.976	5.117	5.108	5.344	5.651

Table F.43: Minimum cost allocation results for $\alpha = \beta = 1$ and $\gamma = 0.5$ using GA18

Run	Network						
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)	6 ($\times 10^6$)	7 ($\times 10^6$)
1	4.996	4.436	4.395	4.645	4.641	4.753	5.228
2	4.996	4.437	4.396	4.645	4.640	4.753	5.228
3	4.996	4.436	4.394	4.645	4.640	4.753	5.228
4	4.995	4.438	4.394	4.645	4.640	4.753	5.228
5	4.994	4.438	4.394	4.646	4.641	4.753	5.228
Best	4.994	4.436	4.394	4.645	4.640	4.753	5.228

Table F.44: Minimum cost allocation results for $\alpha = \beta = 1$ and $\gamma = 1$ using GA19

Run	Network						
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)	6 ($\times 10^6$)	7 ($\times 10^6$)
1	6.471	5.338	5.435	5.861	5.561	5.958	6.007
2	6.188	5.417	5.362	5.499	5.684	5.787	6.001
3	6.310	5.454	5.598	5.910	5.858	6.108	6.029
4	6.116	5.428	5.384	5.672	5.523	5.781	6.005
5	6.217	5.228	5.422	5.749	5.714	5.954	6.294
Best	6.116	5.228	5.362	5.499	5.523	5.781	6.001

Table F.45: Minimum cost allocation results for $\alpha = \beta = 1$ and $\gamma = 0$ using GA20

Run	Network						
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)	6 ($\times 10^6$)	7 ($\times 10^6$)
1	4.616	4.011	4.017	4.136	4.123	4.098	4.001
2	4.450	3.941	3.866	4.093	4.117	4.065	3.982
3	4.627	3.909	4.016	4.062	4.040	4.169	3.965
4	4.532	3.909	3.928	4.144	4.151	4.185	4.034
5	4.650	3.933	3.918	4.103	4.076	4.110	4.000
Best	4.450	3.909	3.866	4.062	4.040	4.065	3.965

Table F.46: Minimum cost allocation results for $\alpha = \beta = 0.8$ and $\gamma = 0.5$ using GA18

Run	Network						
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)	6 ($\times 10^6$)	7 ($\times 10^6$)
1	4.020	3.531	3.541	3.718	3.717	3.633	3.686
2	4.014	3.531	3.541	3.717	3.716	3.633	3.686
3	4.015	3.531	3.545	3.717	3.717	3.633	3.686
4	4.016	3.533	3.541	3.732	3.717	3.633	3.686
5	4.018	3.532	3.541	3.716	3.717	3.633	3.686
Best	4.014	3.531	3.541	3.716	3.716	3.633	3.686

Table F.47: Minimum cost allocation results for $\alpha = \beta = 0.8$ and $\gamma = 1$ using GA19

Run	Network						
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)	6 ($\times 10^6$)	7 ($\times 10^6$)
1	5.005	4.518	4.597	4.444	4.625	4.585	4.300
2	4.910	4.335	4.281	4.546	4.581	4.346	4.381
3	5.147	4.292	4.271	4.463	4.363	4.491	4.228
4	4.895	4.249	4.374	4.473	4.474	4.344	4.237
5	4.946	4.265	4.463	4.556	4.524	4.624	4.230
Best	4.895	4.249	4.271	4.444	4.363	4.344	4.228

Table F.48: Minimum cost allocation results for $\alpha = \beta = 0.8$ and $\gamma = 0$ using GA20

Parameter	Value	Description
`ww95.theK`	`8` or `4`	Number of alternative paths, k
`ww95.sortPaths`	`true`	Allocate the paths in hop count order?
`ww95.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wadi.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wahr.theK`	`8` or `4`	Number of alternative paths, k
`wahr.firstFibreOnly`	`false`	Use only first fibre on each cable?

Table F.49: Non-default parameters for W&0, W&D1 and W&D2 heuristics (minimum cost allocation)

Run	Network						
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)	6 ($\times 10^6$)	7 ($\times 10^6$)
1	5.919	5.151	5.185	5.445	5.524	5.732	6.093
2	5.980	5.231	5.172	5.434	5.696	5.692	6.077
3	6.032	5.161	5.213	5.413	5.441	5.811	6.164
4	5.917	5.171	5.153	5.416	5.424	5.662	6.101
5	5.909	5.169	5.154	5.477	5.526	5.761	6.232
Best	5.909	5.151	5.153	5.413	5.424	5.662	6.077

Table F.50: Minimum cost allocation results for $\alpha = \beta = 1$ and $\gamma = 0.5$ using W&O

Run	Network						
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)	6 ($\times 10^6$)	7 ($\times 10^6$)
1	5.790	5.081	5.053	5.367	5.386	5.540	5.966
2	5.845	5.159	5.067	5.320	5.543	5.516	5.907
3	5.897	5.109	5.089	5.313	5.338	5.569	5.960
4	5.704	5.104	5.051	5.330	5.302	5.533	5.968
5	5.782	5.088	5.078	5.365	5.411	5.615	6.025
Best	5.704	5.081	5.051	5.313	5.302	5.516	5.907

Table F.51: Minimum cost allocation results for $\alpha = \beta = 1$ and $\gamma = 1$ using W&O

Run	Network						
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)	6 ($\times 10^6$)	7 ($\times 10^6$)
1	6.048	5.222	5.317	5.523	5.662	5.925	6.220
2	6.115	5.303	5.278	5.548	5.850	5.868	6.247
3	6.167	5.213	5.337	5.513	5.544	6.053	6.368
4	6.131	5.237	5.254	5.503	5.547	5.790	6.234
5	6.036	5.251	5.229	5.589	5.641	5.907	6.439
Best	6.036	5.213	5.229	5.503	5.544	5.790	6.220

Table F.52: Minimum cost allocation results for $\alpha = \beta = 1$ and $\gamma = 0$ using W&O

Run	Network						
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)	6 ($\times 10^6$)	7 ($\times 10^6$)
1	4.667	4.090	4.162	4.328	4.392	4.295	4.215
2	4.715	4.146	4.159	4.317	4.527	4.260	4.203
3	4.755	4.089	4.186	4.312	4.329	4.354	4.263
4	4.662	4.107	4.138	4.312	4.313	4.246	4.217
5	4.653	4.105	4.140	4.350	4.394	4.315	4.306
Best	4.653	4.089	4.138	4.312	4.313	4.246	4.203

Table F.53: Minimum cost allocation results for $\alpha = \beta = 0.8$ and $\gamma = 0.5$ using W&O

Run	Network						
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)	6 ($\times 10^6$)	7 ($\times 10^6$)
1	4.565	4.036	4.058	4.272	4.283	4.148	4.124
2	4.605	4.088	4.072	4.230	4.402	4.123	4.082
3	4.651	4.050	4.090	4.230	4.248	4.170	4.115
4	4.489	4.056	4.056	4.242	4.216	4.149	4.123
5	4.552	4.041	4.081	4.260	4.302	4.203	4.157
Best	4.489	4.036	4.056	4.230	4.216	4.123	4.082

Table F.54: Minimum cost allocation results for $\alpha = \beta = 0.8$ and $\gamma = 1$ using W&O

Run	Network						
	1 ($\times 10^6$)	2 ($\times 10^6$)	3 ($\times 10^6$)	4 ($\times 10^6$)	5 ($\times 10^6$)	6 ($\times 10^6$)	7 ($\times 10^6$)
1	4.768	4.144	4.266	4.384	4.501	4.441	4.306
2	4.824	4.204	4.245	4.404	4.651	4.398	4.323
3	4.859	4.128	4.283	4.395	4.410	4.538	4.412
4	4.836	4.158	4.220	4.382	4.409	4.343	4.311
5	4.755	4.169	4.200	4.439	4.487	4.427	4.455
Best	4.755	4.128	4.200	4.382	4.409	4.343	4.306

Table F.55: Minimum cost allocation results for $\alpha = \beta = 0.8$ and $\gamma = 0$ using W&O

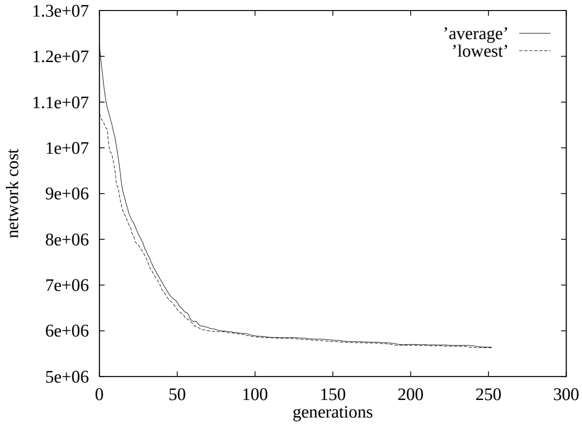
F.5 Operator-probability Adaptation

F.5.1 Basic Adaptation Algorithm

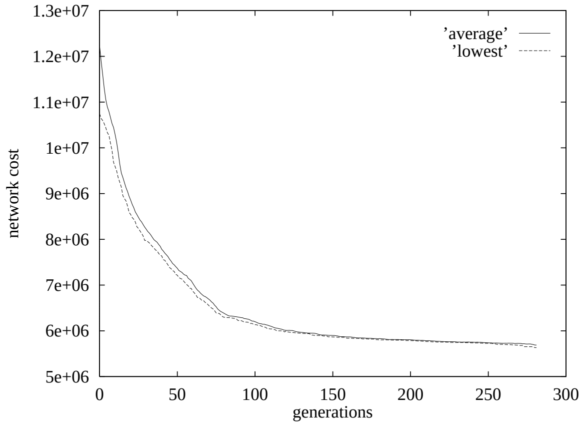
The non-default parameters values common to the five GAs discussed in §5.3.1 on p. 118 are given in Table F.56. Typical results, showing progress over single runs of GA21–GA25, each with a different combination of adaptation parameters (ψ, κ), on Network 1 are illustrated in Fig. F.14. The corresponding operator-probability adaptation curves for GA21, GA23–GA25 are given in Fig. 5.10 on p. 120. Moreover, the complete results from applying all five GAs to Networks 1, 3 & 5 are recorded in Tables F.57–F.61 (these were summarised in Tables 5.24 and 5.25 on p. 121).

Parameter	Value	Description
`net.metric`	`PNCT`	Metric used for network self-assessment
`net.router`	`WNOP`	Router used to route the network prior to self-assessment
`maxTrials`	`100,000`	Maximum number of trials; must be ≥ 1
`pop.size`	`500`	Population size
`pop.select`	`tournament`	Selection operator
`pop.initOp`	`WRKS`	Population initialisation operator
`pop.phyNetwork`	`true`	Construct the individual using a `PhyNetwork` (`true`) or a `Network` (`false`)
`opp.adaptProp`	`0.00`, `0.15` or `0.30`	Adaptation proportion, ψ; $0 \leq \psi \leq 1$
`opp.creditK`	`0.0`, `0.5` or `1.0`	Credit constant, κ; must be ≥ 0
`wrks.theK`	`8`	Number of alternative paths, k
`wrks.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wkm.theK`	`8`	Number of alternative paths k
`wkm.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wwrr.theK`	`8`	Number of alternative paths, k
`wwrr.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wwso.theK`	`8`	Number of alternative paths, k
`wwso.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wkco.theK`	`8`	Number of alternative paths, k
`wkco.firstFibreOnly`	`false`	Use only first fibre on each cable?

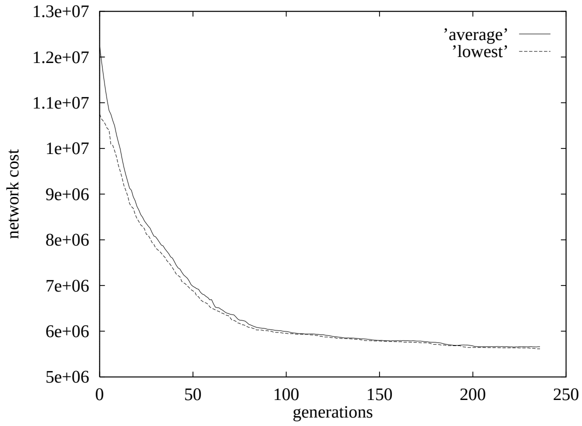
Table F.56: Non-default parameters for GA21–GA25



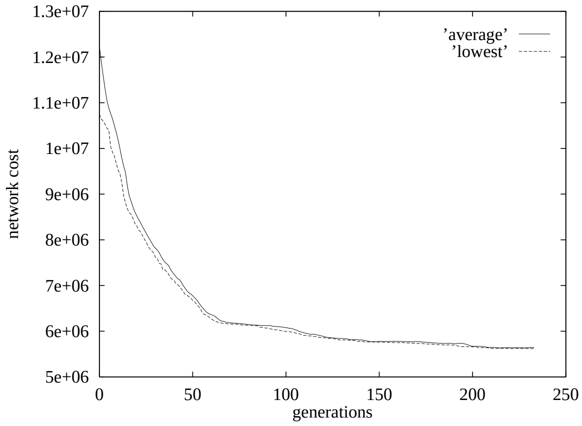
(a) GA21, $(\psi, \kappa) = (0.15, 0.5)$



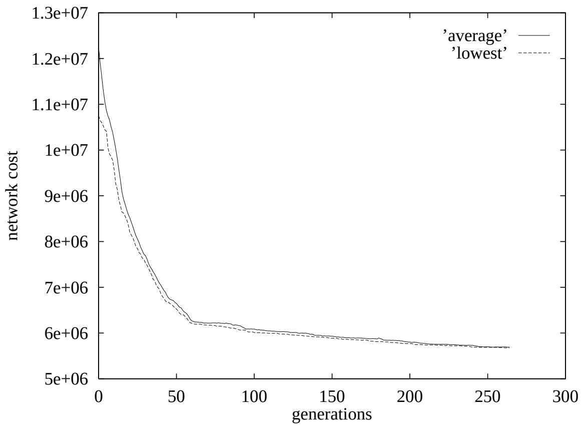
(b) GA22, $(\psi, \kappa) = (0.00, 0.5)$



(c) GA23, $(\psi, \kappa) = (0.30, 0.5)$



(d) GA24, $(\psi, \kappa) = (0.15, 0.0)$



(e) GA25, $(\psi, \kappa) = (0.15, 1.0)$

Figure F.14: Typical progress over single runs of GA21–GA25 (Network 1, Run 1)

Run	Network					
	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)
	25,000 trials			100,000 trials		
1	6.222	5.468	5.954	5.632	4.960	5.126
2	6.582	5.524	6.075	5.614	4.889	5.111
3	6.506	5.838	5.869	5.645	4.911	5.256
4	6.528	5.576	5.913	5.780	4.991	5.091
5	6.442	5.635	6.272	5.717	4.967	5.081
6	6.454	5.618	5.896	5.582	4.879	5.161
7	6.379	5.511	5.979	5.699	4.950	5.199
8	6.663	5.591	6.034	5.680	4.940	5.233
9	6.595	5.684	5.922	5.673	5.028	5.249
10	6.682	5.649	6.074	5.597	4.928	5.101
Best	6.222	5.468	5.869	5.582	4.879	5.081
Median	6.517	5.604	5.967	5.659	4.945	5.144

Table F.57: Operator-probability adaptation results for GA21

Run	Network					
	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)
	25,000 trials			100,000 trials		
1	6.783	5.654	6.147	5.634	4.871	5.225
2	6.780	5.700	6.293	5.736	4.934	5.063
3	6.682	5.850	5.922	5.743	4.861	5.179
4	6.709	5.747	5.953	5.703	4.885	5.127
5	6.543	5.730	6.318	5.718	4.976	5.115
6	6.630	5.778	6.085	5.621	4.924	5.144
7	6.551	5.788	6.030	5.680	4.871	5.238
8	6.754	5.642	6.085	5.682	4.965	5.188
9	6.678	5.692	6.055	5.671	4.910	5.164
10	6.527	5.713	6.042	5.636	4.937	5.194
Best	6.527	5.642	5.922	5.621	4.861	5.063
Median	6.680	5.722	6.070	5.681	4.917	5.171

Table F.58: Operator-probability adaptation results for GA22

Run	Network					
	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)
	25,000 trials			100,000 trials		
1	6.628	5.535	6.115	5.614	4.978	5.276
2	6.306	5.625	6.172	5.609	4.913	5.183
3	6.727	5.763	5.784	5.630	4.908	5.090
4	6.534	5.800	5.972	5.650	5.016	5.108
5	6.484	5.818	6.207	5.717	4.853	5.165
6	6.703	5.820	6.002	5.667	4.981	5.142
7	6.442	5.691	6.074	5.701	4.936	5.175
8	6.700	5.586	6.073	5.735	4.971	5.135
9	6.581	5.824	5.845	5.707	4.867	5.138
10	6.442	5.714	5.984	5.647	4.833	5.090
Best	6.306	5.535	5.784	5.609	4.833	5.090
Median	6.558	5.739	6.038	5.658	4.924	5.140

Table F.59: Operator-probability adaptation results for GA23

Run	Network					
	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)
	25,000 trials			100,000 trials		
1	6.331	5.461	6.029	5.613	4.934	5.209
2	6.529	5.566	6.205	5.811	4.932	5.207
3	6.635	5.743	5.875	5.695	4.936	5.242
4	6.598	5.662	6.157	5.681	4.985	5.140
5	6.379	5.607	6.168	5.694	4.926	5.191
6	6.621	5.725	5.929	5.624	4.905	5.141
7	6.463	5.674	6.023	5.655	4.887	5.138
8	6.511	5.460	5.927	5.648	4.933	5.175
9	6.611	5.583	5.884	5.637	4.921	5.109
10	6.382	5.613	6.109	5.662	4.937	5.079
Best	6.331	5.460	5.875	5.613	4.887	5.079
Median	6.520	5.610	6.026	5.659	4.932	5.158

Table F.60: Operator-probability adaptation results for GA24

Run	Network					
	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)
	25,000 trials			100,000 trials		
1	6.222	5.546	5.944	5.675	4.976	5.113
2	6.364	5.678	6.254	5.641	5.019	5.150
3	6.679	5.748	5.819	5.635	4.887	5.113
4	6.680	5.668	6.144	5.646	4.974	5.150
5	6.359	5.627	6.001	5.548	4.875	5.140
6	6.460	5.605	5.995	5.445	4.890	5.118
7	6.413	5.636	5.950	5.635	4.880	5.153
8	6.620	5.578	6.009	5.656	4.961	5.079
9	6.492	5.714	5.904	5.614	4.895	5.215
10	6.562	5.545	6.184	5.608	5.001	5.229
Best	6.222	5.545	5.819	5.445	4.875	5.079
Median	6.476	5.631	5.998	5.635	4.928	5.145

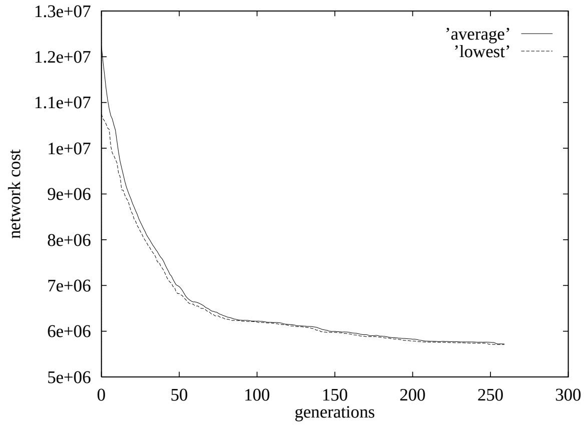
Table F.61: Operator-probability adaptation results for GA25

F.5.2 Variants on the Basic Algorithm

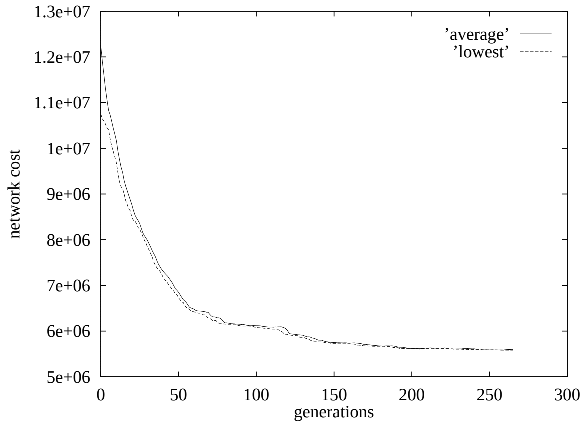
The non-default parameters values common to the eight GAs discussed in §5.3.2 on p. 122 are given in Table F.62. Typical results, showing progress over single runs of GA26-GA29, each with a different combination of adaptation parameters (ψ, κ) and using parental credit assignment, on Network 1 are illustrated in Fig. F.15. The corresponding operator-probability adaptation curves are given in Fig. 5.11 on p. 124. Similarly, for median credit assignment, typical results showing progress over single runs of GA30-GA33 on Network 1 are presented in Fig. F.16. The corresponding operator-probability adaptation curves to these are shown in Fig. 5.12 on p. 127. In addition, the complete results from applying all eight GAs to Networks 1, 3 & 5 are recorded in Tables F.63–F.70 (these were summarised in Tables 5.27–5.30 on pp. 125–127).

Parameter	Value	Description
`net.metric`	`PNCT`	Metric used for network self-assessment
`net.router`	`WNOP`	Router used to route the network prior to self-assessment
`maxTrials`	`100,000`	Maximum number of trials; must be ≥ 1
`pop.size`	`500`	Population size
`pop.select`	`tournament`	Selection operator
`pop.initOp`	`WRKS`	Population initialisation operator
`pop.phyNetwork`	`true`	Construct the individual using a `PhyNetwork` (`true`) or a `Network` (`false`)
`opp.adaptProp`	`0.15` or `0.30`	Adaptation proportion, ψ; $0 \leq \psi \leq 1$
`opp.creditK`	`0.0`, `0.5` or `1.0`	Credit constant, κ; must be ≥ 0
`opp.fitCredType`	`parent` or `median`	Credit type: `best`, `winBest`, `median` or `parent`
`wrks.theK`	`8`	Number of alternative paths, k
`wrks.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wkm.theK`	`8`	Number of alternative paths, k
`wkm.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wwrr.theK`	`8`	Number of alternative paths, k
`wwrr.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wwso.theK`	`8`	Number of alternative paths, k
`wwso.firstFibreOnly`	`false`	Use only first fibre on each cable?
`wkco.theK`	`8`	Number of alternative paths, k
`wkco.firstFibreOnly`	`false`	Use only first fibre on each cable?

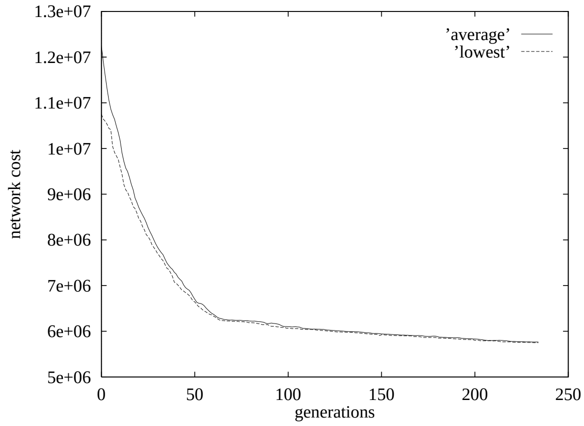
Table F.62: Non-default parameters for GA26–GA33



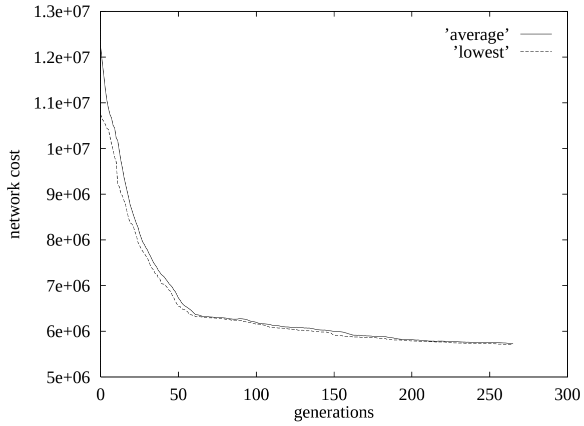
(a) GA26, $(\psi, \kappa) = (0.15, 0.5)$



(b) GA27, $(\psi, \kappa) = (0.30, 0.5)$



(c) GA28, $(\psi, \kappa) = (0.15, 0.0)$

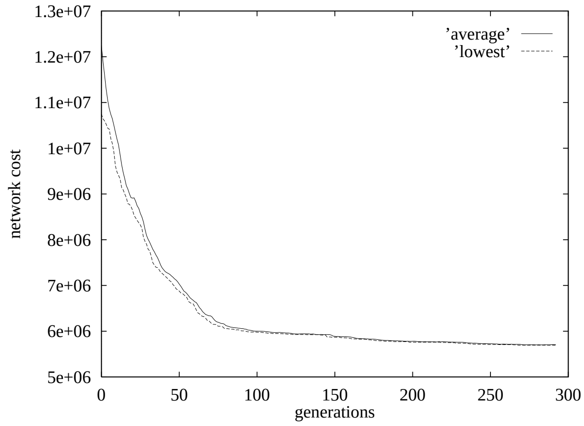


(d) GA29, $(\psi, \kappa) = (0.15, 1.0)$

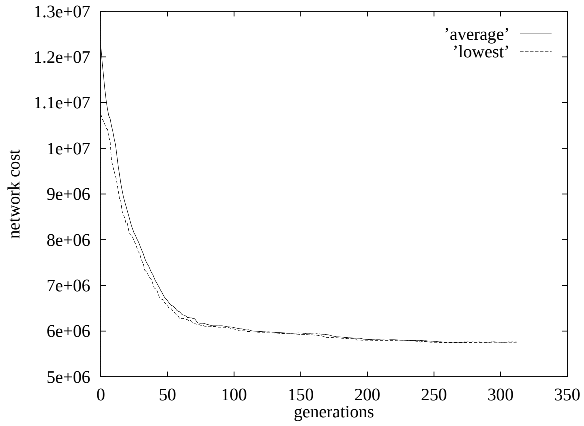
Figure F.15: Typical progress over single runs of GA26–GA29 (Network 1, Run 1)

Run	Network					
	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)
	25,000 trials			100,000 trials		
1	6.557	5.523	5.990	5.710	4.968	5.112
2	6.461	5.639	6.035	5.642	4.875	5.124
3	6.579	5.838	5.806	5.731	4.958	5.137
4	6.684	5.613	5.982	5.584	4.904	5.095
5	6.371	5.605	5.999	5.636	4.892	5.164
6	6.652	5.651	6.073	5.678	4.878	5.189
7	6.526	5.571	6.024	5.638	4.981	5.165
8	6.439	5.527	5.823	5.704	4.874	5.198
9	6.510	5.599	5.707	5.771	4.933	5.173
10	6.545	5.553	5.895	5.655	4.911	5.137
Best	6.371	5.523	5.707	5.584	4.874	5.095
Median	6.536	5.602	5.986	5.666	4.907	5.150

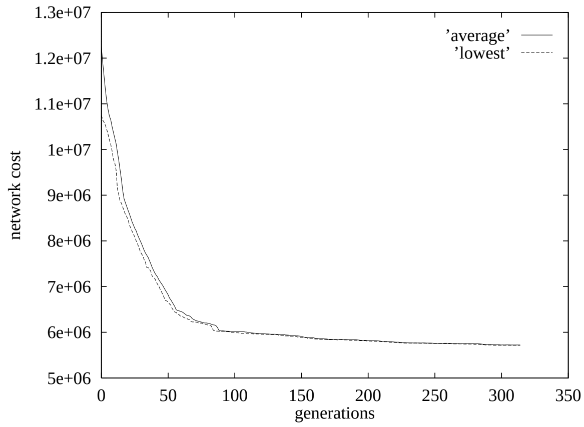
Table F.63: Operator-probability adaptation results for GA26



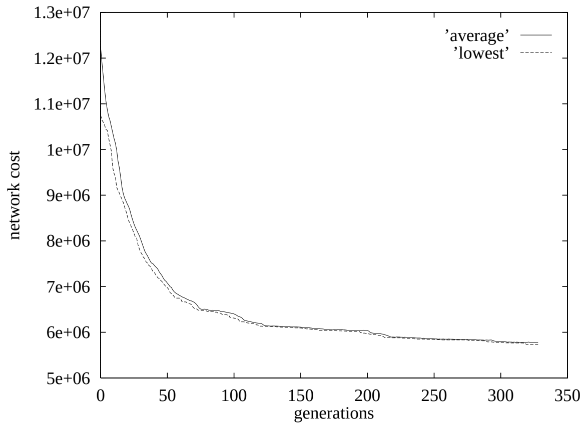
(a) GA30, $(\psi, \kappa) = (0.15, 0.5)$



(b) GA31, $(\psi, \kappa) = (0.30, 0.5)$



(c) GA32, $(\psi, \kappa) = (0.15, 0.0)$



(d) GA33, $(\psi, \kappa) = (0.15, 1.0)$

Figure F.16: Typical progress over single runs of GA30–GA33 (Network 1, Run 1)

Run	Network					
	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)
	25,000 trials			100,000 trials		
1	6.427	5.470	5.932	5.583	5.051	5.183
2	6.521	5.622	5.944	5.697	5.006	5.211
3	6.509	5.801	5.751	5.668	4.941	5.143
4	6.499	5.545	5.841	5.647	4.894	5.204
5	6.434	5.686	6.109	5.642	5.000	5.167
6	6.402	5.655	6.065	5.602	4.915	5.166
7	6.450	5.587	6.107	5.644	4.923	5.161
8	6.550	5.515	5.952	5.668	4.869	5.138
9	6.499	5.613	5.955	5.786	4.840	5.155
10	6.420	5.615	5.862	5.576	4.950	5.180
Best	6.402	5.470	5.751	5.576	4.840	5.138
Median	6.474	5.614	5.948	5.646	4.932	5.166

Table F.64: Operator-probability adaptation results for GA27

Run	Network					
	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)
	25,000 trials			100,000 trials		
1	6.332	5.541	5.969	5.750	4.941	5.180
2	6.468	5.586	6.069	5.628	4.941	5.095
3	6.564	5.887	5.760	5.815	4.961	5.139
4	6.519	5.798	5.964	5.803	4.997	5.118
5	6.378	5.601	6.234	5.563	4.981	5.171
6	6.462	5.738	6.024	5.616	4.912	5.095
7	6.370	5.665	5.909	5.589	4.919	5.091
8	6.529	5.551	5.949	5.608	4.958	5.248
9	6.542	5.640	6.004	5.698	4.884	5.171
10	6.515	5.536	5.969	5.608	4.896	5.129
Best	6.332	5.536	5.760	5.563	4.884	5.091
Median	6.491	5.621	5.969	5.622	4.941	5.134

Table F.65: Operator-probability adaptation results for GA28

Run	Network					
	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)
	25,000 trials			100,000 trials		
1	6.312	5.589	6.017	5.713	4.930	5.229
2	6.506	5.539	6.154	5.626	5.001	5.213
3	6.667	5.883	5.765	5.592	4.939	5.206
4	6.657	5.666	6.049	5.568	4.971	5.159
5	6.219	5.490	5.967	5.637	4.965	5.168
6	6.473	5.727	6.010	5.676	4.937	5.148
7	6.409	5.645	5.962	5.662	4.983	5.123
8	6.653	5.563	6.050	5.603	4.893	5.100
9	6.452	5.565	5.884	5.744	5.035	5.042
10	6.502	5.530	6.039	5.630	4.994	5.266
Best	6.219	5.490	5.765	5.568	4.893	5.042
Median	6.487	5.577	6.013	5.634	4.968	5.164

Table F.66: Operator-probability adaptation results for GA29

Run	Network					
	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)
	25,000 trials			100,000 trials		
1	6.400	5.520	5.943	5.693	4.956	5.255
2	6.396	5.500	6.092	5.760	4.976	5.319
3	6.420	5.774	5.900	5.719	4.919	5.169
4	6.576	5.690	5.970	5.834	4.966	5.134
5	6.463	5.658	5.949	5.721	4.973	5.113
6	6.491	5.628	5.969	5.733	4.946	5.253
7	6.401	5.580	6.103	5.668	4.987	5.126
8	6.567	5.589	5.877	5.678	5.136	5.153
9	6.508	5.395	5.896	5.694	5.002	5.147
10	6.396	5.590	5.871	5.595	0.000	5.199
Best	6.396	5.395	5.871	5.595	4.919	5.113
Median	6.442	5.590	5.946	5.706	4.971	5.161

Table F.67: Operator-probability adaptation results for GA30

Run	Network					
	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)	1 ($\times 10^6$)	3 ($\times 10^6$)	5 ($\times 10^6$)
	25,000 trials			100,000 trials		
1	6.249	5.420	5.916	5.745	4.935	5.153
2	6.381	5.441	6.108	5.720	5.035	5.254
3	6.626	5.643	5.733	5.700	4.959	5.135
4	6.517	5.743	5.895	5.705	4.899	5.264
5	6.437	5.545	5.924	5.780	4.982	5.168
6	6.427	5.659	5.913	5.678	4.900	5.346
7	6.530	5.594	5.833	5.657	4.906	5.190
8	6.423	5.666	6.138	5.818	4.982	5.194
9	6.409	5.524	5.788	5.778	4.890	5.115
10	6.390	5.523	5.892	5.637	4.914	5.107
Best	6.249	5.420	5.733	5.637	4.890	5.107
Median	6.425	5.569	5.904	5.712	4.925	5.179

Table F.68: Operator-probability adaptation results for GA31

Run	Network					
	1 $(\times 10^6)$	3 $(\times 10^6)$	5 $(\times 10^6)$	1 $(\times 10^6)$	3 $(\times 10^6)$	5 $(\times 10^6)$
	25,000 trials			100,000 trials		
1	6.305	5.450	5.971	5.716	4.913	5.183
2	6.449	5.615	6.079	5.698	4.904	5.203
3	6.393	5.687	5.791	5.701	4.972	5.126
4	6.601	5.833	5.928	5.752	5.086	5.126
5	6.443	5.691	6.147	5.720	5.047	5.136
6	6.571	5.594	5.939	5.674	4.918	5.242
7	6.370	5.616	6.052	5.660	4.975	5.099
8	6.502	5.478	5.892	5.652	4.890	5.133
9	6.611	5.598	5.823	5.623	4.952	5.159
10	6.392	5.496	5.886	5.657	4.960	5.119
Best	6.305	5.450	5.791	5.623	4.890	5.099
Median	6.446	5.607	5.933	5.686	4.956	5.134

Table F.69: Operator-probability adaptation results for GA32

Run	Network					
	1 $(\times 10^6)$	3 $(\times 10^6)$	5 $(\times 10^6)$	1 $(\times 10^6)$	3 $(\times 10^6)$	5 $(\times 10^6)$
	25,000 trials			100,000 trials		
1	6.664	5.475	5.895	5.737	4.882	5.174
2	6.455	5.608	6.243	5.670	5.026	5.258
3	6.431	5.770	5.827	5.578	4.840	5.148
4	6.550	5.789	5.923	5.649	5.018	5.177
5	6.578	5.523	5.980	5.682	4.942	5.075
6	6.488	5.579	5.946	5.668	4.931	5.200
7	6.401	5.536	6.085	5.632	4.999	5.205
8	6.683	5.531	6.034	5.680	4.982	5.180
9	6.511	5.607	5.940	5.547	5.007	5.086
10	6.523	5.571	5.884	5.740	4.935	5.118
Best	6.401	5.475	5.827	5.547	4.840	5.075
Median	6.517	5.575	5.943	5.669	4.962	5.175

Table F.70: Operator-probability adaptation results for GA33