NOMaD: Initial architecture of an optical network optimisation, modelling and design tool

M. C. Sinclair*

1 Introduction

This paper describes the initial architecture of a tool for optical network modelling, optimisation and design (NOMaD) being developed by the author at the University of Essex. NOMaD will be used as part of the author's own research into the application of genetic-algorithm/heuristic hybrid optimisation techniques to network design, as well as in several research projects, including two Advanced Communications Technologies and Services (ACTS) projects [1]: WOTAN (Wavelength-agile Optical Transport and Access Network) and OPEN (Optical Pan-European Network). It is envisaged that the overall NOMaD architecture developed by the author will be extended by others to meet the requirements of their individual projects.

After introducing the application areas for the NOMaD tool, its overall architecture will be presented, with additional detail then given on the object-oriented network toolset, genetic-algorithm/heuristic hybrid optimisation, heuristic network design, the tcl scripting layer and the graphical user interface (GUI).

2 Application Areas

The design problems that NOMaD will seek to address will, at least initially, be those presented by multi-wavelength all-optical networks, including combined access/core networks at national level and transport networks

*Dept. of Electronic Systems Engineering, University of Essex, Wivenhoe Park, Colchester, Essex CO4 3SQ. Tel: 01206-872477; Fax: 01206-872900; Email: mcs@essex.ac.uk

at the international, as well as consideration of both ‘flat’ and hierarchical approaches. The two key application areas are:

- Topological design, starting by making the choice of links in both the virtual and physical topologies (the virtual topology ignores the underlying physical reality of which ducts and fibres will actually be used), and then building on that to network extensions (additional node placement), and perhaps eventually node placement for an entire network.

- Routing and wavelength allocation, initially taking account of only the virtual topology, but then recognising the need to include duct and fibre placement.

To begin with, NOMaD will model networks with static traffic demands, but subsequently it is hoped to incorporate dynamic traffic into the network model. Further, although at the start the focus will be on cost optimisation (whilst meeting certain performance and reliability constraints), additional performance metrics will be added later. It is also envisaged that NOMaD will allow the interactive study of network failure scenarios as part of the overall assessment of a design.

3 Overall Architecture

NOMaD is being developed as an overall architecture that will be extended by several people in a variety of directions to serve the needs of their individual projects. Consequently, its users will not just be given a single unchangeable piece of software, but rather a framework which they can actively add to for themselves. As a result, NOMaD will have to be easy to understand, flexible and extensible. The overall layered architecture of NOMaD is shown in Fig. 1.

To aid both understanding and extensibility, the network optimisation, modelling and design toolset is being developed using object technology. The analysis and design is being documented using Booch Notation [2], and the implementation done in the C++ programming language. This process is aided by the use of a CASE tool, Rational Rose/C++ [3], which generates skeletal C++ code directly from the diagrams and specifications created by the author.

To provide flexibility, rather than only providing a C++ interface for users of the NOMaD network toolset, a NOMaD-specific tcl extension is also being developed. Tcl [4, 5] is a scripting language designed to be extended by the incorporation of application-specific commands, coded in C or C++.

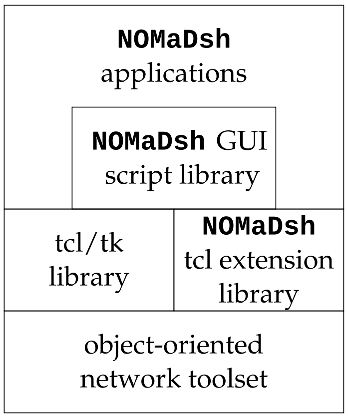

Figure 1: *Overall NOMaD architecture*

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. This latter option enables new high-level algorithms to be developed with much faster edit-compile-debug cycle times, because no compile time is needed—the scripts can be immediately run by the `NOMaDsh` interpreter. Only if the user needs to extend the underlying C++ network toolset does the `NOMaDsh` itself need to be recompiled. These benefits are particularly important in a multi-person research environment, where the rapid prototyping of ideas is of paramount importance.

Finally, a library of GUI scripts is also being developed, using the tk toolkit (tcl/tk), that will allow `NOMaDsh` scripts to edit and display networks.

4 Object-Oriented Network Toolset

Even at this early stage in the development of NOMaD, the object-oriented network toolset is too large to present in detail here, however the overall approach adopted in developing a class library for optical network optimisation, modelling and design can be illustrated by looking at a few aspects of the current class hierarchy.

Fig. 2 is the top-level NOMaD class diagram, illustrating the partitioning of the library into five class categories: `NOMaD`, `NetworkModel`, `GA`, `Operators` and `Utilities`. As yet, the network toolset is at an early stage of development, and the focus has been on simple heuristics for the virtual-topology design of both flat and hierarchical networks. This has been accomplished mainly through the development of the `NetworkModel` and `Operators` class categories.

The main `NetworkModel` class diagram is given in Fig. 3, which shows the `Network` class, composed not only of `Node`s and `Link`s, but also several implementation classes used to represent the `Network`'s adjacency matrix, connection matrix (which records `Link` levels in hierarchical networks), paths and their 'lengths', as well as the traffic requirements. It is envisaged that the network model will in future incorporate classes to represent a network's physical topology (*e.g.* `Duct` and `Fibre`), as well as 'true' classes for paths and ordered sequences of paths.

Fig. 4 is the main `Operators` class diagram, illustrating the abstract `Operator` class, its two abstract subclasses, `UnaryOp` and `BinaryOp`, as well as the latter two classes' use of the `Network` class. Further details of the descendants of `UnaryOp` are given in Fig. 5, which shows some eight subclasses of `UnaryOp` that can be applied to `Network`s. `RandomLinks`, for example, randomly determines whether there is a direct `Link` between each `Node` pair in a `Network`, whereas `RandomAddLinks` iteratively and exhaustively adds a `Link` between random `Node` pairs in a `Network` until no further improvement can be made.

With this approach, rather than hard-coding in only a limited number of simple heuristics for topology design, the class hierarchy that has been adopted should ensure that by inheritance from the `UnaryOp` class, new heuristics can be added in a straightforward fashion. Indeed, as the class library is further developed, it is envisaged that suitable points in the inheritance lattice will be identified for users to supply their own heuristics for virtual and physical topology design, routing and wavelength allocation, as well as classes for cost and performance assessment.

5 GA/Heuristic Hybrid Optimisation

The simple or 'traditional' genetic algorithm (GA) is based on a simplification of the mechanisms of natural selection and genetics [6].

Instead of searching directly through the desired problem space for an optimum, GAs decouple the problem into genotypes, analogous to the chromosomes inherited by an individual, and phenotypes, analogous to the organism that develops from those chromosomes. The only link between them in the algorithm is that any given genotype can be decoded into a phenotype and its raw fitness assessed; this is analogous to the organism's ability to survive and reproduce in the environment. In network design, for example, the genotypes could be binary strings, and the phenotypes the networks decoded from them, with their raw fitness assessed in terms of some overall objective

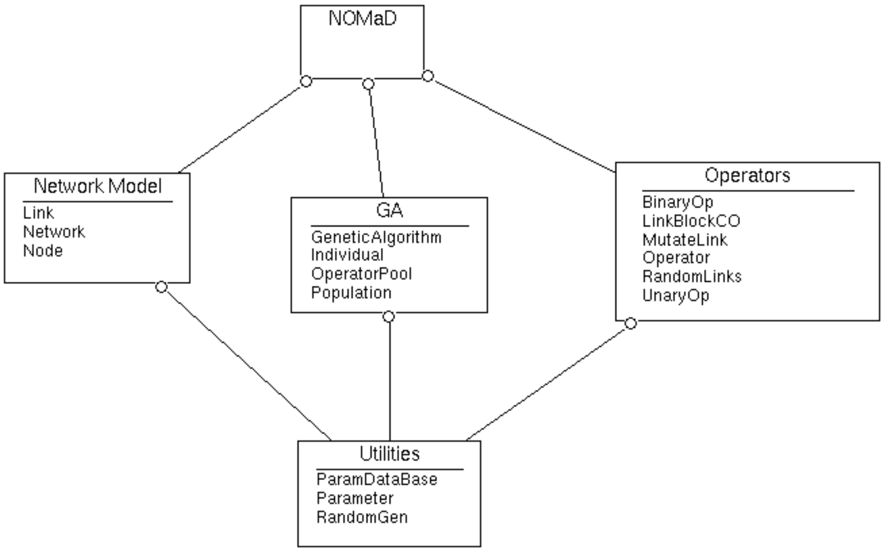

Figure 2: *NOMaD class diagram*

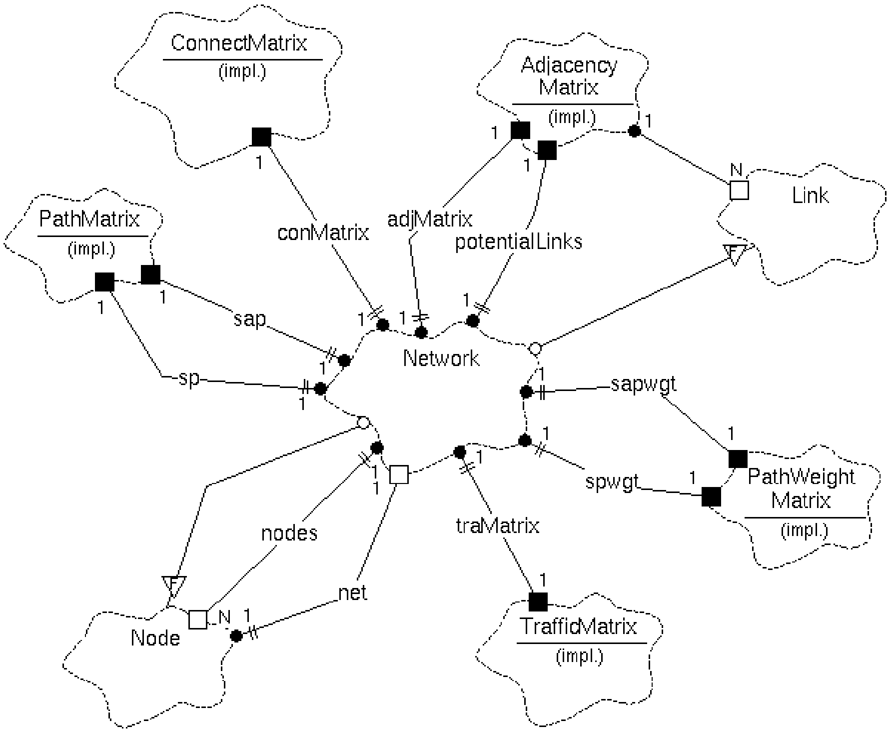

Figure 3: `NetworkModel` *class diagram*

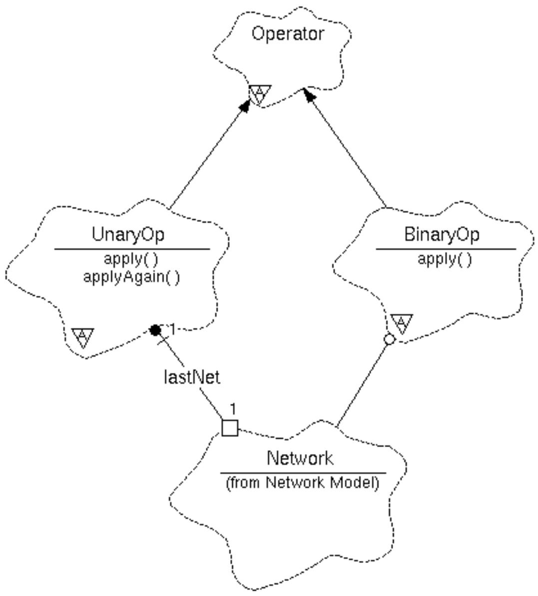

Figure 4: `Operators` *class diagram*

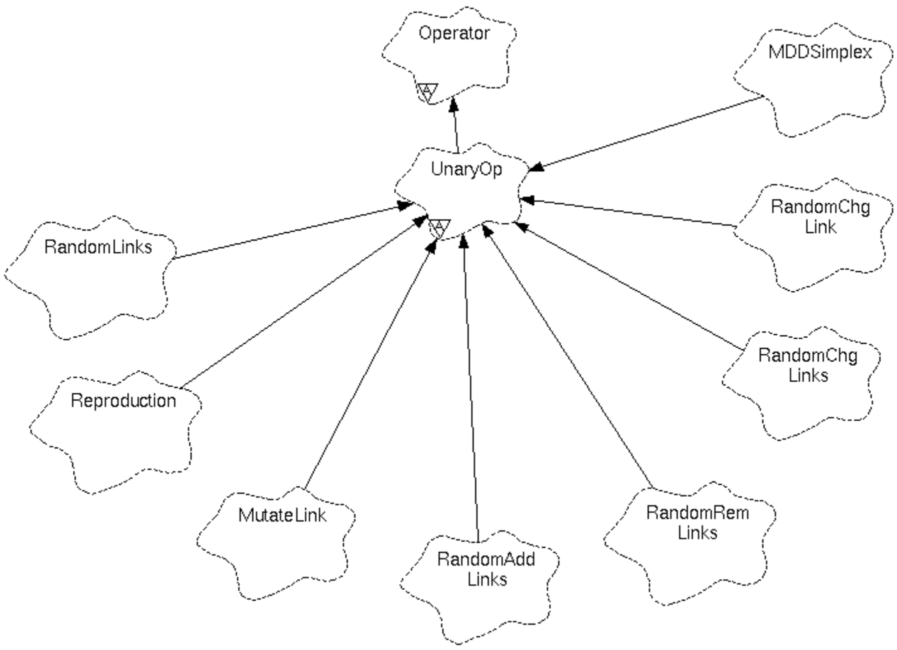

Figure 5: `UnaryOp` *class diagram*

function.

The simple GA searches by starting with a randomly generated population of genotypes, each of which has its raw fitness assessed, and then randomly selecting pairs of genotypes, with their probabilities of selection determined by their fitness. Each such pair is mated to produce two child genotypes, which are modified from their parents using two mechanisms: crossover and mutation. Rather than being direct copies of their parents, in any given mating, with a given probability, there will be an exchange of genetic material between the genotypes passed down by the parents to the two children. In this case, one of the children will consist of a copy of the start of one parent genotype (with the break at a random point), and the end of the other, and vice versa for the other child: crossover has occurred. The other mechanism is mutation: with a given (but usually very low) probability any of the bits in the child genotypes may be altered. The selection and mating of pairs of genotypes from the original population continues until a new population of identical size has been generated; the old population is then discarded, and replaced by the new one. The process continues until a given stopping condition, such as a maximum number of such generations, is reached. In addition, the genotype with the highest fitness found during the run will be recorded.

The mechanisms of selection and crossover ensure that short highly-fit substrings are searched out much faster than might be supposed. This is as a result of implicit parallelism: all the substrings at different positions within the genotypes are competing simultaneously. Mutation ensures that new substrings, not present in the original population, are introduced into the competition as the algorithm proceeds.

However, within the field of genetic algorithm research, there are two 'schools of thought' with respect to GAs. One approach [6] is to try and develop an algorithm that is robust and general in its application *i.e.* it makes use of no problem-specific information—the focus of the research is on the development of a better GA for all problems. The other [7] is to blend the traditional GA with problem-specific operators, often based on existing heuristic algorithms for a particular problem (or problem domain), such that the combined algorithm performs better than either a 'pure' GA or the heuristic alone. This approach is more pragmatic in nature, with the better solution of particular engineering problems being the primary focus of research.

The GA/heuristic hybrid approach thus adopts a problem-specific encoding, and then develops and employs problem-specific operators that together combine the best existing heuristics for the problem with an overall GA framework. The motivation for this approach is simple, according to Davis:

"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" [7]. It is almost always possible to develop a problem-specific heuristic/GA hybrid that will outperform either the heuristic or a traditional GA alone, and this is one of the approaches that will be used in NOMaD. Indeed, current work on the class library is focusing on this area, with several classes under development in the `GA` class category (*e.g.* `GeneticAlgorithm`, `OperatorPool`, `Population` and `Individual`), as can be seen in Fig. 2.

For a brief survey of work on the application of genetic algorithms to network design, see [8].

6 Heuristic Network Design

In support of its GA/heuristic hybrid optimisation capabilities, NOMaD will need to include a variety of simple heuristics, and early work on some of these has been detailed elsewhere [9]. By way of contrast, though, it is also envisaged that NOMaD will eventually be extended by others to incorporate several advanced heuristics—each a complete set of network design rules—in support of individual research projects. Early work on advanced network design heuristics for ACTS WOTAN can be found in [8], and for ACTS OPEN in [10].

7 `NOMaDsh`: The Tcl Scripting Layer

Tcl [4, 5] 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 [4, 5], 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 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 ifs("central.net"); Network net(ifs);
```

could have been created in `NOMaDsh` with the command:

```
set net [nomad_network file "central.net"]
```

The `Network` can then be queried for the number of nodes it contains in C++:

```
int numNodes = net.getNumNodes();
```

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

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

This semantic similarity (though partially masked by syntax!), will hopefully 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 should be swiftly realised[1].

8 Graphical User Interface

The final element of the NOMaD architecture is the GUI. As well as developing a default user interface that will allow access to all of the core elements of the architecture (and which will mainly serve the author's own work), a library of GUI scripts will also be built (and used to create the default interface), making the development of interfaces by other users for their own projects straightforward. In particular, this will include widgets for both displaying and editing networks, as networks are much more readily understood in visual form than by the equivalent textual representation.

A prototype `NOMaDsh` application for iterative stochastic hill climbing (ISHC) has been developed, and a short run of 500 trials on a 9-node network is illustrated in Fig. 6. Similarly, Fig. 7 shows the NOMaD Prototype Network Editor in action. Both of these simple applications were written using short scripts interpreted by `NOMaDsh`, although clearly it is anticipated that far more complex and sophisticated applications will be written using `NOMaDsh` as its development proceeds.

9 Conclusions

The overall architecture of the NOMaD tool for optical network modelling, optimisation and design has been presented. Although the tool is still being developed, early results are encouraging, and it is envisaged that NOMaD will play a central role in both network design and genetic-algorithm/heuristic hybrid research at the University of Essex.

[1]The interface for `Network` creation is currently being changed to employ a `ParamDataBase` class that will allow random access to the (variable, value) pairs that specify a `Network`. For simplicity though, the earlier `iostream` interface is presented here.

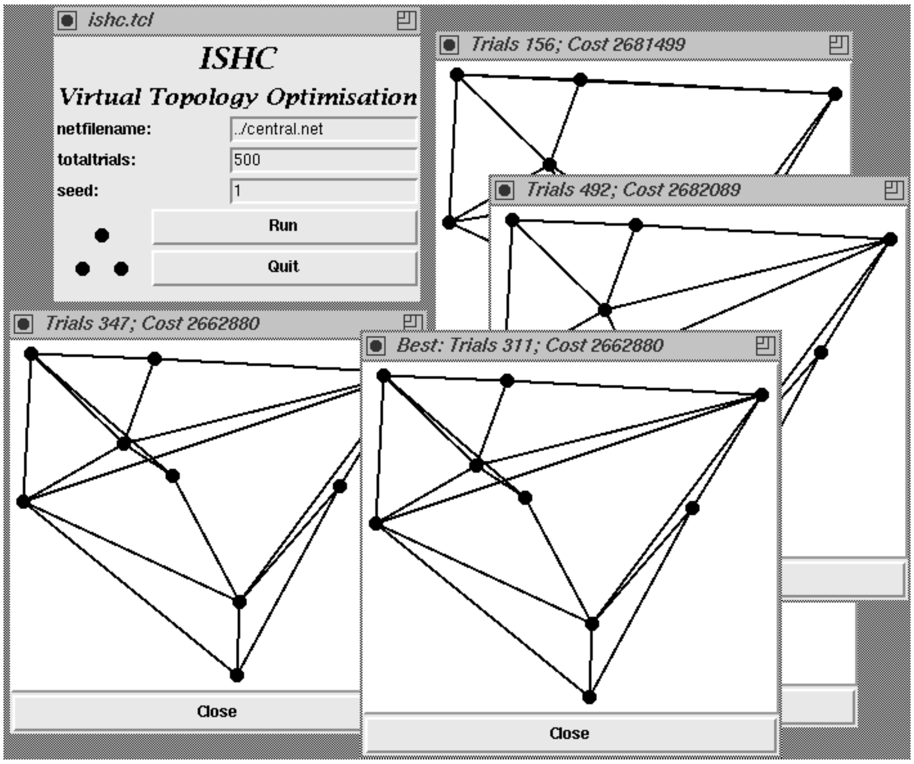

Figure 6: *Prototype ISHC* `NOMaDsh` *application*

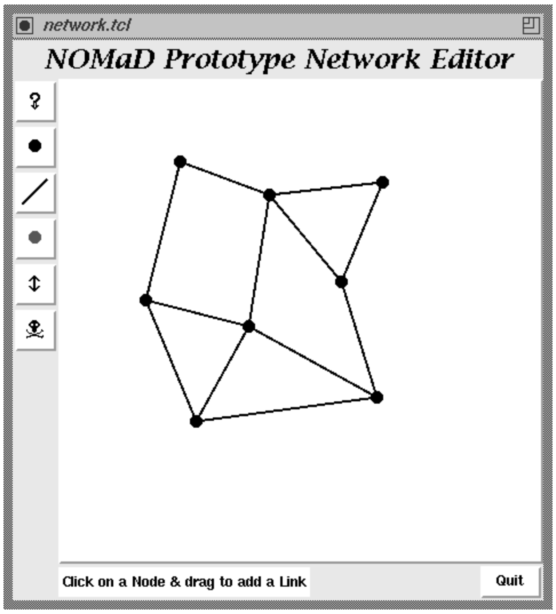

Figure 7: *Prototype network editor* `NOMaDsh` *application*

Acknowledgements

The author is grateful to Katerina Proestaki and Noel Parnis (also of E.S.E., University of Essex) for the many interesting and stimulating discussions during the continuing development of NOMaD. They are both supported by the European Commission under the ACTS programme: Ms. Proestaki on project AC029 (WOTAN) and Mr. Parnis on project AC066 (OPEN).

References

[1] Hill, A.M. & Houghton, A.J.N.: Optical networking in the European ACTS programme *OFC'96* San Jose, USA, 1996

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

[3] Rational Software Corporation: *Using Rational Rose/C++ (Revision 2.7)* 1995

[4] Ousterhout, J,K.: *Tcl and the Tk Toolkit* Addison-Wesley, 1994

[5] Welch, B.B.: *Practical Programming in Tcl and Tk* Prentice-Hall, 1995

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

[7] Davis, L.: *Handbook of Genetic Algorithms* Van Nostrand Reinhold, 1991

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

[9] Griffith, P.S., Proestaki, A. & Sinclair, M.C.: Heuristic topological design of low-cost optical telecommunication networks *Proc. 12th UK Performance Engineering Workshop* Edinburgh University, 1996

[10] 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 University, 1996