Metaheuristic Optimization with Evolver, Genocop and OptQuest

MANUEL LAGUNA

Graduate School of Business Administration
University of Colorado, Boulder, CO 80309-0419
Manuel.Laguna@Colorado.EDU

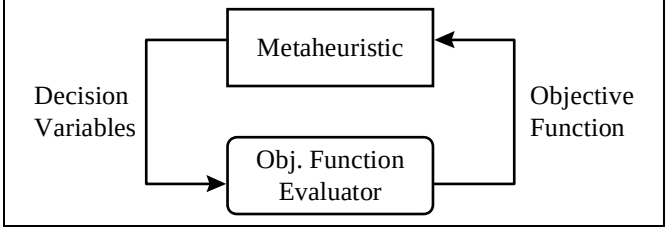
Last revision: April 29, 1997.

ABSTRACT — Metaheuristic optimization has experienced an evolution toward the development of general purpose optimizers. Although the most effective metaheuristics for the solution of hard (in the computational sense) optimization problems are procedures customized to each particular situation, there is an increased interest in developing systems that can effectively operate as general-purpose solvers. In this tutorial paper, we describe and compare the functionality of three general-purpose optimizers that were developed using metaheuristic frameworks. Evolver is a commercial genetic-algorithm software that in its most simple form operates as an add-in function to Microsoft Excel. While Genocop is an experimental genetic algorithm implementation, OptQuest, based on the scatter search methodology, has been commercially implemented to add optimization capabilities to simulation software.

1. Introduction

The purpose of this paper is to describe three metaheuristic systems and compare their features. These systems can be used as general-purpose optimizers because they do not assume any knowledge about the problem to be optimized. We will start by describing each system, concentrating on their functionality rather than on their algorithmic composition. We will then compare these systems using constrained nonlinear optimization problem. The basic model that this systems use to operate is one that treats the objective function evaluator as a black box (see Figure 1).

Fig. 1 Black box model.



The black box model in Figure 1 illustrates that the metaheuristic searches for the optimal values for decision variables without the knowledge of how the objective function value is calculated. The objective function value along with other strategies, however, are used to drive the search for improved solutions. As it will become apparent, the three systems described in this paper use not only different search strategies but also different ways of both communicating with the objective function evaluator and extracting information from the returned values.

2. Evolver

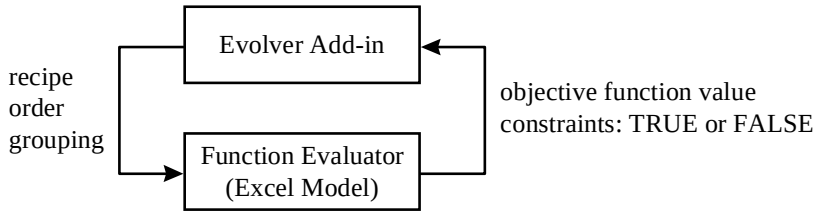
Evolver (a trademark of Axcelis, Inc.) is an genetic algorithm (GA) implementation that is commercially available to run under the Windows operating system. Although it is possible to create customized systems based on the Evolver function library, in this tutorial paper we will concentrate on the Microsoft Excel interface used by the Evolver Add-in. When used as an add-in tool, the system assumes that the user has constructed a model using an Excel worksheet and related functions. Suppose, for example, that we desire to solve the following knapsack problem using Evolver:

$$\begin{aligned} \text{Maximize} \quad & 5x_1 + 9x_2 + 12x_3 + 4x_4 + 3x_5 + 9x_6 \\ \text{subject to} \quad & \\ & 8x_1 + 7x_2 + 16x_3 + 3x_4 + 3x_5 + 11x_6 \leq 25 \\ & x_i \geq 0 \text{ for i} = 1, \ldots, 6. \end{aligned}$$

The first step in the solution process would be to create a simple worksheet and enter the objective function coefficients as well as the constraint coefficients. Then, six cells must be chosen to contain the values of the decision variables. Finally, formulas that calculate the objective function and the left-hand-side value of the constraint must be entered in two separate cells. At this point, the problem has been modeled and can be described in Evolver. The user-interface of Evolver is similar to the user-interface for the Solver tool in Excel. That is, the user must indicate the cell where the objective function is calculated and the direction of the optimization (maximize or minimize). Then, the decision variables cells (or "adjusting cells") are specified. There are several options associated with Evolver's treatment of the

adjusting cells, which are referred to as "solving methods." In the context of our simple example, the appropriate solving method is the so-called "recipe." This method assigns values to the adjusting cells independently, within a specified range. The "order" method, on the other hand, is appropriate in contexts where solutions are best represented as a permutation of numbers (e.g., the traveling salesman problem). There is also the "grouping" method, for problems that involve variables to be assigned to sets (e.g., a clustering problem). In this method, the number of different groups that the system creates is equal to the number of unique values present in the adjusting cells at the beginning of the search. As indicated in Figure 2, the solving method is specified in the Evolver interface and chosen values for each variable type are temporarily placed in the corresponding cells of the Excel model in order to evaluate the objective function. A problem can be formulated using more than one solving method. Each solving method has a specialized version. For example, the "budget" method is a specialized version of the "recipe" method in which the sum of the variable values must be equal to a certain number. That number is the total calculated at the time the search begins (as given by the initial values provided by the user). Similarly, the "project" method is a specialized version of the "order" method and the "schedule" method is specialized version of the "grouping" method. The user is, of course, encouraged to use specialized problem solving methods whenever possible in order to improve upon the quality of the solutions obtained.

Fig. 2 Evolver representation.



Each solving method handles constraints slightly differently. For example, when the recipe method is chosen for a set of adjusting cells, there is the option of requiring the values to be integer within given bounds. Evolver deals with external constraints (i.e., those that cannot be explicitly defined within the solving method) in two different ways. A cell can be used to summarize the feasibility of a solution trial. (This is referred to as the "hard constraint" method.) The cell must evaluate to the Boolean TRUE when all the constraints are satisfied by the proposed solution and to FALSE otherwise (see Figure 2). If the constraint cell evaluates to FALSE, Evolver attempts to modify the infeasible solution so it becomes feasible, by a procedure referred to as "backtracking." If the procedure fails, the solution is discarded. The second, and most recommended form of constraint satisfaction relates to the use of a penalty function. (This is referred to as the "soft constraint" method.) This method relaxes the original problem by creating a Lagrangean function, for which the user is expected to provide appropriate multipliers.

We first use Evolver's recipe solving method with a hard constraint to solve our illustrative problem. The optimal solution to this problem, when no integer restrictions are imposed on the decision variables, is simply $x_4 = 8\ 1/3$ and $x_i = 0$ for $i \neq 4$, with a total profit of 33 1/3. We impose the following bounds on the decision variables $(0, 25/a_i)$, where a_i is the constraint coefficient for decision variable i. Starting from a solution where all the variables are set to zero, Evolver finds a solution with a total profit of 25.41 after 5 minutes of search on a 200 MHz Pentium machine (using default values for crossover rate, mutation rate and population size). An additional 5 minutes of search starting from the best solution found, those not improve the objective function. This simple experiment illustrates the difficulties that Evolver's search engine encounters in solving constrained problems using the hard constraint method.

When the problem is relaxed by creating a penalty function (with a penalty of 100 for every unit of violation of the single constraint), Evolver finds a solution with a total profit of 28.47. After similar experiments, the user will find that Evolver is better suited for unconstrained optimization problems, and therefore, solution quality generally improves when using penalty functions to handle constraints. Given enough time, it is possible that Evolver actually converges to the optimal solution to this simple problem.

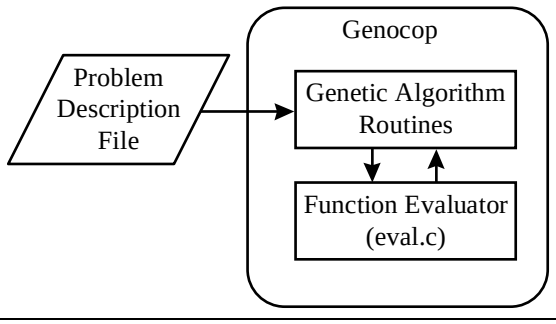
By restricting the variables to be integers within the ranges given by $(0, \lceil 25/a_i \rceil)$, where $\lceil x \rceil$ represents the largest integer less than or equal to x, the problem becomes a knapsack problem. Starting with a solution that assigns a value of zero to all decision variables and using the soft constraint method, Evolver is now able to find the solution with $x_2 = 1$, $x_4 = 5$, $x_5 = 1$ and a total profit of 32 (after 5 minutes of search time). This solution is only one unit away from the optimal of 33, which assigns $x_2 = 1$ and $x_4 = 6$. Finally, we transform the problem into a 0-1 knapsack instance, by modifying the upper bounds on the decision variables to be all 1. Evolver is now able to consistently find the optimal solution ($x_2 = x_4 = x_5 = x_6 = 1$ and $x_1 = x_3 = 0$ with a total profit of 25) within the first 10 seconds of the search. This result is not affected by the use of the soft or hard constraint methods. It is interesting to note the progressive improvement of the solutions found by Evolver as the problem was transformed from continuous, to general integer, and finally to zero-one discrete optimization.

3. Genocop

Genocop is a genetic algorithm-based program for constrained and unconstrained optimization developed by Michalewicz (1994). Genocop 4.0 expands the functionality of earlier versions by allowing decision variables to be discrete (both general integers and Boolean). The system aims at finding a global optimum (minimum or maximum) of a function, while meeting linear constraints. Although the constraints must be linear, the objective function can consist of nonlinear terms. (A recent version of the system, Genocop III, is designed to handle nonlinear constraints as well as nonlinear objective functions.) All Genocop versions are written in C and run without changes on most UNIX systems. The system, however, can also be compiled in DOS and Windows machines. This software is the property of Zbigniew Michalewicz, but permission is granted to be copied and used for scientific, noncommercial purposes. A copy of the software can be obtained from ftp.uncc.edu:/coe/evol/.

We now undertake to solve our illustrative example with Genocop. We will solve the three versions of the problem used to illustrate Evolver's functionality, i.e., the continuous case, the general integer case, and the 0-1 case. Genocop's system configuration is depicted in Figure 3.

Fig. 3 Genocop representation.



Genocop also uses that "black box" approach for the function evaluator, however, the evaluation routine becomes part of the system before the search starts. In other words, although the GA routines are not

adapted to exploit the form of the objective function, the function evaluator becomes part of the compiled system before the optimization process is initiated. Except for the objective function evaluation (coded in the C program file eval.c), the optimization problem is described in a text file that the system reads at an initialization stage. This input file contains the problem description (e.g., number of variables, constraints, constraint coefficients and optimization direction) and search parameters (e.g., population size, number of generations, crossover and mutation information). While Evolver provides four adjustable parameters (population size, crossover rate, mutation rate and search time), a user can adjust up to 18 parameters in Genocop. This reflects the different nature of these two systems: while Evolver is a commercial implementation, Genocop is an experimental research code. In our illustrations we will use the default parameters provided by the developers. At the end of the run, Genocop creates an output file with search information and the best solution found.

In order to solve our illustrative optimization problem with Genocop, we first create an eval.c file with an evaluation function of the following form:

$$5*X[1]+9*X[2]+12*X[3]+4*X[4]+3*X[5]+9*X[6].$$

We assume that the vector X is an argument to the function and that the specific values for each element of this vector will be chosen by the GA routines. Then, the input file is created to describe the rest of the optimization problem (in this case, a single constraint). We will start by making all variables continuous and by placing lower and upper bounds as described in the previous section. Using the default parameters, Genocop is able to find the optimal solution in less than 500 generations, which represent 4 seconds of CPU time in an Alpha workstation. When the problem is changed to a general-integer knapsack problem, Genocop finds the optimal solution with a total profit of 33 in the first generation. It takes Genocop 3 generations to find the optimal solution to the 0-1 knapsack case.

4. OptQuest

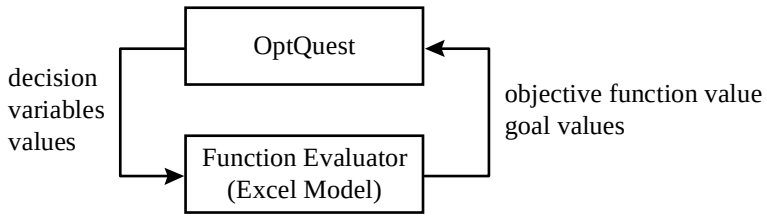
OptQuest is a general-purpose optimizer developed by Glover, Kelly and Laguna (1996) using the scatter search methodology. Scatter search is a population-based approach that shares some common elements with genetic algorithms but uses a fundamentally different search philosophy. A detailed description of scatter search can be found in Glover and Laguna (1997).

The first commercial version of OptQuest was developed for Micro Saint 2.0, a discrete event simulation software (Micro Analysis and Design, Inc.). This implementation treats simulation models created in Micro Saint as “black box” function evaluators. The simulation model software returns the objective function value corresponding to a particular set of input factors (i.e., decision variables). OptQuest generates new input factors based on scatter search strategies and the process repeats. A second version of OptQuest is currently in beta testing to be integrated in Crystal Ball, a risk analysis software (Decisioneering, Inc.). Crystal Ball operates as an add-in function to Excel. In Crystal Ball, users model risk by assigning probability distribution functions to key input data. For each set of values for the decision variables, a simulation is performed and observations are collected for forecast values of interest. OptQuest can then be invoked from the Crystal Ball menu to search for optimal values of the decision variables.

In this section, we use a version of OptQuest that operates in similar way as the one being integrated with Crystal Ball (Laguna 1997). That is, the user is asked to create a function evaluator using Excel and the rest of the optimization problem (constraints and bounds) is described in the OptQuest interface. In this sense, this configuration of OptQuest is also similar to the way Evolver operates, with two exceptions: (1) OptQuest can handle more than one return value from the evaluator, from which one of these values is

associated with the objective function and the others are considered goals, and (2) the constraint set on the decision variables is defined inside the system and not within the function evaluator (see Figure 4).

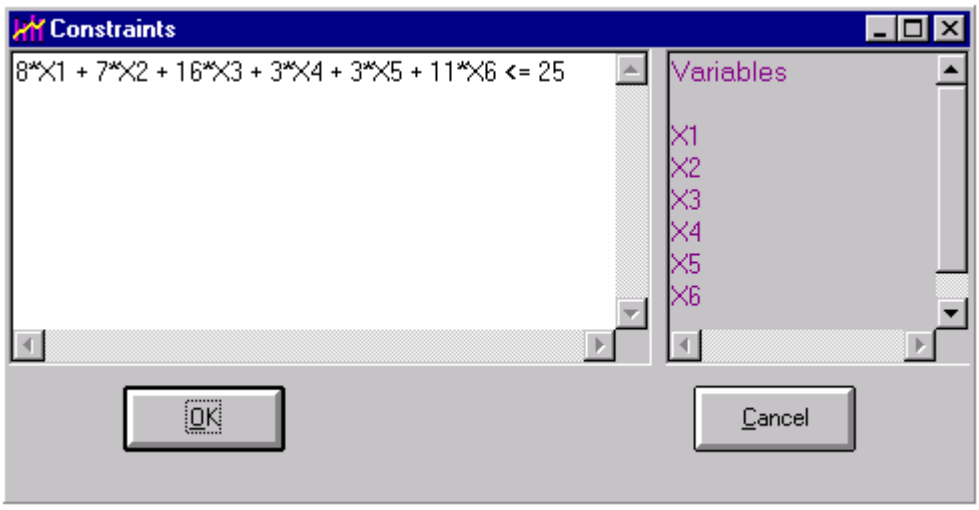
Fig. 4 OptQuest representation.



In OptQuest, goals are values that can only be known after the function evaluation is performed. In risk analysis, for example, a user may wish to optimize the expected return of an investment portfolio while limiting the standard deviation to be less than a desired value. Evolver and Genocop would handle this problem by constructing a nonlinear function that penalizes portfolios with standard deviations that are larger than the desired limit. OptQuest, on the other hand, handles this situation by defining a an objective to be maximized (i.e., the expected return) and a goal to be satisfied (i.e., a bound on the standard deviation of returns). In particular, the problem is described to OptQuest as a maximization problem on the cell that calculates the expected return, with an "upper bound" goal which restricts the value standard deviation cell to be less than the maximum desired limit. The system then considers feasible solutions to be only those that meet the constraints on the decision variables (which are inputs to the function evaluator) while also satisfying the goals (which are outputs to the function evaluator).

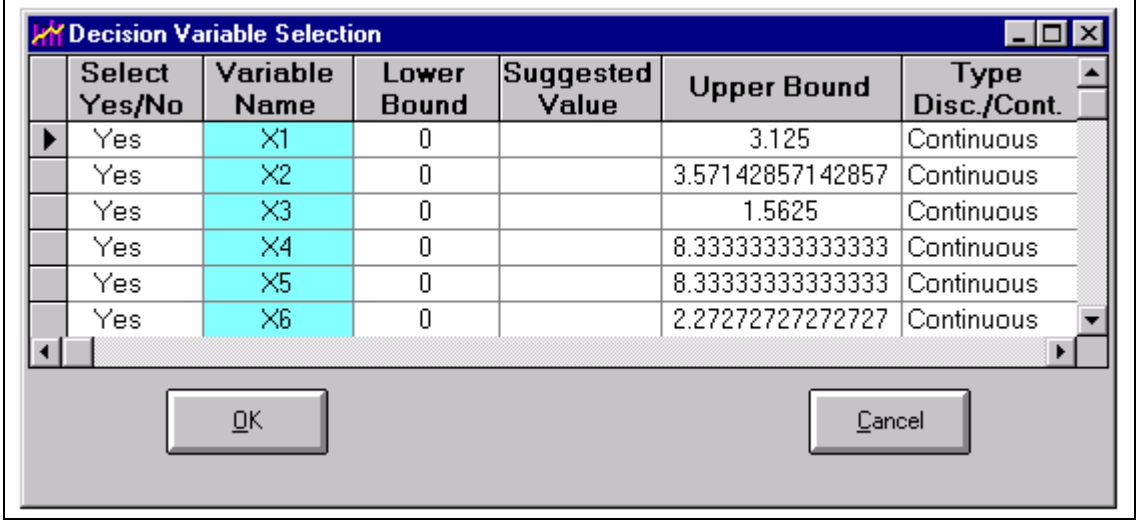
Constraints in OptQuest are handled by a linear programming solver. This is a major departure from Evolver, which tries to manipulate an infeasible solution to make it feasible (a process that may fail). The use of linear programming (or mixed integer programming, when discrete decision variables are part of the model) guarantees that a feasible solution (with respect to the decision variables) will be found, if one exists. The constraints are entered in an algebraic form, using the constraint editor in the system (see Figure 5).

Fig. 5 Constraint editor.



Note that the lower and upper bounds are not included in the constraint set. This information can be entered either in the Excel model or the decision variable window in the OptQuest interface (Figure 6). Unlike Evolver and Genocop, the OptQuest system is design to automatically adjust search parameters based on the maximum search time selected by the user. Therefore, the user has no direct access to the search parameters, such as the population size or the number of new population points ("offspring" in GAs) generated by a linear combination of two reference points. In addition to the total search time, the user can decide whether to activate a neural network filter and a solution log file.

Fig. 6 Decision variable selection.



The neural network and the log file options are particularly relevant in situations where the function evaluation is computationally expensive. The log file records all the solutions generated during the search, and before a new solution is sent to the evaluator, the system searches for the solution in this file. If the solution is found, then there is no need for evaluating it. For obvious reasons, the search in the log file is only justified when the evaluation of solutions is computationally expensive.

The neural network, when activated, serves as a filter of potentially inferior solutions. After a number of initial iterations, a neural network is trained to predict the output of the function evaluator based on the values of the decision variables. Once the training is completed, new solutions are sent to the neural network routine to predict the output of the function evaluator. If the prediction indicates that the solution is "likely" to be inferior to the best known, then the solution is not evaluated by the function evaluator. Again, the training and use of the neural network is only justified when the function evaluation requires excessive computational time.

We now use OptQuest to solve our illustrative example. The log file and the neural network options are not activated in this case. We allow one minute to search for the optimal solutions. In all cases (continuous, general integer knapsack and 0-1 knapsack) the system finds the optimal solution within the first 10 seconds of the search (using less than 100 calls to the function evaluator). These results are not surprising, since infeasible solutions are "mapped" into the feasible region by the solution of a linear or mixed-integer program, which makes the single constraint satisfaction a trivial problem.

5. A Nonlinear Optimization Problem

In this section, we undertake to compare Evolver, Genocop and OptQuest using a nonlinear optimization problem, for which decision variables are continuous, integer and Boolean. This is a very limited test of these systems, since each has features that the others lack. For example, Evolver has three different solving methods, from which only the recipe method is tested in our experiment. OptQuest includes mechanisms to efficiently handle the search time when the function evaluation is expensive, which are deactivated in this context. However, this limited experiment allows us to assess the relative efficiency of these systems in the context of nonlinear, discrete optimization. The problem to be solved can be expressed as follows:

Maximize $\sum_{i=1}^{10}(x_i - 2)^3 + \frac{1}{2}\sum_{i=11}^{20} i(x_i - 2)^2$

subject to

$-3x_1 + 7x_2 - 5x_4 + x_5 + x_6 + 2x_8 - x_9 - x_{10}$
$-9x_{11} + 3x_{12} + 5x_{13} + x_{16} + 7x_{17} - 7x_{18} - 4x_{19} - 6x_{20} \le -15$
$7x_1 - 5x_3 + x_4 + x_5 + 2x_7 - x_8 - x_9 - 9x_{10} + 3x_{11} + 5x_{12} + x_{15} + 7x_{16} - 7x_{17} - 4x_{18} - 6x_{19} - 3x_{20} \le 12$
$-5x_2 + x_3 + x_4 + 2x_6 - x_7 - x_8 - 9x_9 + 3x_{10} + 5x_{11} + x_{14} + 7x_{15} - 7x_{16} - 4x_{17} - 6x_{18} - 3x_{19} + 7x_{20} \le -11$
$7x_{14} - 7x_{15} - 4x_{16} - 6x_{17} - 3x_{18} + 7x_{19} \le -13$
$x_1 + x_2 + 2x_4 - x_5 - x_6 - 9x_7 + 3x_8 + 5x_9 + x_{12} + 7x_{13} - 7x_{14} - 4x_{15} - 6x_{16} - 3x_{17} + 7x_{18} - 5x_{20} \le 15$
$x_1 + 2x_3 - x_4 - x_5 - 9x_6 + 3x_7 + 5x_8 + x_{11} + 7x_{12} - 7x_{13} - 4x_{14} - 6x_{15} - 3x_{16} + 7x_{17} - 5x_{19} + x_{20} \le 14$
$2x_2 - x_3 - x_4 - 9x_5 + 3x_6 + 5x_7 + x_{10} + 7x_{11} - 7x_{12} - 4x_{13} - 6x_{14} - 3x_{15} + 7x_{16} - 5x_{18} + x_{19} + x_{20} \le -11$
$2x_1 - x_2 - x_3 - 9x_4 + 3x_5 + 5x_6 + x_9 + 7x_{10} - 7x_{11} - 4x_{12} - 6x_{13} - 3x_{14} + 7x_{15} - 5x_{17} + x_{18} + x_{19} \le 0$
$-x_1 - x_2 - 9x_3 + 3x_4 + 5x_5 + x_8 + 7x_9 - 7x_{10} - 4x_{11} - 6x_{12} - 3x_{13} + 7x_{14} - 5x_{16} + x_{17} + x_{18} + 2x_{20} \le 19$
$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17} + x_{18} + x_{19} + x_{20} \le 40$

$0 \le x_3 \le 5$
$0 \le x_5 \le 6$
$0 \le x_7 \le 4$
$0 \le x_8 \le 3$
$0 \le x_9 \le 2$
$0 \le x_{10} \le 2$
$0 \le x_{11} \le 3$
$0 \le x_{16} \le 15$
$0 \le x_{18} \le 10$
$0 \le x_{20} \le 20$

$x_1, x_2, x_4, x_6, x_{12}, x_{13}, x_{14}, x_{15}, x_{17}, x_{19} = \{0, 1\}$
$x_3, x_8, x_{11}, x_{16}, x_{20}$ integer.

In order to compare the systems, we chose a termination criterion of 6,000 trials for OptQuest and Evolver and 6,000 generations for Genocop. A trial in both OptQuest and Evolver equals a single call to the function evaluator. A generation in Genocop, on the other hand, equals *p* calls to the function evaluator, where *p* is the population size. Therefore, in this experiment, Genocop is allowed to call the function evaluator *p* more times than either OptQuest or Evolver. (The computational times, however, still favor Genocop because of the lack of graphical interface and the compiled form of the evaluation routine.) Measuring computational effort in terms of calls to the function evaluator is specially relevant in contexts

where the evaluation takes a considerable amount of computer time. In other words, when the function evaluation is computationally expensive, it is then desirable that the algorithm can efficiently search with a limited number of calls to the evaluator.

We solve two versions of the nonlinear problem formulated above. The first version considers the original mix of discrete and continuous variables, while the second version relaxes the integrality restrictions, making all variables continuous within the specified bounds. Our goal is to explore the effectiveness of the systems in both situations. The results are given in Table 1.

Table 1. Solutions of nonlinear problem.

	Mixed Integer			Continuous		
Variable	Genocop	Evolver	OptQuest	Genocop	Evolver	OptQuest
x_1	1.00000	0.00000	0.00000	0.06719	0.74073	0.00000
x_2	1.00000	0.00000	0.00000	0.44335	0.20934	1.00000
x_3	1.00000	1.00000	0.00000	1.41920	0.34545	1.59370
x_4	1.00000	1.00000	1.00000	0.78522	4.17563	0.00000
x_5	1.56781	5.16011	0.57921	1.09667	9.55514	0.00000
x_6	0.00000	0.00000	0.00000	0.00000	0.13035	0.00000
x_7	0.17249	1.09156	0.67736	0.00001	0.19799	0.00000
x_8	0.00000	0.00000	0.00000	0.22073	0.97593	0.00000
x_9	1.01183	1.02825	1.12946	1.99980	0.05334	2.00000
x_{10}	0.24787	0.26551	0.61196	0.01389	0.41716	0.00000
x_{11}	0.00000	2.00000	0.00000	0.00000	0.00940	0.00000
x_{12}	1.00000	1.00000	1.00000	0.72092	0.01650	1.00000
x_{13}	1.00000	0.00000	1.00000	0.18658	0.01434	1.00000
x_{14}	0.00000	0.00000	1.00000	0.72687	0.04354	1.00000
x_{15}	0.00000	0.00000	0.00000	0.00000	0.00416	0.00000
x_{16}	7.00000	6.00000	6.00000	6.34831	0.32306	5.53124
x_{17}	0.00000	0.00000	1.00000	0.08162	1.89413	0.00000
x_{18}	10.00000	9.43835	10.00000	10.00000	0.92331	10.00000
x_{19}	0.00000	0.00000	1.00000	0.00000	7.14386	1.00000
x_{20}	14.00000	12.00000	15.00000	15.88963	12.80841	15.87500
Objective	2347.972	1778.634	2433.984	2777.114	2033.182	2658.846

Table 1 shows that OptQuest finds the best solution when the problem contains discrete variables (with an objective value that is 3.66% better than Genocop's and 36.85% better than Evolver's). However, the best solution to the continuous problem is given by Genocop (beating OptQuest by 4.45% and Evolver by 39.59%). The solutions found by Evolver in both cases are clearly inferior.

6. Conclusions

The general-purpose optimizers described in this tutorial paper extend the application of metaheuristics by employing an operational framework that treats the objective function evaluation as a black box. Evolver and OptQuest allow users to model function evaluators as electronic spreadsheets in Excel. Genocop internalizes the function evaluation by requiring the user to write C code in the eval.c file. Because the Genocop is rebuilt every time the function evaluation changes, this solver is orders of magnitude faster

than either Evolver or OptQuest. The downside, of course, is that the user must have working knowledge of the C computer language to be able to use Genocop.

Evolver is the only system, out of the three, with more than one solving method. While OptQuest and Genocop are designed to operate on a set of discrete or continuous variables, Evolver can also operate directly on sequences by means of the "order" solving method. The additional solving methods (and the corresponding specialized versions) allow Evolver to use context information to improve efficiency. Although we only tested the "recipe" method here, experimentation outside the scope of this paper has shown that the "order" solving method is capable of providing high-quality solutions to small and medium size traveling salesman problem instances. In the limited experimentation shown here, however, Evolver's "recipe" solving method was shown inferior to both Genocop and OptQuest.

As mentioned earlier, OptQuest was originally designed to deal with optimization problems for which the objective function is evaluated by a computationally expensive procedure (e.g., a simulation model). Hence this system includes features, such as a neural network filter and a log file of solutions, that are specifically designed for the purpose of limiting the number of calls to the function evaluator. This approach is significantly different than the one taken by Genocop, which in each generation the number of function evaluation calls equals the population size. Table 2 compares the three systems considering a few different categories.

Table 2. Comparison of systems' features.

Category	*Evolver*	*Genocop*	*OptQuest*
User interface	Windows / Excel	Files / C	Windows / Excel
Solving methods	3	1	1
Goals	no	no	yes
Neural net filter	no	no	yes
Constraints	external	internal	internal
Output	graphical	text file	graphical

The "Goals" category in Table 2 refers to the explicit handling of bounds enforced on output values (i.e., values generated by the function evaluator). While Evolver and Genocop can handle goals if the user creates an appropriate penalty function, an OptQuest user can specify goals (independently from the objective function) and the system will search for solutions that optimize the objective function value while meeting the given goals. The neural network filter is another feature that is exclusive to OptQuest in this context.

In this tutorial paper we undertook to provide general descriptions of three metaheuristic-based optimizers and to compare their functionality. Research in the area of metaheuristics has made possible the development of optimization software that has the goal of providing high-quality solutions to complex systems. An appealing feature, that is common to metaheuristic optimizers, is that optimization problems do not have to be formulated as mathematical programs, a requirement shared by linear, nonlinear and mixed-integer programming solvers. As the complexity of systems increases, metaheuristics may prove a viable framework to fulfill decision makers' expectations of finding high-quality solutions to difficult optimization problems.

References

Glover, F., J. P. Kelly and M. Laguna (1996) "New Advances and Applications of Combining Simulation and Optimization," *Proceedings of the 1996 Winter Simulation Conference,* J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain (eds.), pp. 144-152.

Glover, F. and M. Laguna (1997) *Tabu Search,* Kluwer Academic Publishers, forthcoming.

Laguna, M. (1997) "Optimization of Complex Systems with OptQuest," Graduate School of Business, University of Colorado.

Michalewicz, Z. (1994) *Genetic Algorithms + Data Structures = Evolution Programs,* Springer-Verlag, New York, NY, Second Edition.