Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

NEW ADVANCES AND APPLICATIONS OF COMBINING SIMULATION AND OPTIMIZATION

Fred Glover
James P. Kelly
Manuel Laguna

Graduate School of Business
University of Colorado
Boulder, Colorado 80309-0419, U.S.A.

ABSTRACT

The area of integrating simulation and optimization has recently undergone remarkable changes. New advances are making available applications of simulation that previously had been considered infeasible or beyond the scope of current technology to handle.

This paper describes recently developed computer software that effectively integrates simulation and optimization. The software employs graphics to show the performance of the search mechanisms that control the integration. We also demonstrate the ability of the system to find optimal and near optimal solutions in minutes for applications where an exhaustive examination of relevant alternatives requires days or months. The scaling of reduced time applies equally to settings where simulation runs require greater time. As a result, for a selected practical time limit, the new approach provides the opportunity to obtain decisions and scenarios whose quality greatly exceeds the quality available in the past.

1 INTRODUCTION

The ability to guide a series of simulations in the most effective way, instead of blindly itemizing scenarios (with the hope that at least one of those itemized will be one that is most desirable to implement) has been a long standing goal. The integration of simulation and optimization is putting this goal within practical reach.

Now, practical software exists that is capable of interfacing simulation and special search processes, to guide a series of simulations to uncover optimal or near optimal scenarios. Applications include the goals of finding:

- the best configuration of machines for production scheduling;
- the best investment portfolio for financial planning;
- the best utilization of employees for workforce planning;
- the best location of facilities for commercial distribution;
- the best operating schedule for electrical power planning;
- the best assignment of medical personnel in hospital administration;
- the best setting of tolerances in manufacturing design;
- the best set of treatment policies in waste management;
- and many other practical objectives.

Current advances not only open new doors for simulation, but also extend the areas to which optimization can be applied. The great advantage of simulation, which has been lacking in traditional optimization, is the ability to handle uncertainties and complex interactions (at a level that can scarcely be formulated in standard optimization models). The marriage of simulation and optimization offers a way to overcome this limitation.

Earlier attempts to create methods for optimizing simulations have largely been based on ad hoc approaches, or have relied on the user to run through a cumbersome "seat of the pants" analysis. Alternatively, they have been based on stochastic approximation designs whose main focus involves the analysis of convergence behavior in an infinite time frame. Not surprisingly, the results of such efforts have left a great deal to be desired from a practical standpoint. As a step in the direction of greater rigor within a finite time horizon, a systematic catalog of all possible alternatives may be examined by complete enumeration algorithms. Although this approach guarantees optimal solutions, it has very limited application. As an example of an exceedingly simple setting where enumeration may be practicable, suppose that a simulation model depends on only two input factors, as for example, a job shop with parallel machines of 2 types. If a feasible shop design would allow from 1 to 10 machines of each type, then 100 simulation runs are needed to enumerate all possibilities. If each simulation is relatively short (e.g., 3 seconds), then the entire process could be done in 5 minutes of computer time. However, if instead of 2 machine types we allow a very modest increase to consider 5 types, then enumerating all alternatives to find an optimal one would require $10^5 = 100,000$ simulations, or approximately 3.5 days of computer time. Of course, most simulation settings are not really so simple as the one described. It is easily possible for complete enumeration to take weeks or even months to carry out.

Recent developments in the area of optimization have led to the creation of intelligent search procedures capable of finding optimal or near optimal solutions to complex problems with large solution spaces, by exploring only a small fraction of the possible alternatives. In particular, the system we describe in this paper is the result of implementing a search technology known as *scatter search,* by customizing its operation to the context of optimizing simulations. The system searches for the best possible solution to an optimization problem, defined on a set of input factors to a simulation model.

2 CLASSICAL METHODS AND META-HEURISTICS

To give a background for understanding the rationale underlying our approach, we provide a brief review of classical optimization perspectives and the emerging role of meta-heuristics.

Formally, optimization deals with finding the best (optimal) solution to problems that in general can be expressed in the form of an objective function (to be optimized) and a set of constraints (which restrict the values of the decision variables). The best known optimization tool is linear programming, which model assumes that the objective function and constraints can be expressed using linear functions. Linear programming techniques are able to find optimal solutions to problems without the need to evaluate all possible alternatives. Models with thousands and even millions of variables can be solved with reasonable amounts of computer time.

Evidently, however, not all business and industrial problems can be expressed by means of a linear objective and linear equalities or inequalities. Many complex systems may not even have a convenient mathematical representation, linear or non-linear. Techniques such as linear programming and its cousins (non-linear programming and integer programming) generally require a number of simplifying assumptions about the real system to be able to properly frame the problem. One usual simplifying assumption is to disregard the so called "statistical fluctuations" of the system. For instance, a production process may be modeled by assuming that production times do not vary, permitting a single time estimate to be used for modeling purposes. The advantage, of course, is that once the problem has been formulated, well-established techniques are often able to find an optimal solution, provided the formulation is "congenial." However, there are problems such as production scheduling where even the deterministic versions remain hard to solve. This is due to the combinatorial nature of these problems, as illustrated by the situation where the goal is to determine an optimal order in which to process a set of jobs. Even for fewer that a hundred jobs to be ordered, the number of alternative configurations is astronomical. (Seventy jobs would require longer than the age of the universe to enumerate using all of today's computers working at once!)

Due to the realization that practical problems are not going to become any easier to solve, and that practitioners are seeking solutions to increasingly more complex problems, researchers have actively developed solution procedures that are referred to as *meta-heuristics.* Heuristics have been used for many years to provide approximate solutions to complex problems. For example, a production heuristic may be to give priority to jobs with the shortest estimated processing time. Depending on the context, this heuristic (or processing rule) may actually work fairly well. However, in some other situations the results may be disastrous, with dire consequences for equipment utilization, production lead times, and work-in-process inventory.

The alternatives seem at first to be less than encouraging: either to seek optimal solutions to simplified problems or to seek sub-optimal and possibly very poor solutions to complex systems. The area of meta-heuristics arose with the goal of providing something better, based on integrating high-level intelligent procedures and fast computer implementations. Numerous successful applications emerged, but with an accompanying limitation: typically they required highly problem-specific designs, customizing the solution procedures to each particular case. Every time a new problem surfaced, a new procedure needed to be developed. Meta-heuristic approaches are based on general principles, but they also owe their efficiency to the knowledge of characteristics particular to each situation. In this sense, there is no separation between the model and the solution procedure. In fact, the solution procedure may be seen as a way of modeling the problem.

Of course, it is preferable to separate the solution procedure from the system we are trying to optimize if such a separation can be achieved successfully. The disadvantage of this "black box" approach (see Figure 1), is that the optimization procedure is generic and does not know anything about what goes on inside of the box.

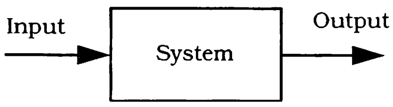


Figure 1: System as a black box.

The clear advantage, on the other hand, is that the same optimizer can be used for many systems. Our approach is an implementation of a generic optimizer that successfully embodies the principle of separating the method from the model. The optimization problem is defined outside the system, which is represented in this case by a simulation model. Therefore, the simulation model can change and evolve to incorporate additional elements, while the optimization routines remain the same. Hence, there is a complete separation between the model that represents the system and the procedure that is used to solve optimization problems defined within this model.

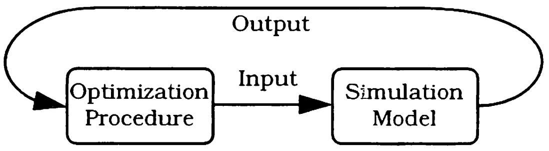


Figure 2: Coordination between optimization and simulation.

The optimization procedure uses the outputs from the simulation model which evaluate the outcomes of the inputs that were fed into the model. On the basis of this evaluation, and on the basis of the past evaluations which are integrated and analyzed with the present simulation outputs, the optimization procedure decides upon a new set of input values (see Figure 2). The optimization procedure is designed to carry out a special "non-monotonic search," where the successively generated inputs produce varying evaluations, not all of them improving, but which over time provide a highly efficient trajectory to the best solutions. The process continues until some termination criterion is satisfied (usually given by a limit expressing the user's preference for the amount of time to be devoted to the search). The underlying components of our method, scatter search and tabu search, are briefly sketched in the next two sections.

3 SCATTER SEARCH

Two of the best-known meta-heuristics are genetic algorithms and tabu search. Genetic Algorithm (GA) procedures were developed by John Holland in the early 1970s, at the University of Michigan (Holland 1975). Parallel to the development of GAs, Fred Glover, at the University of Colorado, established the principles and operational rules for tabu search (TS) and a related methodology know as *scatter search* (Glover 1977).

Scatter search has some interesting commonalties with GA ideas, although it also has a number of quite distinct features. Several of these features have come to be incorporated into GA approaches after an intervening period of approximately a decade, while others remain largely unexplored in the GA context.

Scatter search (Glover 1994) is designed to operate on a set of points, called *reference points,* that constitute good solutions obtained from previous solution efforts. The approach systematically generates linear combinations of the reference points to create new points, each of which is mapped into an associated feasible point. Tabu search is then superimposed to control the composition of reference points at each stage. Tabu Search (see e.g., Glover and Laguna 1993) has its roots in the field of artificial intelligence as well as in the field of optimization. The heart of tabu search lies in its use of adaptive memory, which provides the ability to take advantage of the search history in order to guide the solution process. In its simplest manifestations, adaptive memory is exploited to prohibit the search from reinvestigating solutions that have already been evaluated. However, the use of memory in our implementation is much more complex and calls upon memory functions that encourage search diversification and intensification. These memory components allow the search to escape from locally optimal solutions and in many cases find a globally optimal solution.

Similarities are immediately evident between scatter search and the original GA proposals. Both are instances of what are sometimes called "population based" approaches. Both incorporate the idea that a key aspect of producing new elements is to generate some form of combination of existing elements. On the other hand, several contrasts between these methods may be noted. The early GA approaches were predicated on the idea of choosing parents randomly to produce offspring, and further on introducing randomization to determine which components of the parents should be combined. By contrast, the scatter search approach does not correspondingly make recourse to randomization, in the sense of being indifferent to choices among alternatives. However, the approach is designed to incorporate strategic probabilistic biases, taking account of evaluations and history. Scatter search focuses on generating relevant outcomes without losing the ability to produce diverse solutions, due to the way the generation process is implemented. For example, the approach includes the generation of new points that are not convex combinations of the original points. The new points then may contain information that is not contained in the original reference points.

Scatter search is an *information driven* approach, exploiting knowledge derived from the search space, high-quality solutions found within the space, and trajectories through the space over time. The combination of these factors creates a highly effective solution process. The incorporation of such designs is responsible for endowing our system with the ability to solve complex simulation-based problems with unprecedented efficiency.

4 TABU SEARCH BASICS

One way of intelligently guiding a search process is to forbid (or discourage) certain solutions from being chosen based on information that suggests these solutions may duplicate, or significantly resemble, solutions encountered in the past. In tabu search, this is often done by defining suitable attributes of moves or solutions, and imposing restrictions on a set of the attributes, depending on the search history. Two prominent ways for exploiting search history in TS are through *recency* and *frequency* memories. Recency memory is typically (though not invariably) a short-term memory that is managed by structures or arrays called "tabu lists," while frequency memory more usually fulfills a long term search function. A standard form of recency memory discourages moves that lead to solutions with attributes shared by other solutions recently visited. A standard form of frequency memory discourages moves leading to solutions whose attributes have often been shared by solutions visited during the search, or alternately encourages moves leading to solutions whose attributes have rarely been seen before. Another standard form of frequency memory is defined over subsets of elite solutions to fulfill an intensification function.

Short and long term components based on recency and frequency memory can be used separately or together in complementary TS search strategies. Note that this approach operates by implicitly modifying the neighborhood of the current solution. Tabu search in general includes many enhancements to the scheme sketched here, and we refer the interested reader to Glover and Laguna (1993) or Glover (1996). The details of the short-term and long-term adaptive memories, and a recovery strategy for both intensifying and diversifying the search are discussed in the following section.

5 AN OVERVIEW OF THE ALGORITHM

We assume that a solution to the optimization problem can be represented by a n-dimensional vector $\mathbf{x}$, where x_i may be a real or an integer bounded variable (for $i = 1, \ldots, n$). In addition, we assume that the objective function value $f(\mathbf{x})$ can be obtained by running a related simulation model that uses $\mathbf{x}$ as the value of its input factors. Finally, a set of linear constraints (equality or inequality) may be imposed on $\mathbf{x}$.

The algorithm starts by generating an initial population of reference points. The initial population may include points suggested by the user, and it always includes the following midpoint:

$$x_i = l_i + (u_i - l_i)/2,$$

where u_i and l_i are the upper and lower bounds on x_i, respectively. Additional points are generated with the goal of creating a diverse population. A population is considered diverse if its elements are "significantly" different from one another. We use a distance measure to determine how "close" a potential new point is from the points already in the population, in order to decide whether the point is included or discarded.

Every reference point $\mathbf{x}$ is subjected to a feasibility test before it is evaluated (i.e., before the simulation model is run to determine the value of $f(\mathbf{x})$). The feasibility test consists of checking (one by one) whether the linear constraints imposed by the user are satisfied. An infeasible point $\mathbf{x}$ is made feasible by formulating and solving a linear programming (LP) problem. The LP (or mixed-integer program, when $\mathbf{x}$ contains integer variables) has the goal of finding a feasible $\mathbf{x}^*$ that minimizes the absolute deviation between $\mathbf{x}$ and $\mathbf{x}^*$.

The population size is automatically adjusted by the system considering the time that is required to complete one evaluation of $f(\mathbf{x})$ and the time limit the user has allowed the system to search. Once the population is generated, the procedure iterates in search of improved outcomes. At each iteration two reference points are selected to create four offspring. Let the parent-reference points be $\mathbf{x}_1$ and $\mathbf{x}_2$, then the offspring $\mathbf{x}_3$ to $\mathbf{x}_6$ are found as follows:

$$\mathbf{x}_3 = \mathbf{x}_1 + d$$
$$\mathbf{x}_4 = \mathbf{x}_1 - d$$
$$\mathbf{x}_5 = \mathbf{x}_2 + d$$
$$\mathbf{x}_6 = \mathbf{x}_2 - d$$

where $d = (\mathbf{x}_1 - \mathbf{x}_2)/3$. The selection of $\mathbf{x}_1$ and $\mathbf{x}_2$ is biased by the values $f(\mathbf{x}_1)$ and $f(\mathbf{x}_2)$ as well as the tabu search memory functions. An iteration ends by replacing the worst parent with the best offspring, and giving the surviving parent a tabu-active status for given number of iterations. In subsequent iterations, the use of two tabu-active parents is forbidden.

5.1 Restarting Strategy

In the course of searching for a global optimum, the population may contain many reference points with similar characteristics. That is, in the process of generating offspring from a mixture of high-quality reference points and ordinary reference points member of the current population, the diversity of the population may tend to decrease. A strategy that remedies this situation considers the creation of new population.

Our implementation of a restarting mechanism has the goal of creating a population that is a blend of high-quality points found in earlier explorations (we call these the *elite* points) complemented with points generated in the same way as during the initialization phase. The restarting procedure, therefore, injects diversity through newly generated points and preserves quality through the inclusion of elite points.

5.2 Adaptive Memory and the Age Strategy

Some of the points in the initial population may have poor objective function values. Therefore, they may never be chosen to play the role of a parent and would remain in the population until restarting. To additionally diversify the search, we increase the "attractiveness" of these unused points over time. This is done by using a form of long-term memory that is different from the conventional frequency-based implementation.

In particular, we introduce the notion of "age" and define a measure of "attractiveness" based on the age and the objective function value of a particular point. The idea is to use search history to make reference points not used as parents "attractive," by modifying their objective function values according to their age.

At the start of the search process, all the reference points $\mathbf{x}$ in a population of size p have zero age. At the end of the first iteration, there will be p-1 reference points from the original population and one new offspring. The ages of the p-1 reference points are made one and that of the new offspring zero. The process then repeats for the subsequent iterations, and the age of every reference point increases by one in each iteration except for the age of the new population member whose age is initialized to zero. (A variant of the above procedure sets the surviving parent's age also to 0.)

Each reference point in the population has an associated age an objective function value. These two values are used to define a function of attractiveness that makes an old high-quality point the most attractive. Low-quality points become more attractive as their age increases.

5.3 Neural Network Accelerator

This strategy is designed to increase the power of the system's search engine. The concept behind embedding a neural network is to "screen out" values $\mathbf{x}$ that are likely to result in a very poor value of $f(\mathbf{x})$. The neural network is a prediction model that helps the system accelerate the search by avoiding simulation runs whose results can be predicted as inferior. Engaging the neural network accelerator is an option to the user. When the neural network is used, information is collected about the objective function values obtained by different optimization variable settings. This information is then used to train the neural network during the search. The system automatically determines how much data is needed and how much training should be done, based once again on both the time to perform a simulation and the optimization time limit provided by the user.

The neural network is trained on the historical data collected during the search and an *error* value is calculated during each training round. This error refers to the accuracy of the network as a prediction model. That is, if the network is used to predict $f(\mathbf{x})$ for $\mathbf{x}$-values found during the search, then the error indicates how good those predictions are. The error term can be calculated by computing the differences between the known $f(\mathbf{x})$ and the predicted $\hat{f}(\mathbf{x})$ objective function values. The training continues until the error reaches a minimum prespecified value.

The neural network accelerator can be used at several risk levels. The risk is associated with the probability of discarding $\mathbf{x}$ when $f(\mathbf{x})$ is better than $f(\mathbf{x}_{best})$, where $\mathbf{x}_{best}$ is the best solution found so far. The risk level is defined by the number of standard deviations used to determine how close a predicted value $\hat{f}(\mathbf{x})$ is of the best value $f(\mathbf{x}_{best})$. A risk-averse user would, for instance, would only discard $\mathbf{x}$ if $\hat{f}(\mathbf{x})$ is at least three standard deviations larger than $f(\mathbf{x}_{best})$, in a minimization problem.

6 THE OPTQUEST SYSTEM

A commercial implementation of the system described above has been recently released under the name of OptQuest (1996). In its current version, OptQuest has been specifically customized to help users to find optimal input parameter settings to simulation models built with Micro Saint 2.0. (Micro Saint is registered trademark of Micro Analysis and Design, Inc.) In order to use OptQuest the user first creates a Micro Saint model. Once the simulation model has been created, an "Optimize" option can be selected within Micro Saint to access OptQuest. The interaction between OptQuest and Micro Saint can be summarized as follows:

1) Micro Saint invokes OptQuest and the most current version of the simulation model is stored in the corresponding model file.

2) OptQuest reads the model file from disk to allow the user to select optimization variables from a "Variable Catalog."

3) The user describes the optimization problem (which may include constraints) in terms of the selected variables and the system variable named *objective*, which calculates $f(\mathbf{x})$ for a given simulation run with input factors $\mathbf{x}$.

4) OptQuest repeatedly calls Micro Saint to perform simulation runs during the search for the optimal solution.

5) Micro Saint returns simulation results (i.e., the value of *objective*) to OptQuest.

6) Upon exiting OptQuest, the model may be updated with the values found during the search.

The OptQuest design is quite flexible, since with limited programming effort the simulator can be replaced to customize the system for different applications.

6.1 A Job Shop Simulation

We illustrate the operation an some of the features of OptQuest with a simulation of a Job Shop. The job shop is simulated as network of queues. The job shop has several different processing centers with one or more identical machines in each center. Several types of jobs have different routings through the job shop. We consider a job shop with five different machine centers (*drills*, *grinders*, *lathes*, *punches*, and *saws*). We also consider that three different types of jobs (A, B, and C) arrive to the job shop for processing. The Micro Saint network representation of this model is shown in Figure 3.

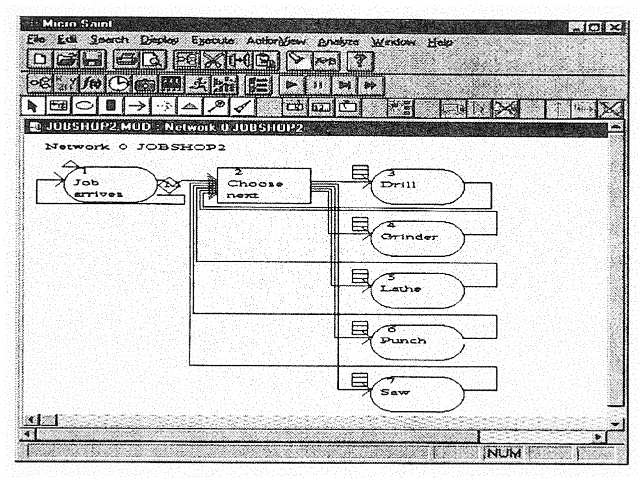
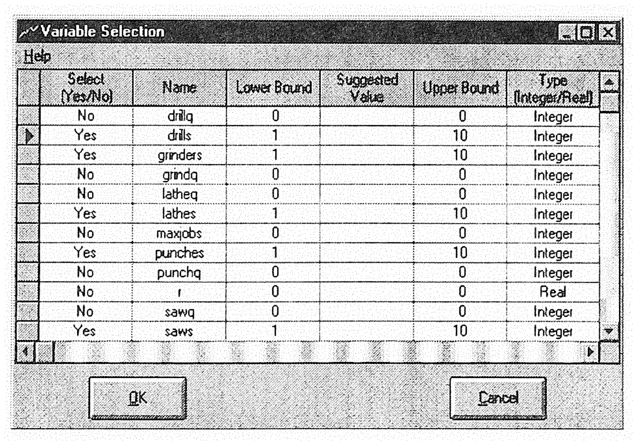

Micro Saint
File Edit Search Display Execute ActionView Analyze Window Help
JOBSHOP2.MOD : Network 0 JOBSHOP2
Network 0 JOBSHOP2
Job arrives
2 Choose next
3 Drill
4 Grinder
5 Lathe
6 Punch
7 Saw
NUM

Figure 3: Micro Saint task network.

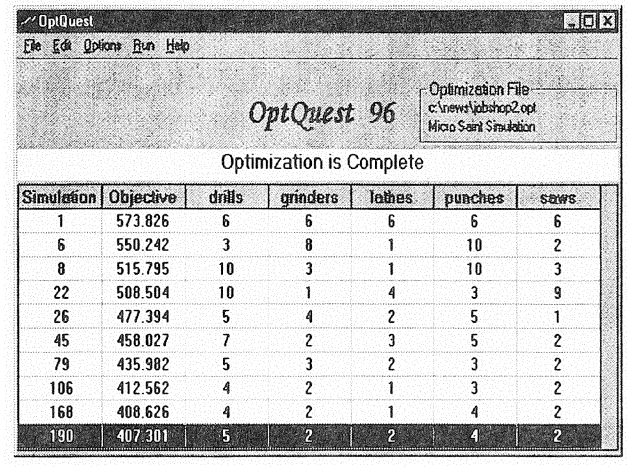
The objective function in this problem creates a tradeoff between minimizing the makespan and the cost associated with the number of machines used in each work center. Let the lower and the upper bounds be 1 and 10, respectively, for the number of machines in each work center. Then the solution $\mathbf{x} = \{1, ..., 1\}$ minimizes the capital cost of equipment, while the solution $\mathbf{x} = \{10, ..., 10\}$ minimizes the makespan. We consider that $\mathbf{x} = \{drills, grinders, lathes, punches, saws\}$. The OptQuest screen shown in Figure 4 is used to select the variables for optimization:



Select (Yes/No)	Name	Lower Bound	Suggested Value	Upper Bound	Type (Integer/Real)
No	drillq	0		0	Integer
Yes	drills	1		10	Integer
Yes	grinders	1		10	Integer
No	grindq	0		0	Integer
No	latheq	0		0	Integer
Yes	lathes	1		10	Integer
No	maxjobs	0		0	Integer
Yes	punches	1		10	Integer
No	punchq	0		0	Integer
No	r	0		0	Real
No	sawq	0		0	Integer
Yes	saws	1		10	Integer

Figure 4: Variable selection screen.

The column labeled "Suggested Value" allows the user to suggest a reference point $\mathbf{x}$ to be included in the initial population. The "Variable Selection" screen shows all the variables defined in the simulation model, however, only a few of them are appropriate to become part of the optimization model. For instance, setting the value of *drillq* as part of the optimization process is meaningless, since this variable is used in the model to keep track of the number of jobs waiting to be processed by an available drill. The main OptQuest screen looks as depicted in Figure 5 after a 10-minute run using the "Aggressive Neural Network" option.



Optimization is Complete

Simulation	Objective	drills	grinders	lathes	punches	saws
1	573.826	6	6	6	6	6
6	550.242	3	8	1	10	2
8	515.795	10	3	1	10	3
22	508.504	10	1	4	3	9
26	477.394	5	4	2	5	1
45	458.027	7	2	3	5	2
79	435.982	5	3	2	3	2
106	412.562	4	2	1	3	2
168	408.626	4	2	1	4	2
190	407.301	5	2	2	4	2

Figure 5: Main OptQuest screen.

6.2 Adding Constraints

We have mentioned that an important feature of OptQuest is the flexibility to impose constraints on the variables selected for optimization. In many situations it is desirable to search for solutions that satisfy restrictions that must be met by the system being simulated. OptQuest allows the user to impose any number of constraints, as long as they can be expressed as linear combinations of the optimization variables.

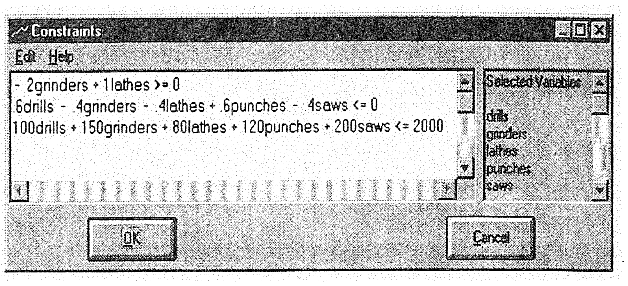
To illustrate the use of constraints, let's turn our attention again to the job shop problem. Suppose that we would like to search for an optimal job shop configuration that meets the following requirements:

1) The ratio of *lathes* to *drills* should be at least 2 to 1.

2) The number of *drills* and *punches* should not represent more than 40% of the total number of machines in the shop.

3) Given that machine costs (in thousands) are $100 for a drill, $150 for a grinder, $80 for a lathe, $120 for a punch, and $200 for a saw, the total cost of the equipment should not exceed $2,000K.

These restrictions can be represented in terms of linear constraints through simple algebraic transformations. Once they have been transformed, the constraints can be entered using the "Constraint Editor" in OptQuest. The screen in Figure 6 shows the result of adding the linear constraints that represent the requirements imposed on the problem.



```
- 2grinders + 1lathes >= 0
.6drills - .4grinders - .4lathes + .6punches - .4saws <= 0
100drills + 150grinders + 80lathes + 120punches + 200saws <= 2000
```

Figure 6: Constraint editor in OptQuest.

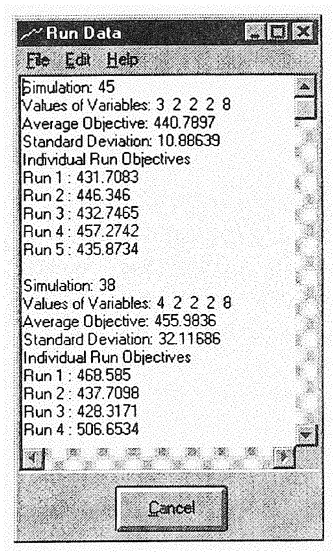
After running the optimization process for 10 minutes, the solution **x** = {2, 2, 5, 2, 2} is reported as the best. It can be easily verified that this solution satisfies all the constraints with $f(\mathbf{x}) = 405.206$.

It is interesting to note that the final objective function value (i.e., the best found in 10 minutes of search) is better than the one found for the unconstrained model (i.e., 407.301). This might seem surprising at first glance. However, restricting the search to regions where good solutions lie allows OptQuest to find high quality solutions faster. Therefore, if the user knows that good shop configurations meet the specifications set by the constraints, then by creating a constrained optimization model, better configurations can be found in a smaller amount of time.

6.3 Optimizing an Average Objective Function Value

We devote this subsection to address the optimization of the average response (or average objective function value). Instead of seeking a minimum or a maximum value of the objective function in a single run, it may sometimes be desirable to optimize its average value. This can be done by changing the number of runs that the simulator is asked to perform. One thing that it must remembered is that an increase in the number of runs causes an increase in the simulation time. So, the number of search iterations that can be performed in a fixed amount of time will decrease.

Suppose we change the number of runs to be 5 in our example. Now, OptQuest attempts to find the best values for *drills*, *grinders*, *lathes*, *punches*, and *saws* in order to minimize the average objective function value obtained after 5 runs of the simulation. In other words, the system attempts to find the best **x** such that $\bar{f}(\mathbf{x})$ is minimized, where the average value is found by running the simulation a given number of times with the same **x** value and different seeds for the random number generator. Detailed information concerning individual runs is displayed by the system on a separate window. This window is labeled "Run Data" and takes on the form depicted in Figure 7 during the search process in the job shop example.



```
Simulation: 45
Values of Variables: 3  2  2  2  8
Average Objective: 440.7897
Standard Deviation: 10.88639
Individual Run Objectives
Run 1 : 431.7083
Run 2 : 446.346
Run 3 : 432.7465
Run 4 : 457.2742
Run 5 : 435.8734

Simulation: 38
Values of Variables: 4  2  2  2  8
Average Objective: 455.9836
Standard Deviation: 32.11686
Individual Run Objectives
Run 1 : 468.585
Run 2 : 437.7098
Run 3 : 428.3171
Run 4 : 506.6534
```

Figure 7: Run data screen.

The 10-minute limit allows the search in this case to only evaluate 85 solutions. This is due to the longer simulation time needed to perform 5 runs instead of one every time a configuration is tested.

7 CONCLUSIONS

In this paper we have described recent advances in optimization and simulation technologies that have made possible the development of a system that can effectively perform the task of optimizing simulations. We showed the benefits of adapting the approach known as scatter search in this context, while complementing this methodology with appropriate tabu search elements.

We have also demonstrated the capabilities of a commercial software package that successfully implements the notion of optimizing a complex system (in this case represented by a simulation model). The software package includes a scatter search / tabu search module, a mixed integer programming solver, and a procedure to configure and train neural networks. All these tools are integrated by a user-friendly interface.

We are currently undertaking additional research in the area of optimizing simulations with the goal of producing systems that are still more effective for dealing with the growing complexity of practical problems. The importance of integrating the complementary realms of optimization and simulation assures that future advances will have a high impact on real world applications

REFERENCES

Glover, F. 1977. Heuristics for integer programming using surrogate constraints. *Decision Sciences* 8:156-166.

Glover, F. 1994. Genetic algorithms and scatter search: unsuspected potentials. *Statistics and Computing* 4:131-140.

Glover, F. 1996. Tabu search and adaptive memory programming—advances, applications and challenges. To appear in *Interfaces in Computer Science and Operations Research,* eds. Barr, Helgason and Kennington, Kluwer Academic Publishers.

Glover, F. and M. Laguna. 1993. Tabu search. In *Modern Heuristic Techniques for Combinatorial Optimization*, ed. C. Reeves, 60-150. Blackwell Publishers: Oxford.

Holland, J. H. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor.

OptQuest 96 User's Guide—The Micro Saint Simulation Optimizer. Optimization Technologies, Inc. Boulder, Colorado.

AUTHOR BIOGRAPHIES

FRED GLOVER is the U S West Chaired Professor in Systems Science at the University of Colorado, Boulder. He has authored or co-authored more than two hundred published articles in the fields of mathematical optimization, computer science and artificial intelligence, with particular emphasis on practical applications in industry and government. In addition to holding editorial posts for journals in the U.S. and abroad, Dr. Glover has been featured as a National Visiting Lecturer by the Institute for Operations Research and the Management Sciences and has served as a host and lecturer in the U.S. National Academy of Sciences Program of Scientific Exchange.

JAMES P. KELLY is an Assistant Professor of Management Science in the College of Business and Administration and Graduate School of Business Administration at the University of Colorado in Boulder.

He received his bachelors and masters degrees in Chemical Engineering from Bucknell University in 1981. He worked four years as a systems analyst and project manager at Texas Instruments in Dallas, Texas. He returned to graduate school and received his doctoral degree in Applied Mathematics and Operations Research from the University of Maryland in 1990. While at the University of Maryland, his research into methods for providing statistical confidentiality was funded by a series of grants from the United States Bureau of the Census. His current research interests are in the area of heuristic combinatorial optimization and applied artificial intelligence. Dr. Kelly has published several papers on topics such as tabu search and simulated annealing in various journals such as *Operations Research, European Journal of Operational Research,* and the *INFORMS Journal on Computing*. He is a member of INFORMS. He is an associate editor for the *INFORMS Journal on Computing* and an editor for the *Journal of Heuristics.*

MANUEL LAGUNA is an Assistant Professor of Operations Management in the College of Business and Administration and Graduate School of Business Administration of the University of Colorado at Boulder. He received master's and doctoral degrees in Operations Research and Industrial Engineering from the University of Texas at Austin. He was the first U S WEST postdoctoral fellow in the Graduate School of Business at the University of Colorado. He has done extensive research in the interface between artificial intelligence and operations research to develop solution methods for problems in the areas of production planning and inventory control, routing and network design in telecommunications, and automated drawing. Dr. Laguna co-edited the Tabu Search volume of *Annals of Operations Research*, and he is currently editor of the *Journal of Heuristics* and *Combinatorial Optimization: Theory and Practice.* He is a member of the Institute for Operations Research and the Management Sciences, the Institute of Industrial Engineers, and the International Honor Society Omega Rho.