# TUTORIAL: HEURISTIC OPTIMIZATION

Ronald L. Rardin
School of Industrial Engineering, Purdue University
West Lafayette, Indiana 47907-1287, U.S.A.

## ABSTRACT

Although it is seriously under-represented in most academic programs, heuristic optimization—optimum-seeking methods explicitly aimed at good feasible solutions that may not be optimal—comprises most of the optimization work actually applied in industrial engineering practice. This tutorial surveys such optimization-based strategies for approximate integer and combinatorial optimization with emphasis on relatively recent developments such as tabu search, simulated annealing and genetic algorithms.

## INTRODUCTION

*Heuristic optimization*[1] encompasses the variety of optimum-seeking methods explicitly addressed to finding good feasible solutions that may not be optimal. Like exact optimization it operates within a formal framework of decisions, constraints and objective functions, but the solution method need not be just a truncated variant of an exact procedure.

Some would say juxtaposing the words "heuristic" and "optimization" produces a contradiction in terms; one is inherently approximate and the other restricted to mathematical optima. However, the overwhelming majority of optimization work that is actually applied in industrial engineering practice is implicitly or explicitly heuristic. The models required are usually just too large and complex to be solved to a provably optimal solution.

Nearly every successful heuristic takes some advantage of domain-specific information about the problem form being optimized. Still, there are common themes and patterns sufficiently general to be viewed as formal methodologies.

This paper provides a brief overview of some of the main heuristic optimization strategies for discrete (integer and combinatorial) optimization models. Emphasis is on relatively recent variations on improving search including tabu search, simulated annealing and genetic algorithms (all covered in Reeves [1993]). The development is strongly based on the outline of a graduate course in heuristic optimization which was introduced at Purdue in 1987 by the author and has been taught several times since with considerable success.

[1] Presented to the Industrial Engineering Research Conference, Nashville, May 1995

## RELAXATION

The one heuristic optimization strategy we will investigate that is explicitly grounded in methods of exact optimization can be termed *relaxation*. We first formulate the problem as an Integer Linear Program (ILP)

$$\begin{array}{lll} \min/\max & \sum_j c_j x_j + \sum_k d_k y_k & \\ \text{s.t.} & \sum_j p_{i,j} x_j + \sum_k q_{i,k} y_k \geq b_i & \text{all } i \\ & x_j \geq 0 & \text{all } j \\ & y_k \geq 0, \text{integer/0-1} & \text{all } k \end{array}$$

(sometimes mildly nonlinear), then form and solve an easier, relaxed version. The relaxation may be the linear program (LP) obtained by dropping integrality requirements, or a Lagrangian relaxation formed by rolling part of the main constraints into the objective function. An approximate optimum is produced by rounding the relaxation optimum in some systematic way to a nearby solution that is feasible in the full model.

Rounding can be very difficult, but it is surprisingly easy in many familiar model forms:

- *Shift scheduling* uses models of the form

$$\begin{array}{lll} \min & \sum_j c_j y_j & \\ \text{s.t.} & \sum_j a_{i,j} y_j \geq b_i & \text{all } i \\ & y_j \geq 0, \text{ integer} & \text{all } j \end{array}$$

  to choose a collection of work shifts covering needed activities over time. With coefficients $a_{i,j}$, which show the amount of $i$ coverage provided by shift pattern $j$, always nonnegative, we may round "up" the LP optimum as $\lceil \bar{y}_j \rceil$ to obtain an feasible integer solution.

- *Capital budgeting* employs similar models

$$\begin{array}{lll} \max & \sum_j c_j y_j & \\ \text{s.t.} & \sum_j a_{i,j} y_j \le b_i & \text{all } i \\ & y_j = 0 \text{ or } 1 & \text{all } j \end{array}$$

  to select a portfolio of investments $j$ subject to budget limitations. Again, resource requirements $a_{i,j} \ge 0$, and we may round "down" the LP optimum as $\lfloor \bar{y}_j \rfloor$ to obtain a heuristic optimum.

- *Fixed charge* models add binary variables $y_j$ to turn on and off fixed/setup costs on continuous variables $x_j$:

$$\begin{array}{lll} \min & \sum_j c_j x_j + \sum_j f_j y_j & \\ \text{s.t.} & \sum_j a_{i,j} x_j \ge b_i & \text{all } i \\ & 0 \le x_j \le u_j y_j & \text{all } j \\ & y_j = 0 \text{ or } 1 & \text{all } j \end{array}$$

  Here, we may obtain an approximate optimum by simply paying the full fixed charge on any $x_j > 0$ in an LP relaxation optimum, i.e. $\lceil \bar{y}_j \rceil$.

- *Lagrangian relaxations* usually partition a problem into parts which are solved separately over subsets of the decision variables. A "rounded" optimum can often be obtained by fixing one set of the decision variables at their relaxation-optimal values and re-optimizing the others with relaxed constraints reimposed.

Relaxation heuristics produce very good solutions in many applications. Still, their applicability depends on the existence of an ILP (or INLP) formulation with a strong relaxation and convenient rounding. When the application involves important details too intricate to model, or known relaxations produce solutions that do not resemble integer-feasible ones, or the objective function is nonlinear in the integer variables, or rounding is too difficult, we must look to other methods. Even if there is a suitable formulation, the size of real-world instances may preclude the use of relaxations. The LP or Lagrangian forms may themselves be too large for effective solution.

## DECOMPOSITION

Many times what makes an optimization model too difficult to approach by exact means is the interplay of more than one decision task: location and allocation, loading and routing, etc. Each may be manageable by itself, but it may be difficult to represent all considerations in a single model.

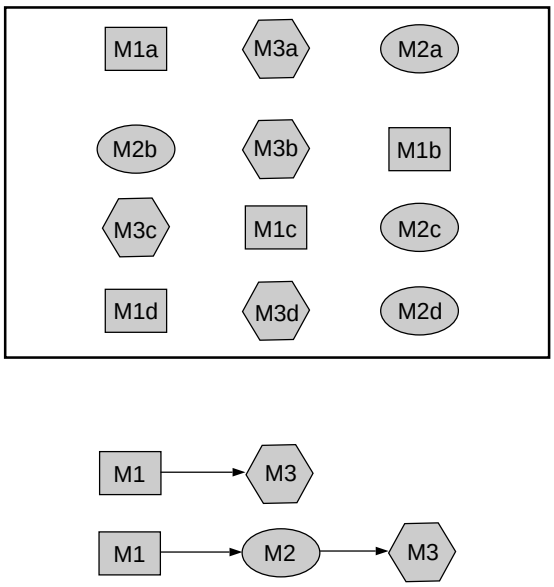


**FIGURE 1 Factory Layout Example**

Such problems invite a *decomposition* heuristic strategy. That is, we divide solution into two or more tasks and solve each separately. Some coordination mechanism must then be imposed to make solutions to the parts consistent in a heuristic optimum.

## ITERATION

The most straight-forward decomposition strategy for producing good feasible solutions can be termed *iteration*. We partition the task into separate subproblems and iterate between optimizations over one set of decision variables subject to constraints implied by fixing others at their best known values.

The factory layout task in Venkatadri, Rardin and Montreuil [1994] provides an example. As indicated in Figure 1, we wish to arrange machines on a factory floor to process known job sequences at minimum total flow-distance. What makes the problem unusually complex is that we have several replicates (copies) of each machine type, yet routings indicate only the sequence of machine types required. We must know how work is assigned to individual replicates in order to properly arrange them in the factory, but we need to know their locations to make a good assignment.

Figure 2 illustrates an iteration strategy for producing a heuristic optimum. We choose an initial layout, and

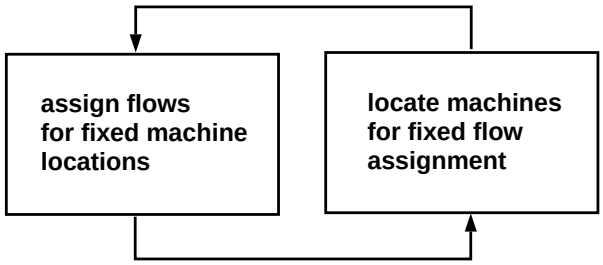


**FIGURE 2 Iteration Strategy for Layout**

then iterate between assigning flows to replicates assuming the current layout vs. revising the layout assuming the current assignment of replicate flows. There is no guarantee of convergence, but experience shows that a good solution often emerges after a few rounds.

## COLUMN GENERATION

*Column generation* is another decomposition strategy best suited to cases with some elements too intricate to represent in a mathematical program. Instead, we proceed as in Figure 3, to decompose the task into one of finding columns or components of a solution that satisfy all the intricacies, together with another selecting an optimal collection of such components subject to shared constraints.

Figure 4 depicts a classic application (see for example Anbil, Gelman, Patty and Tanga [1991]). An airline must organize or "pair" flights into closed sequences that will be staffed by a particular flight crew. For example a crew might originate at CHI in Figure 4, staff flight 203

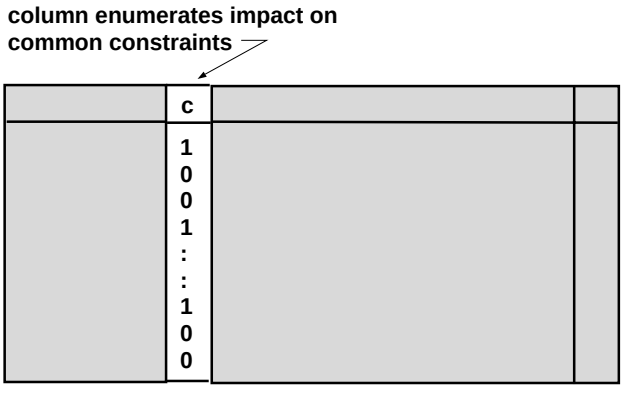


**FIGURE 3 Column Generation**

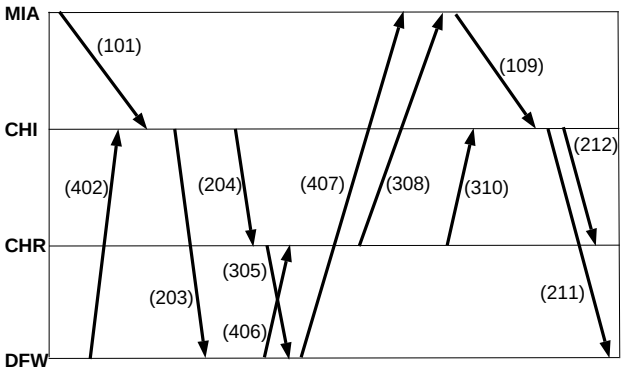


**FIGURE 4 Flight Crew Scheduling Example**

to DFW, then cover flight 407 to MIA, before returning to CHI with flight 109. We wish to staff all flights at minimum total pay and travel expense.

Besides involving an enormous number of crews and flights, this problem is difficult because crew schedules are subject to a great many union and regulatory rules. It would be nearly impossible to represent all such considerations in a single mathematical.

The column generation approach to crew scheduling follows the format of Figure 3. Special programs enumerate and evaluate the cost of possible pairings that meet all constraints. These become columns of the linking program, with $a_{ij} = 1$ when flight $i$ is covered by pairing $j$, and $= 0$ otherwise. What remains is an ILP over the columns to select a minimum cost collection of pairings that covers all flights.

Solution iterates between solving (at least the LP relaxation of) the linking problem and generating new columns. Dual variable values from the main problem may guide the construction of promising new columns, but any convenient method can be used.

## CONSTRUCTIVE SEARCH

Perhaps the most familiar heuristic optimization strategy to industrial engineers is *constructive search*. Starting from a null solution, we make decisions one by one until a full solution has been constructed. Each decision is made in a *greedy* or *myopic* way, doing the best we can with the information available from what has already been decided.

Constructive heuristics are obviously limited by the sequence dependence of this decision process. That is, an

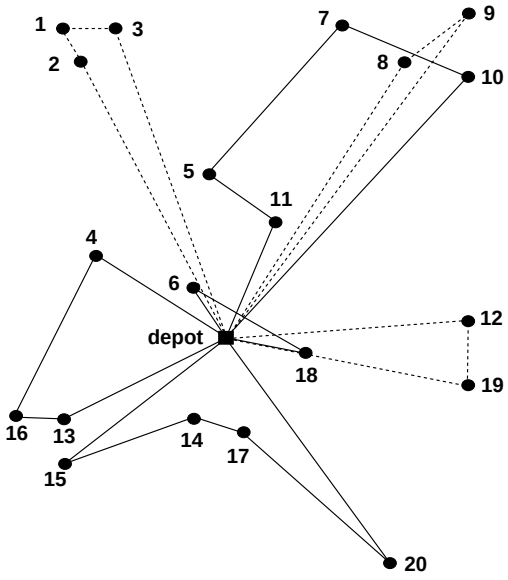


**FIGURE 5 Truck Routing Example**

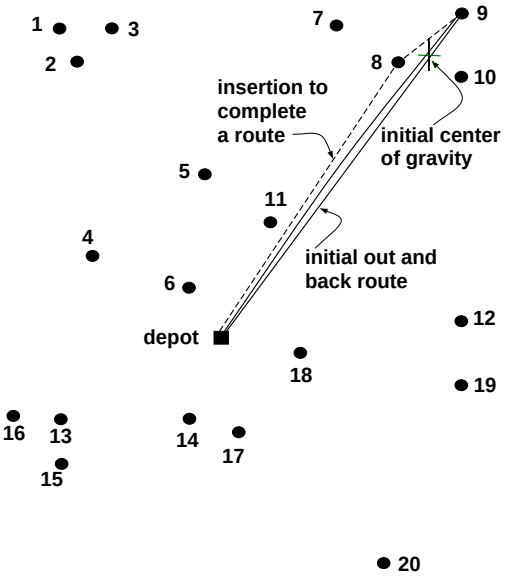


**FIGURE 6 Constructive Routing Heuristic**

early bad decision can seriously degrade later ones. *Backtracking* is sometimes added to allow undoing an early decision and repeating the constructive search. However, backtracking quickly becomes explosively time consuming if the model is large.

Constructive heuristics are routinely employed in a variety of applications including machine scheduling, capital budgeting, and many more. We will illustrate with the truck routing problem of Figure 5 (see for example Chung and Norback [1991]). A given set of stops with known shipment loads must be organized into a collection of routes like those depicted in the figure.

Figure 6 sketches a constructive heuristic for the problem. Route generation begins with one or more "seed" routes running from the depot to a distant stop and returning. Each round of the heuristic then assigns an unallocated stop to a route based on its distance from a computed center of gravity of current points in the route. Limits on truck capacity may also be considered. The idea is to collect stops in natural clusters that can be efficiently routed, but there is no guarantee. Computation terminates when all stops have been assigned.

## IMPROVING SEARCH

In contrast to the one by one constructive approach, *improving search* works with full solutions. Search begins at one or more feasible choices for all decision variables. Each step of the algorithm then considers a defined set of *moves* or modifications to the current collection. Moves producing the greatest improvement are adopted, and the search continues.

One issue in the design of an improving heuristic is the choice of the starting solution. Sometimes the search is started at a random solution. Other applications employ a constructive heuristic to obtain the first full solution.

Another concern is the selection of an appropriate move set, or equivalently, the definition the *neighborhood* composed of all solutions reachable from the current in a single move. A good choice usually takes advantage of whatever domain-specific knowledge may be available about the specific application at hand.

The trivial maximizing example of Figure 7 illustrates the dilemma that must be confronted in choosing a neighborhood. Neighborhood 1 allows only moves to the points immediately above, below, left or right of the current solution. None yields a better objective value than the current 100. Neighborhood 2 includes the 4 diagonal moves to produce a richer range of alternatives. Now there is an improving point with value 120.

Real applications involve much more complex choices of move sets, but the issue is the same. If a neighborhood is too large, it cannot be searched efficiently at each step. If it is too small, the search is unlikely to uncover a good enough solution.

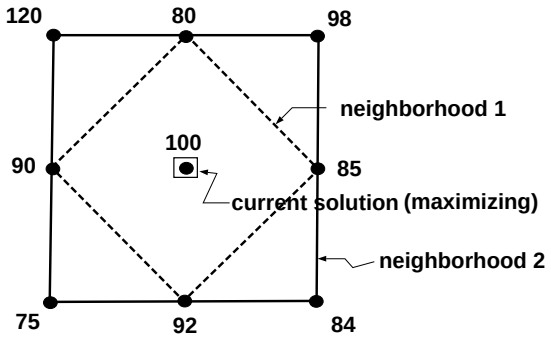


**FIGURE 7 Alternative Neighborhoods**

Another consideration is feasibility. All moves from a current feasible solution would ideally lead to feasible alternatives. However, limiting moves in this way may exclude most of the simplest to implement. Instead, improving search schemes often allow infeasibility, but penalize it in the objective function.

Our survey of improving search strategies will draw upon a real, but easy to understand example (see for example Horan and Coates [1990]). Each term universities must formulate a timetable or schedule of final examinations like the one illustrated in Figure 8. At Purdue, one of $m = 30$ exam periods must be chosen for each of over $n = 2,000$ sections or groups of sections taking a common final.

Costs arise from "conflicts". If two sections have exams scheduled at the same hour, then a common instructor and any students enrolled in both sections experience an inconvenience. We seek a choice of period for each exam that minimizes conflicts.

Our illustrations will begin from a randomly chosen schedule and employ a very simple move set. Each move changes the exam period of a single class. For example in Figure 8, one move would reassign IE101 to period 2.

| Per 1 | Per 2 | $\cdots$ | Per m |
|---|---|---|---|
| IE 101 | IE 314 | $\cdots$ | IE 414 |
| IE 121 | MA 212 | $\cdots$ | EE 201 |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |

**FIGURE 8 Exam Scheduling Example**

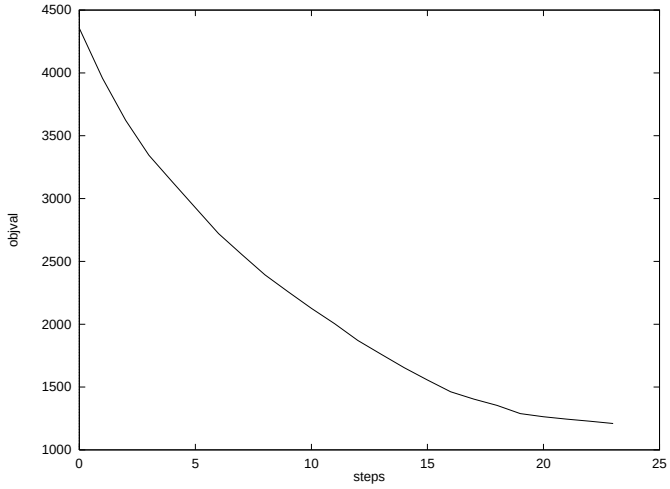


**FIGURE 9 Local Search of Example**

## LOCAL SEARCH

The simplest implementation of the improving search heuristic strategy is called *local search* or *hillclimbing* (see for example Parker and Rardin [1988, Chapter 7]). Starting from any initial feasible solution, each iteration considers all solutions in the neighborhood of the current. While any is feasible and better in the objective, the search advances to the best and repeats. If no neighbor is feasible and improving, we terminate with a *local optimum*.

Figure 9 shows the evolution of such a local search of our exam scheduling example. The 23 steps produce better and better solutions until a local optimum is encountered with 1,210 conflicts. No single exam move can now produce an improvement, so the local search terminates.

Many variations can improve the efficiency of straightforward local search. Only parts of the neighborhood may be searched at each step, or different move sets may be used at different times.

## MULTISTART

The *multistart* variation on local search takes advantage of the fact that the local optimum produced depends on where the search started. Several different starting points are employed, with each carried to a local solution. The best of these results is our heuristic optimum.

Figure 10 illustrates the evolution of multistart on our exam example. Three different starts lead to local op-

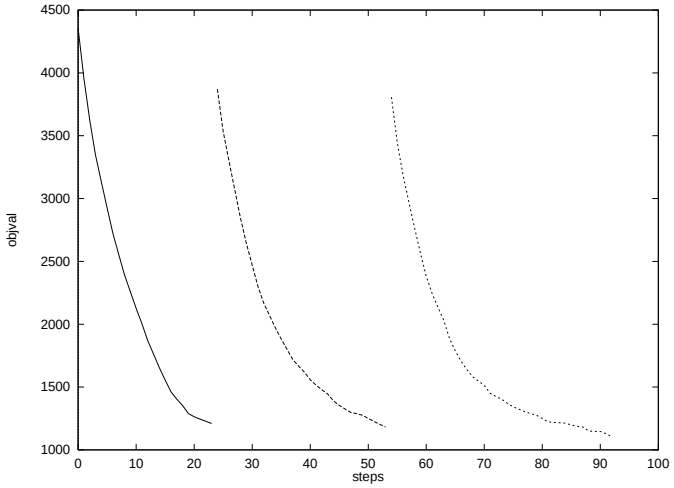


**FIGURE 10 Multistart Search of Example**

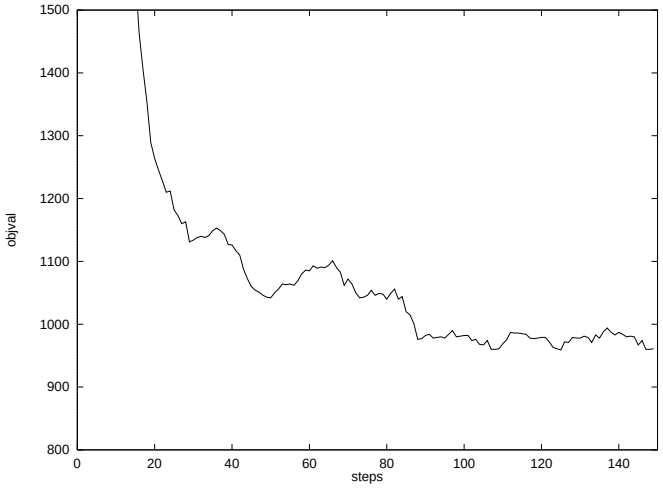


**FIGURE 11 Tabu Search of Example**

tima of value 1,210, 1,182 and 1,105. The best would be reported as an approximate optimum.

## TABU SEARCH

Another obvious extension of the local search idea is to allow nonimproving moves. For example, with neighborhood 1 in Figure 7, we could escape the local optimum by moving to the best neighbor with value 92 even though it is not improving.

There is an immediate difficulty. The only improving neighbor of the new (value 92) current point may be the previous one (value 100). Thus, simply moving to the best neighbor will usually lead to infinite cycling.

*Tabu search* (see Glover and Laguna [1993]) is a variant of local improvement that prevents cycling by temporarily forbidding (hence the name tabu) some moves including any that would return us to the point just visited. Each iteration takes the best non-tabu move, whether or not it improves. The best solution encountered within a specified iteration limit is reported as a heuristic optimum.

Much of the design of a tabu search revolves around the policy for forbidding moves. If too many moves are forbidden, the search will make little progress. If too few are restricted, the search will tend to remain in a local area, and thus not pass through a diverse enough set of points to produce a good heuristic optimum.

In our exam scheduling example, these considerations lead to a tabu policy forbidding a move of the exam just reassigned for 5 steps. Typically for tabu implementations, this policy includes more than just the move undoing the most recent choice. Experience shows that a wider class of moves should be restricted. Notice also that the restriction is temporary. After 5 steps the forbidden moves will be reactivated.

Figure 11 shows the evolution of a tabu search using this rule. The best solution, with 959 conflicts, was encountered at step 107 of a search allowed to run 150 iterations.

The plot in Figure 11 shows a general downward trend reflecting the underlying steepest descent structure of the tabu search. Notice, however, that occasional nonimproving moves ultimately led to a better result than pure hillclimbing Figure 9.

Many refinements have been tried in different applications of tabu search. It is customary to override the tabu status of a move if it would lead to an unusually good solution (a feature termed *aspiration*). Other implementations keep several lists of tabu moves, search only part of the neighborhood at each iteration, bias the search towards moves that have proved especially helpful, or employ multiple starting solutions.

Nearly all effective tabu algorithms also have some mechanism for balancing the diversification of the search over the feasible set vs. intensification in a promising region. By tightening the tabu policy, or favoring underutilized moves, the search can be guided into new parts of the space. A later stage can then focus the search in

a more limited region in order to discover its best local optimum.

## SIMULATED ANNEALING

*Simulated annealing* is another popular variant of improving search (see for example Aarts and Korst [1989]). It uses randomness to avoid cycling on nonimproving moves.

Like all improving searches, it begins with a feasible solution and a defined set of moves. At each step, however, a move is chosen randomly. If the move improves, it is immediately adopted. If the move does not improve, it is accepted with probability

$$e^{-degradation/temperature} \quad (1)$$

and otherwise a new move is chosen and the process repeated. The best solution encountered within a specified iteration limit is reported as a heuristic optimum.

The "degradation" in the exponent of probability (1) measures how much the test solution worsens the objective function value (usually expressed as the absolute difference of the current and test point values). Thus a solution that greatly degrades the objective value has some chance to be accepted, but one with a less dramatic worsening has a higher probability (less negative exponent).

The *temperature* in simulated annealing controls its randomness. A very high temperature in expression (1) makes the exponent nearly zero, so that almost all moves are accepted. A temperature approaching zero renders the exponent highly negative for any degradation at all.

Practical implementations of simulated annealing begin with the temperature relatively high and reduce it toward zero over the life of the search. The idea is to allow the search to wander randomly in its early stages, focusing only after it has discovered a particularly promising region. A typical pattern begins at a given temperature $T_0$ and multiplies by a factor $\alpha \in (0, 1)$ after each $k$ steps.

Figure 12 shows the result of such a policy on our exam scheduling example. Initially, the search jumps around dramatically. As the temperature declines, however, the search performs much more like pure hillclimbing. The best solution (value 933) was encountered after about 2,700 moves of 3,000 tried.

Notice that the number of steps allowed in simulated annealing Figure 12 is far greater that the earlier approaches. This is typical, and reflects two considerations.

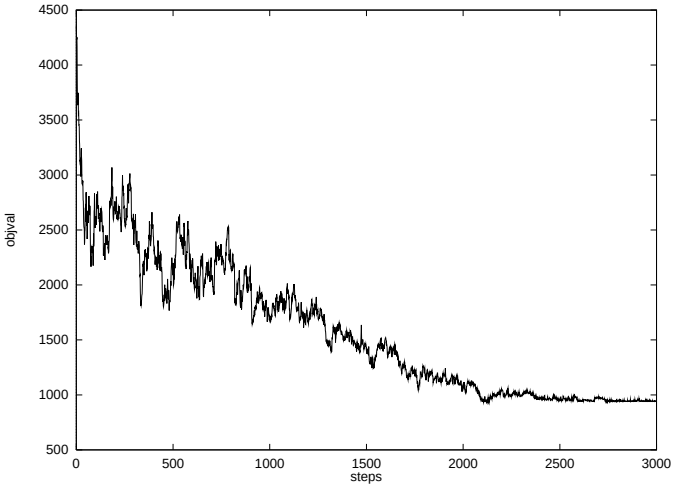


**FIGURE 12 Simulated Annealing on Example**

First, the cost per step is usually much lower than hillclimbing or tabu. There may be a few rejections, but a typical simulated annealing step requires the evaluation of only a small number of neighbors. By contrast local search and tabu methods usually test all or much of the neighborhood at each step.

The other consideration is pragmatic. Experience in most applications shows that a fairly large number of steps will be required if the highly random simulated annealing strategy is to yield a good heuristic optimum.

Like all the other heuristic methods, specific implementations of simulated annealing may introduce a host of variations on the main strategy. Particular schemes may raise and lower the temperature more than once, or vary the move sets as the search evolves, or start from several initial solutions.

Another appeal of simulated annealing is a theoretical convergence property. If a sequence of the allowed moves can reach any solution from any other, and temperature is allowed to approach zero, simple Markov chain analysis can show that simulated annealing will come upon an exact optimal solution in a finite number of steps with probability one. It is important to realize, however, that the real number of steps required may be hopelessly vast.

## GENETIC ALGORITHMS

The most different of the recent variations on local improvement attempts to mimic the biological evolution process for discovering good solutions. Such *genetic algo-*

| Solution | Objval | Probability |
|---|---|---|
| (0,1,1,0,0) | 40 | 0.20 |
| (1,1,0,1,0) | 60 | 0.30 |
| (0,1,1,1,0) | 100 | 0.50 |
| total | 200 | 1.00 |

**FIGURE 13 Population Example**

*rithms* (see for example Goldberg [1989]) maintain a collection or *population* of solutions throughout the search. One or more pairs of members are chosen randomly at each iteration with a bias toward those with best objective values. Then the chosen pairs breed by exchanging parts of their content in an operation called *crossover*. The best of the descendants form the next population, and the process continues. On occasion one or more of the solutions in the population may also be randomly *mutated* by changing a single component.

To illustrate how breeding pairs are selected, consider the 3-member population of solutions for a maximize problem in Figure 13. The common computation depicted first sums the objective values of solutions in the population. Any individual is then selected for breeding with probability

$$\frac{(individual\ solution\ value)}{(population\ total)} \qquad (2)$$

The idea is to allow all individuals some chance, but favor those with best objective values.

The main move form in genetic algorithms is crossover. Figure 14 illustrates. A cut point is randomly chosen within the breeding pair of solutions. Descendants are formed by combining the initial components of the first solution with the last components of the second and vice versa.

In theory such genetic algorithms are domain independent because they require no explicit notion of a neighborhood. However, this can be a serious disadvantage. There is no guarantee that crossover operations produce anything like a feasible solution. Thus practical implementations of genetic algorithms tend to become deeply involved in schemes for pricing infeasibility.

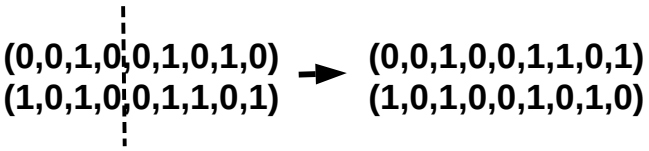


**FIGURE 14 Crossover Example**

A related issue concerns the arrangement of decision variables along solution vectors. A pair of components located next to each other will tend to be unchanged as crossover proceeds. If the same components are located at opposite ends of the solution vector, almost every step will change their mix. Thus domain specific information will almost certainly be used to sequence the solution vector of the genetic algorithm.

A final issue concerns the sampling scheme illustrated in Figure 13 and probability (2). The computation makes sense if we are maximizing the objective function, but not if we are minimizing. It is natural to weight minimize solutions on the basis of some complement

$$(constant) - (individual\ solution\ value)$$

Still, probabilities will change with the selected constant.

## CONCLUSIONS

Which heuristic optimization strategy is best for any application obviously varies with the environment. Still, this brief overview shows there are a wide range of possibilities that deserve consideration. A case admitting strong LP or Lagrangian formulations is a good candidate for the relaxation approach. Problems where much, but not all of the structure can be modeled as a mathematical program invite decomposition. If time or problem size permits generation of only a single solution, constructive algorithms are likely to be preferred. When natural solution neighborhoods are available, one of the improving search implementations may be most appropriate. Within the improving category, tabu search will work better when its steepest descent perspective is supported by strong domain knowledge, while simulated annealing or genetic algorithms might do better with a less well defined setting.

All the methods are also open to creative combination. Relaxation or improving search may be used in some parts of a decomposition strategy. A constructive search may provide the starting solution for tabu or simulated annealing. Best features of one of the variations on improving search can usually be adapted to improve another.

All these combinations mean there is room for much more heuristic optimization research. Systematic presentations like this one will also hopefully lead to more treatment of the subject in academic programs.

## REFERENCES


Aarts, E. and J. Korst [1989], *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*, John Wiley.

Anbil, E., E. Gelman, B. Patty and R. Tanga [1991], "Recent Advances in Crew-Pairing Optimization at American Airlines," *Interfaces, 21:1*, 62-74.

Chung, H.K. and J.B. Norback [1991], "A Clustering and Insertion Heuristic Applied to a Large Routing Problem in Food Distribution," *Journal of the Operational Research Society, 42*, 555-564.

Glover, F. and M. Laguna [1993], in C. Reeves editor, *Modern Heuristic Techniques for Combinatorial Problems,* John Wiley.

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

Horan, C.J. and W.D. Coates [1990], "Using More Than ESP to Schedule Final Exams: Purdue's Examinations Scheduling Procedure II (ESP II), *College and University Computer Users Conference Proceedings, 35*, 133-142.

Parker, R.G. and R.L. Rardin [1988], *Discrete Optimization*, Academic Press.

Reeves, C. [1993], editor, *Modern Heuristic Techniques for Combinatorial Problems,* John Wiley.

Venkatadri, U., R.L. Rardin and B. Montreuil [1994], "Fractal Layout of Factories: A Design Optimization Methodology," presentation to the ORSA/TIMS Detroit meeting, October.


## BIOGRAPHICAL SKETCH


RONALD L. RARDIN is a Professor in the School of Industrial Engineering at Purdue University. He joined Purdue in 1982 after nine years on the faculty of the School of Industrial and Systems Engineering at the Georgia Institute of Technology. Dr. Rardin's teaching and research interests center on large-scale discrete optimization—especially network design. He is co-author of numerous journal articles in that field and a comprehensive book, *Discrete Optimization*. He is presently nearing completion of an undergraduate text on mathematical programming entitled *Modeling and Analysis in Operations Research*.