GLOBAL OPTIMIZATION AND META-HEURISTICS

Manuel Laguna, College of Business, University of Colorado at Boulder, USA

Keywords: optimization, heuristic search, meta-heuristics

Short Contents List

Glossary

Meta-heuristic — A master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality.

Linear programming — An optimization methodology that consists of formulating a problem as a linear function to be maximized or minimized and a set of linear inequalities that represent the constraints.

Integer programming — A methodology related to linear programming with the additional restriction that the decision variables can only take on integer values.

Neighborhood — The set of solutions that can be reached from a given solution by way of applying a move mechanism.

Move — A change that transforms the current solution into a neighbor solution.

Crossover — A move mechanism used in genetic algorithms.

Summary

This article describes the origin and significant developments associated with the field of meta-heuristics as they relate to global optimization. Meta-heuristics provide a means for approximately solving complex optimization problems. These methods are designed to search for global optima, however, they cannot guarantee that the best solution found after termination criteria are satisfied is indeed and global optimal solution to the problem. Experimental testing of meta-heuristic implementations show that the search strategies embedded in such procedures are capable of finding solutions of high quality to hard problems in industry, business and science.

1. Introduction

The theory of optimization refers to the quantitative study of optima and the methods for finding them. The technical verb *optimize* means to achieve the optimum and optimization is the act of optimizing. To achieve the optimum entails in some cases to obtain the most of some measure of success (e.g., revenue) or in some other cases to obtain the least of another measure (e.g., cost). Choosing a quantitative measure of effectiveness and then optimizing it has become the typical way

in which many important decisions are made. Decisions involving how to design, build or operate a physical or economics system are reached in three steps:

1. Identify the decision variables in the system and determine, accurately and qualitatively, how they interact.
2. Identify a measure of system effectiveness that can be expressed in terms of the system variables. This measure is often referred to as the objective function.
3. Choose those values of the system variables that yield optimum effectiveness.

In classical optimization methods, such as linear programming, these three steps result in a model formulation of the type:

Maximize or minimize $f(x)$
Subject to $g(x) \leq b$

In this formulation, $f(x)$ is the quantitative measure of effectiveness (or objective function) and x are the decision variables. The set of constraints is also formulated in terms of the decision variables and represented as bounds on the function $g(x)$. In the case of linear programming both $f(x)$ and $g(x)$ are linear functions. Linear programming is considered a general-purpose tool because the only requirement is to represent the optimization model as a linear objective function subject to a set of linear constraints. The state-of-the-art linear programming solvers are quite powerful and can successfully solve models with thousands and even millions of variables employing reasonable amounts of computer effort. Evidently, however, not all business, industrial and scientific 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 nonlinear. Techniques such as linear programming and its cousins (nonlinear programming and integer programming) generally require a number of simplifying assumptions about the real system to be able to properly frame the problem.

Linear programming solvers are design to exploit the structure of a well-defined and carefully studied problem. The disadvantage to the user is that in order to formulate the problem as linear program, simplifying assumptions and abstractions may be necessary. This leads to the well-known dilemma of choosing between finding the optimal solution to a model that does not represent the real system accurately and developing a model that is a good abstraction of the real system but for which only inferior sub-optimal solutions can be obtained. When dealing with the optimization of complex systems, a course of action taking for many years has been to develop specialized heuristic procedures that, in general, do not require a mathematical formulation of the problem. These procedures were appealing from the standpoint of simplicity, but generally lacked the power to provide high quality solutions to complex problems.

For example, consider the well-known traveling salesman problem. This is the problem of finding the shortest route that visits each of a given collection of locations precisely once and eventually returns to the starting point. One simple heuristic for this problem may be to arbitrarily select a starting location and then always choose from a candidate list the next location that is closest to the current location. Once a location is chosen, it is deleted from the candidate list of unvisited locations. While this heuristic may occasionally give acceptable results in problems that consist of only a hand-full of locations, in general, its performance is predicted to be extremely poor. This procedure falls within the class of heuristics called myopic, because they make decisions based on limited (also called “local”) information without considering the consequences of implementing those decisions. In the case of the traveling salesman problem, choosing the nearest city at every point of the way may build a route that systematically moves the salesman farther from the starting point, making the return trip unnecessarily long.

In addition to heuristics designed to construct solutions, there are also procedures for improving solutions. The most common way of improving a solution is via the application of a *local search.* To continue with our traveling salesman illustration, suppose that a delivery truck driver is asked to visit five locations in the following order:

$$\text{Warehouse} \rightarrow \text{A} \rightarrow \text{B} \rightarrow \text{C} \rightarrow \text{D} \rightarrow \text{E} \rightarrow \text{Warehouse}$$

Also suppose that the route was constructed using the "nearest location" rule discussed above. A local search procedure would attempt to modify the current route by performing a *move* (or change). One possible move is to exchange the position of two locations and measure the impact on the objective function. We assume that the objective is to minimize the total length of the route, so typically a move that decreases the length of the route is considered "better, in the local sense, than one that increases it. If we limit our local search to moves that exchange locations that immediately follow each other in the current route, then we only have to test 4 moves, i.e., (A,B), (B,C), (C,D), and (D,E). We would pick the "best" one of those moves. However, if we would like to test all possible exchanges of two locations as part of the local search effort, the number of moves to be examined is 10. The amount of exploration, which is directly related to the amount of computational effort, is an important design issue in local search procedures. The effort to explore the neighborhood of a solution (that is the set of solutions reachable from the current solution by applying a move mechanism) can vary considerably. In a traveling salesman problem with n locations, there are n-1 neighbors if the move is defined as exchanging the positions of two locations that immediately follow each other in the current solution. However, if the move is defined as the exchange of positions of any two locations, the size of the neighborhood (i.e., the cardinality of the set of solutions reachable with such a move) is $(n^2-n)/2$. Regardless of the move mechanism, local search typically explores only a small fraction of the solution space. In the case of the traveling salesman problem, for example, the solution space consists of $n!$ solutions. So, local search procedures that explore in the order of n^2 or even n^3 solutions are only dealing with a fairly small fraction of the entire solution space as the dimension of the problem increases.

Heuristics designed for constructing solutions are typically combined with local search procedures to create what is called a *hill climbing* method. These methods start from a solution and apply a local search in an attempt to find an improved solution. If an improved solution is found, the search moves to it and the local search is applied again. The method stops when the local search is not capable of finding a solution that improves upon the current solution, i.e., when the "best" possible move cannot improve upon the objective function value of the current solution. The hill climbing terminology refers to the trajectory of the objective function values in a maximization problem. The steps in a hill climbing procedure can be summarized as follows:

1. Generate an initial solution s
2. Apply a local search to find the best neighbor solution s'
3. If s' is better than s then make the current solution s equal to s' and go to step 2, else terminate

The main shortcoming of a hill-climbing method is its inability to escape local optimality. Figure 1 shows a nonlinear function with a single variable for which a hill-climbing method would be able to find the local maximum only if the initial solution happens to fall within the range (b, c). However, since the range (a, b) is considerably wider, a hill climbing method that starts from a randomly generated initial point would likely be "trapped" in the inferior local optimal point x' instead of finding the global maximum point x^*.

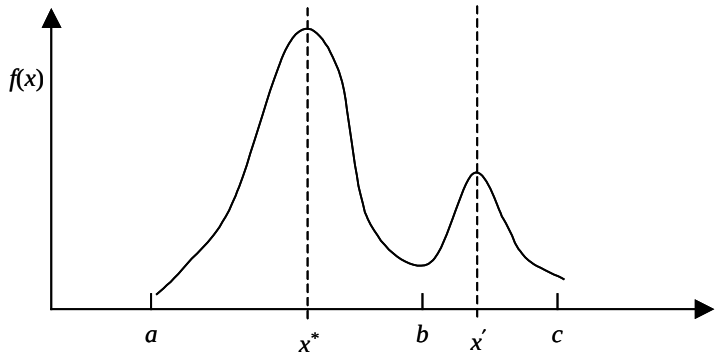

Figure 1 Bimodal function

Meta-heuristics provided a way of considerably improving the performance of simple heuristic procedures, such as those based on hill climbing. The search strategies proposed by meta-heuristic methodologies result in iterative procedures with the ability to escape local optimal points. Meta-heuristics have been developed to solve complex optimization problems in many areas, with combinatorial optimization being one of the most fruitful. Generally, the best procedures achieve their efficiencies by relying on context information. The solution method can be viewed as the result of adapting meta-heuristic strategies to specific optimization problems.

The term *meta-heuristic* (also written *metaheuristic*) was coined by Fred Glover in 1986 and has come to be widely applied in the literature, both in the titles of comparative studies and in the titles of volumes of collected research papers. A meta-heuristic refers to a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality. The heuristics guided by such a meta-strategy may be high level procedures or may embody nothing more than a description of available moves for transforming one solution into another, together with an associated evaluation rule.

The contrast between the meta-heuristic orientation and the "local optimality" orientation is significant. For many years, the primary conception of a heuristic procedure (a conception still prevalent today) was to envision either a clever rule of thumb or an iterative rule that terminates as soon as no solutions immediately accessible could improve the last one found. Such iterative heuristics are often referred to as descent methods, ascent methods, or local search methods. (A sign of the times is that "local search" now sometimes refers to search that is not limited to being local in character.) Consequently, the emergence of methods that departed from this classical design — and that did so by means of an organized master design — constituted an important advance. Widespread awareness of this advance only began to dawn during the last decade, though its seeds go back much farther.

The evolution of meta-heuristics during the past ten years has taken an explosive upturn. Meta-heuristics in their modern forms are based on a variety of interpretations of what constitutes "intelligent" search. These interpretations lead to design choices that in turn can be used for classification purposes. However, a rigorous classification of different meta-heuristics is a difficult and risky enterprise, because the leading advocates of alternative methods often differ among themselves about the essential nature of the methods they espouse. This may be illustrated by considering the classification of meta-heuristics in terms of their features with respect to three basic design choices: (1) the use of adaptive memory, (2) the kind of neighborhood exploration used, and (3) the number of current solutions carried from one iteration to the next. These options can be embedded in a classification scheme of the form *x*/*y*/*z*, where the choices for *x* are A (if the meta-

heuristic employs adaptive memory) and M (if the method is "memoryless"). The choices for *y* are N (for a method that employs some systematic neighborhood search either to select the next move or to improve a given solution) and S (for those methods relying on random sampling). Finally, *z* may be 1 (if the method moves from one current solution to the next after every iteration) or P (for a population-based approach with a population of size P). This simple 3-dimensional scheme gives us a preliminary basis of classification, which discloses that agreement on the proper way to label various meta-heuristics is far from uniform. We show this by providing classifications for a few well-known meta-heuristics in Table 1.

Meta-heuristic	*Classification* 1	*Classification* 2
Genetic algorithms	M/S/P	M/N/P
Scatter search	M/N/P	A/N/P
Simulated annealing	M/S/1	M/N/1
Tabu search	A/N/1	A/N/P

Table 1. Meta-heuristic classification.

Two different ways are given for classifying each of these procedures. The first classification most closely matches the "popular conception" and the second is favored by a significant (if minority) group of researchers. The differences in these classifications occur for different reasons, depending on the method. Some differences have been present from the time the methods were first proposed, while others represent recent changes that are being introduced by a subgroup of ardent proponents. For example, the original form of simulated annealing has come to be modified by a group that believes stronger elements of neighborhood search should be incorporated. A similar change came about in genetic algorithms, a few years before it was introduced in simulated annealing, in the mid 1980s. Still, it should be pointed out that not all the advocates of simulated annealing and genetic algorithms view these changes as appropriate.

On the other hand, among those examples where different classifications were present from the start, the foundation papers for tabu search included population-based elements in the form of strategies for exploiting collections of elite solutions saved during the search. Yet a notable part of the literature has not embraced such population-based features of tabu search until recently. Similarly, scatter search was accompanied by adaptive memory elements as a result of being associated with early tabu search ideas, but this connection is likewise only beginning to be pursued.

A few proponents of simulated annealing and genetic algorithms have recently gone farther in modifying the original conceptions than indicated in Table 1, to propose the inclusion of elements of adaptive memory as embodied in tabu search. Such proposals are often described by their originators as *hybrid* methods, due to their marriage of aspects from different frameworks.

2. Meta-Heuristic Features

In addition to the three basic design elements used in the classification in section 1, meta-heuristics incorporate other strategies with the goal of guiding the search. A meta-heuristic may strategically modify the evaluation provided by a component heuristic (which normally consists of identifying the change in an objective function value produced by a move). For example, simulated annealing relies on a problem objective function to provide each evaluation, but then amends this evaluation based on the current solution. In the amended form, all improving moves are considered equally attractive, and any such move encountered is accepted. Moves that deteriorate the value of the objective function are accepted or rejected by a probabilistic criterion that initially assigns a high probability (when the temperature is high) to accepting any move generated, regardless of its

quality. However, a bias is incorporated that favors smaller deteriorating moves over larger ones, and over time this bias is increased, ultimately reducing the probability of accepting a non-improving move to zero. The set of available moves can be taken from another heuristic, but classical SA pre-empts all other move generation processes to generate moves randomly from the proposed domain.

A meta-heuristic may also modify the neighborhood of moves considered to be available, by excluding some members and introducing others. This amended neighborhood definition may itself necessitate a change in the nature of evaluation. The strategic oscillation approach of tabu search illustrates this intimate relationship between changes in neighborhood and changes in evaluation. A standard neighborhood that allows moves only among feasible solutions is enlarged by this approach to encompass infeasible solutions. The search is then strategically driven to cross the feasibility boundary to proceed into the infeasible region. After a selected depth is reached, the search changes direction to drive back toward feasibility, and upon crossing the feasibility boundary similarly continues in the direction of increased feasibility. (One-sided oscillations are employed in some variants to remain predominantly on a particular side of the boundary.) To guide these trajectories, the approach modifies customary evaluations to take account of the induced direction of movement and the region in which the movement occurs. The result generates a controlled behavior that exploits the theme of non-monotonic exploration.

The emphasis on guidance differentiates a meta-heuristic from a simple random restart procedure or a random perturbation procedure. However, sometimes these naive restarting and perturbation procedures are also classed as low-level meta-heuristics, since they allow an opportunity to find solutions that are better than a first local optimum encountered. “Noising” procedures, which introduce controlled randomized changes in parameters such as cost or resource availability coefficients, provide one of the popular mechanisms for implementing such approaches. Another popular mechanism is simply to randomly modify evaluations, or to choose randomly from evaluations that fall within a chosen window. Such randomized processes are also applied to selecting different types of moves (neighborhood definitions) at different junctures.

In contrast to an orientation that still often appears in the literature, the original conception of a meta-heuristic does not exclude consideration of constructive moves for generating initial solutions, but likewise allows these moves to be subjected to meta-heuristic guidance. (A popular orientation in the literature is to suppose that meta-heuristics are only used in connection with “transition” moves, which operate on fully constructed solutions.) From a broader perspective, a partial solution created by a constructive process is simply viewed as a solution of a particular type, and procedures for generating such solutions are natural candidates to be submitted to higher-level guidance. This view has significant consequences for the range of strategies available to a meta-heuristic approach.

Strategic oscillation again provides an illustration. By the logical restructuring theme of tabu search, constructive moves are complemented by creating associated destructive moves, allowing the oscillation to proceed constructively to (and beyond) a stipulated boundary, and then to reverse direction to proceed destructively to various depths, in alternating waves. Transition moves permit refinements at varying levels of construction and destruction.

The perspective that restricts attention only to transition moves is gradually eroding, as researchers are coming to recognize that such a restriction can inhibit the development of effective methods. However, there remain pockets where this recognition is slow to dawn. (For example, methods that alternate between construction and transition moves — affording a simple subset of options provided by strategic oscillation — have recently been characterized in a segment of the literature as a “new development.”)

Our earlier meta-heuristic classification, which differentiates between population-based strategies and adaptive memory strategies, is often taken to be a fundamental distinction in the literature. Population-based strategies manipulate a collection of solutions rather than a single solution at each stage. Such procedures are now often referred to as composing the class of *evolutionary methods.* A prominent subclass of these methods is based on strategies for "combining" solutions, as illustrated by genetic algorithms, scatter search and path relinking methods. Another prominent subclass consists of methods that are primarily driven by utilizing multiple heuristics to generate new population members. This incorporation of multiple heuristics for generating trial solutions, as opposed to relying on a single rule or decision criterion, is a very old strategy whose origins are probably not traceable. Some of the recent evolutionary literature occasionally cites work of the mid 1960s as embodiments of such ideas, but such work was clearly preceded by earlier developments. The key to differentiating the contributions of such methods obviously rests on the novelty of the component heuristics and the ingenuity of the strategies for coordinating them. Such concerns are more generally the focus of parallel processing solution methods, and many "evolutionary" contributions turn out chiefly to be a subset of the strategies that are being developed to a higher level of sophistication under the parallel processing rubric.

The adaptive memory classification provides a more precise means of differentiation, although it is not without pitfalls. From a naive standpoint, virtually all heuristics other than complete randomization induce a pattern whose present state depends on the sequence of past states, and therefore incorporate an implicit form of "memory." Given that the present is inherited from the past, the accumulation of previous choices is in a loose sense "remembered" by current choices. This sense is slightly more pronounced in the case of solution combination methods such as genetic algorithms and scatter search, where the mode of combination more clearly lends itself to transmitting features of selected past solutions to current solutions. Such an implicit memory, however, does not take a form normally viewed to be a hallmark of an intelligent memory construction. In particular, it uses no conscious design for recording the past and no purposeful manner of comparing previous states or transactions to those currently contemplated. By contrast, at an opposite end of the spectrum, procedures such as branch and bound and A* search use highly (and rigidly) structured forms of memory — forms that are organized to generate all non-dominated solution alternatives with little or no duplication.

Adaptive memory procedures, properly conceived, embody a use of memory that falls between these extremes, based on the goal of combining flexibility and ingenuity. Such methods typically seek to exploit history in a manner inspired by (but not limited to) human problem solving approaches. They are primarily represented by tabu search and its variations that sometimes receive the "adaptive memory programming" label. In recent years, as previously intimated, other approaches have undertaken to incorporate various aspects of such memory structures and strategies, typically in rudimentary form. Developments that produce hybrids of tabu search with other approaches at a more advanced level have become an important avenue for injecting adaptive memory into other methods, and constitute an active area of research.

Another distinction based on memory is introduced by neural network (NN) approaches. Such methods emphasize an associative form of memory, which has its primary application in prediction and pattern matching problems. Neural network procedures also implicitly involve a form of optimization, and in recent years such approaches have been adapted to several optimization settings. Performance is somewhat mixed, but researchers in optimization often regard neural networks as appropriate to be included within the meta-heuristic classification. Such an inclusion is reinforced by the fact that NN-based optimization approaches sometimes draw on standard heuristics, and produce solutions by transformations that are not limited to ordinary notions of local optimality. A number of initiatives have successfully combined neural networks with simulated annealing, genetic algorithms and, most recently, tabu search.

Meta-heuristics are often viewed as composed of processes that are intelligent, but in some instances the intelligence belongs more to the underlying design than to the particular character (or behavior) of the method itself. The distinction between intelligent design and intelligent behavior can be illustrated by considering present day interior point methods of linear programming. Interior point methods (and more general barrier function methods) exploit a number of ingenious insights, and are often remarkably effective for achieving the purposes for which they were devised. Yet it seems doubtful whether such methods should be labeled intelligent, in the sense of being highly responsive to varying conditions, or of changing the basis for their decisions over time as a function of multiple considerations. Similar distinctions arise in many other settings. It must be conceded that the line that demarks intelligent methods from other methods is not entirely precise. For this reason it is not necessary for a master procedure to qualify as intelligent in a highly rigorous sense in order to be granted membership in the category of meta-heuristics.

3. Brief Description of Some Meta-heuristics

In this section we briefly describe three well-known meta-heuristics: tabu search, scatter search and genetic algorithms. The descriptions focus on the main features of these methodologies and leave out implementation details that can be found in specialized research articles. Note that we have also briefly addressed simulated in section 2 as part of our discussion on meta-heuristic features.

3.1 Tabu Search

We have mentioned some of the fundamentals of tabu search (TS) in connection with our general description of meta-heuristics in the previous sections. In particular, we mentioned that tabu search is based on principles of intelligent search. The TS premise is that problem solving, in order to qualify as intelligent, must incorporate *adaptive memory* and *responsive exploration.* The adaptive memory feature of TS allows the implementation of procedures that are capable of searching the solution space economically and effectively. Since local choices are guided by information collected during the search, TS contrasts with memoryless designs that heavily rely on semirandom processes that implement a form of sampling. Examples of memoryless methods include semigreedy heuristics and the prominent “genetic” and “annealing” approaches inspired by metaphors of physics and biology. Adaptive memory also contrasts with rigid memory designs typical of branch and bound strategies. It can be argued that some types of evolutionary procedures that operate by combining solutions, such as genetic algorithms, embody a form of implicit memory.

The emphasis on responsive exploration in tabu search, whether in a deterministic or probabilistic implementation, derives from the supposition that a bad strategic choice can yield more information than a good random choice. In a system that uses memory, a bad choice based on strategy can provide useful clues about how the strategy may profitably be changed. Even in a space with significant randomness a purposeful design can be more adept at uncovering the imprint of structure.

Responsive exploration integrates the basic principles of intelligent search, i.e., exploiting good solution features while exploring new promising regions. Tabu search is concerned with finding new and more effective ways of taking advantage of the mechanisms associated with both adaptive memory and responsive exploration. The development of new designs and strategic mixes makes TS a fertile area for research and empirical study.

3.1.1 Use of Memory

The memory structures in tabu search operate by reference to four principal dimensions, consisting of recency, frequency, quality, and influence. *Recency-based* and *frequency-based* based memory complement each other. The *quality* dimension refers to the ability to differentiate the merit of solutions visited during the search. In this context, memory can be used to identify elements that are common to good solutions or to paths that lead to such solutions. Operationally, quality becomes a foundation for incentive-based learning, where inducements are provided to reinforce actions that lead to good solutions and penalties are provided to discourage actions that lead to poor solutions. The flexibility of these memory structures allows the search to be guided in a multi-objective environment, where the goodness of a particular search direction may be determined by more than one function. The tabu search concept of quality is broader than the one implicitly used by standard optimization methods.

The fourth dimension, *influence*, considers the impact of the choices made during the search, not only on quality but also on structure. Recording information about the influence of choices on particular solution elements incorporates an additional level of learning. By contrast, in branch and bound, for example, the separation rules are pre-specified and the branching directions remain fixed, once selected, at a given node of a decision tree. It is clear however that certain decisions have more influence than others as a function of the neighborhood of moves employed and the way that this neighborhood is negotiated (e.g., choices near the root of a branch and bound tree are quite influential when using a depth-first strategy). The assessment and exploitation of influence by a memory more flexible than embodied in such tree searches is an important feature of the TS framework.

The memory used in tabu search is both *explicit* and *attributive.* Explicit memory records complete solutions, typically consisting of elite solutions visited during the search. An extension of this memory records highly attractive but unexplored neighbors of elite solutions. The memorized elite solutions (or their attractive neighbors) are used to expand the local search.

Alternatively, TS uses attributive memory for guiding purposes. This type of memory records information about solution attributes that change in moving from one solution to another. For example, in a graph or network setting, attributes can consist of nodes or arcs that are added, dropped or repositioned by the moving mechanism. In production scheduling, the index of jobs may be used as attributes to inhibit or encourage the method to follow certain search directions.

3.1.2 Intensification and Diversification

Two highly important components of tabu search are intensification and diversification strategies. Intensification strategies are based on modifying choice rules to encourage move combinations and solution features historically found good. They may also initiate a return to attractive regions to search them more thoroughly. Since elite solutions must be recorded in order to examine their immediate neighborhoods, explicit memory is closely related to the implementation of intensification strategies. The main difference between intensification and diversification is that during an intensification stage the search focuses on examining neighbors of elite solutions.

Here the term “neighbors” has a broader meaning than in the usual context of “neighborhood search.” That is, in addition to considering solutions that are adjacent or close to elite solutions by means of standard move mechanisms, intensification strategies generate “neighbors” by either grafting together components of good solution or by using modified evaluation strategies that favor the introduction of such components into a current (evolving) solution. The diversification stage on the other hand encourages the search process to examine unvisited regions and to generate solutions

that differ in various significant ways from those seen before. Again, such an approach can be based on generating subassemblies of solution components that are then "fleshed out" to produce full solutions, or can rely on modified evaluations as embodied, for example, in the use of penalty / incentive functions.

Intensification strategies require a means for identifying a set of elite solutions as basis for incorporating good attributes into newly created solutions. Membership in the elite set is often determined by setting a threshold that is connected to the objective function value of the best solution found during the search. However, considerations of clustering and "anti-clustering" are also relevant for generating such a set, and more particularly for generating subsets of solutions that may be used for specific phases of intensification and diversification. The TS notions of intensification and diversification are beginning to find their way into other meta-heuristics. It is important to keep in mind that these ideas are somewhat different than the old control theory concepts of "exploitation" and "exploration," especially in their implications for developing effective problem solving strategies.

3.2 Scatter Search

Scatter search, from the standpoint of meta-heuristic classification, may be viewed as an evolutionary (or also called population-based) algorithm that constructs solutions by combining others. It derives its foundations from strategies originally proposed for combining decision rules and constraints (in the context of integer programming). The goal of this methodology is to enable the implementation of solution procedures that can derive new solutions from combined elements. The way scatter search combines solutions and updates the set of reference solutions used for combination sets this methodology apart from other population-based approaches.

The approach of combining existing solutions or rules to create new solutions originated in the 1960s. In the area of scheduling, researchers introduced the notion of combining rules to obtain improved local decisions. Numerically weighted combinations of existing rules, suitably restructured so that their evaluations embodied a common metric, generated new rules. The conjecture that information about the relative desirability of alternative choices is captured in different forms by different rules motivated this approach. The combination strategy was devised with the belief that this information could be exploited more effectively when integrated than when treated in isolation (i.e., when existing selection rules are selected one at a time). In general, the decision rules created from such combination strategies produced better empirical outcomes than standard applications of local decision rules. They also proved superior to a "probabilistic learning approach" that used stochastic selection of rules at different junctures, but without the integration effect provided by generating combined rules.

In integer and nonlinear programming, associated procedures for combining constraints were developed, which likewise employed a mechanism for creating weighted combinations. In this case, nonnegative weights were introduced to create new constraint inequalities, called *surrogate constraints*. The approach isolated subsets of constraints that were gauged to be most critical, relative to trial solutions based on the surrogate constraints. This critical subset was used to produce new weights that reflected the degree to which the component constraints were satisfied or violated.

The main function of surrogate constraints was to provide ways to evaluate choices that could be used to create and modify trial solutions. A variety of heuristic processes that employed surrogate constraints and their evaluations evolved from this foundation. As a natural extension, these processes led to the related strategy of combining solutions. Combining solutions, as manifested in scatter search, can be interpreted as the primal counterpart to the dual strategy of combining constraints.

Scatter search operates on a set of solutions, the *reference set*, by combining these solutions to create new ones. When the main mechanism for combining solutions is such that a new solution is created from the linear combination of two other solutions, the reference set may evolve as illustrated in Figure 2. This figure assumes that the original reference set of solutions consists of the circles labeled A, B and C. After a non-convex combination of reference solutions A and B, solution 1 is created. More precisely, a number of solutions in the line segment defined by A and B are created; however, only solution 1 is introduced in the reference set. In a similar way, convex and non-convex combinations of original and newly created reference solutions create points 2, 3 and 4. The complete reference set shown in Figure 2 consists of 7 solutions (or elements).

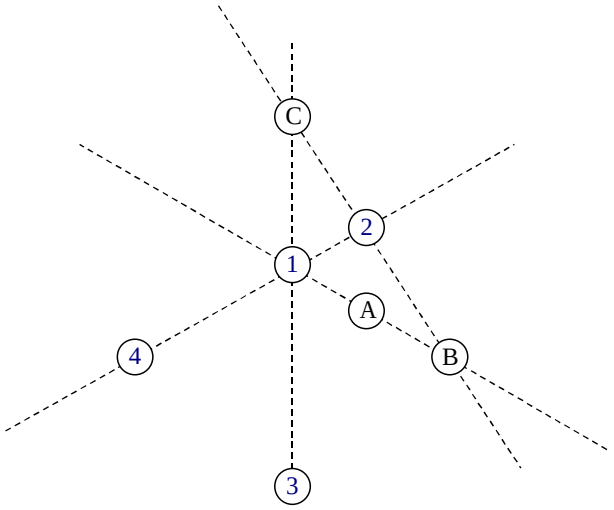

Figure 2. Two-dimensional reference set.

Unlike a "population" in genetic algorithms, the reference set of solutions in scatter search tends to be small. In genetic algorithms, two solutions are randomly chosen from the population and a "crossover" or combination mechanism is applied to generate one or more offspring. A typical population size in a genetic algorithm consists of 100 elements, which are randomly sampled to create combinations. In contrast, scatter search chooses two or more elements of the reference set in a systematic way with the purpose of creating new solutions. Since the combination process considers at least all pairs of solutions in the reference set, there is a practical need for keeping the cardinality of the set small. Typically, the reference set in scatter search has 20 solutions or less. In general, if the reference set consists of b solutions, the procedure examines approximately $(3b-7)b/2$ combinations of four different types. The basic type consists of combining two solutions; the next type combines three solutions, and so on and so forth. Limiting the scope of the search to a selective group of combination types can be used as a mechanism for controlling the number of possible combinations in a given reference set.

3.2.1 Scatter Search Template

The scatter search process, building on the principles that underlie the surrogate constraint design, is organized to (1) capture information not contained separately in the original vectors, (2) take advantage of auxiliary heuristic solution methods to evaluate the combinations produced and to generate new vectors. Specifically, the scatter search approach may be sketched as follows:

1. Generate a starting set of solution vectors to guarantee a critical level of diversity and apply heuristic processes designed for the problem considered as an attempt for improving these

solutions. Designate a subset of the best vectors to be reference solutions. (Subsequent iterations of this step, transferring from Step 4 below, incorporate advanced starting solutions and best solutions from previous history as candidates for the reference solutions.) The notion of "best" in this step is not limited to a measure given exclusively by the evaluation of the objective function. In particular, a solution may be added to the reference set if the diversity of the set improves even when the objective value of such solution is inferior to other solutions competing for admission in the reference set.

2. Create new solutions consisting of structured combinations of subsets of the current reference solutions. The structured combinations are:

   a) chosen to produce points both inside and outside the convex regions spanned by the reference solutions.
   b) modified to yield acceptable solutions. (For example, if a solution is obtained by a linear combination of two or more solutions, a generalized rounding process that yields integer values for integer-constrained vector components may be applied. Note that an acceptable solution may or may not be feasible with respect to other constraints in the problem.)

3. Apply the heuristic processes used in Step 1 to improve the solutions created in Step 2. (Note that these heuristic processes must be able to operate on infeasible solutions and may or may not yield feasible solutions.)

4. Extract a collection of the "best" improved solutions from Step 3 and add them to the reference set. The notion of "best" is once again broad; making the objective value one among several criteria for evaluating the merit of newly created points. Repeat Steps 2, 3 and 4 until the reference set does not change. Diversify the reference set, by re-starting from Step 1. Stop when reaching a specified iteration limit.

The first notable feature in scatter search is that its structured combinations are designed with the goal of creating weighted centers of selected subregions. This adds non-convex combinations that project new centers into regions that are external to the original reference solutions (see, e.g., solution 3 in Figure 2). The dispersion patterns created by such centers and their external projections have been found useful in several application areas.

Another important feature relates to the strategies for selecting particular subsets of solutions to combine in Step 2. These strategies are typically designed to make use of a type of clustering to allow new solutions to be constructed "within clusters" and "across clusters". Finally, the method is organized to use ancillary improving mechanisms that are able to operate on infeasible solutions, removing the restriction that solutions must be feasible in order to be included in the reference set.

The following principles summarize the foundations of the scatter search methodology:

- Useful information about the form (or location) of optimal solutions is typically contained in a suitably diverse collection of elite solutions.
- When solutions are combined as a strategy for exploiting such information, it is important to provide mechanisms capable of constructing combinations that extrapolate beyond the regions spanned by the solutions considered. Similarly, it is also important to incorporate heuristic processes to map combined solutions into new solutions. The purpose of these combination mechanisms is to incorporate both diversity and quality.
- Taking account of multiple solutions simultaneously, as a foundation for creating combinations, enhances the opportunity to exploit information contained in the union of elite solutions.

The fact that the mechanisms within scatter search are not restricted to a single uniform design allows the exploration of strategic possibilities that may prove effective in a particular implementation. These observations and principles lead to the following template for implementing scatter search.

1. A *Diversification Generation Method* to generate a collection of diverse trial solutions, using an arbitrary trial solution (or seed solution) as an input.

2. An *Improvement Method* to transform a trial solution into one or more enhanced trial solutions. (Neither the input nor the output solutions are required to be feasible, though the output solutions will more usually be expected to be so. If no improvement of the input trial solution results, the "enhanced" solution is considered to be the same as the input solution.)

3. A *Reference Set Update Method* to build and maintain a *reference set* consisting of the *b* "best" solutions found (where the value of *b* is typically small, e.g., no more than 20), organized to provide efficient accessing by other parts of the method. Solutions gain membership to the reference set according to their quality or their diversity.

4. A *Subset Generation Method* to operate on the reference set, to produce a subset of its solutions as a basis for creating combined solutions.

5. A *Solution Combination Method* to transform a given subset of solutions produced by the Subset Generation Method into one or more combined solution vectors.

The success of scatter search and related strategies is evident in a variety of application areas such as vehicle routing, arc routing, quadratic assignment, financial product design, neural network training, job shop scheduling, flow shop scheduling, crew scheduling, graph drawing, linear ordering, unconstrained optimization, bit representation, multi-objective assignment, optimizing simulation, tree problems, mixed integer programming.

3.3 Genetic Algorithms

The idea of applying the biological principle of natural evolution to artificial systems, introduced more than three decades ago, has seen impressive growth in the past few years. Usually grouped under the term evolutionary algorithms or evolutionary computation, we find the domains of genetic algorithms, evolution strategies, evolutionary programming, and genetic programming. Evolutionary algorithms have been successfully applied to numerous problems from different domains, including optimization, automatic programming, machine learning, economics, ecology, population genetics, studies of evolution and learning, and social systems.

A genetic algorithm (GA) is an iterative procedure that consists of a constant-size population of individuals, each one represented by a finite string of symbols, known as the genome, encoding a possible solution in a given problem space. This space, referred to as the search space, comprises all possible solutions to the problem at hand. Generally speaking, the genetic algorithm is applied to spaces that are too large to be exhaustively searched (such as those in combinatorial optimization). Solutions to a problem were originally encoded as binary strings due to certain computational advantages associated with such encoding. Also the theory about the behavior of algorithms was based on binary strings. Because in many instances it is impractical to represent solutions using binary strings, the solution representation has been extended in recent years to include character-based encoding, real-valued encoding, and tree representations.

The standard genetic algorithm proceeds as follows: an initial population of individuals is generated at random or heuristically. Every evolutionary step, known as a generation, the individuals in the current population are decoded and evaluated according to some predefined quality criterion, referred to as the fitness, or fitness function. To form a new population (the next generation), individuals are selected according to their fitness. Many selection procedures are currently in use, one of the simplest being Holland's original fitness-proportionate selection, where individuals are selected with a probability proportional to their relative fitness. This ensures that the expected number of times an individual is chosen is approximately proportional to its relative performance in the population. Thus, high-fitness ("good") individuals stand a better chance of "reproducing", while low-fitness ones are more likely to disappear.

Genetically inspired operators are used to introduce new individuals into the population, i.e., to generate new points in the search space. The best known of such operators are crossover and mutation. Crossover is performed, with a given probability p_c (the "crossover probability" or "crossover rate"), between two selected individuals, called parents, by exchanging parts of their genomes (i.e., encoding) to form two new individuals, called offspring; in its simplest form, sub-strings are exchanged after a randomly selected crossover point. This operator tends to enable the evolutionary process to move toward "promising" regions of the search space. The mutation operator is introduced to prevent premature convergence to local optima by randomly sampling new points in the search space. Mutation entails flipping bits at random, with some (small) probability p_m. Genetic algorithms are stochastic iterative processes that are not guaranteed to converge; the termination condition may be specified as some fixed, maximal number of generations or as the attainment of an acceptable fitness level for the best individual.

Let us consider the following simple example to illustrate the genetic algorithm's workings. The population consists of 4 individuals, which are binary-encoded strings (genomes) of length 8. The fitness value equals the number of ones in the bit string, with $p_c = 0.7$, and $p_m = 0.001$. More typical values of the population size and the genome length are in the range 50-1000. Also note that fitness computation in this case is extremely simple since no complex decoding nor evaluation is necessary. The initial (randomly generated) population might look like this:

Label	Genome	Fitness
A	00000110	2
B	11101110	6
C	00100000	1
D	00110100	3

Using fitness-proportionate selection we must choose 4 individuals (two sets of parents), with probabilities proportional to their relative fitness values. In our example, suppose that the two parent pairs are {B,D} and {B,C} (note that A did not get selected as our procedure is probabilistic). Once a pair of parents is selected, the crossover operation is performed with probability p_c, resulting in two offspring. If the crossover operation is not performed (with probability $1-p_c$), then the offspring are exact copies of each parent. Suppose, in our example, that crossover takes place between parents B and D at the (randomly chosen) first bit position, forming offspring E=10110100 and F=01101110, while the crossover operation is not performed between parents B and C, forming offspring that are exact copies of B and C. Next, each offspring is subject to mutation with probability p_m per bit. For example, suppose offspring E is mutated at the sixth position to form E'=10110000, offspring B is mutated at the first bit position to form B'=01101110, and offspring F and C are not mutated at all. The next generation population, created by the above operators of selection, crossover, and mutation is therefore:

Label	Genome	Fitness
E'	10110000	3
F	01101110	5
C	00100000	1
B'	01101110	5

Note that in the new population, although the best individual with fitness 6 has been lost, the average fitness has increased. Iterating this procedure, the genetic algorithm will eventually find a perfect string, i.e., with maximal fitness value of 8. More sophisticated implementations of GAs include the use of local search and several crossover operators that are chosen probabilistically to be applied to each pair of selected parents.

4. Metaphors of Nature

A popular thrust of many research initiatives, and especially of publications designed to catch the public eye, is to associate various methods with processes found in nature. The most notable case is the so-called *ant colony* optimization. The idea behind this meta-heuristic is to emulate the behavior of an ant colony in search for food. In particular, the procedure simulates how pheromone trails are built and also how they evaporate. The simulation of this behavior is achieved with memory functions that are remarkably similar to those in tabu search. For example, the method uses both frequency-based memory and recency-based memory to control attractiveness of a route that an artificial ant might take.

The trend of using metaphors of nature embodies a wave of "New Romanticism," reminiscent of the Romanticism of the 18^{th} and 19^{th} centuries (distinguished by their preoccupation with Nature with a capital "N"). The current fascination with natural phenomena as a foundation for problem-solving methods undoubtedly is fueled by our sense of mystery concerning the ability of such phenomena to generate outcomes that are still far beyond our comprehension. However, the New Romanticism goes farther, to suggest that by mimicking the rules we imagine to operate in nature (especially "rudimentary" processes of nature) we will similarly be able to produce remarkable outcomes.

Models of nature that are relied upon for such inspiration are ubiquitous, and it is easy to conjure up examples whose metaphorical possibilities have not yet been tapped. To take an excursion in the lighter side of such possibilities (though not too far from the lanes currently traveled), we may observe that a beehive offers a notable example of a system that possesses problem-solving abilities. Bees produce hives of exceptional quality and complexity, coordinate diverse tasks among different types of individuals, perform spatial navigation, and communicate via multiple media. (It is perhaps surprising in retrospect that the behavior of bees has not been selected as a basis for one of the "new" problem solving methods.)

Nor is it necessary to look simply to sentient creatures for analogies that inspire templates for effective problem solving. The root system of a tree, for example, provides an intriguing model for parallel computation. In order to find moisture and nutrients (analogous to a quest for "solutions"), roots distribute themselves across different regions, sending out probes that multiply or atrophy according to the efficacy of their progress. The paths of such a system may cross, as different channels prove promising by virtue of the regions in which they lie and also according to the directions in which they are explored. Obstacles are effectively skirted, or over time are surmounted by longer-range strategies — as by extending finer probes, which ultimately expand until the medium is broached. There exist some root systems, as in groves of aspen, where roots of one entity can merge with those of another, thus enlarging the potential sources of communication and contact available to each. (Such an interlinked community gives rise to the largest known organisms on the planet.)

These analogies to systems in nature invite us to ponder a key question. If we were allowed to place our bets on the probable success of a hive of bees or a grove of aspen, as opposed to that of a group of humans, when confronted with a challenging task that requires intelligence and the ability to learn from the past, how would we wager? Undoubtedly we would be drawn to reflect that our goals and problem structures might often be different than those to which "natural processes" apply. In addition, we ourselves — as products of a rather special and extended chain of natural developments — may incorporate capabilities not present in the processes that produced us.

Metaphors of nature have a place. They appear chiefly to be useful for spurring ideas to launch the first phases of an investigation. As long as care is taken to prevent such metaphors from cutting off lines of inquiry beyond their scope, they provide a means for "dressing up" the descriptions of various meta-heuristics in a way that appeals to our instinct to draw parallels between simple phenomena and abstract designs.

Invoking such parallels may sometimes appear to embody a primitive mysticism, akin to chanting about campfires in the night, but it gives us a foundation for connecting the new to the old, and for injecting passion into our quests. It is up to prudence to determine when the symbolism of the New Romanticism obscures rather than illuminates the pathway to improved understanding. Within the realm of meta-heuristic design, there is a great deal we have yet to learn. The issue of whether the analogies that underlie some of our models may limit or enhance our access to further discovery deserves careful reflection.

Acknowledgments

The author wishes to thank Dr. Moshe Sipper for his permission to use materials from his web site for section 3.3.

Bibliography

Corne, D., M. Dorigo and F. Glover (1999) *New Ideas in Optimization*, McGraw-Hill, England. [This book addresses novel meta-heuristic methods as well as the emerging ideas for future designs.]

Glover, F. (1977) "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Sciences*, vol. 8, pp. 156-166. [Seminal work on tabu search and scatter search.]

Glover, F. and Laguna (1997) *Tabu Search*, Kluwer Academic Publishers, Boston. [This book presents a comprehensive treatment of the meta-heuristic known as tabu search.]

Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley. [One of the first comprehensive books on genetic algorithms.]

Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, Michigan. [Seminal work on genetic algorithms.]

Kelly, J. and I. Osman (1996) *Meta-heuristics: Theory and Applications*, Kluwer Academic Publishers, Boston. [It contains articles presented in the first conference on meta-heuristics held in Breckenridge, Colorado.]

Kirkpatrick, S., C. D. Gellat and M. P. Vecchi (1983) "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-680. [Seminal work on simulated annealing.]

Michalewicz, Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin, Third Edition. [The associated computer code called GENOCOP has been extensively used as a benchmark for multi-modal function optimization.]

Reeves, C. (1993) *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, Oxford. [This addresses in detail well-known meta-heuristics and also includes material on experimental testing.]