TABU SEARCH*

Fred Glover[a]
Manuel Laguna[b]

[a] OptTek Systems, Inc., 2241 17th Street, Boulder, CO 80304 USA
glover@opttek.com
[b] Leeds School of Business, University of Colorado, Boulder, CO 80309 USA
laguna@colorado.edu

Chapter to appear in the *Handbook of Combinatorial Optimization*, 2nd Edition, Panos Pardalos, Ding-Zu Du and Ronald Graham, eds.

Abstract

Tabu Search, also called Adaptive Memory Programming, is a method for solving challenging problems in the field of optimization. The goal is to identify the best decisions or actions in order to maximize some measure of merit (such as maximizing profit, effectiveness, quality, social or scientific benefit), or to minimize some measure of demerit (cost, inefficiency, waste, social or scientific loss).

Practical applications in optimization addressed by Tabu Search are exceedingly challenging and pervade the fields of business, engineering, economics and science. Everyday examples include problems in resource management, financial and investment planning, healthcare systems, energy and environmental policy, pattern classification, biotechnology and a host of other areas. The complexity and importance of such problems has motivated a wealth of academic and practical research throughout the past several decades, in an effort to discover methods that are able to find solutions of higher quality than many found in the past and capable of producing such solutions within feasible time limits or at reduced computational cost.

Tabu search has emerged as one of the leading technologies for handling optimization problems that have proved difficult or impossible to solve with classical procedures that dominated the attention of textbooks and were considered the mainstays of available alternatives until recent times. A key feature of tabu search, underscored by its *adaptive memory programming* alias, is the use of special strategies designed to exploit adaptive memory. The idea is that an effective search for optimal solutions should involve a process of flexibly responding to the solution landscape in a manner that permits it to learn appropriate directions to take along with appropriate departures to explore new terrain. The adaptive memory feature of tabu search and allows the implementation of procedures that are capable of searching this terrain economically and effectively.

* The material of this chapter is in part adapted from the book *Tabu Search*, by Fred Glover and Manuel Laguna, Kluwer Academic Publishers, 1997.

Introduction

Faced with the challenge of solving hard optimization problems that abound in the real world, classical methods often encounter great difficulty. Vitally important applications in business, engineering, economics and science cannot be tackled with any reasonable hope of success, within practical time horizons, by solution methods that have been the predominant focus of academic research throughout the past three decades (and which are still the focus of many textbooks).

The meta-heuristic approach called tabu search (TS) is dramatically changing our ability to solve problems of practical significance. Current applications of TS span the realms of resource planning, telecommunications, VLSI design, financial analysis, scheduling, space planning, energy distribution, molecular engineering, logistics, pattern classification, flexible manufacturing, waste management, mineral exploration, biomedical analysis, environmental conservation and scores of others. In recent years, journals in a wide variety of fields have published tutorial articles and computational studies documenting successes by tabu search in extending the frontier of problems that can be handled effectively — yielding solutions whose quality often significantly surpasses that obtained by methods previously applied. Table 1.1 gives a partial catalog of example applications. A more comprehensive list, including summary descriptions of gains achieved from practical implementations, can be found in Glover and Laguna, 1997. Recent TS developments and applications can also be found in the Tabu Search Vignettes section of the web page http://spot.colorado.edu/~glover.

Main Text

1. Tabu Search Features and Relevance

A distinguishing feature of tabu search is embodied in its exploitation of adaptive forms of memory, which equips it to penetrate complexities that often confound alternative approaches. Yet we are only beginning to tap the rich potential of adaptive memory strategies, and the discoveries that lie ahead promise to be as important and exciting as those made to date. The knowledge and principles that have emerged from the TS framework give a foundation to create practical systems whose capabilities markedly exceed those available earlier. At the same time, there are many untried variations that may lead to further advances. A conspicuous feature of tabu search is that it is dynamically growing and evolving, drawing on important contributions by many researchers.

Table 1.1. Illustrative tabu search applications.

Scheduling	**Telecommunications**
Flow-Time Cell Manufacturing	Call Routing
Heterogeneous Processor Scheduling	Bandwidth Packing
Workforce Planning	Hub Facility Location
Classroom Scheduling	Path Assignment
Machine Scheduling	Network Design for Services
Flow Shop Scheduling	Customer Discount Planning
Job Shop Scheduling	Failure Immune Architecture
Sequencing and Batching	Synchronous Optical Networks
Design	**Production, Inventory and Investment**
Computer-Aided Design	Flexible Manufacturing
Fault Tolerant Networks	Just-in-Time Production
Transport Network Design	Capacitated MRP
Architectural Space Planning	Part Selection
Diagram Coherency	Multi-item Inventory Planning
Fixed Charge Network Design	Volume Discount Acquisition
Irregular Cutting Problems	Fixed Mix Investment
Location and Allocation	**Routing**
Supply Chain Analysis	Vehicle Routing
Multicommodity Location/Allocation	Capacitated Routing
Quadratic Assignment	Time Window Routing
Quadratic Semi-Assignment	Multi-Mode Routing
Multilevel Generalized Assignment	Mixed Fleet Routing
Lay-Out Planning	Traveling Salesman
Off-Shore Oil Exploration	Traveling Purchaser
Logic and Artificial Intelligence	**Graph Optimization**
Maximum Satisfiability	Graph Partitioning
Probabilistic Logic	Graph Coloring
Clustering	Clique Partitioning
Pattern Recognition/Classification	Maximum Clique Problems
Data Integrity	Maximum Planner Graphs
Neural Network |Training and Design	P-Median Problems
Technology	**General Combinational Optimization**
Seismic Inversion	Zero-One Programming
Electrical Power Distribution	Fixed Charge Optimization
Engineering Structural Design	Nonconvex Nonlinear Programming
Coordination of Energy Resources	All-or-None Networks
Space Station Construction	Bilevel Programming
DNA Sequencing	Multi-objective Discrete Optimization
Circuit Cell Placement	Hyperplane Splitting
Computer Aided Molecular Design	General Mixed Integer Optimization

1.1 General Tenets

The word *tabu* (or *taboo)* comes from Tongan, a language of Polynesia, where it was used by the aborigines of Tonga island to indicate things that cannot be touched because they are sacred.

According to Webster's Dictionary, the word now also means “a prohibition imposed by social custom as a protective measure" or of something "banned as constituting a risk.” These current more pragmatic senses of the word accord well with the theme of tabu search. The risk to be avoided in this case is that of following a counter-productive course, including one which may lead to entrapment without hope of escape. On the other hand, as in the broader social context where “protective prohibitions” are capable of being superseded when the occasion demands, the “tabus” of tabu search are to be overruled when evidence of a preferred alternative becomes compelling.

The most important association with traditional usage, however, stems from the fact that tabus as normally conceived are transmitted by means of a social memory which is subject to modification over time. This creates the fundamental link to the meaning of "tabu" in tabu search. The forbidden elements of tabu search receive their status by reliance on an evolving memory, which allows this status to shift according to time and circumstance.

More particularly, tabu search is based on the premise 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. Special links with evolutionary methods, and implications for establishing more effective variants of them, are discussed in Section 5.)

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.

1.2 Use of Memory

The memory structures in tabu search operate by reference to four principal dimensions, consisting of recency, frequency, quality, and influence (Figure 1.1). *Recency-based* and *frequency-based* based memory complement each other, and have important characteristics we amplify in later sections. 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.

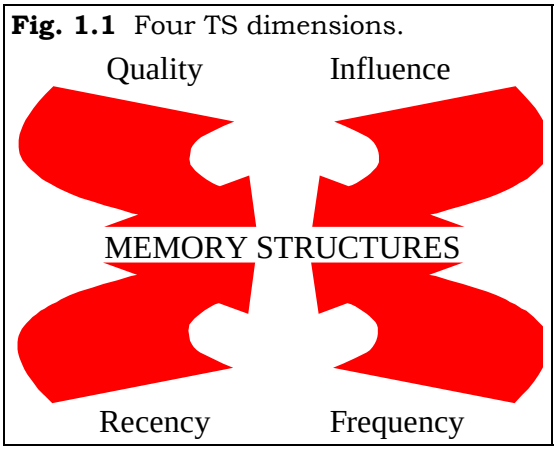


Fig. 1.1 Four TS dimensions.

The fourth dimension, *influence*, considers the impact of the choices made during the search, not only on quality but also on structure. (In a sense, quality may be regarded as a special form of influence.) 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 prespecified 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, as indicated in Section 3. In some cases explicit memory has been used to guide the search and avoid visiting solutions more than once. This application is limited, because clever data structures must be designed to avoid excessive memory requirements.

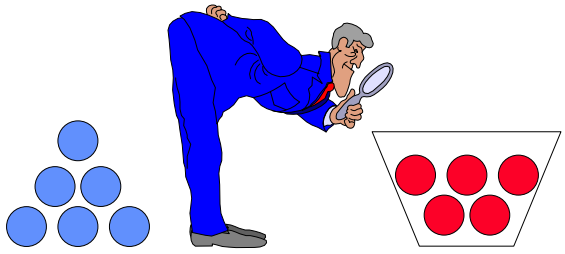
Alternatively, TS uses attributive memory for guiding purposes. This type of memory records information about solution attributes that change in moving from one solution to another. For example, in a graph or network setting, attributes can consist of nodes or arcs that are added, dropped or repositioned 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.

1.3 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. As Figure 1.2 illustrates, the main difference between intensification and diversification is that during an intensification stage the search focuses on examining neighbors of elite solutions.

Fig. 1.2 Intensification and diversification.



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 which 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. In the following sections, we show how the treatment of such concerns can be enhanced by making use of special memory structures. The TS notions of intensification and diversification are beginning to find their way into other meta-heuristics, and it is important to keep in mind (as we subsequently demonstrate) 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.

2. Tabu Search Foundations and Short Term Memory

Tabu search can be applied directly to verbal or symbolic statements of many kinds of decision problems, without the need to transform them into mathematical formulations. Nevertheless, it is useful to introduce mathematical notation to express a broad class of these problems, as a basis for describing certain features of tabu search. We characterize this class of problems as

that of optimizing (minimizing or maximizing) a function $f(x)$ subject to $x \in \mathbf{X}$, where $f(x)$ may be linear or nonlinear, and the set $\mathbf{X}$ summarizes constraints on the vector of decision variables x. The constraints may include linear or nonlinear inequalities, and may compel all or some components of x to receive discrete values. While this representation is useful for discussing a number of problem solving considerations, we emphasize again that in many applications of combinatorial optimization, the problem of interest may not be easily formulated as an objective function subject to a set of constraints. The requirement $x \in \mathbf{X}$, for example, may specify logical conditions or interconnections that would be cumbersome to formulate mathematically, but may be better be left as verbal stipulations that can be then coded as rules.

Tabu search begins in the same way as ordinary local or neighborhood search, proceeding iteratively from one point (solution) to another until a chosen termination criterion is satisfied. Each $x \in \mathbf{X}$ has an associated neighborhood $\mathbf{N}(x) \subset \mathbf{X}$, and each solution $x' \in \mathbf{N}(x)$ is reached from x by an operation called a *move.*

As an initial point of departure, we may contrast TS with a simple descent method where the goal is to minimize $f(x)$ (or a corresponding ascent method where the goal is to maximize $f(x)$). Such a method only permits moves to neighbor solutions that improve the current objective function value and ends when no improving solutions can be found. A pseudo-code of a generic descent method is presented in Figure 2.1. The final x obtained by a descent method is called a local optimum, since it is at least as good or better than all solutions in its neighborhood. The evident shortcoming of a descent method is that such a local optimum in most cases will not be a global optimum, i.e., it usually will not minimize $f(x)$ over all $x \in \mathbf{X}$.

Fig. 2.1 Descent method.

1) Choose $x \in \mathbf{X}$ to start the process.
2) Find $x' \in \mathbf{N}(x)$ such that $f(x') < f(x)$.
3) If no such x' can be found, x is the local optimum and the method stops.
4) Otherwise, designate x' to be the new x and go to 2).

The version of a descent method called *steepest descent* scans the entire neighborhood of x in search of a neighbor solution x' that gives a smallest $f(x')$ value over $x' \in \mathbf{N}(x)$. Steepest descent implementations of some types of solution approaches (such as certain path augmentation algorithms in networks and matroids) are guaranteed to yield globally optimal solutions for the problems they are designed to handle, while other forms of descent may terminate with local optima that are not global optima. In spite of this attractive feature, in certain settings steepest descent is sometimes impractical because it is computationally too expensive, as where $\mathbf{N}(x)$ contains many elements or each element is costly to retrieve or evaluate. Still, it is often valuable to choose an x' at each iteration that yields a "good" if not smallest $f(x')$ value.

The relevance of choosing good solutions from current neighborhoods is magnified when the guidance mechanisms of tabu search are introduced to go beyond the locally optimal termination point of a descent method. Thus, an important first level consideration for tabu search is to determine an appropriate *candidate list strategy* for narrowing the examination of

elements of $\mathbf{N}(x)$, in order to achieve an effective tradeoff between the quality of x' and the effort expended to find it. Here quality may involve considerations beyond those narrowly reflected by the value of $f(x')$. If a neighborhood space is totally random, then of course nothing will work better than a totally random choice. (In such a case there is no merit in trying to devise an effective solution procedure.) Assuming that neighborhoods can be identified that are reasonably meaningful for a given class of problems, the challenge is to define solution quality appropriately so that evaluations likewise will have meaning. By the TS orientation, the ability to use history in creating such evaluations then becomes important for devising effective methods

To give a foundation for understanding the basic issues involved, we turn our attention to the following illustrative example, which will also be used as a basis for illustrating various aspects of tabu search in later sections.

2.1 Memory and Tabu Classifications

An important distinction in TS arises by differentiating between short term memory and longer term memory. Each type of memory is accompanied by its own special strategies. However, the effect of both types of memory may be viewed as modifying the neighborhood $\mathbf{N}(x)$ of the current solution x. The modified neighborhood, which we denote by $\mathbf{N}^*(x)$, is the result of maintaining a selective history of the states encountered during the search.

In the TS strategies based on short term considerations, $\mathbf{N}^*(x)$ characteristically is a subset of $\mathbf{N}(x)$, and the tabu classification serves to identify elements of $\mathbf{N}(x)$ excluded from $\mathbf{N}^*(x)$. In TS strategies that include longer term considerations, $\mathbf{N}^*(x)$ may also be expanded to include solutions not ordinarily found in $\mathbf{N}(x)$. Characterized in this way, TS may be viewed as a dynamic neighborhood method. This means that the neighborhood of x is not a static set, but rather a set that can change according to the history of the search. This feature of a dynamically changing neighborhood also applies to the consideration of selecting different component neighborhoods from a *compound* neighborhood that encompasses multiple types or levels of moves, and provides an important basis for parallel processing. Characteristically, a TS process based strictly on short term strategies may allow a solution x to be visited more than once, but it is likely that the corresponding reduced neighborhood $\mathbf{N}^*(x)$ will be different each time. With the inclusion of longer term considerations, the likelihood of duplicating a previous neighborhood upon revisiting a solution, and more generally of making choices that repeatedly visit only a limited subset of $\mathbf{X}$, is all but nonexistent. From a practical standpoint, the method will characteristically identify an optimal or near optimal solution long before a substantial portion of $\mathbf{X}$ is examined.

A crucial aspect of TS involves the choice of an appropriate definition of $\mathbf{N}^*(x)$. Due to the exploitation of memory, $\mathbf{N}^*(x)$ depends upon the trajectory followed in moving from one solution to the next (or upon a collection of such trajectories in a parallel processing environment).

The approach of storing complete solutions (*explicit* memory) generally consumes an enormous amount of space and time when applied to each solution generated. A scheme that emulates this approach with limited memory requirements is given by the use of hash functions. (Also, as will be seen, explicit memory has a valuable role when selectively applied in strategies that record and analyze certain "special" solutions.) Regardless of the implementation details, short term memory functions provide one of the important cornerstones of the TS methodology. These functions give the search the opportunity to continue beyond local optima, by allowing the execution of nonimproving moves coupled with the modification of the neighborhood

structure of subsequent solutions. However, instead of recording full solutions, these memory structures are generally based on recording attributes (*attributive* memory). In addition, short term memory is often based on the most recent history of the search trajectory.

2.2 Recency-Based Memory

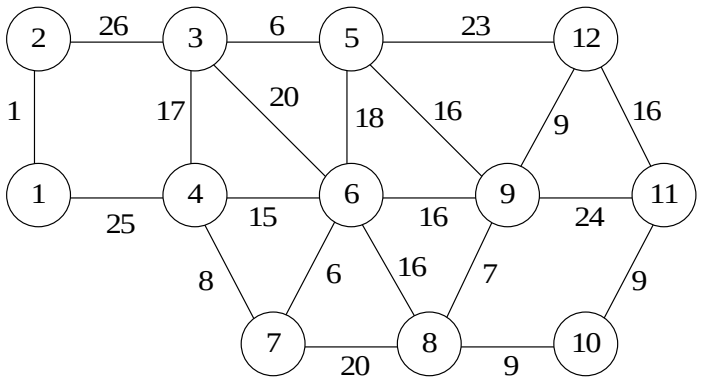
The most commonly used short term memory keeps track of solutions attributes that have changed during the recent past, and is called *recency-based* memory. This is the kind of memory that is included in most short descriptions of tabu search in the literature (although a number of its aspects are often left out by popular summaries).

To exploit this memory, selected attributes that occur in solutions recently visited are labeled *tabu-active*, and solutions that contain tabu-active elements, or particular combinations of these attributes, are those that become tabu. This prevents certain solutions from the recent past from belonging to $\mathbf{N}^*(x)$ and hence from being revisited. Other solutions that share such tabu-active attributes are also similarly prevented from being visited. Note that while the tabu classification strictly refers to solutions that are forbidden to be visited, by virtue of containing tabu-active attributes (or more generally by violating certain restriction based on these attributes), we also often refer to moves that lead to such solutions as being tabu. We illustrate these points with the following example.

Minimum k-Tree Problem Example

The *Minimum k-Tree* problem seeks a tree consisting of k edges in a graph so that the sum of the weights of these edges is minimum (Lokketangen, et al. 1994). An instance of this problem is given in Figure 2.2, where nodes are shown as numbered circles, and edges are shown as lines that join pairs of nodes (the two "endpoint" nodes that determine the edge). Edge weights are shown as the numbers attached to these lines. A tree is a set of edges that contains no cycles, i.e., that contains no paths that start and end at the same node (without retracing any edges).

Fig. 2.2 Weighted undirected graph.



Assume that the move mechanism is defined by edge-swapping, as subsequently described, and that a greedy procedure is used to find an initial solution. The greedy construction starts by choosing the edge (i, j) with the smallest weight in the graph, where i and j are the indexes of the nodes that are the endpoints of the edge. The remaining k-1 edges are chosen successively

to minimize the increase in total weight at each step, where the edges considered meet exactly one node from those that are endpoints of edges previously chosen. For $k = 4$, the greedy construction performs the steps in Table 2.1.

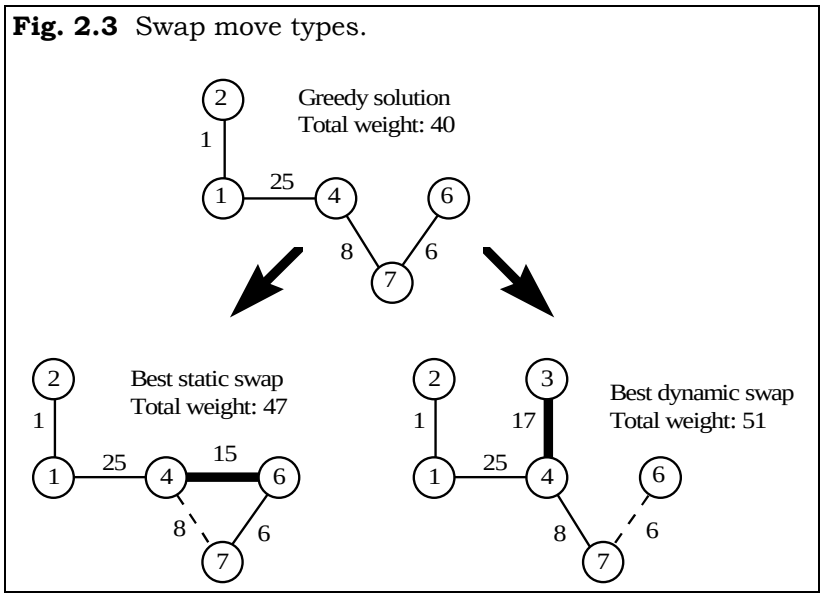
Table 2.1 Greedy construction.

Step	Candidates	Selection	Total Weight
1	(1,2)	(1,2)	1
2	(1,4), (2,3)	(1,4)	26
3	(2,3), (3,4), (4,6), (4,7)	(4,7)	34
4	(2,3), (3,4), (4,6), (6,7), (7,8)	(6,7)	40

The construction starts by choosing edge (1,2) with a weight of 1 (the smallest weight of any edge in the graph). After this selection, the candidate edges are those that connect the nodes in the current partial tree with those nodes not in the tree (i.e., edges (1,4) and (2,3)). Since edge (1,4) minimizes the weight increase, it is chosen to be part of the partial solution. The rest of the selections follow the same logic, and the construction ends when the tree consists of 4 edges (i.e., the value of k). The initial solution in this particular case has a total weight of 40.

The swap move mechanism, which is used from this point onward, replaces a selected edge in the tree by another selected edge outside the tree, subject to requiring that the resulting subgraph is also a tree. There are actually two types of such edge swaps, one that maintains the current nodes of the tree unchanged (static) and one that results in replacing a node of the tree by a new node (dynamic). Figure 2.3 illustrates the best swap of each type that can be made starting from the greedy solution. The added edge in each case is shown by a heavy line and the dropped edge is shown by a dotted line.

The best move of both types is the static swap of Figure 2.3, where for our present illustration we are defining *best* solely in terms of the change on the objective function value. Since this best move results in an increase of the total weight of the current solution, the execution of such move abandons the rules of a descent approach and sets the stage for a tabu search process. (The feasibility restriction that requires a tree to be produced at each step is particular to this illustration, since in general the TS methodology may include search trajectories that violate various types of feasibility conditions.)



Given a move mechanism, such as the swap mechanism we have selected for our example, the next step is to choose the key attributes that will be used for the tabu classification. Tabu search is very flexible at this stage of the design. Problem-specific knowledge can be used as guidance to settle on a particular design. In problems where the moves are defined by adding and deleting elements, the labels of these elements can be used as the attributes for enforcing tabu status. Here, in the present example, we can simply refer to the edges as attributes of the move, since the condition of being *in* or *out of the tree* (which is a distinguishing property of the current solution*)* may be assumed to always be automatically known by a reasonable solution representation.

Choosing Tabu Classifications

Tabu classifications do not have to be symmetric, that is, the tabu structure can be designed to treat added and dropped elements differently. Suppose for example that after choosing the static swap of Figure 2.3, which adds edge (4,6) and drops edge (4,7), a tabu status is assigned to both of these edges. Then one possibility is to classify both of these edges tabu-active for the same number of iterations. The tabu-active status has different meanings depending on whether the edge is added or dropped. For an added edge, tabu-active means that this edge is not allowed to be dropped from the current tree for the number of iterations that defines its tabu tenure. For a dropped edge, on the other hand, tabu-active means the edge is not allowed to be included in the current solution during its tabu tenure. Since there are many more edges outside the tree than in the tree, it seems reasonable to implement a tabu structure that keeps a recently dropped edge tabu-active for a longer period of time than a recently added edge. Notice also that for this problem the tabu-active period for added edges is bounded by k, since if no added edge is allowed to be dropped for k iterations, then within k steps all available moves will be classified tabu.

The concept of creating asymmetric tabu classifications can be readily applied to settings where add/drop moves are not used.

Illustrative Tabu Classifications for the Min k-Tree Problem

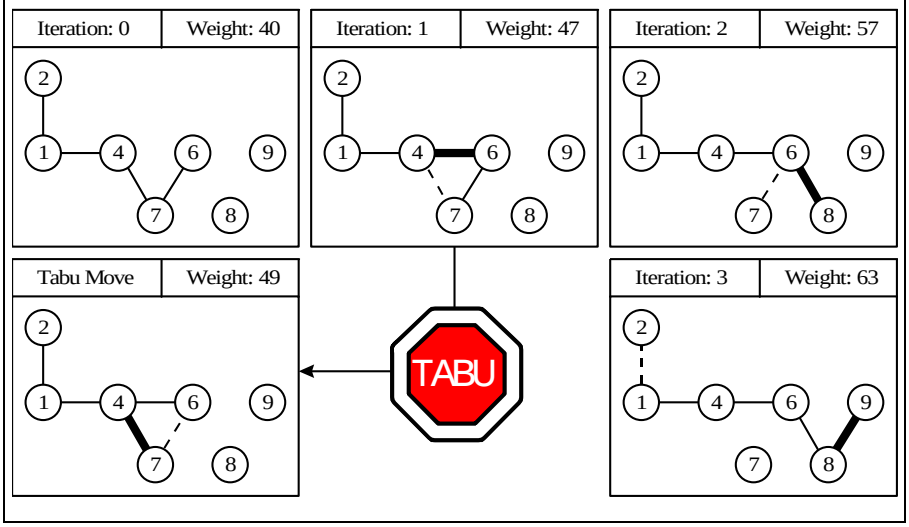
As previously remarked, the tabu-active classification may in fact prevent the search from visiting solutions that have not been examined yet. We illustrate this phenomenon as follows. Suppose that in the Min *k*-Tree problem instance of Figure 2.2, dropped edges are kept tabu-active for 2 iterations, while added edges are kept tabu-active for only one iteration. (The number of iterations an edge is kept tabu-active is called the *tabu tenure* of the edge.) Also assume that we define a swap move to be tabu if either its added or dropped edge is tabu-active. If we examine the full neighborhood of available edge swaps at each iteration, and always choose the best that is not tabu, then the first three moves are as shown in Table 2.2 below (starting from the initial solution found by the greedy construction heuristic). The move of iteration 1 is the static swap move previously identified in Figure 2.3. Diagrams showing the successive trees generated by these moves, starting with the initial greedy solution, are given in Figure 2.4.

Table 2.2 TS iterations.

Iteration	Tabu-active net tenure		Add	Drop	Weight
	1	2			
1			(4,6)	(4,7)	47
2	(4,6)	(4,7)	(6,8)	(6,7)	57
3	(6,8), (4,7)	(6,7)	(8,9)	(1,2)	63

The net tenure values of 1 and 2 in Table 2.2 for the currently tabu-active edges indicate the number of iterations that these edges will remain tabu-active (including the current iteration).

Fig. 2.4 Effects of attributive short term memory.



At iteration 2, the reversal of the move of iteration 1 (that is, the move that now adds (4,7) and drops (4,6)) is clearly tabu, since both of its edges are tabu-active at iteration 2. In addition, the move that adds (4,7) and drops (6,7) is also classified tabu, because it contains the tabu-

active edge (4,7) (with a net tenure of 2). This move leads to a solution with a total weight of 49, a solution that clearly has not been visited before (see Figure 2.4). The tabu-active classification of (4,7) has modified the original neighborhood of the solution at iteration 2, and has forced the search to choose a move with an inferior objective function value (i.e., the one with a total weight of 57). In this case, excluding the solution with a total weight of 49 has little effect on the quality of the best solution found (since we have already obtained one with a weight of 40).

In other situations, however, additional precautions must be taken to avoid missing good solutions. These strategies are known as aspiration criteria and are the subject of Section 2.6. For the moment we observe simply that if the tabu solution encountered at the current step instead had a weight of 39, which is better than the best weight of 40 so far seen, then we would allow the tabu classification of this solution to be overridden and consider the solution admissible to be visited. The aspiration criterion that applies in this case is called the *improved-best* aspiration criterion. (It is important to keep in mind that aspiration criteria do not compel particular moves to be selected, but simply make them available, or alternately rescind evaluation penalties attached to certain tabu classifications.)

One other comment about tabu classification deserves to be made at this point. In our preceding discussion of the Min k-Tree problem we consider a swap move tabu if either its added edge or its dropped edge is tabu-active. However, we could instead stipulate that a swap move is tabu only if both its added and dropped edges are tabu-active. In general, the tabu status of a move is a function of the tabu-active attributes of the move (i.e., of the new solution produced by the move).

2.3 A First Level Tabu Search Approach

We now have on hand enough ingredients for a first level tabu search procedure. Such a procedure is sometimes implemented in an initial phase of a TS development to obtain a preliminary idea of performance and calibration features, or simply to provide a convenient staged approach for the purpose of debugging solution software. While this naive form of a TS method omits a number of important short term memory considerations, and does not yet incorporate longer term concerns, it nevertheless gives a useful starting point for demonstrating several basic aspects of tabu search.

We start from the solution with a weight of 63 as shown previously in Figure 2.4 which was obtained at iteration 3. At each step we select the least weight non-tabu move from those available, and use the improved-best aspiration criterion to allow a move to be considered admissible in spite of leading to a tabu solution. The reader may verify that the outcome leads to the series of solutions shown in Table 2.3, which continues from iteration 3, just executed. For simplicity, we select an arbitrary stopping rule that ends the search at iteration 10.

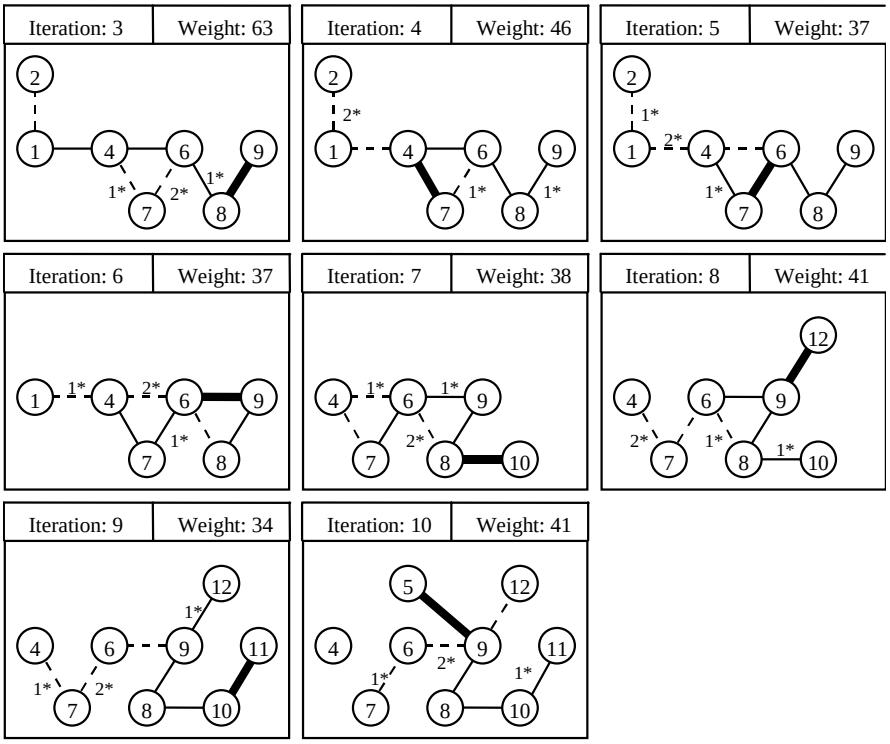
Table 2.3 Iterations of a first level TS procedure.

Iteration	Tabu-active net tenure		Add	Drop	Move Value	Weight
	1	2				
3	(6,8), (4,7)	(6,7)	(8,9)	(1,2)	6	63
4	(6,7), (8,9)	(1,2)	(4,7)	(1,4)	-17	46
5	(1,2), (4,7)	(1,4)	(6,7)	(4,6)	-9	37*
6	(1,4), (6,7)	(4,6)	(6,9)	(6,8)	0	37
7	(4,6), (6,9)	(6,8)	(8,10)	(4,7)	1	38
8	(6,8), (8,10)	(4,7)	(9,12)	(6,7)	3	41
9	(4,7), (9,12)	(6,7)	(10,11)	(6,9)	-7	34*
10	(6,7), (10,11)	(6,9)	(5,9)	(9,12)	7	41

The successive solutions identified in Table 2.3 are shown graphically in Figure 2.5 below. In addition to identifying the dropped edge at each step as a dotted line, we also identify the dropped edge from the immediately preceding step as a dotted line which is labeled 2^*, to indicate its current net tabu tenure of 2. Similarly, we identify the dropped edge from one further step back by a dotted line which is labeled 1^*, to indicate its current net tabu tenure of 1. Finally, the edge that was added on the immediately preceding step is also labeled 1^* to indicate that it likewise has a current net tabu tenure of 1. Thus the edges that are labeled with tabu tenures are those which are currently tabu-active, and which are excluded from being chosen by a move of the current iteration (unless permitted to be chosen by the aspiration criterion).

As illustrated in Table 2.3 and Figure 2.5 the method continues to generate different solutions, and over time the best known solution (denoted by an asterisk) progressively improves. In fact, it can be verified for this simple example that the solution obtained at iteration 9 is optimal. (In general, of course, there is no known way to verify optimality in polynomial time for difficult discrete optimization problems, i.e., those that fall in the class called NP-hard. The Min *k*-Tree problem is one of these.)

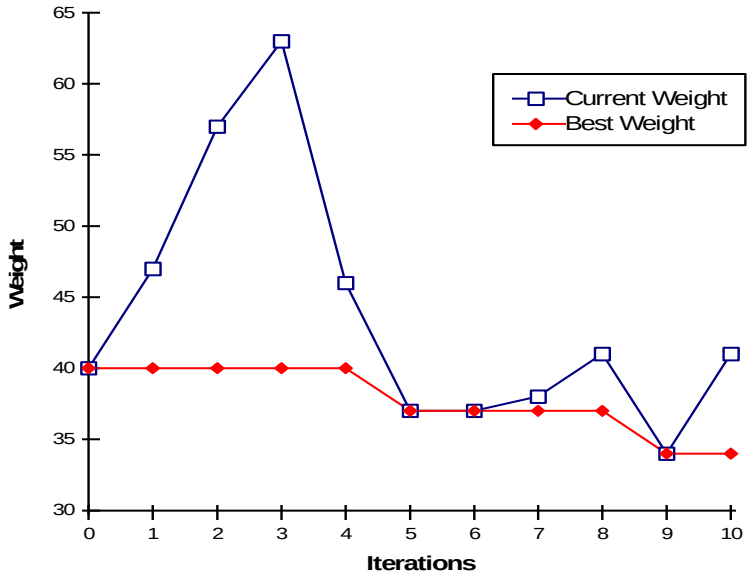
Fig. 2.5 Graphical representation of TS iterations.



It may be noted that at iteration 6 the method selected a move with a move value of zero. Nevertheless, the configuration of the current solution changes after the execution of this move, as illustrated in Figure 2.5.

The selection of moves with certain move values, such as zero move values, may be strategically controlled, to limit their selection as added insurance against cycling in special settings. We will soon see how considerations beyond this first level implementation can lead to an improved search trajectory, but the non-monotonic, gradually improving, behavior is characteristic of TS in general. Figure 2.6 provides a graphic illustration of this behavior for the current example.

Fig. 2.6 TS search trajectory.



We have purposely chosen the stopping iteration to be small to illustrate an additional relevant feature, and to give a foundation for considering certain types of longer term considerations. One natural way to apply TS is to periodically discontinue its progress, particularly if its rate of finding new best solutions falls below a preferred level, and to restart the method by a process designated to generate a new sequence of solutions.

Classical restarting procedures based on randomization evidently can be used for this purpose, but TS often derives an advantage by employing more strategic forms of restarting. We illustrate a simple instance of such a restarting procedure, which also serves to introduce a useful memory concept.

2.3.1 Critical Event Memory

Critical Event memory in tabu search, as its name implies, monitors the occurrence of certain *critical events* during the search, and establishes a memory that constitutes an aggregate summary of these events. For our current example, where we seek to generate a new starting solution, a critical event that is clearly relevant is the generation of the previous starting solution. Correspondingly, if we apply a restarting procedure multiple times, the steps of generating all preceding starting solutions naturally qualify as critical events. That is, we would prefer to depart from these solutions in some significant manner as we generate other starting solutions.

Different degrees of departure, representing different levels of *diversification*, can be achieved by defining solutions that correspond to critical events in different ways (and by activating critical event memory by different rules). In the present setting we consider it important that new starting solutions not only differ from preceding starting solutions, but that they also differ from other solutions generated during previous passes. One possibility is to use a blanket approach that considers each complete solution previously generated to represent a critical event. The aggregation of such events by means of critical event memory makes this entirely

practicable, but often it is quite sufficient (and, sometimes preferable) to isolate a smaller set of solutions.

For the current example, therefore, we will specify that the critical events of interest consist of generating not only the starting solution of the previous pass(es), but also each subsequent solution that represents a "local TS optimum," i.e. whose objective function value is better (or no worse) than that of the solution immediately before and after it. Using this simple definition we see that four solutions qualify as critical (i.e., are generated by the indicated critical events) in the first solution pass of our example: the initial solution and the solutions found at iterations 5, 6 and 9 (with weights of 40, 37, 37 and 34, respectively).

Since the solution at iteration 9 happens to be optimal, we are interested in the effect of restarting before this solution is found. Assume we had chosen to restart after iteration 7, without yet reaching an optimal solution. Then the solutions that correspond to critical events are the initial solution and the solutions of iterations 5 and 6. We treat these three solutions in aggregate by combining their edges, to create a subgraph that consists of the edges (1,2), (1,4), (4,7), (6,7), (6,8), (8,9) and (6,9). (Frequency-based memory, as discussed in Section 4, refines this representation by accounting for the number of times each edge appears in the critical solutions, and allows the inclusion of additional weighting factors.)

To execute a restarting procedure, we penalize the inclusion of the edges of this subgraph at various steps of constructing the new solution. It is usually preferable to apply this penalty process at early steps, implicitly allowing the penalty function to decay rapidly as the number of steps increases. It is also sometimes useful to allow one or more intervening steps after applying such penalties before applying them again.

For our illustration, we will use the memory embodied in the subgraph of penalized edges by introducing a large penalty that effectively excludes all these edges from consideration on the first two steps of constructing the new solution. Then, because the construction involves four steps in total, we will not activate the critical event memory on subsequent construction steps, but will allow the method to proceed in its initial form.

Applying this approach, we restart the method by first choosing edge (3,5), which is the minimum weight edge not in the penalized subgraph. This choice and the remaining choices that generate the new starting solution are shown in Table 2.4.

Table 2.4 Restarting procedure.

Step	Candidates	Selection	Total Weight
1	(3,5)	(3, 5)	6
2	(2,3), (3,4), (3,6), (5,6), (5,9), (5,12)	(5, 9)	22
3	(2,3), (3,4), (3,6), (5,6), (5,12), (6,9), (8,9), (9,12)	(8, 9)	29
4	(2,3), (3,4), (3,6), (5,6), (5,12), (6,8), (6,9), (7,8), (8,10), (9,12)	(8, 10)	38

Beginning from the solution constructed in Table 2.4, and applying the first level TS procedure exactly as it was applied on the first pass, generates the sequence of solutions shown in Table

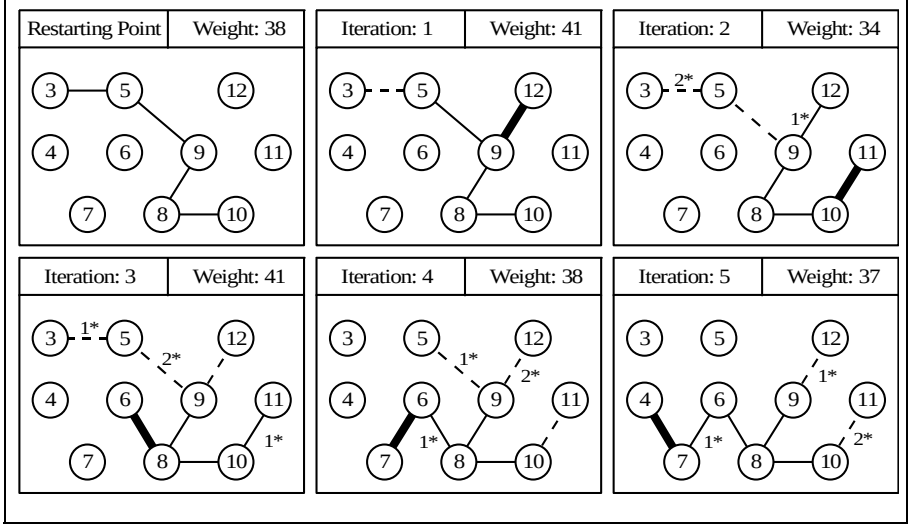
2.5 and depicted in Figure 2.7. (Again, we have arbitrarily limited the total number of iterations, in this case to 5.)

Table 2.5 TS iterations following restarting.

Iteration	Tabu-active net tenure		Add	Drop	Move Value	Weight
	1	2				
1			(9,12)	(3,5)	3	41
2	(9,12)	(3,5)	(10,11)	(5,9)	-7	34*
3	(3,5), (10,11)	(5,9)	(6,8)	(9,12)	7	41
4	(5,9), (6,8)	(9,12)	(6,7)	(10,11)	-3	38
5	(9,12), (6,7)	(10,11)	(4,7)	(8,10)	-1	37

It is interesting to note that the restarting procedure generates a better solution (with a total weight of 38) than the initial solution generated during the first construction (with a total weight of 40). Also, the restarting solution contains 2 "optimal edges" (i.e., edges that appear in the optimal tree). This starting solution allows the search trajectory to find the optimal solution in only two iterations, illustrating the benefits of applying an critical event memory within a restarting strategy. As will be seen in Section 4, related memory structures can also be valuable for strategies that drive the search into new regions by "partial restarting" or by directly continuing a current trajectory (with modified decision rules).

Fig. 2.7 Graphical representation of TS iterations after restarting.



Now we return from our example to examine elements of TS that take us beyond these first level concerns, and open up possibilities for creating more powerful solution approaches. We continue to focus primarily on short term aspects, and begin by discussing how to generalize the use of recency-based memory when neighborhood exploration is based on add/drop moves. From these foundations we then discuss issues of logical restructuring, tabu activation rules

and ways of determining tabu tenure. We then examine the important area of aspiration criteria, together with the role of influence

2.4 Recency-Based Memory for Add / Drop Moves

To understand procedurally how various forms of recency-based memory work, and to see their interconnections, it is useful to examine a convenient design for implementing the ideas illustrated so far. Such a design for the Min *k*-Tree problem creates a natural basis for handling a variety of other problems for which add/drop moves are relevant. In addition, the ideas can be adapted to settings that are quite different from those where add/drop moves are used.

As a step toward fuller generality, we will refer to items added and dropped as *elements*, though we will continue to make explicit reference to edges (as particular types of elements) within the context of the Min *k*-Tree problem example. (Elements are related to, but not quite the same as, solution attributes. The difference will be made apparent shortly.) There are many settings where operations of adding and dropping paired elements are the cornerstone of useful neighborhood definitions. For example, many types of exchange or swap moves can be characterized by such operations. Add/drop moves also apply to the omnipresent class of *multiple choice* problems, which require that exactly one element must be chosen from each member set from a specified disjoint collection. Add/drop moves are quite natural in this setting, since whenever a new element is chosen from a given set (and hence is "added" to the current solution), the element previously chosen from that set must be replaced (and hence "dropped"). Such problems are represented by discrete *generalized upper bound* (GUB) formulations in mathematical optimization, where various disjoint sets of 0-1 variables must sum to 1 (hence exactly one variable from each set must equal 1, and the others must equal 0). An add/drop move in this formulation consists of choosing a new variable to equal 1 (the "add move") and setting the associated (previously selected) variable equal to 0 (the "drop move").

Add/drop moves further apply to many types of problems that are not strictly discrete, that is, which contain variables whose values can varying continuously across specified ranges. Such applications arise by taking advantage of *basis exchange* (pivoting) procedures, such as the simplex method of linear programming. In this case, an add/drop move consists of selecting a new variable to enter (add to) the basis, and identifying an associated variable to leave (drop from) the basis. A variety of procedures for nonlinear and mixed integer optimization rely on such moves, and have provided a useful foundation for a number of tabu search applications. Additional related examples will be encountered throughout the course of this book.

2.4.1. Some Useful Notation

The approach used in the Min *k*-Tree problem can be conveniently described by means of the following notation. For a pair of elements that is selected to perform an add/drop move, let *Added* denote the element that is added, and *Dropped* the element that is dropped. Also denote the current iteration at which this pair is selected by *Iter*. We maintain a record of *Iter* to identify when *Added* and *Dropped* start to be tabu-active. Specifically, at this step we set:

TabuDropStart(*Added*) = *Iter*
TabuAddStart(*Dropped*) = *Iter.*

Thus, *TabuDropStart* records the iteration where *Added* becomes tabu-active (to prevent this element from later being dropped), and *TabuAddStart* records the iteration where *Dropped* becomes tabu-active (to prevent this element from later being added).

For example, in the Min *k*-Tree problem illustration of Table 2.3, where the edge (4,6) was added and the edge (4,7) was dropped on the first iteration, we would establish the record (for *Iter* = 1)

TabuDropStart(4,6) = 1
TabuAddStart(4,7) = 1

To identify whether or not an element is currently tabu-active, let *TabuDropTenure* denote the tabu tenure (number of iterations) to forbid an element to be dropped (once added), and let *TabuAddTenure* denote the tabu tenure to forbid an element from being added (once dropped). (In our Min *k*-Tree problem example of Section 2.2, we selected *TabuAddTenure* = 2 and *TabuDropTenure* = 1.)

As a point of clarification, when we speak of an element as being tabu-active, our terminology implicitly treats elements and attributes as if they are the same. However, to be precise, each element is associated with two different attributes, one where the element belongs to the current solution and one where the element does not. Elements may be viewed as corresponding to variables and attributes as corresponding to specific value assignments for such variables. There is no danger of confusion in the add/drop setting, because we always know when an element belongs or does not belong to the current solution, and hence we know which of the two associated attributes is currently being considered.

We can now identify precisely the set of iterations during which an element (i.e., its associated attribute) will be tabu-active. Let *TestAdd* and *TestDrop* denote a candidate pair of elements, whose members are respectively under consideration to be added and dropped from the current solution. If *TestAdd* previously corresponded to an element *Dropped* that was dropped from the solution and *TestDrop* previously corresponded to an element *Added* that was added to the solution (not necessarily on the same step), then it is possible that one or both may be tabu-active and we can check their status as follows. By means of the records established on earlier iterations, where *TestAdd* began to be tabu-active at iteration *TabuAddStart*(*TestAdd*) and *TestDrop* began to be tabu-active at iteration *TabuDropStart*(*TestDrop*), we conclude that as *Iter* grows the status of these elements will be given by:

TestAdd is tabu-active when:
$Iter \leq TabuAddStart(TestAdd) + TabuAddTenure$
TestDrop is tabu-active when:
$Iter \leq TabuDropStart(TestDrop) + TabuDropTenure$

Consider again the Min *k*-Tree problem illustration of Table 2.3. As previously noted, the move of Iteration 1 that added edge (4.6) and dropped edge (4,7) was accompanied by setting the *TabuDropStart*(4,6) = 1 and *TabuAddStart*(4,7) = 1, to record the iteration where these two edges start to be tabu-active (to prevent (4,6) from being dropped and (4,7) from being added). The edge (4,6) will then remain tabu-active on subsequent iterations, in the role of *TestDrop* (as a candidate to be dropped), as long as

$Iter \leq TabuDropStart(4,6) + TabuDropTenure.$

Hence, since we selected *TabuDropTenure* = 1 (to prevent an added edge from being dropped for 1 iteration), it follows that (4,6) remains tabu-active as long as

$$Iter \leq 2.$$

Similarly, having selected *TabuAddTenure* = 2, we see that the edge (4,7) remains tabu-active, to forbid it from being added back, as long as

$$Iter \leq 3.$$

An initialization step is needed to be sure that elements that have never been previously added or dropped from the solutions successively generated will not be considered tabu-active. This can be done by initially setting *TabuAddStart* and *TabuDropStart* equal to a large negative number for all elements. Then, as *Iter* begins at 1 and successively increases, the inequalities that determine the tabu-active status will not be satisfied, and hence will correctly disclose that an element is not tabu-active, until it becomes one of the elements *Added* or *Dropped*. (Alternately, *TabuAddStart* and *TabuDropStart* can be initialized at 0, and the test of whether an element is tabu-active can be skipped when it has a 0 value in the associated array.)

2.4.2 Streamlining

The preceding ideas can be streamlined to allow a more convenient implementation. First, we observe that the two arrays, *TabuAddStart* and *TabuDropStart*, which we have maintained separately from each other in to emphasize their different functions, can be combined into a single array *TabuStart*. The reason is simply that we can interpret *TabuStart*(*E*) to be the same as *TabuDropStart*(*E*) when the element *E* is in the current solution, and to be the same as *TabuAddStart*(*E*) when *E* is not in the current solution. (There is no possible overlap between these two states of *E*, and hence no danger of using the *TabuStart* array incorrectly.) Consequently, from now on, we will let the single array *TabuStart* take the role of both *TabuAddStart* and *TabuDropStart*. For example, when the move is executed that (respectively) adds and drops the elements *Added* and *Dropped*, the appropriate record consists of setting:

$$TabuStart(Added) = Iter$$
$$TabuStart(Dropped) = Iter.$$

The *TabuStart* array has an additional function beyond that of monitoring the status of tabu-active elements. (As shown in Section 4, this array is also useful for determining a type of frequency measure called a *residence frequency*.) However, sometimes it is convenient to use a different array, *TabuEnd*, to keep track of tabu-active status for recency-based memory, as we are treating here. Instead of recording when the tabu-active status starts, *TabuEnd* records when it ends. Thus, in place of the two assignments to *TabuStart* shown above, the record would consist of setting:

$$TabuEnd(Added) = Iter + TabuDropTenure$$
$$TabuEnd(Dropped) = Iter + TabuAddTenure.$$

(The element *Added* is now available to be dropped, and the element *Dropped* is now available to be added.) In conjunction with this, the step that checks for whether a candidate pair of elements *TestAdd* and *TestDrop* are currently tabu-active becomes:

TestAdd is tabu-active when:
Iter ≤ *TabuEnd*(*TestAdd*)
TestDrop is tabu-active when:
Iter ≤ *TabuEnd*(*TestDrop*).

This is a simpler representation than the one using *TabuStart*, and so it is appealing when *TabuStart* is not also used for additional purposes. (Also, *TabuEnd* can simply be initialized at 0 rather than at a large negative number.)

As will be discussed more fully in the next section, the values of *TabuAddTenure* and *TabuDropTenure* (which are explicitly referenced in testing tabu-active status with *TabuStart*, and implicitly referenced in testing this status with *TabuEnd*), are often preferably made variable rather than fixed. The fact that we use different tenures for added and dropped elements discloses that it can be useful to differentiate the tenures applied to elements of different classes. This type of differentiation can also be based on historical performance, as tracked by frequency-based measures. Consequently, tenures may be individually adjusted for different elements (as well as modified over time). Such adjustment can be quite effective in some settings (e.g., see Laguna, et al. 1995). These basic considerations can be refined to create effective implementations and also can be extended to handle additional move structures, as shown in Glover and Laguna (1997).

2.5 Tabu Tenure

In general, recency-based memory is managed by creating one or several tabu lists, which record the tabu-active attributes and implicitly or explicitly identify their current status. Tabu tenure can vary for different types or combinations of attributes, and can also vary over different intervals of time or stages of the search. This varying tenure makes it possible to create different kinds of tradeoffs between short term and longer term strategies. It also provides a dynamic and robust form of search.

The choice of appropriate types of tabu lists depends on the context. Although no single type of list is uniformly best for all applications, some guidelines can be formulated. If memory space is sufficient (as it often is) to store one piece of information (e.g., a single integer) for each solution attribute used to define the tabu activation rule, it is usually advantageous to record the iteration number that identifies when the tabu-active status of an attribute starts or ends as illustrated by the add/drop data structure described in Sections 2.3 and 2.4. This typically makes it possible to test the tabu status of a move in constant time. The necessary memory space depends on the attributes and neighborhood size, but it does not depend on the tabu tenure.

Depending on the size of the problem, it may not be feasible to implement the preceding memory structure in combination with certain types of attributes. In general, storing one piece of information for each attribute becomes unattractive when the problem size increases or attribute definition is complex. Sequential and circular tabu lists are used in this case, which store the identities of each tabu-active attribute, and explicitly (or implicitly, by list position) record associated tabu tenures.

Effective tabu tenures have been empirically shown to depend on the size of the problem instance. However, no single rule has been designed to yield an effective tenure for all classes of problems. This is partly because an appropriate tabu tenure depends on the strength of the tabu activation rule employed (where more restrictive rules are generally coupled with shorter

tenures). Effective tabu tenures and tabu activation rules can usually be determined quite easily for a given class of problems by a little experimentation. Tabu tenures that are too small can be recognized by periodically repeated objective function values or other function indicators, including those generated by hashing, that suggest the occurrence of cycling. Tenures that are too large can be recognized by a resulting deterioration in the quality of the solutions found (within reasonable time periods). Somewhere in between typically exists a robust range of tenures that provide good performance.

Once a good range of tenure values is located, first level improvements generally result by selecting different values from this range on different iterations. (A smaller subrange, or even more than one subrange, may be chosen for this purpose.) Problem structures are sometimes encountered where performance for some individual fixed tenure values within a range can be unpredictably worse than for other values in the range, and the identity of the isolated poorer values can change from problem to problem. However, if the range is selected to be good overall then a strategy that selects different tenure values from the range on different iterations typically performs at a level comparable to selecting one of the best values in the range, regardless of the problem instance.

Short term memory refinements subsequently discussed, and longer term considerations introduced in later sections, transform the method based on these constructions into one with considerable power. Still, it occasionally happens that even the initial short term approach by itself leads to exceptionally high quality solutions. Consequently, some of the TS literature has restricted itself only to this initial part of the method.

In general, short tabu tenures allow the exploration of solutions "close" to a local optimum, while long tenures can help to break free from the vicinity of a local optimum. These functions illustrate a special instance of the notions of *intensification* and *diversification* that will be explored in more detail later. Varying the tabu tenure during the search provides one way to induce a balance between closely examining one region and moving to different parts of the solution space.

In situations where a neighborhood may (periodically) become fairly small, or where a tabu tenure is chosen to be fairly large, it is entirely possible that iterations can occur when all available moves are classified tabu. In this case an *aspiration-by-default* is used to allow a move with a "least tabu" status to be considered admissible. Such situations rarely occur for most problems, and even random selection is often an acceptable form of aspiration-by-default. When tabu status is translated into a modified evaluation criterion, by penalties and inducements, then of course aspiration-by-default is handled automatically, with no need for to monitor the possibility that all moves are tabu.

There are several ways in which a *dynamic* tabu tenure can be implemented. These implementations may be classified into *random* and *systematic* dynamic tabu tenures.

2.5.1 Random Dynamic Tenure

Random dynamic tabu tenures are often given one of two forms. Both of these forms use a tenure range defined by parameters t_{min} and t_{max}. The tabu tenure t is randomly selected within this range, usually following a uniform distribution. In the first case, the chosen tenure is maintained constant for αt_{max} iterations, and then a new tenure is selected by the same process. The second form draws a new t for every attribute that becomes tabu at a given iteration. The first form requires more bookkeeping than the second one, because one must remember the last time that the tabu tenure was modified.

Either of the two arrays *TabuStart* or *TabuEnd* discussed in Section 2.4 can be used to implement these forms of dynamic tabu tenure. For example, a 2-dimensional array *TabuEnd* can be created to control a dynamic recency-based memory for the sequencing problem introduced at the beginning of this section. As in the case of the Min *k*-Tree problem, such an array can be used to record the time (iteration number) at which a particular attribute will be released from its tabu status. Suppose, for example, that t_{min} = 5 and t_{max} = 10 and that swaps of jobs are used to move from one solution to another in the sequencing problem. Also, assume that *TabuEnd*(*j,p*) refers to the iteration that job *j* will be released from a tabu restriction that prevents it from being assigned to position *p*. Then, if at iteration 30, job 8 in position 2 is swapped with job 12 in position 25, we will want to make the attribute (8,2) and (12,25) tabu-active for some number of iterations to prevent a move that will return one or both of jobs 8 and 12 from re-occupying their preceding positions. If *t* is assigned a value of 7 from the range t_{min} = 5 and t_{max} = 10, then upon making the swap at iteration 30 we may set *TabuEnd*(8,2) = 37 and *TabuEnd*(12,25) = 37.

This is not the only kind of *TabuEnd* array that can be used for the sequencing problem, and we examine other alternatives and their implications in Section 3. Nevertheless, we warn against a potential danger. An array *TabuEnd*(*i,j*) that seeks to prevent jobs *i* and *j* from exchanging positions, without specifying what these positions are, does not truly refer to attributes of a sequencing solution, and hence entails a risk if used to determine tabu status. (The pair (*i,j*) here constitutes an attribute of a move, in a lose sense, but does not serve to distinguish one solution from another.) Thus, if at iteration 30 we were to set *TabuEnd*(8,12) = 37, in order to prevent jobs 8 and 12 from exchanging positions until after iteration 37, this still might not prevent job 8 from returning to position 2 and job 12 from returning to position 25. In fact, a sequence of swaps could be executed that could return to precisely the same solution visited before swapping jobs 8 and 12.

Evidently, the *TabuEnd* array can be used by selecting a different *t* from the interval (t_{min}, t_{max}) at every iteration. As remarked in the case of the Min *k*-Tree problem, it is also possible to select *t* differently for different solution attributes.

2.5.2 Systematic Dynamic Tenure

Dynamic tabu tenures based on a random scheme are attractive for their ease of implementation. However, relying on randomization may not be the best strategy when specific information about the context is available. In addition, certain diversity-inducing patterns can be achieved more effectively by not restricting consideration to random designs. A simple form of *systematic* dynamic tabu tenure consists of creating a sequence of tabu search tenure values in the range defined by t_{min} and t_{max}. This sequence is then used, instead of the uniform distribution, to assign the current tabu tenure value. Suppose it is desired to vary *t* so that its value alternately increases and decreases. (Such a pattern induces a form of diversity that will rarely be achieved randomly.) Then the following sequence can be used for the range defined above:

$$\{ 5, 8, 6, 9, 7, 10 \}.$$

The sequence may be repeated as many times as necessary until the end of the search, where additional variation is introduced by progressively shifting and/or reversing the sequence before repeating it. (In a combined random/systematic approach, the decision of the shift value and the forward or backward direction can itself be made random.) Another variation is to retain a selected tenure value from the sequence for a variable number of iterations before

selecting the next value. Different sequences can be created and identified as effective for particular classes of problems.

The foregoing range of values (from 5 to 10) may seem relatively small. However, some applications use even smaller ranges, but adaptively, increase and decrease the midpoint of the range for diversification and intensification purposes. Well designed adaptive systems can significantly reduce or even eliminate the need to discover a best range of tenures by preliminary calibration. This is an important area of study.

These basic alternatives typically provide good starting tabu search implementations. In fact, most initial implementations apply only the simplest versions of these ideas.

2.6 Aspiration Criteria and Regional Dependencies

Aspiration criteria are introduced in tabu search to determine when tabu activation rules can be overridden, thus removing a tabu classification otherwise applied to a move. (The improved-best and aspiration-by-default criteria, as previously mentioned, are obvious simple instances.) The appropriate use of such criteria can be very important for enabling a TS method to achieve its best performance levels. Early applications employed only a simple type of aspiration criterion, consisting of removing a tabu classification from a trial move when the move yields a solution better than the best obtained so far. This criterion remains widely used. However, other aspiration criteria can prove effective for improving the search.

A basis for one of these criteria arises by introducing the concept of *influence,* which measures the degree of change induced in solution structure or feasibility. This notion can be illustrated for the Min *k*-Tree problem as follows. Suppose that the current solution includes edges (1,2), (1,4), (4,7) and (6,7), as illustrated in Figure 2.9, following. A high influence move, that significantly changes the structure of the current solution, is exemplified by dropping edge (1,2) and replacing it by edge (6,9). A low influence move, on the other hand, is exemplified by dropping edge (6,7) and adding edge (4,6). The weight difference of the edges in the high influence move is 15, while the difference is 9 for the low influence move. However, it is important to point out that differences on weight or cost are not the only — or even the primary — basis for distinguishing between moves of high and low influence. In the present example, the move we identify as a low influence move creates a solution that consists of the same set of nodes included in the current solution, while the move we identified as a high influence move includes a new node (number 9) from which new edges can be examined. (These moves correspond to those labeled static and dynamic in Figure 2.3.)

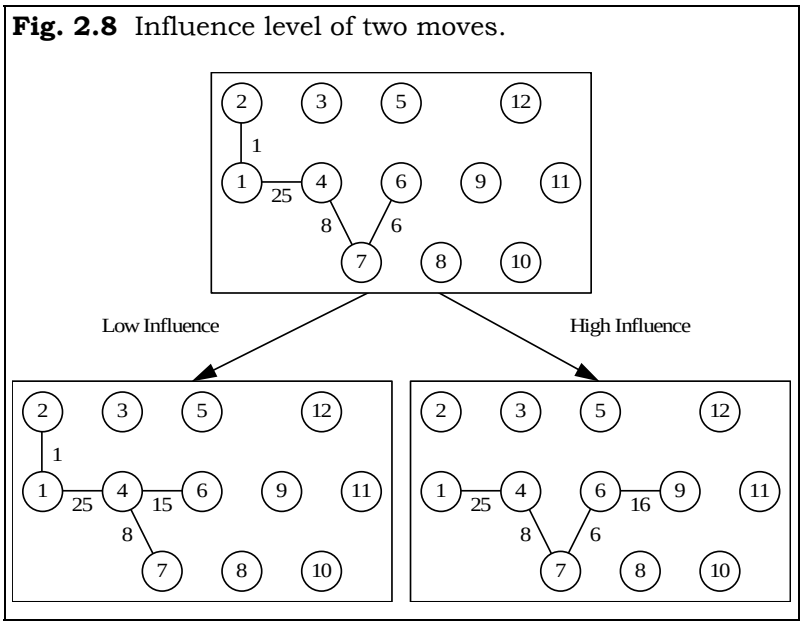


Fig. 2.8 Influence level of two moves.

As illustrated here, high influence moves may or may not improve the current solution, though they are less likely to yield an improvement when the current solution is relatively good. But high influence moves are important, especially during intervals of breaking away from local optimality, because a series of moves that is confined to making only small structural change is unlikely to uncover a chance for significant improvement. Executing the high influence move in Figure 2.8, for example, allows the search to reach the optimal edges (8,9) and (9,12) in subsequent iterations. Of course, moves of much greater influence than those shown can be constructed by considering compound moves. Such considerations are treated in later sections.

Influence often is associated with the idea of move distance. Although important, move influence is only one of several elements that commonly underlie the determination of aspiration criteria. We illustrate a few of these elements in Table 2.6.

Table 2.6 Illustrative aspiration criteria.

Aspiration by	*Description*	*Example*
Default	If all available moves are classified tabu, and are not render admissible by some other aspiration criteria, then a "least tabu" move is selected.	Revoke the tabu status of all moves with minimum *TabuEnd* value.
Objective	Global: A move aspiration is satisfied if the move yields a solution better than the best obtained so far.	Global: The best total tardiness found so far is 29. The current sequence is (4, 1, 5, 3, 6, 2) with T = 39. The move value of the tabu swap (5,2) is -20. Then, the tabu status of the swap is revoked and the search moves to the new best sequence (4, 1, 2, 3, 6, 5) with T = 19.
	Regional: A move aspiration is satisfied if the move yields a solution better than the best found in the region where the solution lies.	Regional: The best sequence found in the region defined by all sequences (1, 2, 3, *, *, *) is (1, 2, 3, 6, 4, 5) with $T = 31$. The current solution is (1, 4, 3, 2, 6, 5) with T = 23. The swap (4, 2) with move value of 6 is tabu. The tabu status is revoked because a new regional best (1, 2, 3, 4, 6, 5) with $T = 29$ can be found.
Search Direction	An attribute can be added and dropped from a solution (regardless of its tabu status), if the direction of the search (improving or nonimproving) has not changed.	For the Min *k*-Tree problem, the edge (11,12) has been recently dropped in the current improving phase making its addition a tabu-active attribute. The improving phase can continue if edge (11,12) is now added, therefore its tabu status may be revoked.
Influence	The tabu status of a low influence move may be revoked if a high influence move has been performed since establishing the tabu status for the low influence move.	If the low influence swap (1,4) described in Table 2.7 is classified tabu, its tabu status can be revoked after the high influence swap (4,5) is performed.

Aspirations such as those shown in Table 2.6 can be applied according to two implementation categories: aspiration by move and aspirations by attribute. A move aspiration, when satisfied, revokes the move's tabu classification. An attribute aspiration, when satisfied, revokes the attribute's tabu-active status. In the latter case the move may or may not change its tabu classification, depending on whether the tabu activation rule is triggered by more than one attribute. For example in our sequencing problem, if the swap of jobs 3 and 6 is forbidden because a tabu activation rule prevents job 3 from moving at all, then an attribute aspiration that revokes job 3's tabu-active status also revokes the move's tabu classification. However, if the swap (3,6) is classified tabu because both job 3 and job 6 are not allowed to move, then revoking job 3's tabu-active status does not result in overriding the tabu status of the entire move.

Different variants of the aspiration criteria presented in Table 2.6 are possible. For example, the regional aspiration by objective can be defined in terms of bounds on the objective function value. These bounds determine the region being explored, and they are modified to reflect the discovery of better (or worse) regions. Another possibility is to define regions with respect to time. For example, one may record the best solution found during the recent past (defined as a number of iterations) and use this value as the aspiration level.

2.7 Concluding Observations for the Min *k*-Tree Example

Influence of tabu tenures.

The tabu tenures used to illustrate the first level TS approach for the Min *k*-Tree problem of course are very small. The risk of using such tenures can be demonstrated in this example from the fact that changing the weight of edge (3,6) in Figure 2.2 from 20 to 17, will cause the illustrated TS approach with *TabuAddTenure* = 2 and *TabuDropTenure* = 1 to go into a cycle that will prevent the optimal solution from being found. The intuition that *TabuDropTenure* has a stronger influence than the *TabuAddTenure* for this problem is supported by the fact that the use of tenures of *TabuAddTenure* = 1 and *TabuDropTenure* = 2 in this case will avoid the cycling problem and allow an optimal solution to be found.

Alternative Neighborhoods

The relevance of considering alternative neighborhoods can be illustrated by reference to the following observation. For any given set of k+1 nodes, an optimal (min weight) k-tree over these nodes can always be found by using the greedy constructive procedure illustrated in Table 2.1 to generate a starting solution (restricted to these nodes) or by beginning with an arbitrary tree on these nodes and performing a succession of static improving moves (which do not change the node set). The absence of a static improving move signals that no better solution can be found on this set.

This suggests that tabu search might advantageously be used to guide the search over a "node-swap" neighborhood instead of an "edge-swap" neighborhood, where each move consists of adding a non-tree node i and dropping a tree node j, followed by finding a min weight solution on the resulting node set. (Since the tree node j may not be a leaf node, and the reconnections may also not make node i a leaf node in the new tree, the possibilities are somewhat different than making a dynamic move in the edge-swap neighborhood.) The tabu tenures may reasonably be defined over nodes added and dropped, rather than over edges added and dropped.

Critical event memory.

The type of critical event memory used in the illustration of restarting the TS approach in Section 2.3.1 may not be best. Generally it is reasonable to expect that the type of critical event memory used for restarting should be different from that used to continue the search from the current solution (when both are applied to drive the search into new regions). Nevertheless, a form that is popularly used in both situations consists of remembering *all* elements contained in solutions previously examined. One reason is that it is actually easier to maintain such memory than to keep track of elements that only occur in selected solutions. Also, instead of keeping track only of which elements occur in past solution, critical event memory is more usually designed to monitor the frequency that elements have appeared in past solutions. Such considerations are amplified in Section 4.

3. Additional Aspects of Short Term Memory

We began the discussion of short term memory for tabu search by contrasting the TS designs with those of memoryless strategies such as simple or iterated descent, and by pointing out how candidate list strategies are especially important for applying TS in the most effective ways. We now describe types of candidate list strategies that often prove valuable in tabu

search implementations. Then we examine the issues of logical restructuring, which provide important bridges to longer term considerations.

3.1 Tabu Search and Candidate List Strategies

The aggressive aspect of TS is manifest in choice rules that seek the best available move that can be determined with an appropriate amount of effort. As addressed in Section 2, the meaning of best in TS applications is customarily not limited to an objective function evaluation. Even where the objective function evaluation may appear on the surface to be the only reasonable criterion to determine the best move, the non-tabu move that yields a maximum improvement or least deterioration is not always the one that should be chosen. Rather, as we have noted, the definition of best should consider factors such as move influence, determined by the search history and the problem context.

For situations where $\mathbf{N}^*(x)$ is large or its elements are expensive to evaluate, candidate list strategies are essential to restrict the number of solutions examined on a given iteration. In many practical settings, TS is used to control a search process that may involve the solution of relatively complex subproblems by way of linear programming or simulation. Because of the importance TS attaches to selecting elements judiciously, efficient rules for generating and evaluating good candidates are critical to the search process. The purpose of these values is to isolate regions of the neighborhood containing moves with desirable features and to put these moves on a list of candidates for current examination.

Before describing the kinds of candidate list strategies that are particularly useful in tabu search implementations, we note that the efficiency of implementing such strategies often can be enhanced by using relatively straightforward memory structures to give efficient updates of move evaluations from one iteration to another. Appropriately coordinated, such updates can appreciably reduce the effort of finding best or near best moves.

In sequencing, for example, the move values often can be calculated without a full evaluation of the objective function. Intelligent updating can be useful even where candidate list strategies are not used. However, the inclusion of explicit candidate list strategies, for problems that are large, can significantly magnify the resulting benefits. Not only search speed but also solution quality can be influenced by the use of appropriate candidate list strategies. Perhaps surprisingly, the importance of such approaches is often overlooked.

3.2 Some General Classes of Candidate List Strategies

Candidate lists can be constructed from context related rules and from general strategies. In this section we focus on rules for constructing candidate lists that are context-independent. We emphasize that the effectiveness of a candidate list strategy should not be measured in terms of the reduction of the computational effort in a single iteration. Instead, a preferable measure of performance for a given candidate list is the quality of the best solution found given a specified amount of computer time. For example, a candidate list strategy intended to replace an exhaustive neighborhood examination may result in more iterations per unit of time, but may require many more iterations to match the solution quality of the original method. If the quality of the best solution found within a desirable time limit (or across a graduated series of such limits) does not improve, we conclude that the candidate list strategy is not effective.

3.2.1 Aspiration Plus

The Aspiration Plus strategy establishes a threshold for the quality of a move, based on the history of the search pattern. The procedure operates by examining moves until finding one that satisfies this threshold. Upon reaching this point, additional moves are examined, equal in number to the selected value *Plus,* and the best move overall is selected.

To assure that neither too few nor too many moves are considered, this rule is qualified to require that at least *Min* moves and at most *Max* moves are examined, for chosen values of *Min* and *Max.* The interpretation of *Min* and *Max* is as follows. Let *First* denote the number of moves examined when the aspiration threshold is first satisfied. Then if *Min* and *Max* were not specified, the total number of moves examined would be *First* + *Plus.* However, if *First* + *Plus* < *Min*, then *Min* moves are examined while if *First* + *Plus* > *Max*, then *Max* moves are examined. (This conditions may be viewed as imposing limits on the move that is "effectively" treated as the *First* move. For example, if as many as *Max* - *Plus* moves are examined without finding one that satisfies the aspiration threshold, then *First* effectively becomes the same as *Max* - *Plus*.)

This strategy is graphically represented in Figure 3.1. In this illustration, the fourth move examined satisfies the aspiration threshold and qualifies as *First.* The value of *Plus* has been selected to be 5, and so 9 moves are examined in total, selecting the best over this interval. The value of *Min*, set at 7, indicates that at least 7 moves will be examined even if *First* is so small that *First* + *Plus* < 7. (In this case, *Min* is not very restrictive, because it only applies if *First* < 2.) Similarly, the value of *Max*, set at 11, indicates that at most 11 moves will be examined even if *First* is so large that *First* + *Plus* > 11. (Here, *Max* is strongly restrictive.) The sixth move examined is the best found in this illustration.

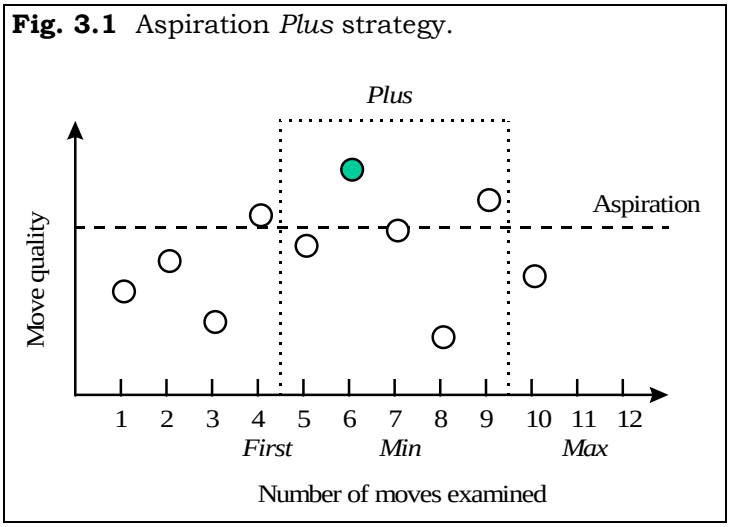


Fig. 3.1 Aspiration *Plus* strategy.

The "Aspiration" line in this approach is an established threshold that can be dynamically adjusted during the search. For example, during a sequence of improving moves, the aspiration may specify that the next move chosen should likewise be improving, at a level based on other recent moves and the current objective function value. Similarly, the values of *Min* and *Max* can be modified as a function of the number of moves required to meet the threshold.

During a nonimproving sequence the aspiration of the Aspiration Plus rule will typically be lower than during an improving phase, but rise toward the improving level as the sequence lengthens. The quality of currently examined moves can shift the threshold, as by encountering moves that significantly surpass or that uniformly fall below the threshold. As an elementary option, the threshold can simply be a function of the quality of the initial *Min* moves examined on the current iteration.

The Aspiration Plus strategy includes several other strategies as special cases. For example, a *first improving* strategy results by setting *Plus* = 0 and directing the aspiration threshold to accept moves that qualify as improving, while ignoring the values of *Min* and *Max*. Then *First* corresponds to the first move that improves the current value of the objective, if such a move can be found. A slightly more advanced strategy can allow *Plus* to be increased or decreased according to the variance in the quality of moves encountered from among some initial number examined. In general, in applying the Aspiration Plus strategy, it is important to assure on each iteration that new moves are examined which differ from those just reviewed. One way of achieving this is to create a circular list and start each new iteration where the previous examination left off.

3.2.2 Elite Candidate List

The Elite Candidate List approach first builds a Master List by examining all (or a relatively large number of) moves, selecting the k best moves encountered, where k is a parameter of the process. Then at each subsequent iteration, the current best move from the Master List is chosen to be executed, continuing until such a move falls below a given quality threshold, or until a given number of iterations have elapsed. Then a new Master List is constructed and the process repeats. This strategy is depicted in Figure 3.2, below.

This technique is motivated by the assumption that a good move, if not performed at the present iteration, will still be a good move for some number of iterations. More precisely, after an iteration is performed, the nature of a recorded move implicitly may be transformed. The assumption is that a useful proportion of these transformed moves will inherit attractive properties from their antecedents.

The evaluation and precise identity of a given move on the list must be appropriately monitored, since one or both may change as result of executing other moves from the list. For example, in the Min k-Tree problem the evaluations of many moves can remain unchanged from one iteration to the next. However, the identity and evaluation of specific moves will change as a result of deleting and adding particular edges, and these changes should be accounted for by appropriate updating (applied periodically if not at each iteration). An Elite Candidate List strategy can be advantageously extended by a variant of the Aspiration Plus strategy, allowing some additional number of moves outside the Master List to be examined at each iteration, where those of sufficiently high quality may replace elements of the Master List.

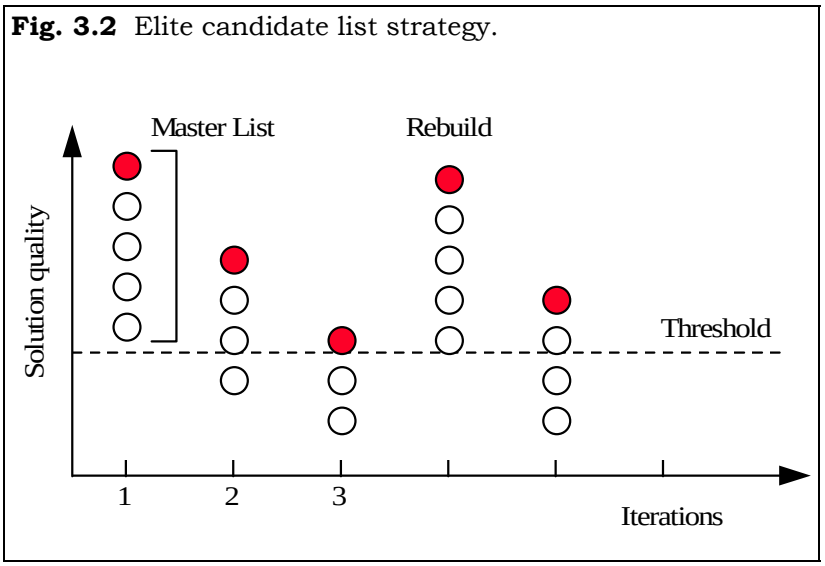


Fig. 3.2 Elite candidate list strategy.

3.2.3 Successive Filter Strategy

Moves can often be broken into component operations, and the set of moves examined can be reduced by restricting consideration to those that yield high quality outcomes for each operation separately. For example, the choice of an exchange move that includes an "add component" and a "drop component" may restrict attention only to exchanges created from a relatively small subset of "best add" and "best drop" components. The gain in efficiency can be considerable. If there are 100 add possibilities and 100 drop possibilities, the number of add/drop combinations is 10,000. However, by restricting attention to the 8 best add and drop moves, considered independently, the number of combinations to examine is only 64. (Values of 8 and even smaller have been found effective in some practical applications.)

The evaluations of the separate components often will give only approximate information about their combined evaluation. Nevertheless, if this information is good enough to insure a significant number of the best complete moves will result by combining these apparently best components, then the approach can yield quite good outcomes. Improved information may be obtained by sequential evaluations, as where the evaluation of one component is conditional upon the prior (restricted) choices of another. Such strategies of subdividing compound moves into components, and then restricting consideration of complete compound moves only to those assembled from components that pass selected thresholds of quality, have proved quite effective in TS methods for partitioning problems and for telecommunication channel balancing problems.

Conditional uses of component evaluations are also relevant for sequencing problems, where a measure can be defined to identify preferred attributes using information such as due dates, processing times, and delay penalties. If swap moves are being used, then some jobs are generally better candidates than others to move early or later in the sequence. The candidate list considers those swaps whose composition includes at least one of these preferred attributes.

In the context of the traveling salesman problem, good solutions are often primarily composed of edges that are among the 20 to 40 shortest edges meeting one of their endpoints (depending

on various factors). Some studies have attempted to limit consideration entirely to tours constructed from such a collection of edges. The successive filter strategy, by contrast, offers greater flexibility by organizing moves that do not have to be entirely composed of such special elements, provided one or more of these elements is incorporated as part of the move. This approach can be frequently controlled to require little more time than the more restricted standard approach, while affording a more desirable set of alternatives to consider.

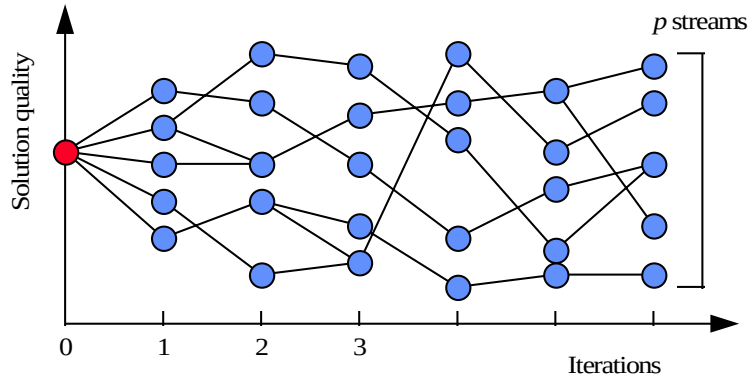
3.2.4 Sequential Fan Candidate List

A type of candidate list that is highly exploitable by parallel processing is the sequential fan candidate list. The basic idea is to generate some p best alternative moves at a given step, and then to create a fan of solution streams, one for each alternative. The several best available moves for each stream are again examined, and only the p best moves overall (where many or no moves may be contributed by a given stream) provide the p new streams at the next step.

In the setting of tree search methods such a sequential fanning process is sometimes called *beam search.* For use in the tabu search framework, TS memory and activation rules can be carried forward with each stream and hence inherited in the selected continuations. Since a chosen solution can be assigned to more than one stream, different streams can embody different missions in TS. Alternatively, when two streams merge into the same solution other streams may be started by selecting a neighbor adjacent to one of the current streams.

The process is graphically represented in Figure 3.3. Iteration 0 constructs an initial solution or alternatively may be viewed as the starting point for constructing a solution. That is, the sequential fan approach can be applied using one type of move to create a set of initial solutions, and then can continue using another type of move to generate additional solutions. (We thus allow a "solution" to be a partial solution as well as a complete solution.) The best moves from this solution are used to generate p streams. Then at every subsequent iteration, the overall best moves are selected to lead the search to p different solutions. Note that since more than one move may lead the search to the same solution, more than p moves may be necessary to continue the exploration of p distinct streams.

Fig. 3.3 Sequential fan candidate list.



A more intensive form of the sequential fan candidate list approach, which is potentially more powerful but requires more work, is to use the process illustrated in Figure 3.3 as a "look

ahead" strategy. In this case a limit is placed on the number of iterations that the streams are generated beyond iteration 0. Then the best outcome at this limiting iteration is used to identify a "best current move" (a single first branch) from iteration 0. Upon executing this move, the step shown as iteration 1 in Figure 3.3 becomes the new iteration 0, that is, iteration 0 always corresponds to the current iteration. Then this solution becomes the source of p new streams, and the process repeats.

There are a number of possible variants of this sequential fan strategy. For example, instead of selecting a single best branch at the limiting iteration, the method can select a small number of best branches, and thus give the method a handful of candidates from which to generate p streams at the new iteration 0.

The iteration limit that determines depth of the look ahead can be variable, and the value of p can change at various depths. Also the number of successors of a given solution that are examined to determine candidates for the p best continuations can be varied as by progressively reducing this number at greater depths.

The type of staging involved in successive solution runs of each stream may be viewed as a means of defining levels in the context of the Proximate Optimality Principle commonly associated with the strategic oscillation component of tabu search. Although we will study this principle in more detail later, we remark that the sequential fan candidate list has a form that is conveniently suited to exploit it.

3.2.5 Bounded Change Candidate List

A bounded change candidate list strategy is relevant in situations where an improved solution can be found by restricting the domain of choices so that no solution component changes by more than a limited degree on any step. A bound on this degree, expressed by a distance metric appropriate to the context, is selected large enough to encompass possibilities considered strategically relevant. The metric may allow large changes along one dimension, but limit the changes along another so that choices can be reduced and evaluated more quickly. Such an approach offers particular benefits as part of an intensification strategy based on decomposition, where the decomposition itself suggests the limits for bounding the changes considered.

3.3 Connections Between Candidate Lists, Tabu Status and Aspiration Criteria

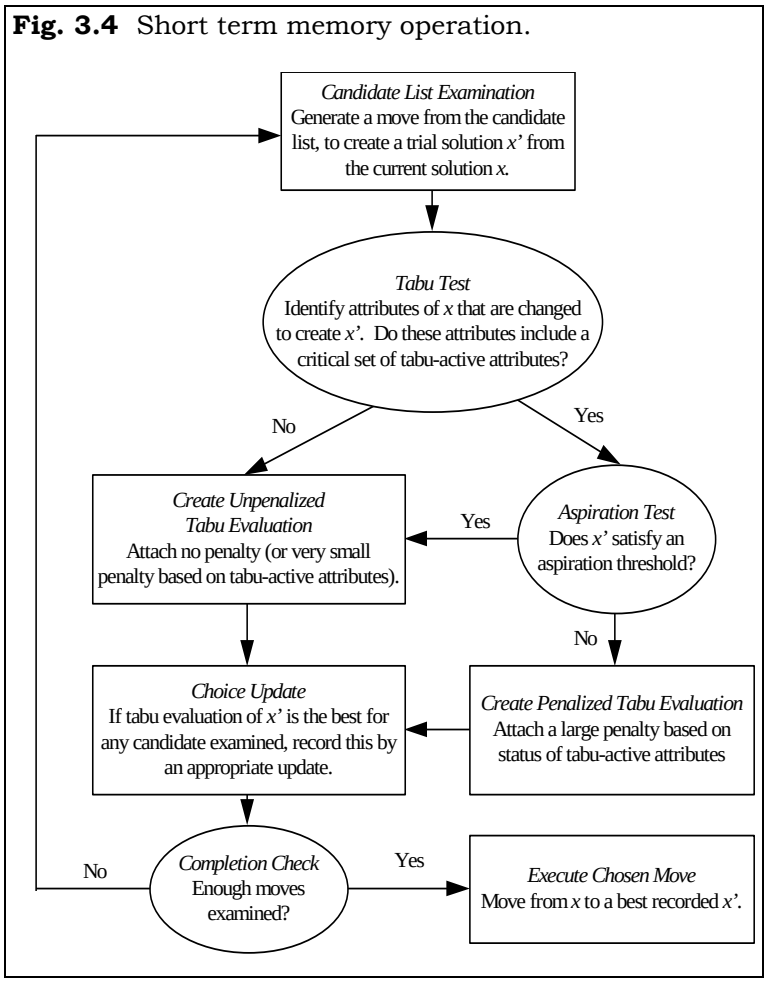
It is useful to summarize the short term memory considerations embodied in the interaction between candidate lists, tabu status and aspiration criteria. The operations of these TS short term elements are shown in Figure 3.4. The representation of penalties in Figure 3.4 either as "large" or "very small" expresses a thresholding effect: either the tabu status yields a greatly deteriorated evaluation or else it chiefly serves to break ties among solutions with highest evaluations. Such an effect of course can be modulated to shift evaluations across levels other than these extremes. If all moves currently available lead to solutions that are tabu (with evaluations that normally would exclude them from being selected), the penalties result in choosing a "least tabu" solution.

The sequence of the *tabu test* and the *aspiration test* in Figure 3.4 evidently can be reversed (that is, by employing the tabu test only if the aspiration threshold is not satisfied). Also, the tabu evaluation can be modified by creating inducements based on the aspiration level, just as

it is modified by creating penalties based on tabu status. In this sense, aspiration conditions and tabu conditions can be conceived roughly as "mirror images" of each other.

For convenience Figure 3.4 expresses tabu restrictions solely in terms of penalized evaluations, although we have seen that tabu status is often permitted to serve as an all-or-none threshold, without explicit reference to penalties and inducements (by directly excluding tabu options from being selected, subject to the outcome of aspiration tests). Whether or not modified evaluations are explicitly used, the selected move may not be the one with the best objective function value, and consequently the solution with the best objective function value encountered throughout the search history is recorded separately.

Fig. 3.4 Short term memory operation.



3.4 Logical Restructuring

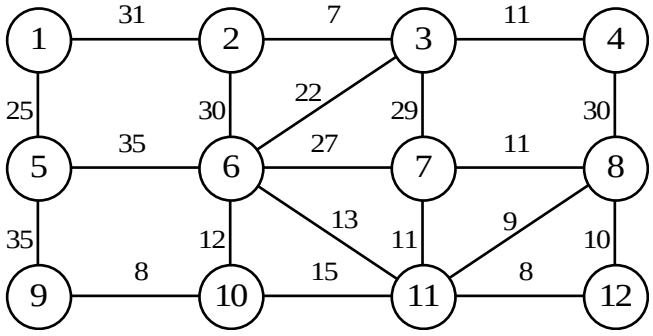
Logical restructuring is an important element of adaptive memory solution approaches, which gives a connection between short and long term strategies. Logical restructuring is implicit in strategic oscillation and path relinking, which we examine in subsequent sections, but its role and significance in these strategies is often overlooked. By extension, the general usefulness of logical restructuring is also often not clearly understood. We examine some of its principal

features before delving into longer term considerations, and show how it can also be relevant for improving the designs of short term strategies.

Logical restructuring emerges as a way to meet the combined concerns of quality and influence. Its goal is to exploit the ways in which influence (structural, local and global) can uncover improved routes to high quality solutions. For this purpose, a critical step is to re-design standard strategies to endow them with the power to ferret out opportunities otherwise missed. This step particularly relies on integrating two elements: (1) the identification of changes that satisfy properties that are essential (and limiting) in order to achieve improvement, in contrast to changes that simply depart from what has previously been seen; (2) the use of anticipatory ("means-ends") analysis to bring about such essential changes. Within the context of anticipatory analysis, logical restructuring seeks to answer the following questions: "What conditions assure the existence of a trajectory that will lead to an improved solution?" and "What intermediate moves can create such conditions?" The "intermediate moves" of the second question may be generated either by modifying the evaluations used to select transitions between solutions or by modifying the neighborhood structure that determines these transitions.

To illustrate the relevant considerations, we return again to the example of the Min *k*-Tree problem discussed in previous sections. We replace the previous graph by the one shown in Figure 3.5, but continue to consider the case of $k = 4$.

Fig. 3.5 Illustrative Min *k*-Tree Problem.



The same rules to execute a first-level tabu search approach as in our earlier illustrations (including the rules for generating a starting solution) produces a sequence of steps that quickly reaches the vicinity of the optimal solution, but requires some effort actually find this solution. In fact, it is readily verified that applying these rules will cause all edges of the optimal solution except one, edge (10,11), to be contained in the union of the two solutions obtained on iterations 4 and 5. Yet an optimal solution will not be found until iteration 11.

This delayed process of finding a route to an optimal solution (which can be greatly magnified for larger or more complex problems) can be substantially accelerated by means of logical restructuring. More generally, such restructuring can make it possible to uncover fertile options that can otherwise be missed entirely.

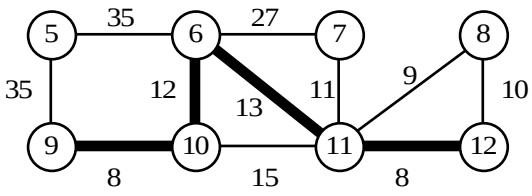
3.4.1 Restructuring by Changing Evaluations and Neighborhoods

The first type of logical restructuring we illustrate makes use both of modified evaluations and an amended neighborhood structure. As pointed out in Section 2.2 earlier, the swap moves we have employed for the Min *k*-Tree problem may be subdivided into two types: *static* swaps, which leave the nodes of the current tree unchanged, and *dynamic* swaps, which replace one of the nodes currently in the tree with another that is not in the tree. This terminology was chosen to reflect the effect that each swap type has on the nodes of the tree. Since dynamic swaps in a sense are more influential, we give them special consideration. We observe that a dynamic swap can select an edge to be dropped only if it is a *terminal edge* — i.e., one that meets a *leaf node* of the tree, which is a node that is met by only a single tree edge (the terminal edge).

Although it is usually advantageous to drop an edge with a relatively large weight, this may not be possible. Thus, we are prompted to consider an "anticipatory goal" of making moves that cause more heavily weighted edges to become terminal edges, and hence eligible to be dropped. By this means, static swaps can be used to set up desirable conditions for dynamic swaps.

The solution obtained at iteration 4 of the process for solving the example problem of Figure 3.5 gives a basis for showing what is involved. We clarify the situation by showing the current solution at this iteration in Figure 3.6 (without bothering to identify the solutions obtained at other iterations), where edges contained in the current tree are shown as heavy edges and the candidate edges to add to the tree are shown as light edges.

Fig. 3.6 Solution and candidate edges to add to iteration 4 tree.



The move that changes the tree at iteration 4 to that of iteration 5 — if the rules illustrated in Section 2 are used — is a dynamic swap that adds edge (8,11) with a weight of 9 and drops edge (9,10) with a weight of 8. We make use of information contained in this choice to construct a more powerful move using logical restructuring, as follows.

Having identified (8,11) as a candidate to be added, the associated anticipatory goal is to identify a static swap that will change a larger weight edge into a terminal edge. Specifically, the static swap that adds edge (10,11) and drops edge (6,10), with a move value of 3, produces a terminal edge from the relatively high weight edge (6,11) (which has a weight of 13). Since the candidate edge (8,11) to be added has a weight of 9, the result of joining the indicated static swap with the subsequent dynamic swap (that respectively adds and drops (8,11) and (6,11)) will be a net gain. (The static move value of 3 is joined with the dynamic move value of -4, yielding a result of -1.)

Effectively, such anticipatory analysis leads to a way to extract a fruitful outcome from a relatively complex set of options by focusing on a simple set of features. It would be possible to

find the same outcome by a more ponderous approach that checks all sequences in which a dynamic move follows a static move. This requires a great deal of computational effort — in fact, considerably more than involved in the approach without logical restructuring that succeeded in finding an optimal solution at iteration 11 (considering the trade-off between number of iterations and work per iteration).

By contrast, the use of logical restructuring allows the anticipatory analysis to achieve the benefits of a more massive exploration of alternatives, but without incurring the burden of undue computational effort. In this example, the restructuring is accomplished directly as follows. First, it is only necessary to identify the two best edges to add for a dynamic swap (independent of matching them with an edge to drop), subject to requiring that these edges meet different nodes of the tree. (In the tree of iteration 4, seen in Figure 3.6, these two edges are (8,11) and (8,12).) Then at the next step, during the process of looking at candidate static swaps, a modified "anticipatory move value" is created for each swap that creates a terminal edge, by subtracting the weight of this edge from the standard move value.

This gives all that is needed to find (and evaluate) a best "combined move sequence" of the type we are looking for. In particular, every static move that generates a terminal edge can be combined with a dynamic move that drops this edge and then adds one of the two "best edges" identified in first of the two preceding steps. Hence, the restructuring is completed by adding the anticipatory move value to the weight of one of these two edges (appropriately identified) thereby determining a best combined move. The illustrated process therefore achieves restructuring in two ways — by modifying customary move values and by fusing certain sequences of moves into a single compound move.

Although this example appears on the surface to be highly problem specific, its basic features are shared by applications that arise in a variety of problem settings. Later the reader will see how variants of logical restructuring embodied in this illustration are natural components of the strategies of path relinking and ejection chain constructions.

3.4.2 Threshold Based Restructuring and Induced Decomposition

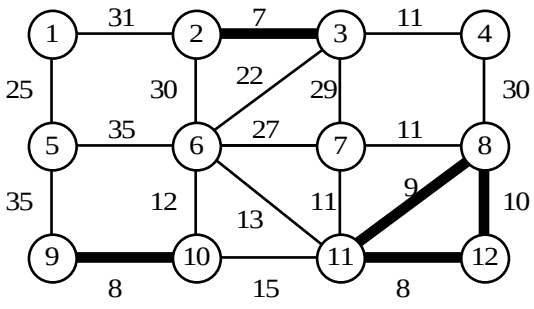
The second mode of logical restructuring that we illustrate by reference to the Min *k*-Tree problem example is more complex (in the sense of inducing a more radical restructuring), but relatively easy to sketch and also potentially more powerful.

Consider again the solution produced at iteration 4. This is a local optimum and also the best solution found up to the current stage of search. We seek to identify a property that will be satisfied by at least one solution that has a smaller weight than the weight of this solution (41), and which will impose useful limits on the composition of such a solution. A property that in fact must be shared by all "better" solutions can be expressed as a threshold involving the average weight of the tree edges. This average weight must be less than the threshold value of 41/4 (i.e., 10 1/4). Since some of the edges in any improved solution must have weights less than this threshold, we are motivated to identify such "preferred" edges as a foundation for a restructured form of the solution approach. In this type of restructuring, we no longer confine attention to swap moves, but look for ways to link the preferred edges to produce an improved solution. (Such a restructuring can be based on threshold values derived from multiple criteria.)

When the indicated strategy is applied to the present example, a large part of the graph is eliminated, leaving only 3 separate connected components: (a) the edge (2,3), (b) the edge (9,10), and (c) the three edges (8,11), (8,12) and (11,12). The graph that highlights these

components is shown in Figure 3.7. At this point a natural approach is to link such components by shortest paths, and then shave off terminal edges if the trees are too large, before returning to the swapping process. Such an approach will immediately find the optimal solution that previously was not found until iteration 11.

Fig. 3.7 Threshold generated components.



This second illustrated form of restructuring is a fundamental component of the strategic oscillation approach which we describe in more detail in the next section. A salient feature of this type of restructuring is its ability to create an *induced decomposition* of either the solution space or the problem space. This outcome, coupled with the goal of effectively joining the decomposed components to generate additional solution alternatives, is also a basic characteristic of path relinking, which is also examined in the next section. More particularly, the special instance of path relinking known as vocabulary building, which focuses on assembling fragments of solutions into larger units, offers a direct model for generalizing the "threshold decomposition" strategy illustrated here.

In some applications, specific theorems can be developed about the nature of optimal solutions and can be used to provide relevant designs for restructuring. The Min *k*-Tree problem is one for which such a theorem is available (Glover and Laguna, 1997). Interestingly, the second form of restructuring we have illustrated, which is quite basic, exploits several aspects of this theorem — although without "knowing" what the theorem is. In general, logical restructuring and the TS strategies such as path relinking and strategic oscillation which embody it, appear to behave as if they similarly have a capacity to exploit underlying properties of optimal solutions in broader contexts — contexts whose features are not sufficiently uniform or easily characterized to permit the nature of optimal solutions to be expressed in the form of a theorem.

4. Longer Term Memory

In some applications, the short term TS memory components are sufficient to produce very high quality solutions. However, in general, TS becomes significantly stronger by including longer term memory and its associated strategies. In the longer term TS strategies, the modified neighborhood produced by tabu search may contain solutions not in the original one, generally consisting of selected elite solutions (high quality local optima) encountered at various points in the solution process. Such elite solutions typically are identified as elements of a regional cluster in intensification strategies, and as elements of different clusters in diversification strategies. In addition, elite solution components, in contrast to the solutions

themselves, are included among the elements that can be retained and integrated to provide inputs to the search process.

Perhaps surprisingly, the use of longer term memory does not require long solution runs before its benefits become visible. Often its improvements begin to be manifest in a relatively modest length of time, and can allow solution efforts to be terminated somewhat earlier than otherwise possible, due to finding very high quality solutions within an economical time span. The fastest methods for some types of routing and scheduling problems, for example, are based on including longer term TS memory. On the other hand, it is also true that the chance of finding still better solutions as time grows — in the case where an optimal solution is not already found — is enhanced by using longer term TS memory in addition to short term memory.

4.1 Frequency-Based Approach

Frequency-based memory provides a type of information that complements the information provided by recency-based memory, broadening the foundation for selecting preferred moves. Like recency, frequency often is weighted or decomposed into subclasses by taking account of the dimensions of solution quality and move influence. Also, frequency can be integrated with recency to provide a composite structure for creating penalties and inducements that modify move evaluations. (Although recency-based memory is often used in the context of short term memory, it can also be a foundation of longer term forms of memory.)

For our present purposes, we conceive frequencies to consist of ratios, whose numerators represent counts expressed in two different measures: a *transition measure* — the number of iterations where an attribute changes (enters or leaves) the solutions visited on a particular trajectory, and a *residence measure* — the number of iterations where an attribute belongs to solutions visited on a particular trajectory, or the number of instances where an attribute belongs to solutions from a particular subset. The denominators generally represent one of three types of quantities: (1) the total number of occurrences of all events represented by the numerators (such as the total number of associated iterations), (2) the sum (or average) of the numerators, and (3) the maximum numerator value. In cases where the numerators represent weighted counts, some of which may be negative, denominator (3) is expressed as an absolute value and denominator (2) is expressed as a sum of absolute values (possibly shifted by a small constant to avoid a zero denominator). The ratios produce *transition frequencies* that keep track of how often attributes change, and *residence frequencies* that keep track of how often attributes are members of solutions generated. In addition to referring to such frequencies, thresholds based on the numerators alone can be useful for indicating when phases of greater diversification are appropriate. (The thresholds for particular attributes can shift after a diversification phase is executed.)

Residence frequencies and transition frequencies sometimes convey related information, but in general carry different implications. They are sometimes confused (or treated identically) in the literature. A noteworthy distinction is that residence measures, by contrast to transition measures, are not concerned with the characteristics of a particular solution attribute or whether it is an attribute that changes in moving from one solution to another. For example in the Min k-Tree problem, a residence measure may count the number of times edge (i,j) was part of the solution, while a transition measure may count the number of times edge (i,j) was added to the solution. (More complex joint measures, such as the number of times edge (i,j) was accompanied in the solution by edge (k,l), or was deleted from the solution in favor of edge (k,l), can also selectively be generated. Such frequencies relate to the issues of creating more complex attributes out of simpler ones, and to the strategies of vocabulary building.)

A high residence frequency may indicate that an attribute is highly attractive if the domain consists of high quality solutions, or may indicate the opposite, if the domain consists of low quality solutions. On the other hand, a residence frequency that is high (or low) when the domain is chosen to include both high and low quality solutions may point to an entrenched (or excluded) attribute that causes the search space to be restricted, and that needs to be jettisoned (or incorporated) to allow increased diversity. For example, an entrenched attribute may be a job that is scheduled in the same position during a sequence of iterations that include both low and high quality objective function evaluations.

As a further useful distinction, a high transition frequency, in contrast to a high residence frequency, may indicate an associated attribute is a "crack filler," that shifts in and out of solutions to perform a fine tuning function. In this context, a transition frequency may be interpreted as a measure of volatility. For example, the Min k-Tree problem instance in Figure 2.2 of Section 2 contains a number of edges whose weight may give them the role of crack fillers. Specifically, edges (3,5) and (6,7) both have a weight of 6, which makes them attractive relative to other edges in the graph. Since these edges are not contained in an optimal solution, there is some likelihood that they may repeatedly enter and leave the current solution in a manner to lure the search away from the optimal region. In general, crack fillers are determined not simply by cost or quality but by structure, as in certain forms of connectivity. (Hence, for example, the edge (3,5) of Figure 2.2 does not repeatedly enter and leave solutions in spite of its cost.) Some subset of such elements is also likely to be a part of an optimal solution. This subset can typically be identified with much less difficulty once other elements are in place. On the other hand, a solution (full or partial) may contain the "right" crack fillers but offer little clue as to the identity of the other attributes that will transform the solution into one that is optimal.

We use a sequencing problem and the Min k-Tree problem as contexts to further illustrate both residence and transition frequencies. Only numerators are indicated, understanding that denominators are provided by the conditions (1) to (3) previously defined. The measures are given in Table 4.1.

Table 4.1. Example of frequency measures.

Problem	*Residence Measure*	*Transition Measure*
Sequencing	Number of times job j has occupied position $\pi(j)$.	Number of times job i has exchanged positions with job j.
	Sum of tardiness of job j when this job occupies position $\pi(j)$.	Number of times job j has been moved to an earlier position in the sequence.
Min k-Tree Problem	Number of times edge (i, j) has been part of the current solution.	Number of times edge (i, j) has been deleted from the current solution when edge (k, l) has been added.
	Sum of total solution weight when edge (i, j) is part of the solution.	Number of times edge (i, j) has been added during improving moves.

Attributes that have greater frequency measures, just as those that have greater recency measures (i.e., that occur in solutions or moves closer to the present), can trigger a tabu activation rule if they are based on consecutive solutions that end with the current solution.

However, frequency-based memory often finds its most productive use as part of a longer term strategy, which employs incentives as well as restrictions to determine which moves are selected. In such a strategy, tabu activation rules are translated into evaluation penalties, and incentives become evaluation enhancements, to alter the basis for qualifying moves as attractive or unattractive.

To illustrate, in a scheduling setting where a swap neighborhood is used, an attribute such as a job j with a high residence frequency in position $\pi(j)$ may be assigned a strong incentive ("profit") to serve as a *swap attribute,* thus resulting in the choice of a move that yields a new sequence π' with $\pi'(j) \neq \pi(j)$. Such an incentive is particularly relevant in the case where the *TabuEnd* value of job j is small compared to the current iteration, since this value (minus the corresponding tabu tenure) identifies the latest iteration that job j was a *swap attribute*, and hence discloses that job j has occupied position $\pi(j)$ in every solution since.

Frequency-based memory therefore is usually applied by introducing graduated tabu states, as a foundation for defining penalty and incentive values to modify the evaluation of moves. A natural connection exists between this approach and the recency-based memory approach that creates tabu status as an all-or-none condition. If the tenure of an attribute in recency-based memory is conceived as a conditional threshold for applying a very large penalty, then the tabu classifications produced by such memory can be interpreted as the result of an evaluation that becomes strongly inferior when the penalties are activated. Conditional thresholds are also relevant to determining the values of penalties and incentives in longer term strategies. Most applications at present, however, use a simple linear multiple of a frequency measure to create a penalty or incentive term. The multiplier is adjusted to create the right balance between the incentive or penalty and the cost (or profit) coefficients of the objective function.

4.2 Intensification 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. A simple instance of this second type of intensification strategy is shown in Figure 4.1. The strategy for selecting elite solutions is italicized in Figure 4.1 due to its importance. Two variants have proved quite successful. One introduces a diversification measure to assure the solutions recorded differ from each other by a desired degree, and then erases all short term memory before resuming from the best of the recorded solutions. A diversification measure may be related to the number of moves that are necessary to transform one solution into another. Or the measure may be defined independently from the move mechanism. For example, in sequencing, two solutions may be considered diverse if the number of swaps needed to move from one to the other is "large." On the other hand, the diversification measure may be the number of jobs that occupy a different position in the two sequences being compared. (This shows that intensification and diversification often work together, as elaborated in the next section.)

Fig. 4.1 Simple TS intensification approach.

Apply TS short term memory.
Apply an *elite selection strategy.*
do {
    Choose one of the elite solutions.
    Resume short term memory TS from chosen solution.
    Add new solutions to elite list when applicable.
} **while** (iterations < limit and list not empty)

The second variant that has also proved successful, keeps a bounded length sequential list that adds a new solution at the end only if it is better than any previously seen. The current last member of the list is always the one chosen (and removed) as a basis for resuming search. However, TS short term memory that accompanied this solution is also saved, and the first move also forbids the move previously taken from this solution, so that a new solution path will be launched.

A third variant of the approach of Figure 4.1 is related to a strategy that resumes the search from unvisited neighbors of solutions previously generated. Such a strategy keeps track of the quality of these neighbors to select an elite set, and restricts attention to specific types of solutions, such as neighbors of local optima or neighbors of solutions visited on steps immediately before reaching such local optima. This type of "unvisited neighbor" strategy has been little examined. It is noteworthy, however, that the two variants previously indicated have provided solutions of remarkably high quality.

Another type of intensification approach is *intensification by decomposition*, where restrictions may be imposed on parts of the problem or solution structure in order to generate a form of decomposition that allows a more concentrated focus on other parts of the structure. A classical example is provided by the traveling salesman problem, where edges that belong to the intersection of elite tours may be "locked into" the solution, in order to focus on manipulating other parts of the tour. The use of intersections is an extreme instance of a more general strategy for exploiting frequency information, by a process that seeks to identify and constrain the values of *strongly determined* and *consistent variables*. We discuss the identification and use of such variables in Section 4.4.1.

Intensification by decomposition also encompasses other types of strategic considerations, basing the decomposition not only on indicators of strength and consistency, but also on opportunities for particular elements to interact productively. Within the context of a permutation problem as in scheduling or routing, for example, where solutions may be depicted as selecting one or more sequences of edges in a graph, a decomposition may be based on identifying subchains of elite solution, where two or more subchains may be assigned to a common set if they contain nodes that are "strongly attracted" to be linked with nodes of other subchains in the set. An edge disjoint collection of subchains can be treated by an intensification process that operates in parallel on each set, subject to the restriction that the identity of the endpoints of the subchains will not be altered. As a result of the decomposition, the best new sets of subchains can be reassembled to create new solutions. Such a process can be applied to multiple alternative decompositions in broader forms of intensification by decomposition.

These ideas are lately finding favor in other procedures, and may provide a bridge for interesting components of tabu search with components of other methodologies. We address the connections with these methodologies in Section 5.

4.3 Diversification Strategies

Search methods based on local optimization often rely on diversification strategies to increase their effectiveness in exploring the solution space defined by a combinatorial optimization problem. Some of these strategies are designed with the chief purpose of preventing searching processes from *cycling,* i.e., from endlessly executing the same sequence of moves (or more generally, from endlessly and exclusively revisiting the same set of solutions). Others are introduced to impart additional robustness or vigor to the search. Genetic algorithms use randomization in component processes such as combining population elements and applying crossover (as well as occasional mutation), thus providing an approximate diversifying effect. Simulated annealing likewise incorporates randomization to make diversification a function of temperature, whose gradual reduction correspondingly diminishes the directional variation in the objective function trajectory of solutions generated. Diversification in GRASP (Greedy Randomized Adaptive Search Procedures) is achieved in a certain sense within repeated construction phases by means of a random sampling over elements that pass a threshold of attractiveness by a greedy criterion.

In tabu search, diversification is created to some extent by short term memory functions, but is particularly reinforced by certain forms of longer term memory. TS diversification strategies, as their name suggests, are designed to drive the search into new regions. Often they are based on modifying choice rules to bring attributes into the solution that are infrequently used. Alternatively, they may introduce such attributes by periodically applying methods that assemble subsets of these attributes into candidate solutions for continuing the search, or by partially or fully restarting the solution process. Diversification strategies are particularly helpful when better solutions can be reached only by crossing barriers or “humps” in the solution space topology.

4.3.1 Modifying Choice Rules

Consider a TS method designed to solve a graph partitioning problem which uses full and partial swap moves to explore the local neighborhood. The goal of this problem is to partition the nodes of the graph into two equal subsets so that the sum of the weights of the edges that join nodes in one subset to nodes in the other subset is minimized. Full swaps exchange two nodes that lie in two different sets of the partition. Partial swaps transfer a single node from one set to the other set. Since full swaps do not modify the number of nodes in the two sets of the partition, they maintain feasibility, while partial swaps do not. Therefore, under appropriate guidance, one approach to generate diversity is to periodically disallow the use of non-improving full swaps for a chosen duration (after an initial period where the search “settles down”). The partial swaps must of course be coordinated to allow feasibility to be recovered after achieving various degrees of infeasibility. (This relates to the approach of strategic oscillation, described in Section 4.4.) Implemented appropriately, this strategy has the effect of intelligently perturbing the current solution, while escaping from a local optimum, to an extent that the search is directed to a region that is different than the one being currently explored. The implementation of this strategy as applied to experimental problems has resulted in significant improvements in problem-solving efficacy.

The incorporation of partial swaps in place of full swaps in the previous example can be moderated by using the following penalty function:

$$MoveValue' = MoveValue + d * Penalty.$$

This type of penalty approach is commonly used in TS, where the *Penalty* value is often a function of frequency measures such as those indicated in Table 4.1, and d is an adjustable diversification parameter. Larger d values correspond to a desire for more diversification. (E.g., nodes that change sets more frequently are penalized more heavily to encourage the choice of moves that incorporate other nodes. Negative penalties, or "inducements," may also be used to encourage low frequency elements.) The penalty can be applied to classes of moves as well as to attributes of moves. Thus, during a phase where full swaps moves are excluded, all such moves receive a large penalty (with a value of d that is effectively infinite).

In some applications where d is used to inhibit the selection of "feasibility preserving" moves, the parameter can be viewed as the reciprocal of a Lagrangean multiplier in that "low" values result in nearly infinite costs for constraint violation, while "high" values allow searching through infeasible regions. The adjustment of such a parameter can be done in a way to provide a strategic oscillation around the feasibility boundary, again as discussed in Section 4.4. The parameter can also be used to control the amount of randomization in probabilistic versions of tabu search.

In TS methods that incorporate the simplex method of linear programming, as in "adjacent extreme point approaches" for solving certain nonlinear and mixed-integer programming problems, a diversification phase can be designed based on the number of times variables become basic. For example, a diversification step can give preference to bringing a nonbasic variable into the basis that has remained out of the basis for a relatively long period (cumulatively, or since its most recent inclusion, or a combination of the two). The number of successive iterations such steps are performed, and the frequency with which they are initiated, are design considerations of the type that can be addressed, for example, by the approach of target analysis (see Section 5).

4.3.2 Restarting

Frequency information can be used in different ways to design restarting mechanisms within tabu search. In a sequencing problem, for example, the overall frequency of jobs occupying certain positions can be used to bias a construction procedure and generate new restarting points.

In a TS method for a location/allocation problem, a diversification phase can be developed using frequency counts on the number of times a depot has changed its status (from open to closed or vice versa). The diversification phase can be started from the best solution found during the search. Based on the frequency information, d depots with the lowest counts are selected and their status is changed. The search starts from the new solution which differs from the best by exactly d components. To prevent a quick return to the best solution, the status of the d depots is also recorded in short term memory. (This is another case where residence frequency measures may provide useful alternatives or supplements to transition frequency measures.)

Additional forms of memory functions are possible when a restarting mechanism is implemented. For example, in the location/allocation problem, it is possible to keep track of recent sets of depots that were selected for diversification and avoid the same selection in the

next diversification phase. Similarly, in a sequencing problem, the positions occupied by jobs in recent starting points can be recorded to avoid future repetition. This may be viewed as a very simple forms of the critical event memory discussed in Section 2, and more elaborate forms will often yield greater benefits. The exploitation of such memory is very important in TS designs that are completely deterministic, since in these cases a given starting point will always produce the same search path. Experience also shows, however, that uses of TS memory to guide probabilistic forms of restarting can likewise yield benefits (Rochat and Taillard, 1995; Fleurent and Glover, 1996; Lokketangen and Glover, 1996).

Before concluding this section, it is appropriate to provide a word of background about the orientation underlying diversification strategies within the tabu search framework. Often there appears to be a hidden assumption that diversification is somehow tantamount to randomization. Certainly the introduction of a random element to achieve a diversifying effect is a widespread theme among search procedures, and is fundamental to the operation of simulated annealing and genetic algorithms. From an abstract standpoint, there is clearly nothing wrong with equating randomization and diversification, but to the extent that diversity connotes differences among elements of a set, and to the extent that establishing such differences is relevant to an effective search strategy, then the popular use of randomization is at best a convenient proxy (and at worst a haphazard substitute) for something quite different.

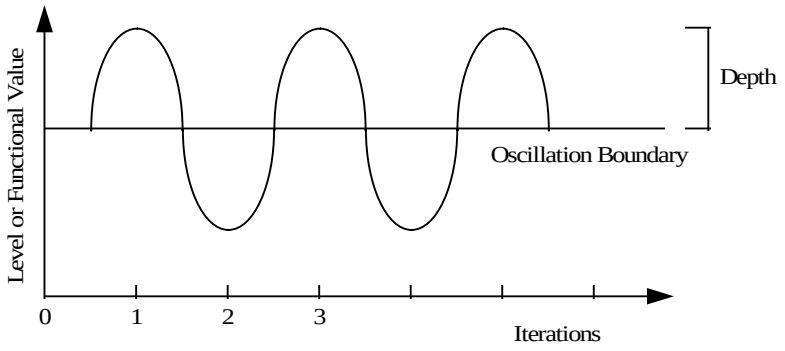
When randomization is used as part of a restarting mechanism, for example, frequency information can be employed to approximate probability distributions that bias the construction process. In this way, randomization is not a “blind” mechanism, but instead it is guided by search history. We examine inappropriate roles of randomization in Section 4.6, where we also explore the intensification / diversification distinction more thoroughly.

4.4 Strategic Oscillation

Strategic oscillation is closely linked to the origins of tabu search, and provides a means to achieve an effective interplay between intensification and diversification over the intermediate to long term. The recurring usefulness of this approach documented in a variety of studies warrants a more detailed examination of its characteristics.

Strategic oscillation operates by orienting moves in relation to a critical level, as identified by a stage of construction or a chosen interval of functional values. Such a critical level or *oscillation boundary* often represents a point where the method would normally stop. Instead of stopping when this boundary is reached, however, the rules for selecting moves are modified, to permit the region defined by the critical level to be crossed. The approach then proceeds for a specified depth beyond the oscillation boundary, and turns around. The oscillation boundary again is approached and crossed, this time from the opposite direction, and the method proceeds to a new turning point (see Figure 4.2).

Fig. 4.2 Strategic oscillation.



The process of repeatedly approaching and crossing the critical level from different directions creates an oscillatory behavior, which gives the method its name. Control over this behavior is established by generating modified evaluations and rules of movement, depending on the region navigated and the direction of search. The possibility of retracing a prior trajectory is avoided by standard tabu search mechanisms, like those established by recency-based and frequency-based memory functions.

A simple example of this approach occurs for the multidimensional knapsack problem, where values of zero-one variables are changed from 0 to 1 until reaching the boundary of feasibility. The method then continues into the infeasible region using the same type of changes, but with a modified evaluator. After a selected number of steps, the direction is reversed by choosing moves that change variables from 1 to 0. Evaluation criteria to drive toward improvement vary according to whether the movement occurs inside or outside the feasible region (and whether it is directed toward or away from the boundary), accompanied by associated restrictions on admissible changes to values of variables. The turnaround towards feasibility can also be triggered by a maximum infeasibility value, which defines the depth of the oscillation beyond the critical level (i.e., the feasibility boundary).

A somewhat different type of application occurs for graph theory problems where the critical level represents a desired form of graph structure, capable of being generated by progressive additions (or insertions) of basic elements such as nodes, edges, or subgraphs. One type of strategic oscillation approach for this problem results by a constructive process of introducing elements until the critical level is reached, and then introducing further elements to cross the boundary defined by the critical level. The current solution may change its structure once this boundary is crossed (as where a forest becomes transformed into a graph that contains loops), and hence a different neighborhood may be required, yielding modified rules for selecting moves. The rules again change in order to proceed in the opposite direction, removing elements until again recovering the structure that defines the critical level.

In the Min k-Tree problem, for example, edges can be added beyond the critical level defined by k. Then a rule to delete edges must be applied. The rule to delete edges will typically be different in character from the one used for adding (i.e., will not simply be its "inverse"). In this case, all feasible solutions lie on the oscillation boundary, since any deviation from this level results in solutions with more or less than k edges.

Such rule changes are typical features of strategic oscillation, and provide an enhanced heuristic vitality. The application of different rules may be accompanied by crossing a boundary to different depths on different sides. An option is to approach and retreat from the boundary while remaining on a single side, without crossing (i.e., electing a crossing of "zero depth").

These examples constitute a constructive/destructive type of strategic oscillation, where constructive steps "add" elements (or set variables to 1) and destructive steps "drop" elements (or set variables to 0). (Types of TS memory structures for add / drop moves discussed in Section 2 are relevant for such procedures.) One-sided oscillations (that remain on a single side of a critical boundary) are appropriate in a variety of scheduling and graph-related applications, where constructive processes are traditionally applied. The alternation with destructive processes that strategically dismantle and then re-build successive trial solutions affords a potent enhancement of more traditional procedures. In both one-sided and two-sided oscillation approaches it is frequently important to spend additional search time in regions close to the critical level, and especially to spend time at the critical level itself. This may be done by inducing a sequence of tight oscillations about the critical level, as a prelude to each larger oscillation that proceeds to a greater depth. Alternately, if greater effort is permitted for evaluating and executing each move, the method may use "exchange moves" (broadly interpreted) to stay at the critical level for longer periods. In the case of the Min *k*-Tree problem, for example, once the oscillation boundary has been reached, the search can stay on it by performing swap moves (either of nodes or edges). An option is to use such exchange moves to proceed to a local optimum each time the critical level is reached.

When the level or functional values in Figure 4.2 refer to degrees of feasibility and infeasibility, a vector-valued function associated with a set of problem constraints can be used to control the oscillation. In this case, controlling the search by bounding this function can be viewed as manipulating a parameterization of the selected constraint set. A preferred alternative is often to make the function a Lagrangean or surrogate constraint penalty function, avoiding vector-valued functions and allowing tradeoffs between degrees of violation of different component constraints.

Intensification processes can readily be embedded in strategic oscillation by altering choice rules to encourage the incorporation of particular attributes — or at the extreme, by locking such attributes into the solution for a period. Such processes can be viewed as designs for exploiting *strongly determined* and *consistent* variables. A strongly determined variable is one that cannot change its value in a given high quality solution without seriously degrading quality or feasibility, while a consistent variable is one that frequently takes on a specific value (or a highly restricted range of values) in good solutions. The development of useful measures of "strength" and "consistency" is critical to exploiting these notions, particularly by accounting for tradeoffs determined by context. However, straightforward uses of frequency-based memory for keeping track of consistency, sometimes weighted by elements of quality and influence, have produced methods with very good performance outcomes.

An example of where these kinds of approaches are also beginning to find favor in other settings occurs in recently developed variants of genetic algorithms for sequencing problems. The more venturesome of these approaches are coming to use special forms of "crossover" to assure offspring will receive attributes shared by good parents, thus incorporating a type of intensification based on consistency. Extensions of such procedures using TS ideas of identifying elements that qualify as consistent and strongly determined according to broader criteria, and making direct use of memory functions to establish this identification, provide an

interesting area for investigation. (Additional links to GA methods, and ways to go beyond current explorations of such methods, are discussed in Section 5.)

Longer term processes, following the type of progression customarily found beneficial in tabu search, may explicitly introduce supplemental diversification strategies into the oscillation pattern. When oscillation is based on constructive and destructive processes, the repeated application of constructive phases (rather than moving to intermediate levels using destructive moves) embodies an extreme type of oscillation that is analogous to a restart method. In this instance the restart point is always the same (i.e., a null state) instead of consisting of different initial solutions, and hence it is important to use choice rule variations to assure appropriate diversification.

A connection can also be observed between an extreme version of strategic oscillation — in this case a relaxed version — and the class of procedures known as *perturbation* approaches. An example is the subclass known as "large-step simulated annealing" or "large-step Markov chain" methods (Martin, Otto and Felten, 1991 and 1992; Johnson, 1990; Lourenco and Zwijnenburg, 1996). Such methods try to drive an SA procedure (or an iterated descent procedure) out of local optimality by propelling the solution a greater distance than usual from its current location.

Perturbation methods may be viewed as loosely structured procedures for inducing oscillation, without reference to intensification and diversification and their associated implementation strategies. Similarly, perturbation methods are not designed to exploit tradeoffs created by parametric variations in elements such as different types of infeasibility, measures of displacement from different sides of boundaries, etc. Nevertheless, at a first level of approximation, perturbation methods seek goals similar to those pursued by strategic oscillation.

4.5 Path Relinking

A useful integration of intensification and diversification strategies occurs in the approach called *path relinking*. This approach generates new solutions by exploring trajectories that connect elite solutions — by starting from one of these solutions, called an *initiating solution*, and generating a path in the neighborhood space that leads toward the other solutions, called *guiding solutions*. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions.

The approach may be viewed as an extreme (highly focused) instance of a strategy that seeks to incorporate attributes of high quality solutions, by creating inducements to favor these attributes in the moves selected. However, instead of using an inducement that merely encourages the inclusion of such attributes, the path relinking approach subordinates all other considerations to the goal of choosing moves that introduce the attributes of the guiding solutions, in order to create a "good attribute composition" in the current solution. The composition at each step is determined by choosing the best move, using customary choice criteria, from the restricted set of moves that incorporate a maximum number (or a maximum weighted value) of the attributes of the guiding solutions. As in other applications of TS, aspiration criteria can override this restriction to allow other moves of particularly high quality to be considered.

Specifically, upon identifying a collection of one or more elite solutions to guide the path of a given solution, the attributes of these guiding solutions are assigned preemptive weights as

inducements to be selected. Larger weights are assigned to attributes that occur in greater numbers of the guiding solutions, allowing bias to give increased emphasis to solutions with higher quality or with special features (e.g., complementing those of the solution that initiated the new trajectory).

More generally, it is not necessary for an attribute to occur in a guiding solution in order to have a favored status. In some settings attributes can share degrees of similarity, and in this case it can be useful to view a solution vector as providing "votes" to favor or discourage particular attributes. Usually the strongest forms of aspiration criteria are relied upon to overcome this type of choice rule.

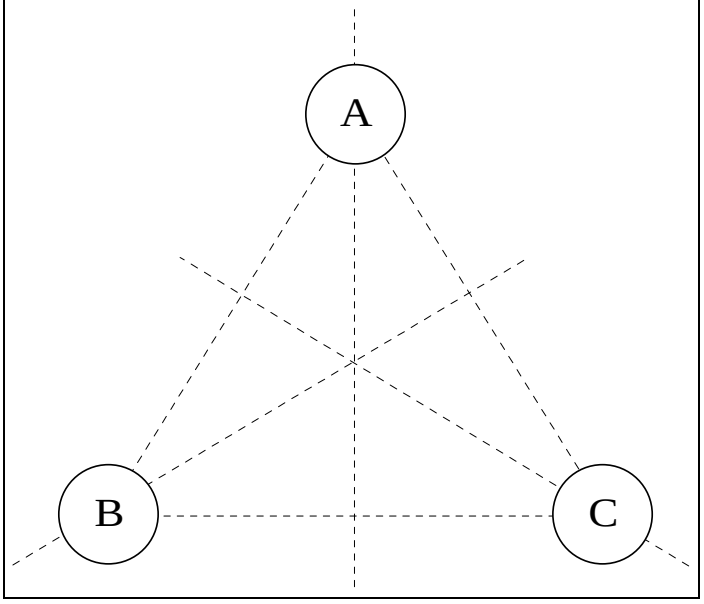
In a given collection of elite solutions, the role of initiating solution and guiding solutions can be alternated. The distinction between initiating solutions and guiding solutions effectively vanishes in such cases. For example, a set of current solutions may be generated simultaneously, extending different paths, and allowing an initiating solution to be replaced (as a guiding solution for others) whenever its associated current solution satisfies a sufficiently strong aspiration criterion.

Because their roles are interchangeable, the initiating and guiding solutions are collectively called *reference solutions*. These reference solutions can have different interpretations depending on the solution framework under consideration. Reference points can be created by any of a number of different heuristics that result in high quality solutions.

An idealized form of such a process is shown in Figure 4.3. The chosen collection of reference solutions consists of the three members, A, B, and C. Paths are generated by allowing each to serve as initiating solution, and by allowing either one or both of the other two solutions to operate as guiding solutions. Intermediate solutions encountered along the paths are not shown. The representation of the paths as straight lines of course is oversimplified, since choosing among available moves in a current neighborhood will generally produce a considerably more complex trajectory. Intensification can be achieved by generating paths from similar solutions, while diversification is obtained creating paths from dissimilar solutions. Appropriate aspiration criteria allow deviation from the paths at attractive neighbors.

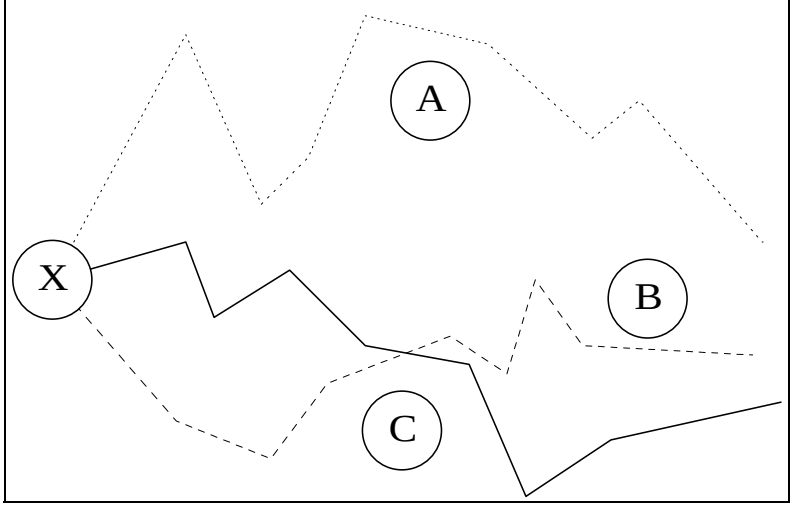
As Figure 4.3 indicates, at least one path continuation is allowed beyond each initiating/guiding solution. Such a continuation can be accomplished by penalizing the inclusion of attributes dropped during a trajectory, including attributes of guiding solutions that may be compelled to be dropped in order to continue the path. (An initiating solution may also be repelled from the guiding solutions by penalizing the inclusion of their attributes from the outset.) Probabilistic TS variants operate in the path relinking setting, as they do in others, by translating evaluations for deterministic rules into probabilities of selection, strongly biased to favor higher evaluations.

Fig. 4.3 Paths relinking in neighborhood space.



Promising regions are searched more thoroughly in path relinking by modifying the weights attached to attributes of the guiding solutions, and by altering the bias associated with solution quality and selected solution features. Figure 4.4 depicts the type of variation that can result, where the point X represents an initiating solution, the points A, B and C represent guiding solutions, and the dashed, dotted and solid lines are different searching paths. For appropriate choices of the reference points (and neighborhoods for generating paths from them), the notion called the *Principle of Proximate Optimality* (Glover and Laguna, 1997) suggests that additional elite points are likely to be found in the regions traversed by the paths, upon launching new searches from high quality points on these paths.

Fig. 4.4 Path relinking by attribute bias.



4.5.1 Roles in Intensification and Diversification

Path relinking, in common with strategic oscillation, gives a natural foundation for developing intensification and diversification strategies. Intensification strategies in this setting typically choose reference solutions to be elite solutions that lie in a common region or that share common features. Similarly, diversification strategies based on path relinking characteristically select reference solutions that come from different regions or that exhibit contrasting features. Diversification strategies may also place more emphasis on paths that go beyond the reference points. Collections of reference points that embody such conditions can be usefully determined by clustering and conditional analysis methods.

These alternative forms of path relinking also offer a convenient basis for parallel processing, contributing to the approaches for incorporating intensification and diversification tradeoffs into the design of parallel solution processes generally.

4.5.2 Incorporating Alternative Neighborhoods

Path relinking strategies in tabu search can occasionally profit by employing different neighborhoods and attribute definitions than those used by the heuristics for generating the reference solutions. For example, it is sometimes convenient to use a constructive neighborhood for path relinking, i.e., one that permits a solution to be built in a sequence of constructive steps (as in generating a sequence of jobs to be processed on specified machines using dispatching rules). In this case the initiating solution can be used to give a beginning partial construction, by specifying particular attributes (such as jobs in particular relative or absolute sequence positions) as a basis for remaining constructive steps. Similarly, path relinking can make use of destructive neighborhoods, where an initial solution is "overloaded" with attributes donated by the guiding solutions, and such attributes are progressively stripped away or modified until reaching a set with an appropriate composition.

When path relinking is based on constructive neighborhoods, the guiding solution(s) provide the attribute relationships that give options for subsequent stages of construction. At an extreme, a full construction can be produced, by making the initiating solution a *null solution*. The destructive extreme starts from a "complete set" of solution elements. Constructive and destructive approaches differ from transition approaches by typically producing only a single new solution, rather than a sequence of solutions, on each path that leads from the initiating solution toward the others. In this case the path will never reach the additional solutions unless a transition neighborhood is used to extend the constructive neighborhood.

Constructive neighborhoods can often be viewed as a special case of feasibility restoring neighborhoods, since a null or partially constructed solution does not satisfy all conditions to qualify as feasible. Similarly, destructive neighborhoods can also represent an instance of a feasibility restoring function, as where an excess of elements may violate explicit problem constraints. A variety of methods have been devised to restore infeasible solutions to feasibility, as exemplified by flow augmentation methods in network problems, subtour elimination methods in traveling salesman and vehicle routing problems, alternating chain processes in degree-constrained subgraph problems, and value incrementing and decrementing methods in covering and multidimensional knapsack problems. Using neighborhoods that permit restricted forms of infeasibilities to be generated, and then using associated neighborhoods to remove these infeasibilities, provides a form of path relinking with useful diversification features. Upon further introducing transition neighborhoods, with the ability to generate successive solutions with changed attribute mixes, the mechanism of path relinking

also gives a way to tunnel through infeasible regions. The following is a summary of the components of path relinking:

Step 1. Identify the neighborhood structure and associated solution attributes for path relinking (possibly different from those of other TS strategies applied to the problem).

Step 2. Select a collection of two or more reference solutions, and identify which members will serve as the initiating solution and the guiding solution(s). (Reference solutions can be infeasible, such as "incomplete" or "overloaded" solution components treated by constructive or destructive neighborhoods.)

Step 3. Move from the initiating solution toward (or beyond) the guiding solution(s), generating one or more intermediate solutions as candidates to initiate subsequent problem solving efforts. (If the first phase of this step creates an infeasible solution, apply an associated second phase with a feasibility restoring neighborhood.)

In Section 5 we will see how the path relinking strategy relates to a strategy called scatter search, which provides additional insights into the nature of both approaches.

4.6 The Intensification / Diversification Distinction

The relevance of the intensification/diversification distinction is supported by the usefulness of TS strategies that embody these notions. Although both operate in the short term as well as the long term, we have seen that longer term strategies are generally the ones where these notions find their greatest application.

In some instances we may conceive of intensification as having the function of an intermediate term strategy, while diversification applies to considerations that emerge in the longer run. This view comes from the observation that in human problem solving, once a short term strategy has exhausted its efficacy, the first (intermediate term) response is often to focus on the events where the short term approach produced the best outcomes, and to try to capitalize on elements that may be common to those events. When this intensified focus on such events likewise begins to lose its power to uncover further improvement, more dramatic departures from a short term strategy are undertaken. (Psychologists do not usually differentiate between intermediate and longer term memory, but the fact that memory for intensification and diversification can benefit from such differentiation suggests that there may be analogous physical or functional differences in human memory structures.) Over the truly long term, however, intensification and diversification repeatedly come into play in ways where each depends on the other, not merely sequentially, but also simultaneously.
There has been some confusion between the terms intensification and diversification, as applied in tabu search, and the terms *exploitation* and *exploration,* as popularized in the literature of genetic algorithms. The differences between these two sets of notions deserves to be clarified, because they have substantially different consequences for problem solving.

The exploitation/exploration distinction comes from control theory, where exploitation refers to following a particular recipe (traditionally memoryless) until it fails to be effective, and exploration then refers to instituting a series of random changes — typically via multi-armed bandit schemes — before reverting to the tactical recipe. (The issue of exploitation versus

exploration concerns how often and under what circumstances the randomized departures are launched.)

By contrast, intensification and diversification in tabu search are both processes that take place when simpler exploitation designs play out and lose their effectiveness — although as we have noted, the incorporation of memory into search causes intensification and diversification also to be manifest in varying degrees even in the short range. (Similarly, as we have noted, intensification and diversification are not opposed notions, for the best form of each contains aspects of the other, along a spectrum of alternatives.)

Intensification and diversification are likewise different from the control theory notion of exploration. Diversification, which is sometimes confused with exploration, is not a recourse to a Game of Chance for shaking up the options invoked, but is a collection of strategies — again taking advantage of memory — designed to move purposefully rather than randomly into uncharted territory.

The source of these differences is not hard to understand. Researchers and practitioners in the area of search methods have had an enduring love affair with randomization, perhaps influenced by the much publicized Heisenberg Uncertainty Principle in Quantum Mechanics. Einstein's belief that God does not roll dice is out of favor, and many find a special enchantment in miraculous events where blind purposelessness creates useful order. (We are less often disposed to notice that this way of producing order requires an extravagant use of time, and that order, once created, is considerably more effective than randomization in creating still higher order.)

Our "scientific" reports of experiments with nature reflect our fascination with the role of chance. When apparently chaotic fluctuations are brought under control by random perturbations, we seize upon the random element as the key, while downplaying the importance of attendant restrictions on the setting in which randomization operates. The diligently concealed message is that under appropriate controls, *perturbation* is effective for creating desired patterned outcomes — and in fact, if the system and attendant controls are sufficiently constrained, perturbation works even when random. (Instead of accentuating differences between workable and unworkable kinds of perturbation, in our quest to mold the universe to match our mystique we portray the central consideration to be randomization versus nonrandomization.)

The tabu search orientation evidently contrasts with this perspective. As manifest in the probabilistic TS variant, elements subjected to random influence are preferably to be strongly confined, and uses of randomization are preferably to be modulated through well differentiated probabilities. In short, the situations where randomization finds a place are very highly structured. From this point of view God may play with dice, but beyond any question the dice are *loaded*.

4.7 Some Basic Memory Structures for Longer Term Strategies

To give a foundation for describing fundamental types of memory structures for longer term strategies, we first briefly review the form of the recency-based memory structure introduced in Section 2 for handling add/drop moves. However, we slightly change the notation, to provide a convenient way to refer to a variety of other types of moves.

4.7.1 Conventions

Let $S = \{1, 2,..., s\}$ denote an index set for a collection of solution attributes. For example, the indexes $i \in S$ may correspond to indexes of zero-one variables x_i, or they may be indexes of edges that may be added to or deleted from a graph, or the job indexes in a production scheduling problem. More precisely, by the attribute/element distinction discussed in Section 2, the attributes referenced by S in these cases consist of the specific values assigned to the variables, the specific add/drop states adopted by the edges, or positions occupied by the jobs. In general, to give a correspondence with developments of Section 3, an index $i \in S$ can summarize more detailed information; e.g., by referring to an ordered pair (j,k) that summarizes a value assignment $x_j = k$ or the assignment of job j to position k, etc. Hence, broadly speaking, the index i may be viewed as a notational convenience for representing a pair or a vector.

To keep our description at the simplest level, suppose that each $i \in S$ corresponds to a 0-1 variable x_i. As before, we let *Iter* denote the counter that identifies the current iteration, which starts at 0 and increases by 1 each time a move is made.

For recency-based memory, following the approach indicated in Section 2, when a move is executed that causes a variable x_i to change its value, we record $TabuStart(i) = Iter$ immediately after updating the iteration counter. (This means that if the move has resulted in $x_i = 1$, then the attribute $x_i = 0$ becomes tabu-active at the iteration $TabuStart(i)$.) Further, we let $TabuTenure(i)$ denote the number of iterations this attribute will remain tabu-active. Thus, by our previous design, the recency-based tabu criterion says that the previous value of x_i is tabu-active throughout all iterations such that

$$TabuStart(i) + TabuTenure(i) \leq Iter.$$

Similarly, in correspondence with earlier remarks, the value $TabuStart(i)$ can be set to 0 before initiating the method, as a convention to indicate no prior history exists. Then we automatically avoid assigning a tabu-active status to any variable with $TabuStart(i) = 0$ (since the starting value for variable x_i has not yet been changed).

4.7.2 Frequency-Based Memory

By our foregoing conventions, allowing the set $S = \{1, ..., s\}$ for illustration purposes to refer to indexes of 0-1 variables x_i, we may indicate structures to handle frequency-based memory as follows.

Transition frequency-based memory is by far the simplest to handle. A transition memory, $Transition(i)$, to record the number of times x_i changes its value, can be maintained simply in the form of a counter for x_i that is incremented at each move where such a change occurs. Since x_i is a zero-one variable, $Transition(i)$ also discloses the number of times x_i changes to and from each of its possible assigned values. In more complex situations, by the conventions already noted, a matrix memory $Transition(j,k)$ can be used to determine numbers of transitions involving assignments such as $x_j = k$. Similarly, a matrix memory may be used in the case of the sequencing problem where both the index of job j and position k may be of interest. In the context of the Min k-Tree problem, an array dimensioned by the number of edges can maintain a transition memory to keep track of the number of times that specific edges have been brought in and out of the solution. A matrix based on the edges can also identify conditional frequencies. For example, the matrix $Transition(j,k)$ can be used to count the number of times edge j replaced edge k. It should be kept in mind in using transition frequency memory that penalties and inducements are often based on *relative* numbers (rather than absolute

numbers) of transitions, hence requiring that recorded transition values are divided by the total number of iterations (or the total number of transitions). As noted earlier, other options include dividing by the current maximum transition value. Raising transition values to a power, as by squaring, is often useful to accentuate the differences in relative frequencies.

Residence memory requires only slightly more effort to maintain than transition memory, by taking advantage of the recency-based memory stored in *TabuStart*(*i*). The following approach can be used to track the number of solutions in which $x_i = 1$, thereby allowing the number of solutions in which $x_i = 0$ to be inferred from this. Start with *Residence*(*i*) = 0 for all *i*. Then, whenever x_i changes from 1 to 0, after updating *Iter* but before updating *TabuStart*(*i*), set

$$Residence(i) = Residence(i) + Iter - TabuStart(i).$$

Then, during iterations when $x_i = 0$, *Residence*(*i*) correctly stores the number of earlier solutions in which $x_i = 1$. During iterations when $x_i = 1$, the true value of *Residence*(*i*) is the right hand side of the preceding assignment, however the update only has to be made at the indicated points when x_i changes from 1 to 0. Table 4.2 illustrates how this memory structure works when used to track the assignments of a variable *x* during 100 iterations. The variable is originally assigned to a value of zero by a construction procedure that generates an initial solution. In iteration 10 a move is made that changes the assignment of *x* from zero to one, however the *Residence* value remains at zero. *Residence* is updated at iterations 22 and 73, when moves are made that change the assignment of *x* from 1 to 0. At iteration 65, for example, *x* has received a value of 1 for 27 iterations (i.e., *Residence* + *Iter* - *TabuStart* = 12 + 65 - 50 = 27), while at iteration 90 the count is 35 (i.e., the value of *Residence*).

Table 4.2 Illustrative residence memory.

Iter	*Assignment*	*Residence*
0	$x = 0$	0
10	$x = 1$	0
22	$x = 0$	22 - 10 = 12
50	$x = 1$	12
73	$x = 0$	12 + 73 -50 = 35

As with transition memory, residence memory should be translated into a relative as a basis for creating penalties and inducements.

The indicated memory structures can readily be applied to multivalued variables (or multistate attributes) by the extended designs illustrated in Section 3. In addition, the 0-1 format can be adapted to reference the number of times (and last time) a more general variable changed its value, which leads to more restrictive tabu conditions and more limiting ("stronger") uses of frequency-based memory than by referring separately to each value the variable receives. As in the case of recency-based memory, the ability to affect larger numbers of alternative moves by these more aggregated forms of memory can be useful for larger problems, not only for conserving memory space but also for providing additional control over solutions generated.

4.7.3 Critical Event Memory

Strategic oscillation offers an opportunity to make particular use of both short term and long term frequency-based memory. To illustrate, let *A*(*Iter*) denote a zero-one vector whose *j*th

component has the value 1 if attribute j is present in the current solution and has the value 0 otherwise. The vector A can be treated "as if" it is the same as the solution vector for zero-one problems, though implicitly it is twice as large, since $x_j = 0$ is a different attribute from $x_j = 1$. This means that rules for operating on the full A must be reinterpreted for operating on the condensed form of A. The sum of the A vectors over the most recent t critical events provides a simple memory that combines recency and frequency considerations. To maintain the sum requires remembering $A(k)$, for k ranging over the last t iterations. Then the sum vector A^* can be updated quite easily by the incremental calculation

$$A^* = A^* + A(Iter) - A(Iter - t + 1).$$

Associated frequency measures, as noted earlier, may be normalized, in this case for example by dividing A^* by the value of t. A long term form of A^* does not require storing the $A(k)$ vectors, but simply keeps a running sum. A^* can also be maintained by exponential smoothing.

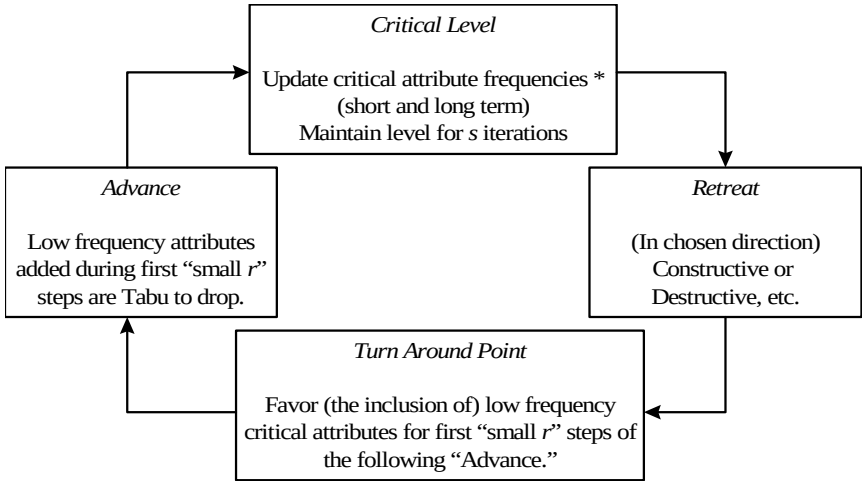
Such frequency-based memory is useful in strategic oscillation where critical events are chosen to be those of generating a complete (feasible) construction, or in general of reaching the targeted boundary (or a best point within a boundary region). Instead of using a customary recency-based TS memory at each step of an oscillating pattern, greater flexibility results by disregarding tabu restrictions until reaching the turning point, where the oscillation process alters its course to follow a path toward the boundary. At this point, assume a choice rule is applied to introduce an attribute that was not contained in any recent solution at the critical level. If this attribute is maintained in the solution by making it tabu to be dropped, then upon eventually reaching the critical level the solution will be different from any seen over the horizon of the last t critical events. Thus, instead of updating A^* at each step, the updating is done only for critical level solutions, while simultaneously enhancing the flexibility of making choices.

In general, the possibility occurs that no attribute exists that allows this process to be implemented in the form stated. That is, every attribute may already have a positive associated entry in A^*. Thus, at the turn around point, the rule instead is to choose a move that introduces attributes which are least frequently used. (Note, "infrequently used" can mean either "infrequently present" or "infrequently absent," depending upon the current direction of oscillation.) This again can be managed conveniently by using penalties and inducements. Such an approach has been found very effective for multidimensional knapsack problems and 0-1 quadratic optimization problems in Glover and Kochenberger (1996) and Glover, Kochenberger and Alidaee (1998).

For greater diversification, this rule can be applied for r steps after reaching the turn around point. Normally r should be a small number, e.g., with a baseline value of 1 or 2, which is periodically increased in a standard diversification pattern. Shifting from a short term A^* to a long term A^* creates a global diversification effect. A template for this approach is given in Figure 4.5.

The approach of Figure 4.5 is not symmetric. An alternative form of control is to seek immediately to introduce a low frequency attribute upon leaving the critical level, to increase the likelihood that the solution at the next turn around will not duplicate a solution previously visited at that point. Such a control enhances diversity, though duplication at the turn around will already be inhibited by starting from different solutions at the critical level.

Fig. 4.5 Strategic oscillation illustrative memory.



* For selected part of critical level iterations: e.g., for first and best solutions of current block

5. Connections, Hybrid Approaches and Learning

Relationships between tabu search and other procedures like simulated annealing and genetic algorithms provide a basis for understanding similarities and contrasts in their philosophies, and for creating potentially useful hybrid combinations of these approaches. We offer some speculation on preferable directions in this regard, and also suggest how elements of tabu search can add a useful dimension to neural network approaches.

From the standpoint of evolutionary strategies, we trace connections between population based models for combining solutions, as in genetic algorithms, and ideas that emerged from surrogate constraint approaches for exploiting optimization problems by combining constraints. We show how this provides the foundation for methods that give additional alternatives to genetic-based frameworks, specifically as embodied in the scatter search approach, which is the "primal complement" to the dual strategy of surrogate constraint approaches. Recent successes by integrating scatter search (and its path relinking extensions) with tabu search disclose potential advantages for evolutionary strategies that incorporate adaptive memory.

Finally, we describe the learning approach called target analysis, which provides a way to determine decision parameters for deterministic and probabilistic strategies — and thus affords an opportunity to create enhanced solution methods.

5.1 Simulated Annealing

The contrasts between simulated annealing and tabu search are fairly conspicuous, though undoubtedly the most prominent is the focus on exploiting memory in tabu search that is absent from simulated annealing. The introduction of this focus entails associated differences in search mechanisms, and in the elements on which they operate. Accompanying the

differences directly attributable to the focus on memory, and also magnifying them, several additional elements are fundamental for understanding the relationship between the methods. We consider three such elements in order of increasing importance.

First, tabu search emphasizes scouting successive neighborhoods to identify moves of high quality, as by candidate list approaches of the form described in Section 3. This contrasts with the simulated annealing approach of randomly sampling among these moves to apply an acceptance criterion that disregards the quality of other moves available. (Such an acceptance criterion provides the sole basis for sorting the moves selected in the SA method.) The relevance of this difference in orientation is accentuated for tabu search, since its neighborhoods include linkages based on history, and therefore yield access to information for selecting moves that is not available in neighborhoods of the type used in simulated annealing.

Next, tabu search evaluates the relative attractiveness of moves not only in relation to objective function change, but in relation to additional factors that represent quality, which are balanced over time with factors that represent influence. Both types of measures are affected by the differentiation among move attributes, as embodied in tabu activation rules and aspiration criteria, and in turn by relationships manifested in recency, frequency, and sequential interdependence (hence, again, involving recourse to memory). Other aspects of the state of search also affect these measures, as reflected in the altered evaluations of strategic oscillation, which depend on the direction of the current trajectory and the region visited.

Finally TS emphasizes guiding the search by reference to multiple thresholds, reflected in the tenures for tabu-active attributes and in the conditional stipulations of aspiration criteria. This may be contrasted to the simulated annealing reliance on guiding the search by reference to the single threshold implicit in the temperature parameter. The treatment of thresholds by the two methods compounds this difference between them. Tabu search varies its thresholds nonmonotonically, reflecting the conception that multidirectional parameter changes are essential to adapt to different conditions, and to provide a basis for locating alternatives that might otherwise be missed. This contrasts with the simulated annealing philosophy of adhering to a temperature parameter that only changes monotonically.

Hybrids are now emerging that are taking preliminary steps to bridge some of these differences, particularly in the realm of transcending the simulated annealing reliance on a monotonic temperature parameter. A hybrid method that allows temperature to be strategically manipulated, rather than progressively diminished, has been shown to yield improved performance over standard SA approaches. A hybrid method that expands the SA basis for move evaluations also has been found to perform better than standard simulated annealing. Consideration of these findings invites the question of whether removing the memory scaffolding of tabu search and retaining its other features may yield a viable method in its own right. For example, experience cited in some of the studies reported in Glover and Laguna (1997) suggests that, while a memoryless version of tabu search called tabu thresholding can outperform a variety of alternative heuristics, it generally does not match the performance of TS methods that appropriately exploit memory.

5.2 Genetic Algorithms

Genetic algorithms offer a somewhat different set of comparisons and contrasts with tabu search. GAs are based on selecting subsets (traditionally pairs) of solutions from a population, called parents, and combining them to produce new solutions called children. Rules of combination to yield children are based on the genetic notion of crossover, which in the

classical form consists of interchanging solution values of particular variables, together with occasional operations such as random value changes. Children that pass a survivability test, probabilistically biased to favor those of superior quality, are then available to be chosen as parents of the next generation. The choice of parents to be matched in each generation is based on random or biased random sampling from the population (in some parallel versions executed over separate subpopulations whose best members are periodically exchanged or shared). Genetic terminology customarily refers to solutions as chromosomes, variables as genes, and values of variables as alleles.

By means of coding conventions, the genes of genetic algorithms may be compared to attributes in tabu search. Introducing memory in GAs to track the history of genes and their alleles over subpopulations would provide an immediate and natural way to create a hybrid with TS.

Some important differences between genes and attributes are worth noting, however. The implicit differentiation of attributes into *from* and *to* components, each having different memory functions, does not have a counterpart in genetic algorithms. A *from* attribute is one that is part of the current solution but is not included in the next solution once a move is made. A *to* attribute is one that is not part of the current solution but becomes part of the next solution once a move is made. The lack of this type of differentiation in GAs results because these approaches are organized to operate without reference to moves (although, strictly speaking, combination by crossover can be viewed as a special type of move

A contrast to be noted between genetic algorithms and tabu search arises in the treatment of context, i.e., in the consideration given to structure inherent in different problem classes. For tabu search, context is fundamental, embodied in the interplay of attribute definitions and the determination of move neighborhoods, and in the choice of conditions to define tabu restrictions. Context is also implicit in the identification of amended evaluations created in association with longer term memory, and in the regionally dependent neighborhoods and evaluations of strategic oscillation.

At the opposite end of the spectrum, GA literature has traditionally stressed the freedom of its rules from the influence of context. Crossover, in particular, is supposedly a *context neutral* operation, which assumes no reliance on conditions that solutions must obey in a particular problem setting, just as genes make no reference to the environment as they follow their instructions for recombination (except, perhaps, in the case of mutation). Practical application, however, generally renders this an inconvenient assumption, making solutions of interest difficult to find. Consequently, a good deal of effort in GA implementation is devoted to developing "special crossover" operations that compensate for the difficulties created by context, effectively reintroducing it on a case by case basis.

The chief method by which modern genetic algorithms handle structure is by relegating its treatment to some other method. For example, genetic algorithms combine solutions by their parent-children processes at one level, and then a descent method may be introduced to operate on the resulting solutions to produce new solutions. These new solutions in turn are submitted to be recombined by the GA processes. In these versions, genetic algorithms already take the form of hybrid methods. Hence there is a natural basis for marrying GA and TS procedures in such approaches. But genetic algorithms and tabu search also can be joined in a more fundamental way.

Specifically, tabu search strategies for intensification and diversification are based on the following question: how can information be extracted from a set of good solutions to help

uncover additional (and better) solutions? From one point of view, GAs provide an approach for answering this question, consisting of putting solutions together and interchanging components (in some loosely defined sense, if traditional crossover is not strictly enforced). Tabu search, by contrast, seeks an answer by utilizing processes that specifically incorporate neighborhood structures into their design.

Augmented by historical information, neighborhood structures are used as a basis for applying penalties and incentives to induce attributes of good solutions to become incorporated into current solutions. Consequently, although it may be meaningless to interchange or otherwise incorporate a set of attributes from one solution into another in a wholesale fashion, as attempted in traditional GA recombination operations, a stepwise approach to this goal through the use of neighborhood structures is entirely practicable. This observation provides a motive for creating *structured combinations* of solutions that embody desired characteristics such as feasibility — as is automatically achieved by the TS approach of path relinking discussed in Section 4. Instead of being compelled to create new types of crossover to remove deficiencies of standard operators upon being confronted by changing contexts, this approach addresses context directly and makes it an essential part of the design for generating combinations.

The current trend of genetic algorithms seems to be increasingly compatible with this perspective, and could provide a basis for a useful hybrid combination of genetic algorithm and tabu search ideas. However, a fundamental question emerges, as posed in the development of the next sections, about whether there is any advantage to introducing genetic crossover-based ideas over introducing the apparently more flexible and exploitable path relinking ideas.

9.2.1 Models of Nature — Beyond "Genetic Metaphors"

An aspect of tabu search that is often misunderstood concerns the relation between a subset of its strategies and certain approaches embodied in genetic algorithms. TS researchers have tended sometimes to overlook the part of the adaptive memory focus that is associated with strategies for combining sets of elite solutions. Complementing this, GA researchers have been largely unaware that such a collection of strategies outside their domain exists. This has quite possibly been due to the influence of the genetic metaphor, which on the one hand has helped to launch a number of useful problem solving ideas, and on the other hand has also sometimes obscured fertile connections to ideas that come from different foundations.

To understand the relevant ties, it is useful to go back in time to examine the origins of the GA framework and of an associated set of notions that became embodied in TS strategies. We will first sketch the original genetic algorithm design (see Figure 5.2), as characterized in Holland (1975). Our description is purposely somewhat loose, to be able to include approaches more general than the specific proposals that accompanied the introduction of GAs. Many variations and changes have come about over the years, as we subsequently observe.

Fig. 5.2 Genetic algorithm template.

1) Begin with a population of binary vectors.

2) Operate repeatedly on the current generation of vectors, for a selected number of steps, choosing two "parent vectors" at random. Then mate the parents by exchanging certain of their components to produce offspring. (The exchange, called "crossover," was originally designed to reflect the process by which chromosomes exchange components in genetic mating and, in common with the step of selecting parents themselves, was organized to rely heavily on randomization. In addition, a "mutation" operation is occasionally allowed to flip bits at random.)

3) Apply a measure of fitness to decide which offspring survive to become parents for the next generation. When the selected number of matings has been performed for the current generation, return to the start of Step 2 to initiate the mating of the resulting new set of parents.

4) Carry out the mating-and-survival operation of Steps 2 and 3 until the population becomes stable or until a chosen number of iterations has elapsed.

A somewhat different model for combining elements of a population comes from a class of relaxation strategies in mathematical optimization known as surrogate constraint methods (Glover, 1965). The goal of these approaches is to generate new constraints that capture information not contained in the original problem constraints taken independently, but which is implied by their union. We will see that some unexpected connections emerge between this development and that of genetic algorithms.

The information-capturing focus of the surrogate constraint framework has the aim of developing improved methods for solving difficult optimization problems by means of (a) providing better criteria for choice rules to guide a search for improved solutions, (b) inferring new bounds (constraints with special structures) to limit the space of solutions examined. (The basic framework and strategies for exploiting it are given in Glover (1965, 1968, 1975b), Greenberg and Pierskalla (1970, 1973), Karwan and Rardin (1976, 1979), and Freville and Plateau (1986, 1993).) Based on these objectives, the generation of new constraints proceeds as indicated in Figure 5.3.

Fig. 5.3 Surrogate constraint template.

1) Begin with an initial set of problem constraints (chosen to characterize all or a special part of the feasible region for the problem considered).

2) Create a measure of the relative influence of the constraints as basis for combining subsets to generate new constraints. The new (surrogate) constraints, are created from nonnegative linear combinations of other constraints, together with cutting planes inferred from such combinations. (The goal is to determine surrogate constraints that are most effective for guiding the solution process.)

3) Change the way the constraints are combined, based on the problem constraints that are not satisfied by trial solutions generated relative to the surrogate constraints, accounting for the degree to which different source constraints are violated. Then process the resulting new surrogate constraints to introduce additional inferred constraints obtained from bounds and cutting planes. (Weaker surrogate constraints and source constraints that are determined to be redundant are discarded.)

4) Change the way the constraints are combined, based on the problem constraints that are not satisfied by trial solutions generated relative to the surrogate constraints, accounting for the degree to which different source constraints are violated. Then process the resulting new surrogate constraints to introduce additional inferred constraints obtained from bounds and cutting planes. (Weaker surrogate constraints and source constraints that are determined to be redundant are discarded.)

A natural first impression is that the surrogate constraint design is quite unrelated to the GA design, stemming from the fact that the concept of combining constraints seems inherently different from the concept of combining vectors. However in many types of problem formulations, including those where surrogate constraints were first introduced, constraints are summarized by vectors. More particularly, over time, as the surrogate constraint approach became embedded in both exact and heuristic methods, variations led to the creation of a "primal counterpart" called *scatter search*. The scatter search approach combines solution vectors by rules patterned after those that govern the generation of new constraints, and specifically inherits the strategy of exploiting linear combinations and inference (Glover, 1977).

5.3 Scatter Search

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.

The original form of scatter search may be sketched as in Figure 5.4.

Fig. 5.4 Scatter search procedure.

1) Generate a starting set of solution vectors by heuristic processes designed for the problem considered, and designate a subset of the best vectors to be *reference solutions*. (Subsequent iterations of this step, transferring from Step 3 below, incorporate advanced starting solutions and best solutions from previous history as candidates for the reference solutions.)
2) Create new points consisting of linear combinations of subsets of the current reference solutions. The linear combinations are:
   (a) chosen to produce points both inside and outside the convex regions spanned by the reference solutions.
   (b) modified by generalized rounding processes to yield integer values for integer-constrained vector components.
3) Extract a collection of the best solutions generated in Step 2 to be used as starting points for a new application of the heuristic processes of Step 1. Repeat these steps until reaching a specified iteration limit.

Three particular features of scatter search deserve mention. First, the linear combinations are structured according to the goal of generating weighted centers of selected subregions, allowing for nonconvex combinations that project these centers into regions external to the original reference solutions. The dispersion pattern created by such centers and their external projections is particularly useful for mixed integer optimization. Second, the strategies for selecting particular subsets of solutions to combine in Step 2 are designed to make use of clustering, which allows different types of strategic variation by generating new solutions "within clusters" and "across clusters". Third, the method is organized to use supporting heuristics that are able to start from infeasible solutions, and hence which remove the restriction that solutions selected as starting points for re-applying the heuristic processes must be feasible. In sum, scatter search is founded on the following premises.

(P1) Useful information about the form (or location) of optimal solutions is typically contained in a suitably diverse collection of elite solutions.

(P2) When solutions are combined as a strategy for exploiting such information, it is important to provide for combinations that can extrapolate beyond the regions spanned by the solutions considered, and further to incorporate heuristic processes to map combined solutions into new points. (This serves to provide both diversity and quality.)

(P3) 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 heuristic processes of scatter search are not restricted to a single uniform design, but represent a varied collection of procedures, affords additional strategic possibilities. This theme also shares a link with the original surrogate constraint proposal, where heuristics for surrogate relaxations are introduced to improve the application of exact solution methods. In combination, the heuristics are used to generate strengthened surrogate constraints and, iteratively applied, to generate trial solutions for integer programming problems.

The catalog in Figure 5.5 traces the links between the conceptions underlying scatter search and conceptions that have been introduced over time as amendments to the GA framework.

These innovations in the GA domain, which have subsequently been incorporated in a wide range of studies, are variously considered to be advances or heresies according to whether they are viewed from liberal or traditional perspectives. Significantly, their origins are somewhat diffuse, rather than integrated within a single framework.

Fig. 5.5 Scatter search features (1977) incorporated into non-traditional GA approaches.

- Introduction of "flexible crossover operations." (Scatter search combinations include all possibilities generated by the early GA crossover operations, and also include all possibilities embedded in the more advanced "uniform" and "Bernoulli" crossovers (Ackley (1987), Spears and DeJong (1991)). Path relinking descendants of scatter search provide further possibilities, noted subsequently.)

- Use of heuristic methods to improve solutions generated from processes for combining vectors (Muhlenbein et al. (1988), Ulder et al. (1991)), (Whitley, Gordon and Mathias (1994)).

- Exploitation of vector representations that are not restricted to binary representations (Davis (1989), Eschelman and Schaffer (1992)).

- Introduction of special cases of linear combinations for operating on continuous vectors (Davis (1989), Wright (1990), Bäck et al. (1991), Michalewicz and Janikow (1991)).

- Use of combinations of more than two parents simultaneously to produce offspring (Eiben et al. (1994), Mühlenbein and Voight (1996)).

- Introduction of strategies that subdivide the population into different groupings (Mühlenbein and Schlierkamp-Voosen (1994)).

It is clear that a number of the elements of the scatter search approach remain outside of the changes brought about by these proposals. A simple example is the approach of introducing adaptive rounding processes for mapping fractional components into integers. There also has conspicuously been no GA counterpart to the use of clustering to create strategic groupings of

points, nor (as a result) to the notion of combining points according to distinctions between membership in different clusters. (The closest approximation to this has been the use of "island populations" that evolve separately, but without concern for analyzing or subdividing populations based on inference and clustering.)

The most important distinction, however, is the link between scatter search and the theme of exploiting history. The prescriptions for combining solutions within scatter search are part of a larger design for taking advantage of information about characteristics of previously generated solutions to guide current search. In retrospect, it is perhaps not surprising that such a design should share an intimate association with the surrogate constraint framework, with its emphasis on extracting and coordinating information across different solution phases. This orientation, which takes account of elements such as the recency, frequency and quality of particular value assignments, clearly shares a common foundation with notions incorporated within tabu search. (The same reference on surrogate constraint strategies that is the starting point for scatter search is also often cited as a source of early TS conceptions.) By this means, the link between tabu search and so-called "evolutionary" approaches also becomes apparent. The term *evolutionary* has undergone an interesting evolution of its own. By a novel turn, the term "mutation" in the GA terminology has become reinterpreted to refer to any form of change, including the purposeful change produced by a heuristic process. As a result, all methods that apply heuristics to multiple solutions, whether or not they incorporate strategies for combining solutions, are now considered kindred to genetic algorithms, and the enlarged collection is labeled "evolutionary methods." (This terminology accordingly has acquired the distinction of embracing nearly every kind of method conceivable.)

5.3.1 Modern Forms and Applications of Scatter Search

Recent implementations of scatter search (cited below) have taken advantage of the implicit learning capabilities provided by the tabu search framework, leading to refined methods for determining reference points and for generating new points. Current scatter search versions have also introduced more sophisticated mechanisms to map fractional values into integer values. This work is reinforced by new theorems about searches over spaces of zero-one integer variables. Special models have also been developed to allow both heuristic and exact methods to transform infeasible trial points into feasible points. Finally, scatter search is the source of the broader class of path relinking methods, as described in Section 4, which offer a wide range of mechanisms for creating productive combinations of reference solutions. A brief summary of some of these developments appears in Figure 5.6.

Implementation of various components of these extensions have provided advances for solving general nonlinear mixed discrete optimization problems with both linear and nonlinear constraints, as noted in the references cited under Recommended Reading.

Fig. 5.6 Scatter Search Extensions.

- Tabu search memory is used to select current reference points from a historical pool (Glover, 1989, 1994a).
- Tabu search intensification and diversification strategies guide the generation of new points (Fleurent et al. 1996; Glover, Laguna and Marti, 2000).
- Solutions generated as "vector combinations" are further improved by explicit tabu search guidance (Trafalis and Al-Harkan, 1995; Glover, Kelly and Laguna, 1996; Fleurent et al., 1996; Cung, et al. 1997).
- Directional rounding processes focus the search for feasible zero-one solutions allowing them to be mapped into convex subregions of hyperplanes produced by valid cutting plane inequalities (Glover, 1995a).
- Neural network learning is applied to filter out promising and unpromising points for further examination, and pattern analysis is used to predict the location of promising new solutions (Glover, Kelly and Laguna, 1996).
- Mixed integer programming models generate sets of diversified points, and yield refined procedures for mapping infeasible points into feasible points (Glover, Kelly and Laguna, 1996).
- Structured combinations of points take the role of linear combinations, to expand the range of alternatives generated (Glover, 1994a).

5.3.2 Scatter Search and Path Relinking Interconnections

The relation between scatter search and path relinking sheds additional light on the character of these approaches. As already remarked, path relinking is a direct extension of scatter search. The way this extension comes about is as follows.

From a spatial orientation, the process of generating linear combinations of a set of reference points may be characterized as generating paths between and beyond these points (where points on such paths also serve as sources for generating additional points). This leads to a broader conception of the meaning of *combinations* of points. That is, by natural extension, we may conceive such combinations to arise by generating paths between and beyond selected points in neighborhood space, rather than in Euclidean space.

The form of these paths in neighborhood space is easily specified by reference to attribute-based memory, as used in tabu search. The path relinking strategy thus emerges as a direct consequence. Just as scatter search encompasses the possibility to generate new solutions by weighting and combining more than two reference solutions at a time, path relinking includes the possibility to generate new solutions by multi-parent path constructions that incorporate attributes from a set of guiding solutions, where these attributes are weighted to determine which moves are given higher priority, as we have seen in Section 4. The name *path relinking* comes from the fact that the generation of such paths in neighborhood space characteristically "relinks" previous points in ways not achieved in the previous search history.

The relevance of these concepts as a foundation for evolutionary procedures is illustrated by recent applications of scatter search and path relinking which have disclosed the promise of these approaches for solving a variety of optimization problems. A sampling of such applications includes:

- Vehicle Routing – Rochat and Taillard (1995); Taillard (1996)
- Quadratic Assignment – Cung *et al.* (1996)
- Financial Product Design – Consiglio and Zenios (1999)
- Neural Network Training – Kelly, Rangaswamy and Xu (1996)
- Job Shop Scheduling – Yamada and Nakano (1996)
- Flow Shop Scheduling – Yamada and Reeves (1997)
- Graph Drawing – Laguna and Marti (1999)
- Linear Ordering – Laguna, Marti and Campos (1997)
- Unconstrained Continuous Optimization – Fleurent *et al.* (1996)
- Bit Representation – Rana and Whitley (1997)
- Optimizing Simulation – Glover, Kelly and Laguna (1996)
- Complex System Optimization – Laguna (1997)

It is additionally useful to note that re-expressing scatter search relative to neighborhood space — as done in path relinking — also leads to more general forms of scatter search in Euclidean space. The form of path relinking manifested in vocabulary building (which results by using constructive and destructive neighborhoods to create and reassemble components of solutions), also suggests the relevance of combining solutions in Euclidean space by allowing different linear combinations to be created for different solution components. The design considerations that underlie vocabulary building generally carry over to this particular instance (see Glover and Laguna, 1997).

The broader conception of solution combinations provided by path relinking has useful implications for evolutionary procedures. The exploitation of neighborhood space and attribute-based memory gives specific, versatile mechanisms for achieving such combinations, and provides a further interesting connection between tabu search proposals and genetic algorithm proposals. In particular, many recently developed "crossover operators," which have no apparent relation between each other in the GA setting, can be shown to arise as special instances of path relinking, by restricting attention to two reference points (taken as parents in GAs), and by replacing the strategic neighborhood guidance of path relinking with a reliance on randomization. In short, the options afforded by path relinking for combining solutions are more unified, more systematic and more encompassing than those provided by the "crossover" concept, which changes from instance to instance and offers no guidance for how to take advantage of any given context.

5.4 Greedy Randomized Adaptive Search Procedures (GRASP)

The GRASP methodology was developed in the late 1980s, and the acronym was coined by Tom Feo (Feo and Resende, 1995). It was first used to solve computationally difficult set covering problems (Feo and Resende, 1989). Each GRASP iteration consists of constructing a trial solution and then applying an exchange procedure to find a local optimum (i.e., the final solution for that iteration). The construction phase is iterative, greedy, and adaptive. It is iterative because the initial solution is built considering one element at a time. It is greedy because the addition of each element is guided by a greedy function. It is adaptive because the element chosen at any iteration in a construction is a function of those previously chosen.

(That is, the method is adaptive in the sense of updating relevant information form iteration to iteration, as in most constructive procedures.) The improvement phase typically consists of a local search procedure.

For illustration purposes, consider the design of a GRASP for the 2-partition problem (see, e.g., Laguna et al., 1994). This problem consists of clustering the nodes of a weighted graph into two equal sized sets such that the weight of the edges between the two sets is minimized. In this context, the iterative, greedy, and adaptive elements of the GRASP construction phase may be interpreted as follows. The initial solution is built considering one node at a time. The addition of each node is guided by a greedy function that minimizes the augmented weight of the partition. The node chosen at any iteration in the construction is a function of the adjacencies of previously chosen nodes. There is also a probabilistic component in GRASP, that is applied to the selection of elements during the construction phase. After choosing the first node for one set, all non-adjacent nodes are of equal quality with respect to the given greedy function. If one of those nodes is chosen by some deterministic rule, then every GRASP iteration will repeat this selection. In such stages within a construction where there are multiple greedy choices, choosing any one of them will not compromise the greedy approach, yet each will often lead to a very different solution.

To generalize this strategy, consider forming a *candidate list* (at each stage of the construction) consisting of high quality elements according to an adaptive greedy function. Then, the next element to be included in the initial solution is randomly selected from this list. A similar strategy has been categorized as a cardinality-based semi-greedy heuristic.

The solution generated by a greedy randomized adaptive construction can generally be improved by the application of an improvement phase following selected construction phases, as by using a descent method based on an exchange mechanism, since usually the result of the construction phase is not a local minimum with respect to simple exchange neighborhoods. There is an obvious computational tradeoff between the construction and improving phases. An intelligent construction requires fewer improving exchanges to reach a local optimum, and therefore, it results in a reduction of the total CPU time required per GRASP iteration. The exchange mechanism can also be used as a basis for a hybrid method, as by incorporating elements of other methodologies such as simulated annealing or tabu search. In particular, given that the GRASP constructions inject a degree of diversification to the search process, the improvement phase may consist of a short term memory tabu search that is fine tuned for intensification purposes. Other connections may be established with methods such as scatter search or the path relinking strategy of tabu search, by using the GRASP constructions (or their associated local optima) as reference points.

Performing multiple GRASP iterations may be interpreted as a means of strategically sampling the solution space. Based on empirical observations, it has been found that the sampling distribution generally has a mean value that is inferior to the one obtained by a deterministic construction, but the best over all trials dominates the deterministic solution with a high probability. The intuitive justification of this phenomenon is based on the ordering statistics of sampling. GRASP implementations are generally robust in the sense that it is difficult to find or devise pathological instances for which the method will perform arbitrarily bad. The robustness of this method has been well documented in applications to production, flight scheduling, equipment and tool selection, location, and maximum independent sets.

An interesting connection exists between GRASP and probabilistic tabu search (PTS). If PTS is implemented in a memoryless form, and restricted to operate only in the constructive phase of a multistart procedure (stripping away memory, and even probabilistic choice, from the

improving phase), then a procedure resembling GRASP results. The chief difference is that the probabilities used in PTS are rarely chosen to be uniform over members of the candidate list, but generally seek to capture variations in the evaluations, whenever these variations reflect anticipated differences in the effective quality of the moves considered.

This connection raises the question of whether a multistart variant of probabilistic tabu search may offer a useful alternative to memoryless multistart approaches like GRASP. A study of this issue for the quadratic assignment problem, where GRASP has been reported to perform well, was conducted by Fleurent and Glover (1996). To provide a basis for comparison, the improving phases of the PTS multistart method excluded the use of TS memory and guidance strategies, and were restricted to employ a standard descent procedure. Probabilistic tabu search mechanisms were used in the constructive phases, incorporating frequency-based intensification to improve the effectiveness of successive constructions. The resulting multistart method proved significantly superior to other multistart approaches previously reported for the quadratic assignment problem. However, it also turned out to be not as effective as the leading tabu search methods that use memory in the improving phases as well as (or instead of) in the constructive phases. Nevertheless, it seems reasonable to conjecture that classes of problems exist where increased reliance on re-starting will prove advantageous, and where the best results may be obtained from appropriately designed multistart strategies such as based on greedy randomized search and multistart variants of PTS.

5.5 Neural Networks

Neural networks have a somewhat different set of goals than tabu search, although some overlaps exist. We indicate how tabu search can be used to extend certain neural net conceptions, yielding a hybrid that may have both hardware and software implications. The basic transferable insight from tabu search is that memory components with dimensions such as recency and frequency can increase the efficacy of a system designed to evolve toward a desired state. We suggest the merit of fusing neural network memory with tabu search memory as follows. (A rudimentary acquaintance with neural network ideas is assumed.)

Recency based considerations can be introduced from tabu search into neural networks by a *time delay feedback loop* from a given neuron back to itself (or from a given synapse back to itself, by the device of interposing additional neurons). This permits firing rules and synapse weights to be changed only after a certain time threshold, determined by the length of the feedback loop. Aspiration thresholds of the form conceived in tabu search can be embodied in inputs transmitted on a secondary level, giving the ability to override the time delay for altering firing thresholds and synaptic weights. Frequency based effects employed in tabu search similarly may be incorporated by introducing a form of cumulative averaged feedback.

Time delay feedback mechanisms for creating recency and frequency effects also can have other functions. In a problem solving context, for example, it may be convenient to disregard one set of options to concentrate on another, while retaining the ability to recover the suppressed options after an interval. This familiar type of human activity is not a customary part of neural network design, but can be introduced by the time dependent functions previously indicated. In addition, a threshold can be created to allow a suppressed option to "go unnoticed" if current activity levels fall in a certain range, effectively altering the interval before the option reemerges for consideration. Neural network designs to incorporate those features may directly make use of the TS ideas that have made these elements effective in the problem solving domain.

Tabu search strategies that introduce longer term intensification and diversification concerns are also relevant to neural network processes. As a foundation for blending these approaches, it is useful to adopt an orientation where a collection of neurons linked by synapses with various activation weights is treated as a set of attribute variables which can be assigned alternative values. Then the condition that synapse j (from a specified origin neuron to a specified destination neuron) is assigned an activation weight in interval p can be coded by the assignment $y_j = p$, where y_j is a component of an attribute vector y as identified in the discussion of attribute creation processes in connection with vocabulary building. A similar coding identifies the condition under which a neuron fires (or does not fire) to activate its associated synapses. As a neural network process evolves, a sequence of these attribute vectors is produced over time. The association between successive vectors may be imagined to operate by reference to a neighborhood structure implicit in the neural architecture and associated connection weights. There also may be an implicit association with some (unknown) optimization problem, or a more explicit association with a known problem and set of constraints. In the latter case, attribute assignments (neuron firings and synapse activation) can be evaluated for efficacy by transformation into a vector x, to be checked for feasibility by $x \in \mathbf{X}$. (We maintain a distinction between y and x since there may not be a one-one association between them.)

Time records identifying the quality of outcomes produced by recent firings, and identifying the frequency particular attribute assignments produce the highest quality firing outcomes, yield a basis for delaying changes in certain weight assignments and for encouraging changes in others. The concept of influence, in the form introduced in tabu search, should be considered in parallel with quality of outcomes.

Early designs to incorporate tabu search into neural networks are provided in the work of de Werra and Hertz (1989) and Beyer and Ogier (1991). These applications, which respectively treat visual pattern identification and nonconvex optimization, are reported to significantly reduce training times and increase the reliability of outcomes generated. More recent uses of tabu search to enhance the function of neural networks are provided by the studies reported in Glover and Laguna (1997).

5.6 Target Analysis

Target analysis (Glover and Greenberg, 1989) links artificial intelligence and operation research perspectives to give heuristic or exact solution procedures the ability to learn what rules are best to solve a particular class of problems. Many existing solution methods have evolved by adopting, a priori, a somewhat limited characterization of appropriate rules for evaluating decisions. An illustration is provided by restricting the definition of a "best" move to be one that produces the most attractive objective function change. However, this strategy does not guarantee that the selected move will lead the search in the direction of the optimal solution. In fact, in some settings it has been shown that the merit of such a decision rule diminishes as the number of iterations increases during a solution attempt.

As seen earlier, the tabu search philosophy is to select a best admissible move (from a strategically controlled candidate list) at each iteration, interpreting best in a broad sense that goes beyond the use of objective function measures, and relies upon historical parameters to aid in composing an appropriate evaluation. Target analysis provides a means to exploit this broader view. For example, target analysis can be used to create a dynamic evaluation function that incorporates a systematic process for diversifying the search over the longer term.

A few examples of the types of questions that target analysis can be used to answer are:

(1) Which decision rule from a collection of proposed alternatives should be selected to guide the search? (In an expanded setting, how should the rules from the collection be combined? By interpreting "decision rule" broadly, this encompasses the issue of selecting a neighborhood, or a combination of neighborhoods, as the source of a move at a given stage.) Similarly, which parameter values should be chosen to provide effective instances of the decision rules?

(2) What attributes are most relevant for determining tabu status, and what associated tabu restrictions, tabu tenures and aspiration criteria should be used?

(3) What weights should be assigned to create penalties or inducements (e.g., as a function of frequency-based memory), and what thresholds should govern their application?

(4) Which measures of quality and influence are most appropriate, and which combinations of these lead to the best results in different search phases?

(5) What features of the search trajectory disclose when to focus more strongly on intensification and when to focus more strongly on diversification? (In general, what is the best relative emphasis between intensification and diversification, and under what conditions should this emphasis change?)

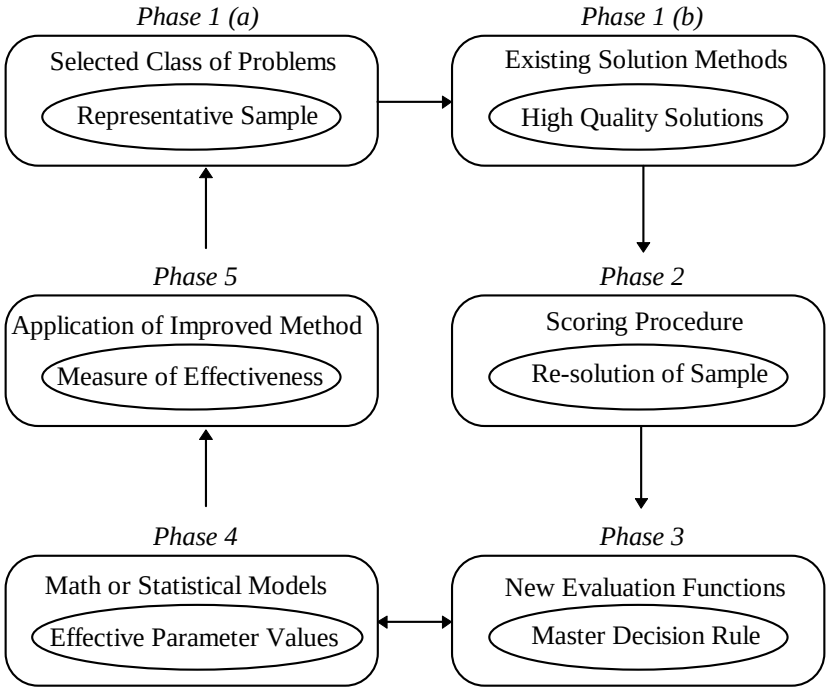
Motivation for using target analysis to answer such questions is provided by contrasting target analysis with the way answers are normally determined. Typically, an experimenter begins with a set of alternative rules and decision criteria which are intended to capture the principal elements of a given method, often accompanied by ranges of associated parameters for implementing the rules. Then various combinations of options are tried, to see how each one works for a preliminary set of test problems. However, even a modest number of rules and parameters may create a large number of possibilities in combination, and there is usually little hope of testing these with any degree of thoroughness. As a result, such testing for preferred alternatives generally amounts to a process of blind groping. Where methods boast the lack of optional parameters and rules, typically it is because the experimenter has already done the advance work to settle upon a particular combination that has been hard-wired for the user, at best with some degree of adaptiveness built in, but the process that led to this hard-wiring still raises the prospect that another set of options may be preferable.

More importantly, in an adaptive memory approach, where information from the history of the search is included among the inputs that determine current choices, a trial and error testing of parameters may overlook key elements of timing and yield no insights about relationships to be exploited. Such a process affords no way to uncover or characterize the circumstances encountered during the search that may cause a given rule to perform well or badly, and consequently gives no way to anticipate the nature of rules that may perform better than those originally envisioned. Target analysis replaces this by a systematic approach to create *hindsight before the fact*, and then undertakes to "reverse engineer" the types of rules that will lead to good solutions.

5.6.1. Target Analysis Features

The main features of target analysis may briefly be sketched by viewing the approach as a five phase procedure (see Figure 5.7). *Phase* 1 of target analysis is devoted to applying existing methods to determine optimal or exceptionally high quality solutions to representative problems of a given class. In order to allow subsequent analysis to be carried out more conveniently, the problems are often selected to be relatively small, provided this can be done in a way to assure these problems will exhibit features expected to be encountered in hard problems from the class examined.

Fig. 957 Overview of the target analysis methodology.



Although this phase is straightforward, the effort allotted to obtaining solutions of the specified quality will generally be somewhat greater than would be allotted during the normal operation of the existing solution procedures, in order to assure that the solutions have the quality sought. (Such an effort may be circumvented in cases where optimal solutions to a particular testbed of problems are known in advance.)

Phase 2 uses the solutions produced by Phase 1 as *targets,* which become the focus of a new set of solution passes. During these passes, each problem is solved again, this time scoring all available moves (or a high-ranking subset) on the basis of their ability to progress effectively toward the target solution. The scoring can be a simple classification, such as “good” or “bad,” or it may capture more refined gradations. In the case where multiple best or near best solutions may reasonably qualify as targets, the scores may be based on the target that is "closest to" the current solution.

In some implementations, choices during Phase 2 are biased to select moves that have high scores, thereby leading to a target solution more quickly than the customary choice rules. In

other implementations, the method is simply allowed to make its regular moves. In either case, the goal is to generate information during this solution effort which may be useful in inferring the solution scores. That is, the scores provide a basis for creating modified evaluations — and more generally, for creating new rules to generate such evaluations in order to more closely match them with the measures that represent "true goodness" (for reaching the targets).

In the case of tabu search intensification strategies such as elite solution recovery approaches, scores can be assigned to parameterized rules for determining the types of solutions to be saved. For example, such rules may take account of characteristics of clustering and dispersion among elite solutions. In environments where data bases can be maintained of solutions to related problems previously encountered, the scores may be assigned to rules for recovering and exploiting particular instances of these past solutions, and for determining which new solutions will be added to the data bases as additional problems are solved. (The latter step, which is part of the target analysis and not part of the solution effort can be performed "off line.") An integration of target analysis with a generalized form of sensitivity analysis for these types of applications has been developed and implemented in financial planning and industrial engineering by Glover, et al. (1998). Such designs are also relevant, for example, in applications of linear and nonlinear optimization based on simplex method subroutines, to identify sets of variables to provide crash-basis starting solution.

In path relinking strategies, scores can be applied to rules for matching initiating solutions with guiding solutions. As with other types of decision rules produced by target analysis, these will preferably include reference to parameters that distinguish different problem instances. The parameter-based rules similarly can be used to select initiating and guiding solutions from pre-existing solutions pools. Tunneling applications of path relinking, which allow traversal of infeasible regions, and strategic oscillation designs that purposely drive the search into and out of such regions, are natural accompaniments for handling recovered solutions that may be infeasible.

Phase 3 constructs parameterized functions of the information generated in Phase 2, with the goal of finding values of the parameters to create a *master decision rule*. This rule is designed to choose moves that score highly, in order to achieve the goal that underlies Phase 2. It should be noted that the parameters available for constructing a master decision rule depend on the search method employed. Thus, for example, tabu search may include parameters that embody various elements of recency-based and frequency-based memory, together with measures of influence linked to different classes of attributes or to different regions from which elite solutions have been derived.

Phase 4 transforms the general design of the master decision rule into a specific design by applying a model to determine effective values for its parameters. This model can be a simple set of relationships based on intuition, or can be a more rigorous formulation based on mathematics or statistics (such as a goal programming or discriminant analysis model, or even a "connectionist" model based on neural networks).

The components of phases 2, 3 and 4 are not entirely distinct, and may be iterative. On the basis of the outcomes of these phases, the master decision rule becomes the rule that drives the solution method. In the case of tabu search, this rule may use feedback of outcomes obtained during the solution process to modify its parameters for the problem being solved.

Phase 5 concludes the process by applying the master decision rule to the original representative problems and to other problems from the chosen solution class to confirm its merit. The process can be repeated and nested to achieve further refinement.

Target analysis has an additional important function. On the basis of the information generated during its application, and particularly during its final confirmation phase, the method produces empirical frequency measures for the probabilities that choices with high evaluations will lead to an optimal (or near-optimal) solution within a certain number of steps. These decisions are not only at tactical levels but also at strategic levels, such as when to initiate alternative solution phases, and which sources of information to use for guiding these phases (e.g., whether from processes for tracking solution trajectories or for recovering and analyzing solutions). By this means, target analysis can provide inferences concerning expected solution behavior, as a supplement to classical "worst case" complexity analysis. These inferences can aid the practitioner by indicating how long to run a solution method to achieve a solution desired quality, according to specified empirical probability.

One of the useful features of target analysis is its capacity for taking advantage of human interaction. The determination of key parameters, and the rules for connecting them, can draw directly on the insight of the observer as well as on supplementary analytical techniques. The ability to derive inferences from pre-established knowledge of optimal or near optimal solutions, instead of manipulating parameters blindly (without information about the relation of decisions to targeted outcomes), can save significant investment in time and energy. The key, of course, is to coordinate the phases of solution and guided re-solution to obtain knowledge that has the greatest utility. Many potential applications of target analysis exist, and recent applications suggest the approach holds considerable promise for developing improved decision rules for difficult optimization problems.

5.6.2 Illustrative Application and Implications

An application of target analysis to a production scheduling problem (Laguna and Glover, 1993) provides a basis for illustrating some of the relevant considerations of the approach. In this study, the moves consisted of a combination of swap and insert moves, and scores were generated to identify the degree to which a move brought a solution closer to the target solution (which consisted of the best known solution before improving the method by means of target analysis). In the case of a swap move, for example, a move might improve, worsen (or, by the measure used, leave unchanged) the "positional value" of each component of the swap, and by the simplification of assigning scores of 1, 0 or -1 to each component, a move could accordingly receive a score ranging from 2 to -2. The application of target analysis then proceeded by tracking the scores of the 10 highest evaluation moves at each iteration, to determine the circumstances under which the highest evaluations tended to correspond to the highest scores. Both tabu and non-tabu moves were included in the analysis, to see whether tabu status was also appropriately defined.

At an early stage of the analysis a surprising relationship emerged. Although the scores of the highest evaluation non-tabu moves ranged across both positive and negative values, the positive values were largely associated with moves that improved the schedule while the negative values were largely associated with moves that worsened the schedule. In short, the highest evaluations were significantly more "accurate" (corresponded more closely to high scores) during phases where the objective function value of the schedule improved than during phases when it deteriorated.

A simple diversification strategy was devised to exploit this discovery. Instead of relying on the original evaluations during "disimproving phases," the strategy supplemented the evaluations over these intervals by assigning penalties to moves whose component jobs had been moved frequently in the past. The approach was initiated at a local optimum after the progress of the

search began to slow (as measured by how often a new best solution was found), and was de-activated as soon as a move was executed that also was an improving move (to become re-activated the next time that all available moves were disimproving moves). The outcome was highly effective, producing new solutions that were often superior to the best previously found, especially for larger problems, and also finding the highest quality solutions more quickly.

The success of this application, in view of its clearly limited scope, provides an incentive for more thorough applications. For example, a more complete analysis would reasonably proceed by first seeking to isolate the high scoring moves during the disimproving phases and to determine how frequency-based memory and other factors could be used to identify these moves more effectively. Comparisons between evaluations proposed in this manner and their associated move scores would then offer a foundation for identifying more intelligent choices. Classifications to segregate the moves based on criteria other than "improving" and "disimproving" could also be investigated. Additional relevant factors that may profitably be taken into account are examined in the illustration of the next subsection.

A Hypothetical Illustration. The following hypothetical example embodies a pattern related to the one uncovered in the scheduling application cited above. However, the pattern in this case is slightly more ambiguous, and less clearly points to options that it may be exploited.

For simplicity in this illustration, suppose that moves are scored to be either "good" or "bad." (If each move changes the value of a single 0-1 variable, for instance, a move may be judged good or bad depending on whether the assigned value is the same as in the target solution. More generally, a threshold can be used to differentiate the two classifications.)

Table 5.1 indicates the percent of time each of the five highest evaluation moves, restricting attention in this case to those that are non-tabu, receives a good score during the search history. (At a first stage of conducting the target analysis, this history could be for a single hard problem, or for a small collection of such problems.) The *Move Rank* in the table ranges from 1 to 5, corresponding to the highest evaluation move, the 2nd highest evaluation move, and so on to the 5th highest evaluation move.

The indicated percent values do not total 100 because good scores may also be assigned to moves with lower evaluations, whose ranks are not included among those shown. Also, it may be expected that some non-tabu moves will also receive good scores. (A fuller analysis would similarly show ranks and scores for these moves.)

Table 5.1 Moves throughout the search history.

Move Rank	1	2	3	4	5
Percent of moves with "good" scores	22	14	10	20	16

At first glance, the table appears to suggest that the fourth and fifth ranked moves are almost as good as the first ranked move, although the percent of moves that receive good scores is not particularly impressive for any of the ranks. Without further information, a strategy might be contemplated that allocates choices probabilistically among the first, fourth and fifth ranked moves (though such an approach would not be assured to do better than choosing the first ranked move at each step). Tables 5.2 and 5.3 below provide more useful information about choices that are potentially favorable, by dividing the iterations into improving and disimproving phases as in the scheduling study previously discussed.

Table 5.2 Moves during improving phases.					
Move Rank	1	2	3	4	5
Percent of moves with "good" scores	34	21	9	14	7

Table 5.3 Moves during disimproving phases.					
Move Rank	1	2	3	4	5
Percent of moves with "good" scores	8	7	11	26	25

These tables are based on a hypothetical situation where improving and disimproving moves are roughly equal in number, so that the percent values shown in Table 9.1 are the average of the corresponding values in Tables 9.2 and 9.3. (For definiteness, moves that do not change the problem objective function may be assumed to be included in the improving phase, though a better analysis might treat them separately.)

The foregoing outcomes to an extent resemble those found in the scheduling study, though with a lower success rate for the highest evaluation improving moves. Clearly Tables 5.2 and 5.3 give information that is more exploitable than the information in Table 5.1. According to these latter tables, it would be preferable to focus more strongly on choosing one of the two highest evaluation moves during an improving phase, and one of the fourth or fifth highest evaluation moves during a disimproving phase. This conclusion is still weak in several respects, however, and we examine considerations that may lead to doing better.

Refining the Analysis. The approach of assigning scores to moves, as illustrated in Tables 5.1, 5.2 and 5.3, disregards the fact that some solution attributes (such as assignments of values to particular 0-1 variables) may be fairly easy to choose "correctly," while others may be somewhat harder. Separate tables of the type illustrated should therefore be created for easy and hard attributes (as determined by how readily their evaluations lead to choices that would generate the target solution), since the preferred rules for evaluating moves may well differ depending on the types of attributes the moves contain. Likewise, an effective strategy may require that easy and hard attributes become the focus of different search phases. The question therefore arises as to how to identify such attributes.

As a first approximation, we may consider an easy attribute to be one that often generates an evaluation that keeps it out of the solution if it belongs out, or that brings it into the solution if it belongs in. A hard attribute behaves oppositely. Thus, a comparison between frequency-based memory and move scores gives a straightforward way to differentiate these types of attributes. Both residence and transition frequencies are relevant, though residence measures are probably more usually appropriate. For example, an attribute that belongs to the current solution a high percentage of the time, and that also belongs to the target solution, would evidently qualify as easy. On the other hand, the number of times the attribute is accepted or rejected from the current solution may sometimes be less meaningful than how long it stays in or out. The fact that residence and transition frequencies are characteristically used in tabu search makes them conveniently available to assist in differentiations that can improve the effectiveness of target analysis.

5.6.3 Conditional Dependencies Among Attributes

Tables 5.1, 5.2 and 5.3 suggest that the search process that produced them is relatively unlikely to find the target solution. Even during improving phases, the highest evaluation

move is almost twice as likely to be bad as good. However, this analysis is limited, and discloses a limitation of the tables themselves. In spite of first appearances, it is entirely possible that these tables could be produced by a search process that successfully obtains the target solution (by a rule that chooses a highest evaluation move at each step). The reason is that the relation between scores and evaluations may change over time. While there may be fairly long intervals where choices are made poorly, there may be other shorter intervals where the choices are made more effectively – until eventually one of these shorter intervals succeeds in bringing all of the proper attributes into the solution.

Such behavior is likely to occur in situations where correctly choosing some attributes may pave the way for correctly choosing others. The interdependence of easy and hard attributes previously discussed is carried a step farther by these conditional relationships, because an attribute that at one point deserves to be classified hard may later deserve to be classified easy, once the appropriate foundations are laid.

Instead of simply generating tables that summarize results over long periods of the search history, therefore, it can be important to look for blocks of iterations where the success rate of choosing good moves may differ appreciably from the success rate overall. These blocks provide clues about intermediate solution compositions that may transform hard attributes into easy ones, and thus about preferred sequences for introducing attributes that may exploit conditional dependencies. The natural step then is to see which additional types of evaluation information may independently lead to identifying such sequences.

A simple instance of this type of effect occurs where the likelihood that a given attribute will correctly be selected (to enter or leave the solution) depends roughly on the number of attributes that already correctly belong to the solution. In such situations, the appropriate way to determine a "best choice" is therefore also likely to depend on this number of attributes correctly in solution. Even though such information will not generally be known during the search, it may be possible to estimate it and adjust the move evaluations accordingly. Such relationships, as well as the more general ones previously indicated, are therefore worth ferreting out by target analysis.

5.6.4 Differentiating Among Targets

In describing the steps of target analysis, it has already been noted that scores should not always be rigidly determined by only one specific target, but may account for alternative targets, and in general may be determined by the target that is closest to the current solution (by a metric that depends on the context). Acknowledging that there may be more than one good solution that is worth finding, such a differentiation among targets can prove useful. Yet even in the case where a particular solution is uniquely the one to be sought (as where its quality may be significantly better than that of all others known), alternative targets may be still be valuable to consider in the role of intermediate solutions, and may provide a springboard to finding additional solutions that are better. Making reference to intermediate targets is another way of accounting for the fact that conditional dependencies may exist among the attributes, as previously discussed. However, such dependencies in some cases may be more easily exploited by explicitly seeking constructions at particular stages that may progressively lead to a final destination.

Some elite solutions may provide better targets than others because they are easier to obtain — completely apart from developing strategies to reach ultimate targets by means of intermediate ones. However, some care is needed in making the decision to focus on such easier targets as a basis for developing choice rules. As in the study of Lokketangen and Glover (1997), it may

be that focusing instead on the harder targets will yield rules that likewise cause the easier targets to be found more readily, and these rules may apply to a wider spectrum of problems than those derived by focusing on easier targets.

5.6.5 Generating Rules by Optimization Models

Target analysis can use optimization models to generate decision rules by finding weights for various decision criteria to create a composite (master) rule. To illustrate, let G and B respectively denote index sets for good moves and bad moves, as determined from move scores, as in the classification embodied in Tables 5.1, 5.2, and 5.3. Incorporate the values of the different decision criteria in a vector A_i for $i \in G$ and $i \in B$; i.e., the j^{th} component a_{ij} of A_i is the value assigned to move i by the decision criterion j. These components need not be the result of rules, but can simply correspond to data considered relevant to constructing rules. In the tabu search setting, such data can include elements of recency-based and frequency-based memory. Then we may consider a master rule which is created by applying a weight vector w to each vector A_i to produce a composite decision value $A_i w = \sum_j a_{ij} w_j$. An ambitious objective is to find a vector w that yields

$$A_i w > 0 \qquad \text{for } i \in G$$
$$A_i w \leq 0 \qquad \text{for } i \in B$$

If such a weight vector w could be found, then all good moves would have higher evaluations by the composite criterion than all bad moves, which of course is normally too much to ask. A step toward formulating a more reasonable goal is as follows. Let $G(iter)$ and $B(iter)$ identify the sets G and B for a given iteration *iter.* Then an alternative objective is to find a w so that, at each such iteration, at least one $i \in G(iter)$ would yield

$$A_i w > A_k w \qquad \text{for all } k \in B(iter)$$

or equivalently

$$Max\{A_i w: i \in G(iter)\} > Max\{A_k w: k \in B(iter)\}$$

This outcome would insure that a highest evaluation move by the composite criterion will always be a good move. Naturally, this latter goal is still too optimistic. Nevertheless, it is possible to devise goal programming models (related to LP discriminant analysis models) that can be used to approximate this goal. A model of this type has proved to be effective for devising branching rules to solve a problem of refueling nuclear reactors (Glover, Klingman and Phillips, 1990).

A variety of opportunities exist for going farther in such strategies. For example, issues of creating nonlinear and discontinuous functions to achieve better master rules can be addressed by using trial functions to transform components of A_i vectors into new components, guided by LP sensitivity and postoptimality analysis. Target analysis ideas previously indicated can also be useful in this quest.

The range of possibilities for taking advantage of target analysis is considerable, and for the most part only the most rudimentary applications of this learning approach have been initiated. The successes of these applications make further exploration of this approach attractive.

6. Neglected Tabu Search Strategies

We briefly review several key strategies in tabu search that are often neglected (especially in beginning studies), but which are important for producing the best results.

Our purpose is to call attention to the relevance of particular elements that are mutually reinforcing, but which are not always discussed "side by side" in the literature, and which deserve special emphasis. In addition, observations about useful directions for future research are included.

A comment regarding implementation: first steps do not have to include the most sophisticated variants of the ideas discussed in the following sections, but the difference between "some inclusion" and "no inclusion" can be significant. Implementations that incorporate simple instances of these ideas will often disclose the manner in which refined implementations can lead to improved performance.

The material that follows brings together ideas described in preceding sections to provide a perspective on how they interrelate. In the process, a number of additional observations are introduced.

6.1 Candidate List Strategies

Efficiency and quality can be greatly affected by using intelligent procedures for isolating effective candidate moves, rather than trying to evaluate every possible move in a current neighborhood of alternatives. This is particularly true when such a neighborhood is large or expensive to examine. The gains to be achieved by using candidate lists have been widely documented, yet many TS studies overlook their relevance.

Careful organization in applying candidate lists, as by saving evaluations from previous iterations and updating them efficiently, can also be valuable for reducing overall effort. Time saved in these ways allows a chance to devote more time to higher level features of the search.

While the basic theme of candidate lists is straightforward, there are some subtleties in the ways candidate list strategies may be used. Considerable benefit can result by being aware of fundamental candidate list approaches, such as the *Subdivision Strategy*, the *Aspiration Plus Strategy*, the *Elite Candidate List Strategy*, the *Bounded Change Strategy* and the *Sequential Fan Strategy* (as discussed in Section 3).

An effective integration of a candidate list strategy with the rest of a tabu search method will typically benefit by using TS memory designs to facilitate functions to be performed by the candidate lists. This applies especially to the use of frequency based memory. A major mistake of some TS implementations, whether or not they make use of candidate lists, is to consider only the use of recency based memory. Frequency based memory — which itself takes different forms in intensification phases and diversification phases — cannot only have a dramatic impact on the performance of the search in general but also can often yield gains in the design of candidate list procedures. A useful way to meld different candidate list procedures is described in Glover (1997).

6.2 Intensification Approaches

Intensification strategies, which are based on recording and exploiting elite solutions or, characteristically, specific features of these solutions, have proved very useful in a variety of applications. Some of the relevant forms of such strategies and considerations for implementing them are as follows.

6.2.1 Restarting with Elite Solutions

The simplest intensification approach is the strategy of recovering elite solutions in some order, each time the search progress slows, and then using these solutions as a basis for re-initiating the search. The list of solutions that are candidates to be recovered is generally limited in size, often in the range of 20 to 40 (although in parallel processing applications the number is characteristically somewhat larger). The size chosen for the list in serial TS applications also corresponds roughly to the number of solution recoveries anticipated to be done during the search, and so may be less or more depending on the setting. When an elite solution is recovered from the list, it is removed, and new elite solutions are allowed to replace less attractive previous solutions — usually dropping the worst of the current list members. However, if a new elite solution is highly similar to a solution presently recorded, instead of replacing the current worst solution, the new solution will compete directly with its similar counterpart to determine which solution is saved.

This approach has been applied very effectively in job shop and flow shop scheduling, in vehicle routing, and in telecommunication design problems. One of the best approaches for scheduling applications keeps the old TS memory associated with the solution, but makes sure the first new move away from this solution goes to a different neighbor than the one visited after encountering this solution the first time. Another effective variant does not bother to save the old TS memory, but uses a probabilistic TS choice design.

The most common strategy is to go through the list from best to worst, but in some cases it has worked even better to go through the list in the other direction. In this approach, it appears effective to allow two passes of the list. On the first pass, when a new elite solution is found that falls below the quality of the solution currently recovered, but which is still better than the worst already examined on the list, the method still adds the new solution to the list and displaces the worst solution. Then a second pass, after reaching the top of the list, recovers any added solutions not previously recovered.

6.2.2 Frequency of Elite Solutions

Another primary intensification strategy is to examine elite solutions to determine the frequency in which particular solution attributes occur (where the frequency is typically weighted by the quality of the solutions in which the attributes are found).

This strategy was originally formulated in the context of identifying “consistent” and “strongly determined” variables — where, loosely speaking, consistent variables are those more frequently found in elite solutions, while strongly determined variables are those that would cause the greatest disruption by changing their values (as sometimes approximately measured by weighting the frequencies based on solution quality). The idea is to isolate the variables that qualify as more consistent and strongly determined (according to varying thresholds), and then to generate new solutions that give these variables their “preferred values.” This can be done either by rebuilding new solutions in a multistart approach or by modifying the choice rules of an ongoing solution effort to favor the inclusion of these value assignments.

Keeping track of the frequency that elite solutions include particular attributes (such as edges of tours, assignments of elements to positions, narrow ranges of values taken on by variables, etc.) and then favoring the inclusion of the highest frequency elements, effectively allows the search to concentrate on finding the best supporting uses and values of other elements. A simple variant is to "lock in" a small subset of the most attractive attributes (value assignments) — allowing this subset to change over time or on different passes.

A Relevant Concern: In the approach that starts from a current (good) solution, and tries to bring in favored elements, it is important to introduce an element that yields a best outcome from among the current contenders (where, as always, best is defined to encompass considerations that are not solely restricted to objective function changes). If an attractive alternative move shows up during this process, which does not involve bringing in one of these elements, aspiration criteria may determine whether such a move should be taken instead. Under circumstances where the outcome of such a move appears sufficiently promising, the approach may be discontinued and allowed to enter an improving phase (reflecting a decision that enough intensification has been applied, and it is time to return to searching by customary means).

Intensification of this form makes it possible to determine what percent of "good attributes" from prior solutions should be included in the solution currently generated. It also gives information about which subsets of these attributes should go together, since it is preferable not to choose attributes during this process that cause the solution to deteriorate compared to other choices. This type of intensification strategy has proved highly effective in the settings of vehicle routing and zero-one mixed integer optimization.

6.2.3 Memory and Intensification

It is clearly somewhat more dangerous to hold elements "in" solution than to hold them "out" (considering that a solution normally is composed of a small fraction of available elements — as where a tree contains only a fraction of the edges of a graph). However, there is an important exception, previously intimated. As part of a longer term intensification strategy, elements may be selected very judiciously to be "locked in" on the basis of having occurred with high frequency in the best solutions found. In that case, choosing different mutually compatible (and mutually reinforcing) sets to lock in can be quite helpful. This creates a *combinatorial implosion* effect (opposite to a combinatorial explosion effect) that shrinks the solution space to a point where best solutions over the reduced space are likely to be found more readily.

The key to this type of intensification strategy naturally is to select an appropriate set of elements to lock in, but the chances appear empirically to be quite high that some subset of those with high frequencies in earlier best solutions will be correct. Varying the subsets selected gives a significant likelihood of picking a good one. (More than one subset can be correct, because different subsets can still be part of the same complete set.) Aspiration criteria make it possible to drop elements that are supposedly locked in, to give this approach more flexibility.

6.2.4 Relevance of Clustering for Intensification

A search process over a complex space is likely to produce clusters of elite solutions, where one group of solutions gives high frequencies for one set of attributes and another group gives high frequencies for a different set. It is important to recognize this situation when it arises. Otherwise there is a danger that an intensification strategy may try to compel a solution to

include attributes that work against each other. This is particularly true in a strategy that seeks to generate a solution by incorporating a collection of attributes "all at once," rather than using a step by step evaluation process that is reapplied at each move through a neighborhood space. (Stepping through a neighborhood has the disadvantage of being slower, but may compensate by being more selective. Experimentation to determine the circumstances under which each of these alternative intensification approaches may be preferable would be quite valuable.)

A strategy that incorporates a block of attributes together may yield benefits by varying both the size and composition of the subsets of high frequency "attractive" attributes, even if these attributes are derived from solutions that lie in a common cluster, since the truly best solutions may not include them all. Threshold based forms of logical restructuring, as discussed in Section 3, may additionally lead to identifying elements to integrate into solutions that may not necessarily belong to solutions previously encountered. The vocabulary building theme becomes important in this connection. The relevance of clustering analysis for logical restructuring and vocabulary building is reinforced by the use of a related conditional analysis, which is examined subsequently in Section 6.5.

6.3 Diversification Approaches

Diversification processes in tabu search are sometimes applied in ways that limit their effectiveness, due to overlooking the fact that diversification is not just "random" or "impulsive," but depends on a purposeful blend of memory and strategy. As noted in Section 3, recency and frequency based memory are both relevant for diversification. Historically, these ideas stem in part from proposals for exploiting surrogate constraint methods. In this setting, the impetus is not simply to achieve diversification, but to derive appropriate weights in order to assure that evaluations will lead to solutions that satisfy required conditions (see Section 5). Accordingly, it is important to account for elements such as how often, to what extent, and how recently, particular constraints have been violated, in order to determine weights that produce more effective valuations.

The implicit *learning effects* that underlie such uses of recency, frequency and influence are analogous to those that motivate the procedures used for diversification (and intensification) in tabu search. Early strategic oscillation approaches exploited this principle by driving the search to various depths outside (and inside) feasibility boundaries, and then employing evaluations and directional search to move toward preferred regions.

In the same way that these early strategies bring diversification and intensification together as part of a continuously modulated process, it is important to stress that these two elements should be interwoven in general. A common mistake in many TS implementations is to apply diversification without regard for intensification. "Pure" diversification strategies are appropriate for truly long term strategies, but over the intermediate term, diversification is generally more effective if it is applied by heeding information that is also incorporated in intensification strategies. In fact, intensification by itself can sometimes cause a form of diversification, because intensifying over part of the space allows a broader search of the rest of the space. A few relevant concerns are as follows.

6.3.1 Diversification and Intensification Links

A simple and natural diversification approach is to keep track of the frequency that attributes occur in non-elite solutions, as opposed to solutions encountered in general, and then to

periodically discourage the incorporation of attributes that have modest to high frequencies (giving greater penalties to larger frequencies). The reference to non-elite solutions tends to avoid penalizing attributes that would be encouraged by an intensification strategy.

More generally, for a "first level" balance, an Intermediate Term Memory matrix may be used, where the high frequency items in elite solutions are not penalized by the long term values, but may even be encouraged. The tradeoffs involved in establishing the degree of encouragement, or the degree of reducing the penalties, represents an area where a small amount of preliminary testing can be valuable. This applies as well to picking thresholds to identify high frequency items. (Simple guesses about appropriate parameter values can often yield benefits, and tests of such initial guesses can build an understanding that leads to increasingly effective strategies.)

By extension, if an element has never or rarely been in a solution generated, then it should be given a higher evaluation for being incorporated in a diversification approach if it was "almost chosen" in the past but didn't make the grade. This observation has not been widely heeded, but is not difficult to implement, and is relevant to intensification strategies as well. The relevant concerns are illustrated in the discussion of "Persistent Attractiveness" and "Persistent Voting" in Chapter 7 of Glover and Laguna (1997).

6.3.2 Implicit Conflict and the Importance of Interactions

Current evaluations also should not be disregarded while diversification influences are activated. Otherwise, a diversification process may bring elements together that conflict with each other, make it harder rather than easier to find improved solutions.

For example, a design that gives high penalties to a wide range of elements, without considering interactions, may drive the solution to avoid good combinations of elements. Consequently, diversification — especially in intermediate term phases — should be carried out for a limited number of steps, accompanied by watching for and sidestepping situations where indiscriminately applying penalties would create incompatibilities or severe deterioration of quality. To repeat the theme: even in diversification, attention to quality is important. And as in "medical remedies," sometimes small doses are better than large ones. Larger doses (i.e., more radical departures from previous solutions) which are normally applied less frequently, can still benefit by coordinating the elements of quality and change.

6.3.3 Reactive Tabu Search

An approach called Reactive Tabu Search (RTS) developed by Battiti and Tecchiolli (1992, 1994a) deserves additional consideration as a way to achieve a useful blend of intensification and diversification. RTS incorporates hashing in a highly effective manner to generate attributes that are very nearly able to differentiate among distinct solutions. That is, very few solutions contain the same hashed attribute, applying standard hash function techniques. Accompanying this, Battiti and Tecchiolli use an automated tabu tenure, which begins with the value of 1 (preventing a hashed attribute from being reinstated if this attribute gives the "signature" of the solution visited on the immediately preceding step). This tenure is then increased if examination shows the method is possibly cycling, as indicated by periodically generating solutions that produce the same hashed attribute.

The tabu tenure, which is the same for all attributes, is increased exponentially when repetitions are encountered, and decreased gradually when repetitions disappear. Under circumstances where the search nevertheless encounters an excessive number of repetitions

within a given span (i.e., where a moving frequency measure exceeds a certain threshold), a diversification step is activated, which consists of making a number of random moves proportional to a moving average of the cycle length.

The reported successes of this approach invite further investigations of its underlying ideas and related variants. As a potential bases for generating such variants, attributes created by hashing may be viewed as *fine grain* attributes, which give them the ability to distinguish among different solutions. By contrast, "standard" solution attributes, which are the raw material for hashing, may be viewed as *coarse grain* attributes, since each may be contained in (and hence provide a signature for) many different solutions. Experience has shown that tabu restrictions based on coarse grain attributes are often advantageous for giving increased vigor to the search. (There can exist a variety of ways of defining and exploiting attributes, particularly at coarser levels, which complicates the issue somewhat.) This raises the question of when particular degrees of granularity are more effective than others.

It seems reasonable to suspect that fine grain attributes may yield greater benefits if they are activated in the vicinity of elite solutions, thereby allowing the search to scour "high quality terrain" more minutely. This effect may also be achieved by reducing tabu tenures for coarse grain attributes — or basing tabu restrictions on attribute conjunctions — and using more specialized aspiration criteria. Closer scouring of critical regions can also be brought about by using strongly focused candidate list strategies, such as a sequential fan candidate list strategy. (Empirical comparisons of such alternatives to hashing clearly would be of interest.) On the other hand, as documented by Nonobe and Ibaraki (1998, 2001), the use of "extra coarse grain" attributes (those that prohibit larger numbers of moves when embodied in tabu restrictions) can prove advantageous for solving large problems over a broadly defined problem domain.

Another type of alternative to hashing also exists, which is to create new attributes by processes that are not so uniform as hashing. A potential drawback of hashing is its inability to distinguish the relative importance (and appropriate influence) of the attributes that it seeks to map into others that are fine grained. A potential way to overcome this drawback is to make use of vocabulary building (Glover and Laguna, 1997) and of conditional analysis (Section 6.5).

6.4 Strategic Oscillation

A considerable amount has been written on strategic oscillation and its advantages. However, one of the uses of this approach that is frequently overlooked involves the idea of oscillating among alternative choice rules and neighborhoods. As stressed in Section 4, an important aspect of strategic oscillation is the fact that there naturally arise different types of moves and choice rules that are appropriate for negotiating different regions and different directions of search. Thus, for example, there are many constructive methods in graph and scheduling problems, but strategic oscillation further leads to the creation of complementary "destructive methods" which can operate together with their constructive counterparts. Different criteria emerge as relevant for selecting a move to take on a constructive step versus one to take on a destructive step. Similarly, different criteria apply according to whether moves are chosen within a feasible region or outside a feasible region (and whether the search is moving toward or away from a feasibility boundary).

The variation among moves and evaluations introduces an inherent vitality into the search that provides one of the sources underlying the success of strategic oscillation approaches. This reinforces the motivation to apply strategic oscillation to the choice of moves and evaluation

criteria themselves, selecting moves from a pool of possibilities according to rules for transitioning from one choice to another. In general, instead of picking a single rule, a process of invoking multiple rules provides a range of alternatives that run all the way from "strong diversification" to "strong intensification."

This form of oscillation has much greater scope than may at first be apparent, because it invokes the possibility of simultaneously integrating decision rules and neighborhoods, rather than only visiting them in a strategically determined sequence.

Such concepts are beginning to find counterparts in investigations being launched by the computer science community. The "agent" terminology is being invoked in such applications to characterize different choice mechanisms and neighborhoods as representing different agents. Relying on this representation, different agents then are assigned to work on (or "attack") the problem serially or in parallel. The CS community has begun to look upon this as a significant innovation — unaware of the literature where such ideas were introduced a decade or more ago — and the potential richness and variation of these ideas still seems not to be fully recognized. For example, there have not yet been any studies that consider the idea of "strategically sequencing" rules and neighborhoods, let alone those that envision the notion of parametric integration. The further incorporation of adaptive memory structures to enhance the application of such concepts also lies somewhat outside the purview of most current CS proposals. At the same time, however, TS research has also neglected to conduct empirical investigations of the broader possibilities. This is clearly an area that deserves fuller study.

6.5 Clustering and Conditional Analysis

To reinforce the theme of identifying opportunities for future research, we provide an illustration to clarify the relevance of clustering and conditional analysis, particularly as a basis for intensification and diversification strategies in tabu research.

An Example: Suppose 40 elite solutions have been saved during the search, and each solution is characterized as a vector $\boldsymbol{x}$ of zero-one variables x_j, for $j \in N = \{1,\ldots,n\}$. Assume the variables that receive positive values in at least one of the elite solutions are indexed x_1 to x_{30}. (Commonly in such circumstances, n may be expected to be somewhat larger than the number of positive valued variables, e.g., in this case, reasonable values may be n = 100 or 1000.)

For simplicity, we restrict attention to a simple weighted measure of consistency which is given by the frequency that the variables x_1 to x_{30} receive the value 1 in these elite solutions. (We temporarily disregard weightings based on solution quality and other aspects of "strongly determined" assignments.) Specifically, assume the frequency measures are as shown in Table 6.1.

Since each of x_1 to x_{15} receives a value of 1 in 24 of the 40 solutions, these variables tie for giving "most frequent" assignments. An intensification strategy that favors the inclusion of some number of such assignments would give equal bias to introducing each of x_1 to x_{15} at the value 1. (Such a bias would typically be administrated either by creating modified evaluations or by incorporating probabilities based on such evaluations.)

Table 6.1 Frequency measures.	
Variables $x_j = 1$	Number of Solutions
x_1 to x_{15}	24
x_{16} to x_{20}	21
x_{21} to x_{25}	17
x_{26} to x_{30}	12

To illustrate the relevance of clustering, suppose the collection of 40 elite solutions can be partitioned into two subsets of 20 solutions each, whose characteristics are summarized in Table 6.2.

Table 6.2 Frequency measures for two subsets.			
Subset 1 (20 solutions)		Subset 2 (20 solutions)	
Variables $x_j = 1$	No. of Solutions	Variables $x_j = 1$	No. of Solutions
x_{11} to x_{15}	20	x_{16} to x_{20}	20
x_{21} to x_{25}	16	x_6 to x_{10}	16
x_1 to x_5	12	x_1 to x_5	12
x_6 to x_{10}	8	x_{26} to x_{30}	8
x_{26} to x_{30}	4	x_{11} to x_{15}	4
x_{16} to x_{20}	1	x_{21} to x_{25}	1

A very different picture now emerges. The variables x_1 to x_{15} no longer appear to deserve equal status as "most favored" variables. Treating them with equal status may be a useful source of diversification, as opposed to intensification, but the clustered data provide more useful information for diversification concerns as well. In short, clustering gives a relevant contextual basis for determining the variables (and combinations of variables) that should be given special treatment.

6.5.1 Conditional Relationships

To go a step beyond the level of differentiation provided by cluster analysis, it is useful to sharpen the focus by referring explicitly to interactions among variables. Such interactions can often be identified in a very straightforward way, and can form a basis for more effective clustering. In many types of problems, the number of value assignments (or the number of "critical attributes") needed to specify a solution is relatively small compared to the total number of problem variables. (For example, in routing, distribution and telecommunication applications, the number of links contained in feasible constructions is typically a small fraction of those contained in the underlying graph.) Using a 0-1 variable representation of possibilities, it is not unreasonable in such cases to create a *cross reference* matrix, which identifies variables (or coded attributes) that simultaneously receive a value of 1 in a specific collection of elite solutions.

To illustrate, suppose the index set $P = \{1,\ldots,p\}$ identifies the variables x_j that receive a value of 1 in at least r solutions from the collection of elite solutions under consideration. (Apart from other strategic considerations, the parameter r can also be used to control the size of p, since larger values of r result in smaller values of p.)

Then create a $p \times p$ symmetric matrix **M** whose entries m_{ij} identify the number of solutions in which x_i and x_j are both 1. (Thus, row M_i of **M** represents the sum of the solution vectors in which $x_i = 1$, restricted to components x_j for $j \in P$.) The value m_{ii} identifies the total number of elite solutions in which $x_i = 1$, and the value m_{ij}/m_{ii} represents the "conditional probability" that $x_j = 1$ in this subset of solutions. Because p can be controlled to be of modest size, as by the choice of r and the number of solutions admitted to the elite set, the matrix **M** is not generally highly expensive to create or maintain.

By means of the conditional probability interpretation, the entries of **M** give a basis for a variety of analyses and choice rules for incorporating preferred attributes into new solutions. Once an assignment $x_j = 1$ is made in a solution currently under consideration (which may be either partly or completely constructed), an updated conditional matrix **M** can be created by restricting attention to elite solution vectors for which $x_j = 1$. (Restricted updates of this form can also be used for look-ahead purposes.) Weighted versions of **M**, whose entries additionally reflect the quality of solutions in which specific assignments occur, likewise can be used.

Critical event memory provides a convenient mechanism to maintain appropriate variation when conditional influences are taken into account. The "critical solutions" associated with such memory in the present case are simply those constituting a selected subset of elite solutions. Frequency measures for value assignments can be obtained by summing these solution vectors for problems with 0-1 representations and the critical event control mechanisms can then assure assignments are chosen to generate solutions that differ from those of previous elite solutions.

Conditional analysis, independent of such memory structures, can also be a useful foundation for generating solution fragments to be exploited by vocabulary building processes.

6.6 Referent-Domain Optimization

Referent-domain optimization is based on introducing one or more optimization models to strategically restructure the problem or neighborhood, accompanied by auxiliary heuristic or algorithmic process to map the solutions back to the original problem space. The optimization models are characteristically devised to embody selected heuristic goals (e.g., of intensification, diversification or both), within the context of particular classes of problems.

There are several ways to control the problem environment as a basis for applying referent-domain optimization. A natural control method is to limit the structure and range of parameters that define a neighborhood (or the rules used to navigate through a neighborhood), and to create an optimization model that operates under these restricted conditions.

The examples that follow assume the approach starts from a current trial solution, which may or may not be feasible. The steps described yield a new solution, and then the step is repeated, using tabu search as a master guiding strategy to avoid cycling, and to incorporate intensification and diversification.

Example 1. A heuristic selects k variables to change values, holding other variables constant. An exact method determines the (conditionally) optimal new values of the k selected variables.

Example 2. A heuristic identifies a set of restrictive bounds that bracket the values of the variables in the current trial solution (where the bounds may compel some variables to take on

a single value). An exact method determines an optimal solution to the problem as modified to include these bounds.

Example 3. A heuristic selects a restructured and exploitable region around the current solution to search for an alternative solution. An exact method finds the best solution in this region.

Example 4. For add/drop neighborhoods, a heuristic chooses k elements to add (or to drop). For example, the heuristic may operate by both adding and dropping k specific elements, as in k-opt moves for the TSP or k-swap moves for graph bipartitioning that add and drop k nodes. Then, attention is restricted to consider only the subset of elements added or the subset of elements dropped (and further restricted in the case of a bipartitioning problem to just one of the two sets). Then an exact method identifies the remaining k elements to drop (or to add), that will complete the move optimally.

Example 5. A heuristic chooses a modified problem formulation, that also admits the current trial solution as a trial solution. (For example, the heuristic may relax some part of the formulation and/or restrict another part.) An exact method then finds an optimal solution to the modified formulation. An illustration occurs where a two phase exact algorithm first finds an optimal solution to a relaxed portion of the problem, and then finds an optimal solution to a restricted portion. Finally, a small part of the feasible region of the original problem close to or encompassing this latter solution is identified, and an exact solution method finds an optimal solution in this region.

Example 6. The use of specially constructed neighborhoods (and aggregations or partitions of integer variables) permits the application of mixed integer programming (MIP) models to identify the best options from all moves of depth at most k (or from associated collections of at most k variables). When k is sufficiently small, such MIP models can be quite tractable, and produce moves considerably more powerful than those provided by lower level heuristics.

Example 7. In problems with graph-related structures, the imposition of directionality or non-looping conditions gives a basis for devising generalized shortest path (or dynamic programming) models to generate moves that are optimal over a significant subclass of possibilities. This type of approach gives rise to a combinatorial leverage phenomenon, where a low order effort (e.g., linear or quadratic) can yield solutions that dominate exponential numbers of alternatives. (See, e.g., Glover, 1992; Punnen and Glover, 1997: Rego and Glover, 2009.)

Example 8. A broadly applicable control strategy, similar to that of a relaxation procedure but more flexible, is to create a proxy model that resembles the original problem of interest, and which is easier to solve. Such an approach must be accompanied with a method to transform the solution to the proxy model into a trial solution for the original problem. A version of such an approach, which also induces special structure into the proxy model, can be patterned after layered surrogate/Lagrangean decomposition strategies for mixed integer optimization.

Referent-domain optimization can also be applied in conjunction with target analysis to create more effective solution strategies. In this case, a first stage learning model, based on controlled solution attempts, identifies a set of desired properties of good solutions, together with target solutions (or target regions) that embody these properties. Then a second stage model is devised to generate neighborhoods and choice rules to take advantage of the outcomes of the learning model. Useful strategic possibilities are created by basing these two models on a

proxy model for referent-domain optimization, to structure the outcomes so that they may be treated by one of the control methods indicated in the foregoing examples.

Conclusion

It is natural to be tempted to implement the most rudimentary forms of a method. More than a few papers on tabu search examine only a small portion of the elements of short term memory, and examine little or nothing at all of longer term memory. Unfortunately, in some cases these papers also present themselves as embodying the essence of tabu search.

A factor that has reinforced the tendency to examine a limited part of tabu search (aside from convenience, which can be sensible in early stages of an investigation), is that such a focus has sometimes produced very appealing results. When reasonably decent outcomes can be found without great effort, the motive to look further is diminished. The danger, of course, lies in failing to discover significant gains that are likely to be achieved by a more complete approach.

It is appropriate to acknowledge that attention may be given to a limited subset of ideas from an overall search framework for the following reasons:

(1) such a focus may help to uncover a better form for the strategies associated with this subset.
(2) weaknesses of this subset, when studied in isolation from other ideas, may stand out more clearly, thus yielding insights into the features of a more complete approach that are required to produce a better method;
(3) for methods which are susceptible to highly "modular" implementations, as typically occurs for tabu search, simpler designs can readily be made a part of more complex designs.

Nevertheless, in many settings, tabu search implementations that incorporate a more comprehensive set of its basic strategies typically perform appreciably better than implementations that restrict consideration to a narrow set of such strategies.

A great deal remains to be learned about tabu search. Evidently, we also still know very little about how we ourselves use memory in our problem solving. It is not inconceivable that discoveries about effective uses of memory within our search methods will provide clues about strategies that humans are adept at employing — or may advantageously be taught to employ. The potential links between the areas of heuristic search and psychology have scarcely been examined. Unquestionably, in the realm of optimization, we have not yet investigated the strategic possibilities at a level that comes close to disclosing their full potential. The numerous successes of tabu search implementations provide encouragement that such issues are profitable to probe more fully. Some of the opportunities and challenges involved are discussed in Glover (2007).

Recent fundamental advances in applications of tabu search have been assembled in a collection of “Tabu Search Vignettes“ which can be accessed via the internet at http://spot.colorado.edu/~glover. These include summaries of key developments in a variety of areas, including:

Constraint Solving and Its Applications (Resource Assignment, Planning and Timetabling, Integer Programming Feasibility, Satisfiability, Mobile Network Frequency Assignment)

Chemical Industry Applications (Computer Aided Molecular Design (CAMD), Heat Exchanger Network (HEN) Synthesis, Phase Equilibrium Calculations, Gibbs Free Energy Minimization, Optimal Component Lumping Problems)
Classification
Feature Selection
Satellite Range Scheduling
Maritime Transportation for International Trade
Conservation Area Network Design
High Level Synthesis
Graph Coloring
Delivery
Routing with Loading and Inventory Constraints
Heterogeneous Routing and Scheduling
Capacitated Facility Location
Multi-period Forest Harvesting
Manpower Scheduling
DNA Sequencing
Airline Disruption Management
Internet Traffic Engineering
Matrix Bandwidth Minimization
Generalized Assignment
Constraint Satisfaction (Work Shift Scheduling, Set-Covering and Nurse Scheduling)
Resource-Constrained Project Scheduling
Dynamic Optimization (Trade Market Prediction, Meteorological Forecast, Robotics Motion Control)

Additional topics and references related to tabu search, including these vignettes, will also be featured in the website http://www.tabusearch.info/ which is scheduled to debut in November 2012.

Cross-References

Algorithms and Metaheuristics for Combinitorial Matrices 00013
Binary Unconstrained Quadratic Optimization Problem 00015
Fuzzy Combinatorial Optimization Problems 00068
Neural Network Models in Combinatorial Optimization 00065

Recommended Reading

Ackley, D. (1987) "*A Connectionist Model for Genetic Hillclimbing*," Kluwer, Dordrecht. Academic Publishers.

Bäck, T., F. Hoffmeister and H. Schwefel (1991) "A Survey of Evolution Strategies," *Proceedings of the Fourth International Conference on Genetic Algorithms*, R. Belew and L. Booker (eds.), pp. 2-9.

Battiti, R. and G. Tecchiolli (1992) "Parallel Based Search for Combinatorial Optimization: Genetic Algorithms and Tabu Search," *Microprocessor and Microsystems*, Vol. 16, pp. 351-367.

Battiti, R. and G. Tecchiolli (1994a) "The Reactive Tabu Search," *ORSA Journal on Computing*, Vol. 6, No. 2, pp. 126-140.

Beyer, D. and R. Ogier (1991) "Tabu Learning: A Neural Network Search Method for Solving Nonconvex Optimization Problems," *Proceedings of the International Conference in Neural Networks,* IEEE and INNS, Singapore.

Consiglio, A. and S.A. Zenios (1999) "Designing Portfolios of Financial Products via Integrated Simulation and Optimization Models," *Operations Research*, Vol. 47, No. 2, pp. 195-208.

Cung, V-D., T. Mautor, P. Michelon, A. Tavares (1996) "Scatter Search for the Quadratic Assignment Problem," Laboratoire PRiSM-CNRS URA 1525.

Davis, L. (1989) "Adapting Operator Probabilities in Genetic Algorithms," *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, pp. 61-69.

Eiben, A. E., P-E Raue and Zs. Ruttkay (1994) "Genetic Algorithms with Multi-Parent Recombination," *Proceedings of the Third International Conference on Parallel Problem Solving from Nature* (PPSN), Y. Davidor, H-P Schwefel and R. Manner (eds.), New York: Springer-Verlag, pp. 78-87.

Eschelman, L. J. and J. D. Schaffer (1992) "Real-Coded Genetic Algorithms and Interval-Schemata," Technical Report, Phillips Laboratories.

Feo, T. and M. G. C. Resende (1989) "A probabilistic Heuristic for a Computationally Difficult Set Covering Problem," *Operations Research Letters,* Vol. 8, pp. 67-71.

Feo, T. and M. G. C. Resende (1995) "Greedy Randomized Adaptive Search Procedures," *Journal of Global Optimization,* Vol. 2, pp. 1-27.

Fleurent, C., F. Glover, P. Michelon and Z. Valli (1996) "A Scatter Search Approach for Unconstrained Continuous Optimization," *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, 643-648.

Freville, A. and G. Plateau (1986) "Heuristics and Reduction Methods for Multiple Constraint 0-1 Linear Programming Problems," *European Journal of Operational Research*, 24, 206-215.

Freville, A. and G. Plateau (1993) "An Exact Search for the Solution of the Surrogate Dual of the 0-1 Bidimensional Knapsack Problem," *European Journal of Operational Research*, 68, 413-421.

Glover, F. (1963) "Parametric Combinations of Local Job Shop Rules," Chapter IV, ONR Research Memorandum no. 117, GSIA, Carnegie Mellon University, Pittsburgh, PA.

Glover, F. (1968) "Surrogate Constraints," *Operations Research*, 16, 741-749.

Glover, F. (1975) "Surrogate Constraint Duality in Mathematical Programming," *Operations Research*, 23, 434-451.

Glover, F. (1977) "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Sciences*, Vol 8, No 1, 156-166.

Glover, F. (1989) "Tabu Search — Part I," *ORSA Journal on Computing*, Vol. 1, pp. 190-206.

Glover, F. (1992) "Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems," University of Colorado. Shortened version published in *Discrete Applied Mathematics*, 1996, 65, 223-253.

Glover, F. (1994a) "Genetic Algorithms and Scatter Search: Unsuspected Potentials," S*tatistics and Computing*, 4, 131-140.

Glover, F. (1995a) "Scatter Search and Star-Paths: Beyond the Genetic Metaphor," *OR Spektrum*, Vol. 17, pp. 125-137.

Glover, F. (1997) "A Template for Scatter Search and Path Relinking," in *Artificial Evolution*: *Lecture Notes in Computer Science*, 1363, J.K. Hao, E. Lutton, E. Ronald, M. Schoenauer, D. Snyers (Eds.), Springer, pp. 13-54.

Glover, F (2007) "Tabu Search – Uncharted Domains," Annals of Operations Research, Vol. 149, No. 1, pp. 89-98.

Glover, F. and H. Greenberg (1989) "New Approaches for Heuristic Search: A Bilateral Linkage with Artificial Intelligence," *European Journal of Operational Research,* Vol. 39, No. 2, pp. 119-130.

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

Glover, F. and G. Kochenberger (1996) "Critical Event Tabu Search for Multidimensional Knapsack Problems," *Meta-Heuristics: Theory and Applications*, I. H. Osman and J. P. Kelly (eds.), Kluwer Academic Publishers, pp. 407-427.

Glover, F., G. Kochenberger and B. Alidaee (1998) "Adaptive Memory Tabu Search for Binary Quadratic Programs," *Management Science*, Vol. 44, No. 3, pp. 336-345

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

Glover, F., M. Laguna and R. Marti (2000) "Fundamentals of Scatter Search and Path Relinking," *Control and Cybernetics*, volume 29, number 3, pp. 653-684.

Glover, F., J. Mulvey, D. Bai, and M. Tapia (1998) "Integrative Population Analysis for Better Solutions to Large-Scale Mathematical Programs," *Industrial Applications of Combinatorial Optimization*, G. Yu, Ed. Kluwer Academic Publishers, Boston, MA, pp. 212-237.

Greenberg, H. J. and Pierskalla, W.P. (1970) "Surrogate Mathematical Programs," *Operations Research*, 18, 924-939.

Greenberg, H. J. and W. P. Pierskalla (1973) "Quasi-conjugate Functions and Surrogate Duality," *Cahiers du Centre d'Etudes de Recherche Operationelle*, Vol. 15, pp. 437-448.

Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

Johnson, D. S. (1990) "Local Optimization and the Traveling Salesman Problem," *Proc. 17th Intl. Colloquium on Automata, Languages and Programming,* pp. 446-460.

Karwan, M.H. and R.L. Rardin (1976) "Surrogate Dual Multiplier Search Procedures in Integer Programming," School of Industrial Systems Engineering, Report Series No. J-77-13, Georgia Institute of Technology.

Karwan, M.H. and R.L. Rardin (1979) "Some Relationships Between Lagrangean and Surrogate Duality in Integer Programming," *Mathematical Programming*, 17, 230-334.

Kelly, J., B. Rangaswamy and J. Xu (1996) "A Scatter Search-Based Learning Algorithm for Neural Network Training," *Journal of Heuristics*, Vol. 2, pp. 129-146.

Laguna, M. (1997) "Optimizing Complex Systems with OptQuest," Research Report, University of Colorado

Laguna, M., T. Feo and H. Elrod (1994) "A Greedy Randomized Adaptive Search Procedure for the 2-Partition Problem," *Operations Research*, Vol. 42, No. 4, pp. 677-687.

Laguna, M. and R. Marti (1999) "GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization," INFORMS Journal on Computing, Vol. 11, No. 1, pp. 44-52.

Laguna, M., R. Martí and V. Campos (1997) "Tabu Search with Path Relinking for the Linear Ordering Problem," Research Report, University of Colorado.

Laguna M., R. Marti and V. Valls (1997) "Arc Crossing Minimization in Hierarchical Digraphs with Tabu Search," *Computers and Operations Research*, Vol. 24, No. 12, pp. 1175-1186.

Lokketangen, A. K. Jornsten and S. Storoy (1994) "Tabu Search within a Pivot and Complement Framework," *International Transactions in Operations Research,* Vol. 1, No. 3, pp. 305-316.

Lokketangen, A. and F. Glover (1996) "Probabilistic Move Selection in Tabu Search for 0/1 Mixed Integer Programming Problems," *Meta-Heuristics: Theory and Applications*, I. H. Osman and J. P. Kelly (eds.), Kluwer Academic Publishers, pp. 467-488.

Lokketangen, A. and Glover, F. (1997) "Surrogate Constraint Analysis — New Heuristics and Learning Schemes for Satisfiability Problems," Proceedings of the DIMACS workshop on Satisfiability Problems: Theory and Applications, D-Z. Du, J. Gu and P. Pardalos (eds.).

Lourenco, H. R. and M. Zwijnenburg (1996) "Combining the Large-Step Optimization with Tabu Search: Application to the Job Shop Scheduling Problem," *Meta-Heuristics: Theory and Applications*, I. H. Osman and J. P. Kelly (eds.), Kluwer Academic Publishers, pp. 219-236.

Martin, O., S. W. Otto and E. W. Felten (1991) "Large-Step Markov Chains for the Traveling Salesman Problem," *Complex Systems,* Vol. 5, No. 3, pp. 299-326.

Martin, O., S. W. Otto and E. W. Felten (1992) "Large-Step Markov Chains for TSP Incorporating Local Search Heuristics," *Operations Research Letters,* Vol. 11, No. 4, pp. 219-224.

Michalewicz, Z. and C. Janikow (1991) "Genetic Algorithms for Numerical Optimization," *Statistics and Computing*, Vol. 1, pp. 75-91.

Mühlenbein, H., M. Gorges-Schleuter, and O. Krämer (1988) "Evolution Algorithms in Combinatorial Optimization," *Parallel Computing*, Vol. 7, pp. 65-88.

Mühlenbein, H. and D. Schlierkamp-Voosen (1994) "The Science of Breeding and its Application to the Breeder Genetic Algorithm," *Evolutionary Computation,* Vol. 1, pp. 335-360.

Mühlenbein, H. and H-M Voigt (1996) "Gene Pool Recombination in Genetic Algorithms," *Meta-Heurisitics: Theory and Applications*, I. H. Osman and J. P. Kelly (eds.), Kluwer Academic Publishers, pp. 53-62.

Nonobe, K. and T. Ibaraki (1998) "A Tabu Search Approach for the Constraint Satisfaction Problem as a General Problem Solver," *European Journal of Operational Research,* Vol. 106, pp. 599-623.

Nonobe, K. and T. Ibaraki (2001) "An improved tabu search method for the weighted constraint satisfaction problem, *INFOR*, Vol. 39, pp. 131–151.

Punnen, A. P. and F. Glover (1997) "Ejection Chains with Combinatorial Leverage for the Traveling Salesman Problem," Graduate School of Business, University of Colorado at Boulder.

Rana, S. and D. Whitley (1997) "Bit Representations with a Twist," Proc. 7th International Conference on Genetic Algorithms, T. Baeck ed. pp: 188-196, Morgan Kaufman.

Rego, C. and F. Glover (2009) "Ejection chain and filter-and-fan methods in combinatorial optimization," *Annals of Operations Research*, Springer Science+Business Media, LLC 2009 DOI: DOI 10.1007/s10479-009-0656-7.

Rochat, Y. and É. D. Taillard (1995) "Probabilistic diversification and intensification in local search for vehicle routing". *Journal of Heuristics* 1, pp. 147-167.

Spears, W.M. and K.A. DeJong (1991) "On the Virtues of Uniform Crossover," 4th International Conference on Genetic Algorithms, La Jolla, CA.

Taillard, É. D. (1996) "A heuristic column generation method for the heterogeneous VRP", Publication CRT-96-03, Centre de recherche sur les transports, Université de Montréal. To appear in *RAIRO-OR*.

Trafalis, T., and I. Al-Harkan (1995) "A Continuous Scatter Search Approach for Global Optimization," Extended Abstract in: *Conference in Applied Mathematical Programming and Modeling (APMOD'95),* London, UK, 1995.

Ulder, N. L. J., E. Pech, P. J. M. van Laarhoven, H. J. Bandelt and E. H. L. Aarts (1991) "Genetic Local Search Algorithm for the Traveling Salesman Problem," *Parallel Problem Solving from Nature*, R. Maenner and H. P. Schwefel (eds.), Springer-Verlag, Berlin, pp. 109-116.

De Werra, D. and A. Hertz (1989) "Tabu Search Techniques: A Tutorial and Applications to Neural Networks," *OR Spectrum,* Vol. 11, pp. 131-141.

Whitley, D., V. S. Gordon and K. Mathias (1994) "Lamarckian Evolution, the Baldwin Effect and Function Optimization," *Proceedings of the Parallel Problem Solving from Nature*, Vol. 3, New York: Springer-Verlag, pp. 6-15.

Wright, A. H. (1990) "Genetic Algorithms for Real Parameter Optimization," *Foundations of Genetic Algorithms*, G. Rawlins (ed.), Morgan Kaufmann, Los Altos, CA, pp. 205-218.

Yamada, T. and C. Reeves (1997) "Permutation Flowshop Scheduling by Genetic Local Search," 2nd IEE/IEEE Int. Conf. on Genetic Algorithms in Engineering Systems (GALESIA '97), pp. 232-238, Glasglow, UK.

Yamada, T. and R. Nakano (1996) "Scheduling by Genetic Local Search with Multi-Step Crossover," 4th International Conference on Parallel Problem Solving from Nature, 960-969.

Zenios, S. (1996) "Dynamic Financial Modeling and Optimizing the Design of Financial Products," Presented at the National INFORMS Meeting, Washington, D.C.