ORSA Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Tabu Search—Part II

Fred Glover,

To cite this article:
Fred Glover, (1990) Tabu Search—Part II. ORSA Journal on Computing 2(1):4-32. https://doi.org/10.1287/ijoc.2.1.4

ORSA Journal on Computing
Vol. 2, No. 1, Winter 1990

0899-1499/90/0201-0004 $01.25

Tabu Search—Part II

FRED GLOVER *US WEST Chair in Systems Science, Center for Applied Artificial Intelligence, Graduate School of Business, Box 419, University of Colorado, Boulder, Colorado 80309-0419, ARPANET: glover-f@cubldr.colorado.edu*

(Received: May 1989; final revision received: November 1989; accepted: November 1989)

This is the second half of a two part series devoted to the tabu search metastrategy for optimization problems. Part I introduced the fundamental ideas of tabu search as an approach for guiding other heuristics to overcome the limitations of local optimality, both in a deterministic and a probabilistic framework. Part I also reported successful applications from a wide range of settings, in which tabu search frequently made it possible to obtain higher quality solutions than previously obtained with competing strategies, generally with less computational effort. Part II, in this issue, examines refinements and more advanced aspects of tabu search. Following a brief review of notation, Part II introduces new dynamic strategies for managing tabu lists, allowing fuller exploitation of underlying evaluation functions. In turn, the elements of staged search and structured move sets are characterized, which bear on the issue of finiteness. Three ways of applying tabu search to the solution of integer programming problems are then described, providing connections also to certain nonlinear programming applications. Finally, the paper concludes with a brief survey of new applications of tabu search that have occurred since the developments reported in Part I. Together with additional comparisons with other methods on a wide body of problems, these include results of parallel processing implementations and the use of tabu search in settings ranging from telecommunications to neural networks.

As a prelude to the considerations of this paper, we briefly review some of the basic notation introduced in Part I.[11] An optimization problem will be represented in the following form:

(*P*) Minimize $c(x)$: $x \in X$

where $x \subseteq R_n$. The objective function $c(x)$ may be linear or nonlinear, and may incorporate penalty function components to drive toward satisfying certain types of constraints. The condition $x \in X$ summarizes constraining conditions which, except in special strategic variations, will be maintained at each step of the search, and in many contexts of interest will require specified components of x to receive discrete values.

A move s that leads from one trial solution (selected $x \in X$) to another may be viewed as a mapping defined on a subset $X(s)$ of X:

$$s\colon X(s) \to X.$$

Associated with $x \in X$ is the set $S(x)$ which consists of those moves $s \in S$ that can be applied to x; i.e., $S(x) = \{s \in S\colon x \in X(s)\}$ (and hence $X(s) = \{x \in X\colon s \in S(X)\}$).

To complete the notation relevant to later sections of Part II, we reiterate the description of a simple form of tabu search used as a starting point in Part I, based on creating a set T of tabu moves and an evaluator function denoted by OPTIMUM.

SIMPLE TABU SEARCH

1. Select an initial $x \in X$ and let $x^* := x$. Set the iteration counter $k = 0$ and begin with T empty.
2. If $S(x) - T$ is empty, go to Step 4.
   Otherwise, set $k := k + 1$ and select $s_k \in S(x) - T$ such that $s_k(x) = \text{OPTIMUM}(s(x)\colon s \in S(X) - T)$.
3. Let $x := s_k(X)$. If $c(x) < c(x^*)$, where x^* denotes the best solution currently found, let $x^* := x$.
4. If a chosen number of iterations has elapsed either in total or since x^* was last improved, or if $S(x) = \varnothing$ upon reaching this step directly from Step 2, stop. Otherwise, update T (as identified in Part I) and return to Step 2.

To provide a basis for understanding the extensions of ideas to be developed here, we briefly comment on the character of the foregoing process, which rests on the way the tabu set T is defined and treated.

A key concept in the management of T is to constrain the search in a manner that allows latitude in selecting "best" (highest evaluation) moves with the OPTIMUM function, while undertaking to assure the method will not re-visit a previous solution except by following a trajectory not traveled before. This is ac-

Subject classification Programming: integer, heuristic.

complished by introducing tabu restrictions (or penalties) which discourage the reversal, and in some cases repetition, of selected moves. In the simplest implementations, an attribute or set of attributes is identified which, if prevented from occurring in a future move, will assure the present move cannot be reversed. The attributes which are classified as forbidden (tabu) are recorded on a tabu list, where they reside for a specified number of iterations and then are removed, freeing them from their tabu status.

This short-term memory function of tabu search is customarily handled by treating the tabu list as a circular list, adding elements in sequence in positions 1 through t, where t is the list size, and then starting over at position 1 again. The addition of each element thus erases the element recorded in its position t iterations ago. Empirical results have indicated that a robust range of t values exists for which such a simple tabu list performs very effectively for driving the search beyond local optima and obtaining progressively improved solutions (in a nonmonotonic sequence).

Longer-term memory processes are incorporated as a means to intensify and diversify the search, as elaborated in Part I. The guiding theme in these processes is to endow the memory structures with a flexibility to choose the "most attractive" moves by evaluation functions that are determined to be most effective. Such effectiveness generally requires evaluations that are based not only on objective function change, but on the state of search. Thus, beyond its simplest forms, tabu search does not restrict evaluations to measures of "ascent" and "descent," but employs more adaptive and varied measures. The use of target analysis[13,14] has proved instrumental in determining the form these measures should take (and the conditions under which they should be modified).

In the short-term memory process, the flexibility to take advantage of such measures is enhanced by means of aspiration criteria which allow a move to be selected regardless of its tabu status. This creates a pattern of removing moves (attributes) from the tabu list on a basis not entirely related to the sequence in which they were added. The identification of appropriate aspiration criteria can have an important effect on the performance of the short-term memory process.[11,17,20]

Beyond this brief review of basic ideas, which provides a background for understanding most of the material that follows in Part II, a more complete familiarity with the concepts and strategies of tabu search, including strategic oscillation, representation of the search by a digraph, and probabilistic tabu search, will prove additionally useful as a foundation for later sections.

1. Dynamic Tabu List Processes

The effectiveness of simple rules for constructing and managing tabu lists leads to consideration of extending those rules to provide more general "dynamic" list handling processes. Such processes can have an important influence on which moves are available to be selected at a given iteration, and hence can also affect the determination of appropriate aspiration criteria. More particularly, a primary goal of such processes is by the evaluation functions to be embodied in OPTIMUM (and, by means of approaches such as target analysis, to promote the development of evaluation functions that enable this freedom to be applied more effectively).

In the following development we focus not only on general procedures but also on data structures which are essential for efficient implementation. A number of relationships are developed that are subject to a more formal theorem-proof characterization, but we undertake to provide sufficient explanation within the narration to allow the main assertions to be substantiated by the reader without difficulty. Our treatment is intended to provide useful details for those interested in practical aspects of design and execution.

1.1. Tabu List Strategies for Single Attribute Moves

We let TL denote the tabu list that implicitly defines the set T of tabu moves, i.e., TL is a vector of attributes which impart a tabu classification to moves that contain these attributes.

As a starting point, consider TL to be given in the following form

$$\mathrm{TL} = (\mathrm{e}(1), \mathrm{e}(2), \ldots, \mathrm{e}(q)).$$

We suppose the elements (attributes) are indexed by iteration, identifying the point at which they were added to the list, and q is the index of the current iteration. For convenience TL is depicted as a list that progressively enlarges as q increases, although we implicitly allow for earlier elements to be discarded by reference to a selected limiting size for TL as in customary tabu list processes (e.g., treating TL as a circular list). Correspondingly, we may identify the list of solutions $(x(1), x(2), \ldots, x(q))$ such that, for each i, $\mathrm{e}(i)$ is the attribute associated with the move applied to $x(i)$ to prevent this move from being reversed to return to $x(i)$.

We will depart slightly from usual notation in the following development by supposing that it is not the attributes $\mathrm{e}(i)$ themselves which are tabu, but rather their inverse (or complementary) attributes $\bar{e}(i)$. By this means, $\mathrm{e}(i)$ may directly refer to the move applied to $x(i)$ rather than to its reversal. For example, in a

zero–one integer programming context, if $x(i)$ is transformed into $x(i + 1)$ by a move that sets $x_j = 1$, then $e(i)$ may be taken to represent this assignment, and hence the attribute $\bar{e}(i)$ represents the tabu assignment $x_j = 0$ that transforms $x(i + 1)$ into $x(i)$. (In this case, the move attribute completely identifies the associated move. However, when a move is defined more broadly to include reference to the solution to which it is applied, additional attributes can be included such as objective function values, linear combinations of selected variables, and so forth.)

In general we will assume, as in the preceding example, that the move attributes satisfy a *sufficiency property*, which specifies that no two solutions, $x(h)$ and $x(k)$, for $h < k$, can be the same unless there exists a matching of the elements $e(h), \ldots, e(k)$ such that, for each pair $e(r)$, $e(s)$ of the matching, $\bar{e}(r) = e(s)$.

It is useful for our subsequent purposes to state this property in a different fashion, based on a process of "successive cancellation." We assume for convenience that the attributes $e(i)$ are defined so that no e can appear twice on TL unless $\bar{e}$ appears in an intermediate position. (For example, $x_j = 1$ cannot occur twice unless $x_j = 0$ intervenes.) In the sequence $e(p), \ldots, e(q)$, if any $e(r)$ is followed by an element $e(s)$ such that $e(r) = \bar{e}(s)$ then $e(r)$ is said to be *canceled* by the first such $e(s)$ (i.e., the $e(s)$ indexed by the least $s > r$) which exhibits this property. Consider all maximal subsequences, beginning with a canceled element that does not cancel any previous element, where each successive element in the subsequence cancels the element that precedes it in the subsequence. (Hence, the last element is uncanceled.) All such subsequences can be identified by a single pass through the elements $e(p), \ldots, e(q)$, checking whether each element encountered cancels an earlier element, and if so, adding it to the appropriate subsequence (and also adding the element it cancels if the earlier element is the start of the subsequence). Then the sufficiency property states that these subsequences contain all the elements $e(p), \ldots, e(q)$, and every such subsequence has an even number of elements.

Note that if each $e(i)$ consists of setting the value of a selected variable to 0 or 1 (in a zero–one IP application), or consists of adding or deleting a selected element from a set (in an optimal set membership application), then these elements satisfy the sufficiency property, and in addition satisfy a corresponding necessity property, which stipulates that $x(p) = x(q)$ will result whenever a matching (or collection of subsequences) exists which has the form used to define the sufficiency property.

The goal of this section will be to focus on two different ways of managing TL to provide a more dynamic (time and event dependent) characterization of tabu status. In each of these, an attribute $\bar{e}(i)$ is not automatically tabu as a result of the membership of $e(i)$ in TL, but only *potentially* tabu. In spite of this relaxation of customary tabu status, TL will be treated in a manner that assures no solution will be duplicated within the span of moves subjected to consideration, except where aspiration criteria or the absence of admissible alternatives may lead to selecting a move designated to be tabu.

1.2. Tabu Status Based on Cancellation Sequences

We associate with the tabu list TL an *active tabu list*, ATL, which consists only of the elements of TL that have not been canceled, and represent ATL in the same general form as TL, i.e.,

$$\text{ATL} = (e(p), \ldots, e(q)).$$

ATL is understood to be a subsequence of TL, which contains the same last element $e(q)$ (derived from the move that transforms $x(q)$ into the solution $x(q + 1)$). We assume that a buffer of the tb most recent elements $e(i)$ (i.e., for $i > q - tb$) is created which serves a purpose similar to that of a standard tabu list by automatically defining the associated elements $\bar{e}(i)$ to be tabu. (Such a buffer will appropriately be somewhat smaller than the size of a standard tabu list, however.)

To understand the way that ATL will be managed, consider a step in which an element $e(q + 1)$ is added to ATL, where $e(q + 1) \neq \bar{e}(q)$ (under the assumption $tb \geq 1$), and suppose that $e(q + 1)$ cancels an earlier element $e(i)$ of ATL as a result of $e(q + 1) = \bar{e}(i)$. (Since ATL consists only of uncanceled elements, the identification of $e(i)$ is unique.) The structure of ATL, upon adding $e(q + 1)$, but before dropping the element $e(i)$ canceled by $e(q + 1)$, may be depicted as follows:

$$\text{ATL} = (e(p), \ldots, e(h), e(i), e(j), \ldots, e(q), e(q + 1)).$$

The elements $e(h)$ and $e(j)$ are respectively the immediate predecessor and the immediate successor of $e(i)$ on the list ATL. The element $e(j)$ may be the same as $e(q)$, and $e(i)$ may be the same as $e(p)$ (in which case the predecessor $e(h)$ of $e(i)$ does not exist or is not in ATL).

If the addition of $e(q + 1)$ constitutes the first time that any element cancels a previous element (hence ATL and TL are the same to this point) then the sufficiency property implies that no two solutions generated so far can be the same. Moreover the solution $x(i)$ cannot be duplicated by the solution resulting from any sequence of future moves unless every element in the nonempty sequence $e(j), \ldots, e(q)$ is canceled. We call this sequence which lies between the canceling element $e(q + 1)$ and the canceled element $e(i)$ the *Cancellation Sequence*, or C-Sequence. Given that we

prevent the cancellation of e(q) by e(q + 1), if we can insure that at least one element in each successive C-Sequence will remain uncanceled, then no solution can ever repeat. (This is a sufficient but not a necessary condition to avoid repetition of solutions.)

This means of avoiding a duplication of an earlier solution can be assured by specifying an attribute $\bar{e}(i)$ to be tabu if and only if the associated element e(i) of ATL is the last remaining member of some C-Sequence, once all other members have been canceled. To enforce this condition successfully, a way must be afforded to remove each canceled element from every C-Sequence to which it belongs, and to identify when one or more of these sequences has been reduced to a single element.

An efficient method for accomplishing this can be based on the observation that whenever one sequence lies within another, the larger sequence is dominated by the smaller and may be discarded, since retaining an element in the smaller sequence assures that one is also retained in the larger. (This observation is also central to establishing the sufficiency of nonempty C-Sequences to avoid duplicate solutions, as long as moves exist that permit this condition to be maintained.) Specifically, we introduce a data structure consisting of two arrays, *startseq*(e) and *endseq*(e), defined for each element e on the active tabu list ATL, where *startseq*(e) denotes the element f on ATL that starts the C-Sequence terminated by e (where f = *void*, a dummy element, if e does not terminate such a sequence), and *endseq*(e) denotes the element g on ATL that ends the C-Sequence initiated by e (where g = *void* if e does not initiate such a sequence). The dominance property implies that these two values are uniquely defined for each e on ATL (i.e., of two contending values for f or g, the element which is closer to e on ATL takes precedence). Moreover, the condition in which e is the only element of a C-Sequence is identified by *startseq*(e) = *endseq*(e) = e.

The use of these arrays and the role of dominance is illustrated in Figures 1 and 2. Figure 1 shows the creation of a C-Sequence as a result of a cancellation step, together with the associated *startseq* and *endseq* assignments. Figure 2 provides an example of dominance, and of the creation of tabu status. (It may be noted in Figure 2 that the "old C-Sequence" will also dominate the new if the canceled element occurs as early in the ATL list as the starting element of the old C-Sequence, since the form of this sequence after the cancellation will still lie inside the new C-Sequence.)

To complete the basis for identifying and updating the C-Sequences, we introduce the two arrays *predecessor*(e) and *successor*(e), which respectively identify the elements that precede and succeed e on ATL. Also, we introduce two elements, first_dummy and last_dummy, which respectively begin and end the ATL

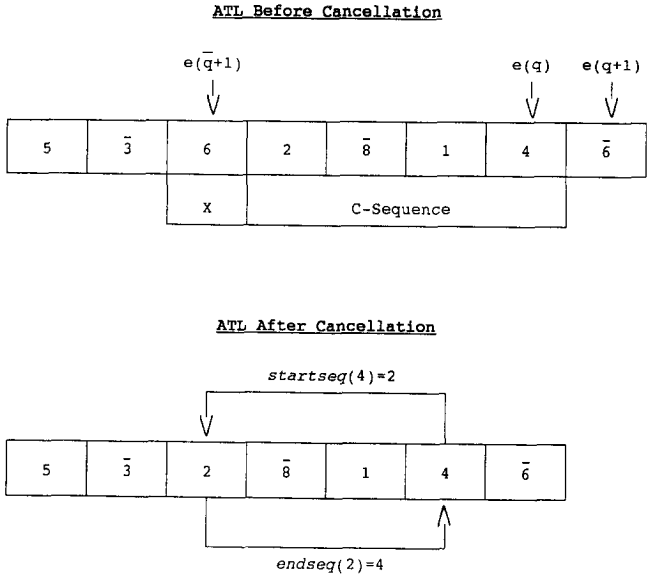

Figure 1. Creation of C-sequence by cancellation.

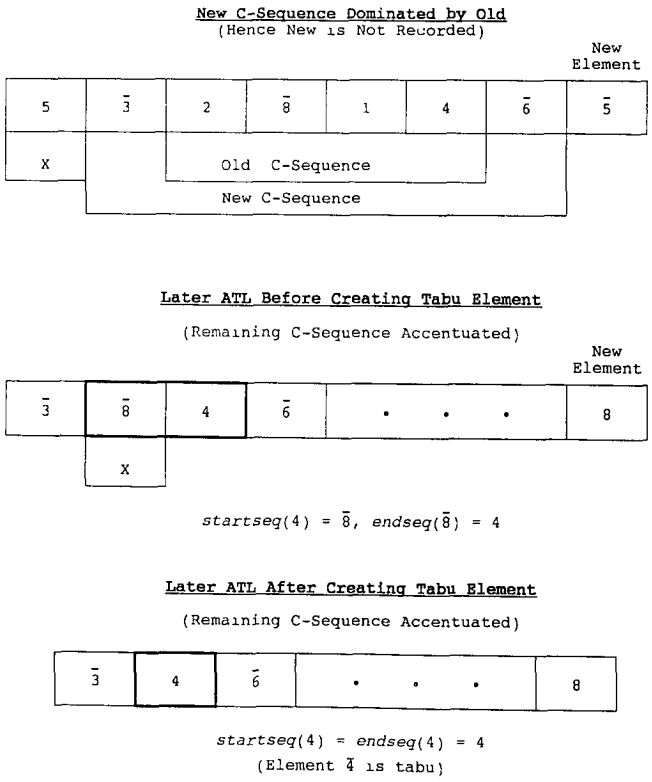

Figure 2. Domination and creation of tabu status.

list, i.e., first_dummy precedes the first (oldest) element and last_dummy succeeds the last (most recent) element. Finally, we define *iteration*(e) = the iteration e was added to ATL, letting this value equal 0 if e is not on ATL. Then for two elements e and f on ATL, *iteration*(e) < *iteration*(f) indicates that e appears before f. The *iteration* array will be used to determine

whether a newly created C-Sequence is dominated by an earlier C-Sequence. This determination is made by reference to a special variable, last_start, which identifies largest value of *iteration*(e) such that e is the starting element of a C-Sequence; i.e., such that *endseq*(e) ≠ *void*. Then if f denotes the element that starts the new C-Sequence, this C-Sequence will be dominated by an earlier one if and only if last_start ≥ *iteration*(f).

The procedure for managing the tabu list, based on the use of cancellation sequences, is indicated as follows. We refer to e(q + 1), the new element to be added to ATL, and to e(q), the previous last element of ATL, without altering notation (though these elements do not represent indexed members of a list in the procedure). The most important part of this procedure is the update of preexisting C-Sequences made possible by the dominance property when a cancellation occurs.

C-SEQUENCE METHOD

This is the main sequence of steps. The data structure includes elemental lists: *iteration*, *startseq*, and *endseq* defined over elements (e); and, ATL (Active Tabu List), which is dynamic (limited by a maximum size), defined by *predecessor* and *successor* lists.

1. Initialize:
   For all elements (e), set *iteration*(e) := 0; *startseq*(e) := *void*; *endseq*(e) := *void*.
   Set last_start := 0; *predecessor*(last_dummy) := first_dummy; *successor*(first_dummy) := last_dummy.
2. Iterate until stopping rule is satisfied (q denotes iteration number):
   Set e(q) := *predecessor*(last_dummy); v := complement of e(q + 1).
   IF *iteration*(v) ≠ 0 THEN perform **Cancel(v)**.
   Insert e(q + 1) on ATL:

   predecessor(e(q + 1)) := e(q);
   successor(e(q)) := e(q + 1);
   predecessor(*last_dummy*) := e(q + 1);
   successor(e(q + 1)) = last_dummy.

   Set *iteration*(e(q + 1)) = q + 1 and complete update of other relevant tabu conditions:
   Add e(q + 1) to the starting buffer of tabu elements, and remove the oldest element if the addition of e(q + 1) causes the buffer to contain more than *tb* elements.
   If a tabu element (excluding those on the starting buffer) has been tabu for more than t iterations, release it from its tabu status.

CANCEL(v)

This cancels the element identified by v (entering argument).

1. Set u := *predecessor*(v); w := *successor*(v).
2. IF *startseq*(v) ≠ *void* THEN update C-Sequence that ends with v (and which necessarily dominates any C-Sequence that ends with its predecessor, u):
   Set f := *startseq*(u); *endseq*(f) := *void*; *startseq*(u) := *startseq*(v).
   IF *startseq*(u) = u and *iteration*(u) ≥ least_cut-off, THEN make u tabu. (u became the only element in a C-Sequence).
3. IF *endseq*(v) ≠ *void* THEN update (v starts a C-Sequence that dominates any C-Sequence that starts with its successor, w):
   Set g := *endseq*(w); *startseq*(g) := *void*; *endseq*(w) := *endseq*(v).
   IF *endseq*(w) = w and *iteration*(w) ≥ least_cut-off, THEN make w tabu. (w became the only element in a C-Sequence).
   Set last_start := maximum{last_start, *iteration*(w)}.
4. IF last_start < *iteration*(w) THEN create new, undominated C-Sequence:
   last_start := *iteration*(w); *startseq*(w) := e(q); *endseq*(e(q)) := w.
5. Re-link u to w and drop v from ATL:
   Set *predecessor*(w) := u; *successor*(u) := w; *iteration*(v) := 0.
6. Free e from its tabu status if it is tabu, and return.

Several aspects of the foregoing procedure may be noted. First, the operations involving the updates of C-Sequences by the dominance principles are very efficient, involving no loops or searches. Second, when u or w becomes the only element of a C-Sequence, and qualifies to become tabu, it is allowed to escape its tabu status if it has resided on ATL long enough (has a small enough value of *iteration*(u) or *iteration*(w)). This cut-off value involving residence on ATL is a parameter different from the limit t, which governs the number of iterations an element is kept tabu after receiving a tabu classification. One of these two parameters may be disregarded, if desired. However, if t is retained, intuition suggests that t + *tb* should not greatly exceed the tabu list sizes found to be effective in standard implementations. (In any event, *tb* should be relatively small, perhaps only 1.)

Third, memory savings are possible due to the fact that an arbitrary element e and its complement $\bar{e}$ cannot appear on ATL simultaneously. Hence the arrays that

use these elements as arguments can be halved in size by means of an appropriate external flag (which introduces at most one half-sized array). However, if the number of such elements is still large, then *predecessor*, *successor*, *startseq*, *endseq* and *iteration* can instead name positions on a circular TL list, which contains the elements of only as many iterations as are chosen to be relevant (essentially twice the number of iterations an element is allowed to reside on TL before being considered ineligible to receive tabu status, noting that elements beyond this age may still define relevant C-Sequences since TL is not the same as ATL). In this case, *iteration*(e) can be inferred from the position on the TL list rather than maintained as a separate array.

1.3. Tabu List Management by the Reverse Elimination Method

The second dynamic process for managing tabu lists is based on generating move restrictions implied by C-Sequences of a more general form, whose boundaries can expand as well as contract. The appropriateness of expanding the boundaries of a C-Sequence derives from the fact that the addition of a new element to ATL protects against the return to solutions otherwise "guarded by" some C-Sequences, and thus should be included among these sequences. There is an obstacle to doing this, however, because the new element is not adjacent to the previous elements, and generally no way exists to rearrange elements and boundaries so that these expanded C-Sequences can be defined by the convenient types of data structures introduced for handling ordinary C-Sequences.

The approach we propose to overcome this obstacle operates by successively eliminating elements of the TL list, not by the sequence of steps that occurs in keeping ATL updated at each iteration, but by the reverse sequence, hence producing what we call the *Reverse Elimination Method.* After a new element is added to TL, becoming the new e(q) in the sequence TL = (e(1), . . . , e(q)), a trace is initiated applying rules for eliminating elements that are slightly different from those used to produce ATL. Specifically, as each e(i) is visited in turn (in the process of tracing backward from e(q) to e(1)), a *Residual Cancellation Sequence* from e(i) through e(q) is identified, along with the number $n(i)$ of elements in this sequence. A Residual C-Sequence is defined to be a subsequence that remains after removing a maximum number of pairs of mutually canceling elements, i.e., pairs of the form (e, $\bar{e}$), where each member of the sequence is allowed to belong to only one such pair.

When the sufficiency assumption holds, it is not difficult to see that the solution $x(i)$ will be the same as $x(q + 1)$, the solution produced by the move associated with e(q), only if $n(i) = 0$, and this becomes an if and only if condition provided the necessity assumption also holds. Moreover, if $n(i) = 1$ and e denotes the unique element belonging to the associated Residual C-Sequence, then under the same assumptions the new solution produced by the move associated with e($q + 1$) will duplicate $x(i)$ only if (or if and only if) e($q + 1$) = $\bar{e}$. Thus, the rule to prevent the new solution from duplicating any preceding solution is to make $\bar{e}$ (and the moves associated with it) tabu at each point of the reverse trace where such an element is found, as signaled by $n(i) = 1$.

Generally, of course, since the size of q can become large, it is appropriate to limit the size of TL as in customary tabu search applications, conducting a reverse trace of the sequence e(p), . . . , e(q) for a selected $p \geq 1$. Similarly, a buffer may be maintained which imparts temporary tabu status to the last tb elements of the list, as in the C-Sequence method. The rule that classifies elements of TL tabu by the Reverse Elimination Method does not change under these conditions.

The data structure requirements of the method are very simple, making use of the same *predecessor* and *successor* arrays used by the C-Sequence approach. However, these arrays do not identify the composition of the ATL list, but instead successively identify the elements of each Residual C-Sequence generated during the reverse trace of TL. Beyond this, the method requires only the TL array itself. The value $n(i)$ can be treated simply as a number n, since we are only interested in its current value as it changes from step to step.

To characterize the method, we indicate that an element e is absent from the current Residual C-Sequence by introducing a dummy element *absent* and setting *successor*(e) = *absent*. The first element of the current Residual C-Sequence (associated with the earliest iteration) will be denoted by the variable first_e. As before, first_dummy precedes the first element of each linked sequence and last_dummy follows the last (associated with the most recent iteration).

An important departure from the C-Sequence Method, in addition to tracing in the reverse direction, is that the elimination operation not only removes a previously canceled element but also omits the new element, that causes the cancellation, from the sequence. Before presenting the details of the procedure, we provide an illustration of how it operates in Figure 3, tracing the TL list back to the first iteration ($p = 1$). For simplicity no buffer is employed. (In contrast to the C-Sequence Method, $tb = 0$ is always possible with the Reverse Elimination Method.) The array structures used for implementation are also not shown, though the processes involved should be clear by reference to the effect of each successively examined e(i) of TL on the composition of the Residual C-Sequence.

TL List

i	1	2	3	4	5	6	7	8
e(i)	6	$\bar{4}$	2	3	$\bar{5}$	4	5	$\bar{3}$

Reverse Elimination Trace

i	Residual C-Sequence	n	Tabu Status
8	$\bar{3}$	1	3 Tabu
7	5, $\bar{3}$	2	
6	4, 5, $\bar{3}$	3	
5	4, $\bar{3}$ ($\bar{5}$ and 5 cancel)	2	
4	4 (3 and $\bar{3}$ cancel)	1	$\bar{4}$ Tabu
3	2, 4	2	
2	2 ($\bar{4}$ and 4 cancel)	1	$\bar{2}$ Tabu
1	6, 2	2	

Figure 3. Reverse elimination method.

It is interesting to observe that the tabu classifications produced by the method for this example are the same as those produced by the more restrictive C-Sequence Method when the latter employs a minimum buffer size of $tb = 1$. However, with the addition of the element e(9) = $\bar{5}$ to TL, the Reverse Elimination Method continues to generate a set of restrictions which is exactly necessary and sufficient to prevent duplications, while the C-Sequence Method produces two additional (unnecessary) tabu restrictions.

The precise details of the method that generates the sequences and tabu classifications shown in Figure 3 are as follows.

REVERSE ELIMINATION METHOD AT EACH ITERATION

comment:
The following local initialization is implemented after selecting the move whose associated attribute is e(q).
For each element e: designate e to be "not tabu" and set

successor(e) := *absent*
predecessor(last_dummy) := first_dummy
successor(first_dummy) := last_dummy
first_e := last_dummy

$n := 0$
comment:
Select the current size of TL by choosing $p < q$, and select the buffer size tb. Then trace the TL list in reverse order, as follows.
for $i := q$ **until** p **do**
begin
e := e(i)
if *successor*($\bar{e}$) = *absent* **then**
begin
comment:
Expansion Step: $\bar{e}$ is not in the current residual C-Sequence, hence e becomes the new first element of this sequence.
successor(e) := first_e
predecessor(first_e) := e
first_e := e
predecessor(e) = first_dummy
$n := n + 1$
if $n = 1$ **then** make $\bar{e}$ tabu
else begin
comment:
Elimination Step: $\bar{e}$ is in the current residual C-Sequence. Hence e is not added and $\bar{e}$ is eliminated.
f := *predecessor*($\bar{e}$)
g := *successor*($\bar{e}$)
successor(f) := g
predecessor(g) := f
successor($\bar{e}$) := *absent*
if $\bar{e}$ = first_e **then** first_e := g
$n := n - 1$
if $n = 1$ **then** make the complement of first_e tabu
comment:
if $n = 0$, the solutions $x(i)$ and $x(q + 1)$ are the same
end;
comment:
create automatic tabu status for elements of the buffer
if $i > q - tb$ **then** make $\bar{e}$ tabu
end;

Reducing Effort and Related Strategic Considerations

The foregoing procedure evidently requires more work than the C-Sequence procedure, since it traces the TL list at each iteration. However, some shortcuts are possible at the expense of more memory. For example, there is no need to continue the trace after adding an element on the Expansion Step which does not contain a complement among earlier elements of TL (a condition easily recorded).

A process that may achieve greater savings is the following. On a given iteration, record a value *least*(n) for each value of n from 0 to a selected cut-off n^*, which identifies the smallest value of i for which a Residual C-Sequence has a size of $n(i) = n$. This value may be determined automatically by starting with

least$(n) = q$ for all n, followed by setting *least*$(n) = i$, where $n = n(i)$, for each i in the reverse sequence from $i = q$ to p. Then, given that $n(i)$ cannot decrease by more than 1 on the next iteration, it is unnecessary at that point to trace earlier than a value of = Minimum(*least*(0), *least*(1), *least*(2)), and in general it is unnecessary on the next r iterations to trace earlier than a value of i = Minimum(*least*(k): $k = 0, \ldots, r + 1$). A simple auxiliary means to save effort is to identify tabu status by setting tabu(e) := q, the current iteration value, thereby avoiding the need to change or erase previous array values at the start of each new iteration.

The allowance for choosing p anew at each iteration, instead of giving it a fixed relationship to q, reflects a strategic property of the Reverse Elimination Method not shared by other tabu list approaches. Specifically, there is no "danger" in choosing p too small (i.e., in making TL too large), except for considerations of computational effort. In other tabu list procedures, if TL grows beyond a certain size, the selection of new moves becomes restricted too severely, and solution quality suffers. With the Reverse Elimination Method, however, the only restriction caused by creating tabu status (except for elements in the buffer, which may allowably be empty) is to prevent returning to an earlier solution. Assuming the existence of multiple paths to the neighbors of such solutions, this restriction is likely to be benign. (In the event that it is not, appropriately designed aspiration criteria may permit tabu status to be overridden and thereby return to an attractive region.) Consequently, to exploit the fact that a fuller examination of TL is generally advantageous, the Reverse Elimination Method can employ a strategy of periodically tracing TL to a deeper level than chosen for "customary" iterations, persisting until generating some number of solutions that are verified to provide no duplication at that level. However, it is probably unnecessary in most applications to trace back much earlier than a first local optimum, since solutions encountered before such a point are unlikely to be revisited in any event.

The Reverse Elimination Method also creates an appealing opportunity for diversifying the search process, as accomplished by the use of long-term memory functions in other versions of tabu search. In particular, rather than making a move attribute tabu only when $n = 1$, it is additionally possible to penalize, or conditionally avoid, the choice of attributes associated with low n values, e.g., for $n \leq n^*$.

This can be accomplished by creating a record *min_n*(e) which identifies the smallest value $n(i)$ value for any Residual C-Sequence that contains element e. To determine this value, start each iteration by setting *min_n*(e) := $n^* + 1$ (a value for n that receives no penalty). On the Expansion Step, when element e is added to the current Residual C-Sequence, and n is increased by 1, set *min_n*(e) := Minimum(*min_n*(e), n). On an Elimination Step, when $\bar{e}$ is eliminated and the resulting n is at most n^*, trace the successive predecessors f of $\bar{e}$, and set *min_n*(f) := Minimum(*min_n*(f), n) for each of these predecessors encountered. (These are the only members of the Residual C-Sequence that may receive a smaller assigned value than previously.) At the completion of the reverse trace, each element e thus receives a penalty (or is classified tabu) according to the size of *min_n*(e).

It is to be noted that situations may arise where no move exists that will avoid duplicating a previous solution. The number of such situations can be strategically reduced by approaches that penalize rather than categorically forbid infeasible moves, but it is still possible to become walled off with no recourse except to repeat an earlier solution. Under such circumstances (i.e., where each available move contains an attribute made tabu by the Reverse Elimination Method), a reasonable choice is to select the tabu attribute e(i) with the smallest i value. To make this choice accurately, the method should keep track for each e of the first (hence largest) i for which e is made tabu. We conjecture that such a choice has implications for finiteness in the zero–one IP and optimal set membership examples, provided infeasible moves are penalized rather than strictly removed from consideration.

An Alternate Characterization of Residual C-Sequences

Before progressing to more advanced considerations, we note that it is possible to characterize the Residual C-Sequence associated with each e(i) of TL, which we denote by RCS(i), in an alternate way that does not require the application of the Reverse Elimination Method. Specifically, for each $h \leq q$ define

$$cancel(h) = \text{Maximum}(k\text{: } k < h \text{ and } e(k) = \bar{e}(h))$$

where *cancel*$(h) = 0$ if no e(k) of the specified form exists. The index i = *cancel*(h) thus identifies the element e(i) that is canceled by e(h) in the progressive construction of ATL, at the point where e(h) is added as the new element of ATL (and e(i) accordingly is dropped). Then, associated with the sequence e(i), ..., e(q), which represents the attributes of the moves that collectively transform $x(i)$ into $x(q + 1)$, the Residual C-Sequence RCS(i) is given by

$$\text{RCS}(i) = (e(h)\text{: } i \leq h \leq q \text{ and } cancel(h) < i).$$

By reference to this characterization, the change in RCS(i) each time a new element e(q) is added to TL is easily specified. By convention, treat RCS(q) as empty before e(q) is added, and let the symbols "+" and "−" denote the operations of adding and deleting elements from sequences (effectively treated as sets). Then the

change in RCS(i) is given by

$$\text{RCS}(i) := \text{RCS}(i) + \text{e}(q) \text{ for } cancel(q) < i \leqslant q$$

and, letting $k := cancel(q)$, if $cancel(q) > 0$, set

$$\text{RCS}(i) := \text{RCS}(i) - \text{e}(k) \text{ for } cancel(k) < i \leqslant k.$$

Repeating this process, we let $h := cancel(k)$ take the role of q (provided $cancel(k) > 0$), and hence in turn let $k := cancel(h)$, thereby identifying successive intervals of i indexes in which RCS(i) either adds the element e(h) or drops the element e(k), where e(h) = e(q) and e(k) = $\bar{e}(q)$. Accordingly, the value of $n(i)$ increases or decreases by 1 over these intervals.

These relationships may be susceptible to exploitation by intermediate level procedures that combine aspects of the C-Sequence and Reverse Elimination Method. For example, a "shadow ATL" may be maintained that encompasses the elements e(h) for $h = cancel(i)$, associated with each e(i) on ATL, noting that updates involving ATL and its shadow can be handled more efficiently than those involving TL, and can generate tabu conditions that are sufficient to prevent repetition of solutions.

1.4. Adaptations for Bounded Variable Integer Programs

Integer programming problems with general upper bounds are often disregarded in treatments of combinatorial optimization, under the supposition that procedures which apply to zero–one problems apply to the general upper bounded case as well. Though theoretically accurate, this assessment disregards the fact that considerable inefficiency and excessive demands on memory may result by "direct adaptations." Consequently, we undertake to show how the foregoing procedures can be adapted in a more effective manner to treat IP problems with general upper bounds, referring specifically to the class of moves that consists of changing the value of a selected variable to an adjacent value.

Perhaps surprisingly, standard techniques to transform the problem into one with zero–one variables are not only inefficient, but fail to work in this context. For example, if a variable x_j is represented by a binary expansion as $x_{1j} + 2x_{2j} + 4x_{3j} + 8x_{4j}, \ldots$, then to change the value of x_j from 7 to 8 involves a composite move that changes x_{1j}, x_{2j} and x_{3j} from 1 to 0, and x_{4j} from 0 to 1. Other representations of x_j as a linear combination of zero–one variables require the use of a composite move that changes the value of one variable from 1 to 0 and another variable from 0 to 1. Such moves belong to the class of *paired attribute* moves which require more complex rules and memory structures, and whose treatment is indicated in Section 1.5.

Bounded Variable Specialization of the Reverse Elimination Method

We first consider how the Reverse Elimination Method can be adapted to the bounded variable setting. This adaptation is considerably simpler than the one for the C-Sequence Method, and allows a convenient means for introducing considerations basic to both procedures.

Move attributes that satisfy the sufficiency and necessity conditions discussed earlier result by the natural device of creating an attribute to represent the increase or decrease of each integer variable to a specific value. For example, by selecting an attribute e to represent increasing x_j from 7 to 8, the complement $\bar{e}$ represents decreasing x_j from 8 to 7. In principle, the Reverse Elimination Method in the form previously described can be applied directly to this representation of move attributes. However, this unfortunately creates an attribute for every value to which x_j can be increased or decreased, producing $2U_j$ attributes for every variable x_j, where U_j is the upper bound for x_j. Thus, a 500 variable IP problem where each variable is constrained to lie between 0 and 100 gives rise to about 1,000,000 elements, and hence requires a corresponding dimension for each of the *predecessor* and *successor* arrays.

There is a simple way to organize the Reverse Elimination Method by the use of a different data structure that will reduce this component of memory by a factor of more than 30. In general, instead of requiring array space of $2(\sum U_j)$, this alternative data structure requires an array space of only $3m$, where m is the total number of variables—essentially introducing a single array of m elements beyond the two arrays of m elements required by the zero–one problem.

The approach is based on the fact that it is not the sequence of elements in the Residual C-Sequences that is important, but only their identity. Thus the *predecessor* and *successor* arrays serve only as a convenient means for tracking this identity (and indeed, by the use of bit string coding, and efficient routines for identifying bits that are "on," the zero–one case can be handled with less memory as well).

The specialization of the Reverse Elimination Procedure to the zero–one IP example allows a simple interpretation for the elements of the Residual C-Sequence that are linked by the *predecessor* and *successor* arrays. In particular, at the point where the element e(i) is either added to the sequence or used to eliminate its complement, the residual C-Sequence elements identify precisely the variables x_j whose values in $x(i)$ differ from their values in the current solution $x(q)$.

By extension, to track the corresponding information for the general upper bound case, we use a *deviation* array which identifies the vector difference $x(i) - x(q)$, i.e., $deviation(j) = x_j(i) - x_j(q)$. The specialization of the Reverse Elimination Procedure to

this setting occurs by starting with all entries of the *deviation* array equal to 0. Each e(h) encountered during the reverse trace from e(q) to e(i) in TL records the information that a particular x_j was increased or decreased, and on the basis of this information the corresponding value of *deviation*(j) is changed in the opposite direction. The value of n in the Reverse Elimination Procedure accordingly represents the sum of the absolute values of the *deviation* array entries in this specialization. The *predecessor* and *successor* arrays are given a changed role, which consists of linking the indexes of nonzero entries of the *deviation* array. Hence, when $n = 1$, the only element remaining linked by these arrays is the unique index j such that *deviation*(j) = 1 or -1, indicating that x_j appropriately becomes tabu to be increased or decreased, respectively, in the current solution. The resulting specialization of the Reverse Elimination Method to the bounded variable IP setting thus becomes both straightforward and easy to execute.

The process which incorporates this specialization is illustrated in Figure 4.

Bounded Variable Specialization of the C-Sequence Method

The handling of general bounded IP problems with the more efficient (though more restrictive) C-Sequence Method requires a different type of approach. The dominance property once again leads to a process that

Specialization for Bounded Variable IP

TL List Code
j means increase x_j by 1
-j means decrease x_j by 1

TL List

i	1	2	3	4	5	6	7	8	9	10	11
e(i)	-3	-3	-1	3	-4	2	-4	3	4	4	-2

Reverse Elimination Trace

	deviation (j)					
i	j = 1	2	3	4	n	Tabu
11		1			1	Increase x_2
10		1		-1	2	
9		1		-2	3	
8		1	-1	-2	4	
7		1	-1	-1	3	
6			-1	-1	2	
5			-1		1	Decrease x_3
4			-2		2	
3	1		-2		3	
2	1		-1		2	
1	1				1	Increase x_1

Figure 4. Reverse elimination method.

is highly efficient and which achieves an economical use of memory.

As a basis for this specialization we introduce an array *ATLrecord*, which is dimensioned to the maximum size to be allotted to the ATL list, and which contains the elements of this list without specifying the order in which they appear. The order is provided by altering the function of the *predecessor* and *successor* arrays to link *ATLrecord* indexes. Specifically for h = *predecessor*(i) and j = *successor*(i), the entries *ATLrecord*(h), and *ATLrecord*(j) identify the elements that immediately precede and follow the element *ATLrecord*(i) on the ATL list. Correspondingly, the array entries *startseq*(i) and *endseq*(i) name the positions in *ATLrecord* of the elements that start and end the C-Sequences that the element *ATLrecord*(i) ends and starts, respectively. Similarly, the *iteration* array is keyed to positions of *ATLrecord*.

We will show by means of this data structure, and two additional arrays, that the C-Sequence Method can be specialized in a way that permits the associated component of memory to be reduced by a factor roughly equivalent to that achieved for the Reverse Elimination Method. To describe how this occurs, we first stipulate that the *predecessor* and *successor* arrays must initially link two different sets of indexes (each beginning and ending with its own fixed dummy indexes), one which consists of the positions on *ATLrecord* where elements of ATL currently are found, and the other which consists of "free" positions. When an element is canceled, its *ATLrecord* index goes on the linked list of free positions. Once *ATLrecord* is full, however, and no free positions remain, the current element added to ATL goes in the position of *ATLrecord* occupied by the element canceled on this step, or by the oldest (earliest) element if no element is canceled, and this position is relinked to become the new "last" position.

The elements recorded on *ATLrecord* embody three pieces of information: the variable x_j whose value was changed on a given iteration, the new value assigned to this variable, and whether the change was an increase or decrease. The first of the two additional arrays for exploiting this representation effectively has the purpose of identifying the location of each e on *ATLrecord*, and will be denoted *location*(e). Instead of defining this array over elements recorded in *ATLrecord*, however, we specify that the elements accessed by *location*(e) are "reduced elements" that indicate only the identity of specific x_j variables and whether they are increased or decreased (without reference to the value attained). This results in much less memory than would otherwise be required, but entails that *location*(e) points only to a subset of the elements on *ATLrecord*. To

compensate for this, the second additional array, denoted *pre*(i), links positions of *ATLrecord* which *location*(e) temporarily omits. Specifically, for each element *ATLrecord*(i), *pre*(i) names the position h such that *ATLrecord*(h) contains the most recent earlier appearance of the same variable x_j and the same direction of change recorded in *ATLrecord*(i). (A dummy position is named if *ATLrecord* contains no such entry.) Then *location*(e) identifies precisely the latest position (associated with the largest iteration value) that a given variable x_j was changed in a given direction.

The ability to restrict *location*(e) in this way derives from the fact that this entry names the only position in *ATLrecord* which is currently of interest (for the associated "reduced" e)—i.e., it identifies the only element that currently can be canceled by a new move that changes x_j in the direction opposite to that indicated by e. Upon cancellation, the new position to be named by *location*(e) is determined by the value of *pre*(i), for $i = location(e)$.

To complete the process, when the attributes of the new moves are recorded in *ATLrecord*(h), then *location*(e) (for e identifying the appropriately reduced subset of these same attributes) gives the proper entry for *pre*(h), and *location*(e) is reset by *location*(e) := h. Indeed, by these observations, the use of the *location* and *pre* arrays makes it possible to record attributes in reduced form (identifying only a variable x_j and a given direction of change) within *ATLrecord* itself, provided current values of the variables are stored separately, and we assume this henceforth.

It remains only to designate how tabu status can be recognized and enforced. As a result of the foregoing process, the condition *startseq*(i) = i, which identifies the element *ATLrecord*(i) to be tabu, can only be relevant for the current solution in the situation where *location*(e) = i. Thus, whenever a change results in setting *startseq*(i) = *endseq*(i) = i, the attribute e = *ATLrecord*(i) is checked to determine whether *location*(e) = i, and similarly, when *location*(e) is updated to name a new position i, the condition *startseq*(i) = i can be checked. Since *location*(e) identifies the index of the only move for x_j (in the given direction) that is relevant to be considered tabu, all tabu moves can be identified by flagging *location*(e) negative for those elements e that currently identify a tabu move in this manner.

An illustration of how the *ATLrecord* array is linked by the *location* and *pre* arrays is provided in Figure 5. For convenience, this illustration assumes the elements of *ATLrecord* occur in the same order as in ATL (hence, appearing in the order of their iteration values, for the subset of iterations encompassed by ATL). By this means, the *predecessor* and *successor* arrays refer to successive indexes of *ATLrecord* and it

Partial Data Structures for Bounded Variable IP

ATL List Code
j means increase x_j by 1
-j means decrease x_j by 1

Note: *ATLrecord* is assumed to have the same order as ATL (hence *predecessor*(i) = i-1).

ATLrecord

i	1	2	3	4	5	6	7	8	9	10	11
ATLrecord(i)	2	-1	2	3	-1	-2	3	-2	-2	2	1
pre(i)	0	0	1	0	2	0	4	6	8	3	0
iteration(i)	3	4	6	7	10	11	12	13	15	18	19

Location(e)

e	1	2	3	4	-1	-2	-3	-4
location(e)	0	10	7	11	5	9	0	0
pre(i) links		3 1 0	4 0	0	2 0	8 6 0		

Figure 5. C-Sequence method.

is unnecessary to identify them separately to determine the "true order" of *ATLrecord*. We note that this type of data structure and the rules for its management can be adapted to a variety of other settings involving large numbers of elements, where these elements belong to ordered classes in which each member (other than the first) can occur in a solution only if its "class predecessor" does also.

1.5. Dynamic Tabu Lists for Paired Attribute Moves

A wide range of procedures for optimization problems operate by means of moves that may be represented by a pair of attributes (e, f). Such *paired attribute* moves are exemplified by a variety of "add/drop" operations, such as adding and dropping elements from a set, edges from a graph, or variables from a basis. Rather than treat the attributes of such moves as ordered pairs, we explicitly consider an element used in two different ways to be two different move attributes; hence, in one move e may represent adding a given element and in another move g may represent dropping the same element (in which case, we may also write $g = \bar{e}$).

Such paired attribute moves can also include single attribute moves by the use of "null attributes" a and d, which respectively take the place of the operations of adding and dropping "ordinary" elements. Thus if e represents the attribute of adding a given edge to a graph, the pair (e, d) represents the move consisting of adding this edge without dropping another (i.e., dropping instead the null attribute d). By the convention that $d = \bar{a}$ the subsequent development can be applied consistently to this type of representation. In the case

where each element is eligible for membership only in one specific set, it is preferable to create a different pair of null elements for each set. (Otherwise, moves that cannot occur may unnecessarily be identified as tabu.)

The importance of paired attribute moves is enhanced by the fact that a variety of methods for discrete and nonlinear optimization are based on the use of basis exchange operations, whose moves fall naturally into this category. (We examine a mixed integer programming method of this type in a later section.) Another area of application for such moves occurs in "multiple choice" problems, where zero–one variables belong to sets, and are governed by the provision that exactly (or at most) one of the variables in a given set can receive the value 1. (The introduction of zero–one slack variables converts the "at most one" case to the "exactly one" case.) The relevant paired attribute moves for these problems take the form of setting $x_i = 1$ and $x_j = 0$, where x_i and x_j belong to the same set.

A variety of moves that do not seem at first to be paired attribute moves can be expressed in the context of multiple choice problems. For example, moves that allow variables to "jump" to nonadjacent values can be viewed in this setting (conceiving the range of values for a variable to be a multiple choice set). Operations that transfer a job between machines, which appear to involve several attributes (e.g., the identity of the job, the machines affected by the transfer, and the sequence position of the job on each machine), likewise can be viewed as multiple choice moves. In this case, the multiple choice set derives from the collection of positions and machines to which the job may be assigned, since the job can be placed in at most one position on at most one machine. In addition to their pervasiveness, multiple choice moves have the attractive feature of permitting an economical data structure for recording tabu status in both the C-Sequence and Reverse Elimination Methods, as will be shown.

Handling Paired Attribute Moves by the C-Sequence Method

As in the single attribute case, we will represent the tabu list TL for paired attribute moves by

$$\text{TL} = (\text{e}(1), \text{e}(2), \ldots, \text{e}(q-1), \text{e}(q)).$$

However, e(i) no longer refers to a move attribute associated with iteration i. Instead, two attributes e(i) and e($i + 1$), for i odd, are generated at each iteration k (where $k = (i + 1)/2$). The value q is then twice the value of the current iteration. By this convention the sufficiency and necessity properties for TL can be defined exactly as earlier, noting that solutions are associated only with odd valued i indexes.

Similarly, this convention allows the active tabu list, ATL, to be given the same representation as before. The *predecessor* and *successor* arrays also operate with no change. However, there is an important distinction concerning the manner in which the attributes of TL are processed. The identity of each block of attributes that corresponds to a single move is maintained throughout all operations, enabling C-Sequences to take a more general form that depends on this identity. (This fact may allow memory savings in certain settings.) Correspondingly, the array *iteration*(e) continues to refer to the iteration when element e was added to ATL, thereby automatically identifying members of a common block by the shared value of *location*(e).

The more general form of the C-Sequence Method that occurs for paired attribute moves may be characterized as follows. Assume that e($q + 1$) and e($q + 2$) constitute the block of elements newly added, and let e denote each of these elements in succession. If $\bar{e}$ belongs to ATL, and hence is canceled by e, then the resulting C-Sequence ends with the predecessor of e, as before. However, the C-Sequence does not necessarily begin with the successor of $\bar{e}$, but rather with the element f which is the other member of the block to which $\bar{e}$ belongs, unless f has previously been canceled. This follows from the fact that the solution which was transformed by the block consisting of $\bar{e}$ and f (i.e., by the move associated with this block) cannot be repeated, under the sufficiency condition, unless both attributes of the block are canceled. The C-Sequence applicable to the single attribute case, where each block consists of just one element, also obeys this rule, since the cancellation of an element eliminates "all members" of the block and thus excludes it from the C-Sequence.

The newly added block of elements is not similarly encompassed by the C-Sequence, since a C-Sequence does not permissibly include an attribute which cancels another. However, the process for creating a C-Sequence acquires a new feature, due to the ability to consider the members of the added block in any order, which allows either one of the new attributes to be designated e($q + 1$), and the remaining attribute to be designated e($q + 2$). Different orderings yield different outcomes, and it is possible to identify an ordering that guarantees the best (least restrictive) collection of C-Sequences from the alternatives available.

The rule to generate the preferable ordering is as follows. Choose e($q + 1$), the "first" member of the new attribute pair, to be an attribute which does not cancel an attribute currently recorded on ATL, or which cancels an attribute recorded earlier than the attribute canceled by the other member of the new pair. If neither member cancels an earlier element, or if both members cancel elements from the same block, then their order is immaterial. (In the latter case, regardless of order, the dominance condition for C-Sequences results in creating only one C-Sequence, which ends immediately before the first member and hence excludes them both.)

This stipulation is accompanied by the requirement that the tabu buffer must contain at least one attribute of the last preceding move, i.e., the added elements $e(q + 1)$ and $e(q + 2)$ are not permitted to cancel both members of the immediately preceding block (consisting of $e(q)$ and $e(q - 1)$).

The application of this rule for creating C-Sequences is illustrated in Figure 6.

To determine tabu status by the application of the preceding rule, null add and drop attributes are disregarded, and effectively dropped out of the C-Sequence representation. It is possible on a given step for a C-Sequence to be reduced to consist of one or two elements. Once a C-Sequence shrinks to a single element, then the complement of this element is made tabu as before, thus rendering any move that contains this complement tabu. However, whenever a C-Sequence is reduced to consist of two elements, the complements of these elements are made tabu as a single pair, rather than separately. (The tabu classification is unnecessary, of course, if the pair does not correspond to an existing move. For example, in an add/drop context it is possible that two remaining elements will both represent "drop attributes," in which case no corresponding move exists.)

The creation of tabu status can be handled for such pairs by coding them as single entities—as, for example,

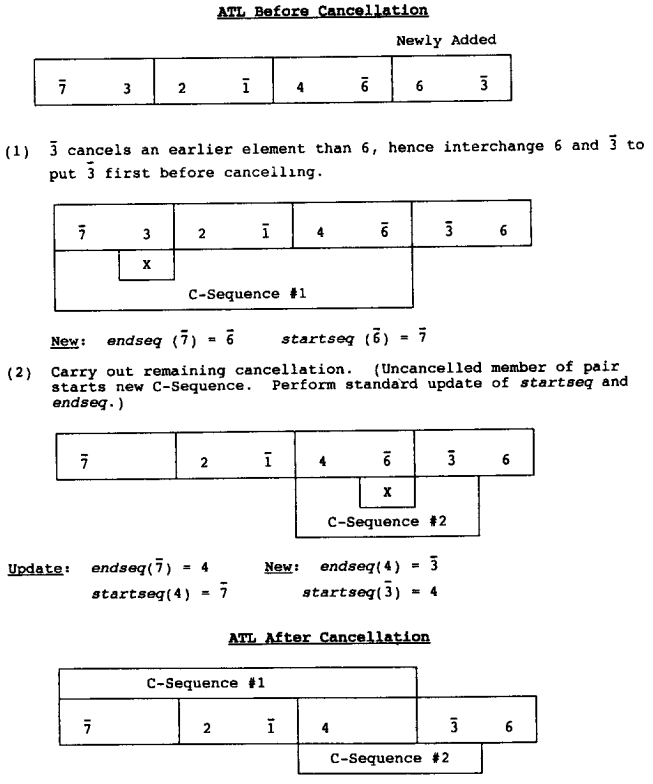

Figure 6. C-Sequence creation-paired attribute moves.

by a matrix representation $M(r, s)$ which refers to adding an element indexed r and dropping an element indexed s, where $M(r, s)$ is assigned a positive value for a tabu pair. However, it is generally possible to do better than this. The positive $M(r, s)$ matrix entries are likely to be a very small fraction of the total, and hence a linked list may be maintained for each index r (or s) that identifies its associated tabu pairs. Only a small search is required to identify tabu status under this scheme.

In the multiple choice context this approach can be made additionally effective, and can be implemented without a need for search. An add/drop pair indexed by (r, s) can eliminate reference to the "drop index" s and still recover its identity from knowledge of the associated multiple choice set. Specifically, tabu status can be established by a single list $tabu(r)$, where $tabu(r) > 0$ signifies that the move associated with the pair (r, s) is tabu, and s is the unique element such that $x_s = 1$ in the multiple choice set that contains x_r.

For problems outside the multiple choice context, an alternative approach can be employed to reduce memory, based on assigning numerical "tabu values" to different attributes recorded on ATL. A move is then treated as tabu if the sum of the tabu values of its attributes is at least 1. Specifically, when a C-Sequence shrinks to one element, the tabu value of the complement of this element is set equal to 1, assuring that any move which contains this attribute will be tabu. By extension, when a C-Sequence shrinks to two elements, the tabu value of each complement is set to one-half. This approach can render a larger number of moves tabu and hence constitutes a more restrictive means of creating tabu status than the approach that codes (r, s) pairs as single entities.

Handling Paired Attribute Moves by the Reverse Elimination Method

The Reverse Elimination Method allows a more rigorous treatment of paired attribute moves, although at the expense of greater computational effort. The TL list is represented with the same conventions indicated for the C-Sequence Method, where the paired attributes for successive moves are recorded in blocks of the form $e(i)$, $e(i + 1)$ for odd values of i.

The operations of adding and eliminating elements occur by the same rules that apply to the situation where each $e(i)$ is created by a separate move. However, during the reverse trace tabu status is only assigned at the conclusion of examining both members of a block. Null add and drop elements are not extracted from the sequences (except as they cancel each other), but are an integral part of the process. Further, tabu status applies only to pairs of attributes defining specific moves, rather than to single attributes, and does not depend on

whether both members of the pair belong to the same block.

In particular, after every second element examined in the reverse trace, the value of n is checked to see if it equals 2. (This value will always be even at this point.) If so, the two attributes that are linked by the predecessor and successor arrays identify the complementary pair of attributes to be made tabu. A pair that does not represent a move accessible to the current solution can be disregarded.) In an add/drop context, it is assured that one member of the tabu pair will represent an element previously added and one will represent an element previously dropped. The M(r, s) matrix coding and the "numerical value" approach for creating tabu status can be employed as with the C-Sequence Method. Similarly, a single *tabu*(r) array can be used to achieve an economical use of memory in the multiple choice problem setting.

To apply a diversification strategy, pursuing the goal of avoiding moves that belong to small sequences (i.e., which separate the current solution from a previous solution by a small number of steps), the rules become somewhat more complex than in the single attribute case. In the add/drop context, every possible pairing of "add" and "drop" attributes contained in the sequence identifies a potential move whose reversal should be avoided, provided the move currently exists. (These two types of attributes will occur in equal numbers, in no specific order, and will respectively belong and not belong to the current solution.) Thus a diversification strategy based on the Reverse Elimination Method will reasonably restrict attention to sequences of only a few such attributes. This limitation may be avoided, at the risk of penalizing more moves than necessary, by a diversification strategy based on assigning numerical tabu values to attributes.

A tabu buffer that automatically assigns a tabu status to the complements of the tb most recent attributes on TL also achieves a degree of "local diversification." Such a buffer is probably more important for paired attribute moves than for single attribute moves. In the paired attribute case, there are often more paths between solutions and more opportunities to follow a trajectory that remains in the vicinity of a solution recently visited. This can create a significant overlap in the solutions implicitly evaluated by examining alternative moves. For example, in a multiple choice problem, if a tabu buffer is not used to encourage diversity, a move that sets $x_r = 1$ and $x_s = 0$ may reasonably be followed by a move that sets $x_r = 0$ and $x_v = 1$. Yet the solutions reached by all moves of the latter type, for $v \neq s$, were previously accessible by the moves that set $x_s = 0$ and $x_v = 1$ on the preceding iteration. Unless there is compelling reason to reconsider these alternatives, a local diversification policy would appropriately designate all such moves tabu. (When the associated solutions should in fact be reconsidered, tabu search typically allows these solutions to be encountered by a more roundabout circuit, which may be shortened by the use of appropriately designed aspiration criteria. We note that structured move sets of the type discussed in Section 2 can allow earlier alternatives to be visited more directly than by such roundabout move sequences.)

The strategic oscillation approach characterized in Section 7 of Part I likewise serves as a means of supplementing local diversification. The number of steps that are selected in this approach for continuing beyond a boundary, before permitting the search to return (and thereby to meet and cross the boundary from the opposite direction), will generally have a direct effect on the degree of diversification. An appropriate depth for penetrating such boundaries is likely to be greater for paired attribute moves than for single attribute moves, for the reasons already suggested.

Apart from the diversification issue, the implementation of the Reverse Elimination Method for paired attribute moves is a straightforward adaptation of the form of the method for single attribute moves, and the implications of the sufficiency and necessity conditions in avoiding duplicate solutions are the same as in the simpler case.

An illustration of the Reverse Elimination Method for paired attribute moves is given in Figure 7. The Residual C-Sequences generated in this example disclose that the add and drop attributes do not necessarily

Paired Attribute Moves

TL List Code
e, $\bar{f}$ means add e, drop f

TL List

i	1 2	3 4	5 6	7 8	9 10	11 12	13 14	15 16	17 18
e(i)	9 $\bar{8}$	7 $\bar{4}$	8 $\bar{5}$	5 $\bar{9}$	3 $\bar{7}$	2 $\bar{1}$	4 $\bar{6}$	1 $\bar{3}$	6 $\bar{2}$

Reverse Elimination Trace

i, i-1	Residual C-Sequence	n	Tabu Status
18, 17	6, $\bar{2}$	2	Tabu (2, $\bar{6}$)
16, 15	1, $\bar{3}$, 6, $\bar{2}$	4	
14, 13	4, 1, $\bar{3}$, $\bar{2}$	4	
12, 11	4, $\bar{3}$	2	Tabu (3, $\bar{4}$)
10, 9	$\bar{7}$ 4	2	Tabu (7, $\bar{4}$)
8, 7	5, $\bar{9}$, $\bar{7}$, 4	4	
6, 5	8, $\bar{9}$, $\bar{7}$, 4	4	
4, 3	8, $\bar{9}$	2	Tabu (9, $\bar{8}$)
2, 1		0	Duplication

Figure 7. Reverse elimination method.

alternate as they do on the original TL list, and that tabu moves may derive their attributes from different blocks of TL. A duplicated solution is also illustrated, depicting the type of situation that may occur when all currently accessible moves are tabu, or when TL is traced to greater depths on some iterations than on others.

1.6. Extensions to Multiattribute Moves

Optimization procedures sometimes employ moves that involve more than two attributes. For example, it is common in the traveling salesmen setting to employ "four-attribute moves" that drop two edges and add two others. The ideas we have presented for handling paired attribute moves can be extended more generally to multiattribute moves, provided a few special conditions are heeded.

For both the C-Sequence and Reverse Elimination Methods, the TL list does not alter its representation, although the blocks of attributes that correspond to moves now consist of an increased number of elements. In the C-Sequence approach, when an attribute of a newly added block cancels an attribute of an earlier block, all uncanceled elements in the earlier block are included in the C-Sequence that results. The preferred ordering of the attributes of the newly added block arises by choosing attributes that do not cancel earlier attributes on ATL to appear first. Then, of the attributes remaining, those which cancel earlier attributes on ATL appear before those that cancel later attributes on ATL. Numerical tabu values may appropriately be employed by assigning a value of $1/k$ to each attribute in a C-Sequence of size k, restricting attention to C-Sequences that do not exceed the size of a block.

The Reverse Elimination Procedure examines all members of each successive block before considering the assignment of tabu status. A tabu move is identified when the linked elements are exactly equal in number to the size of a block. These elements can be fewer in number without implying that a move should be identified as tabu. (An exception may occur if a type of move that contains such a smaller number of attributes is admissible.)

Except for these changes, the multiattribute setting is handled by the same data structures and processing rules previously described.

1.7. Intensification/Diversification Tradeoffs and Frequency-Based Tabu List Management

In contrast to the approaches of the preceding sections, which organize all attributes on a single tabu list, there are occasions where the use of different tabu lists for different types of attributes is warranted. Multiple tabu lists, which can be based on the simpler structures that classify the complements of all elements on the list tabu (instead of merely potentially tabu), give an opportunity to exploit tradeoffs between intensifying and diversifying the search. Intensification strategies, which reinforce attribute combinations characteristic of a particular region (or combinations historically found good), seem inherently opposed to diversification strategies, which seek to drive the search into new regions. However, these two strategies are not always mutually exclusive, and sometimes can be handled more effectively by seeking a balance, rather than an alternating dominance, between the two.

A notable illustration of this occurs in the setting of add/drop moves, as in the edge swapping moves for transforming one tree into another or one traveling salesman tour into another (in the latter case, adding and dropping pairs of edges at a time). Experimental evidence suggests that it is better to maintain a tabu list to prevent dropped elements from being added back than it is to maintain such a list to prevent added elements from being dropped. Such findings also suggest that the best sizes for the first type of list should be somewhat smaller than for the second.[20] Intuition supports both of these outcomes. The difference in tabu list sizes derives from the fact that the number of edges that lie in a tree (or tour) is typically much smaller than the number that lie outside. Under such circumstances, a tabu condition that locks an edge into a tree is much more restrictive than one that keeps an edge out of the tree, and a list that accomplishes the former result should be smaller than one that accomplishes the latter result. At the same time, the general superiority of the list that prevents dropped elements from being added back seems reasonable from the standpoint of flexibility. Although such a list may be larger, and therefore render more elements tabu, it still generally allows a larger number of moves to be constructed from nontabu elements, giving the search more degree of freedom. (Of course, these arguments are not relevant to the more advanced dynamic tabu list strategies.)

Additional useful insight into the tradeoffs between the two types of lists can be gained by considering their effects on intensifying and diversifying the search. Two perspectives appear relevant. From the first perspective, a restriction that locks an edge into a tree has an intensification role by focusing the search in the subspace where the tabu portion of the tree remains relatively invariant (changing by one edge each iteration), while a tabu restriction that keeps a dropped edge from being added back to a tree has a diversification role by assuring that new edges will thereby be added to the tree. From the second perspective, locking an edge into a tree also has a diversification role, because it assures that edges of the tree which are not tabu will change

their identities, while preventing a dropped edge from being added back has an intensification role by limiting the search to the moves that incorporate the remaining non-tree edges.

This complementarity of the intensification and diversification strategies suggests the value of not simply using a tabu list for a single type of attribute (when simpler lists are used), but of maintaining multiple lists whose sizes reflect the relative restrictiveness of their associated tabu conditions (as influenced by the relative number of choices available in different categories; e.g., non-tree versus tree edges). To date, applications involving multiple tabu lists have focused on creating different lists for different components or stages of search, or for preventing repetitions as well as reversals. By contrast, the use of multiple lists that incorporate different types of attributes has been largely unexplored, and the issue of exploiting tradeoffs between intensification and diversification by this means remains open to investigation.

Such considerations, which invite a closer look at the standard tabu list structures, also invite examination of different types of dynamic strategies that maintain these structures. Accordingly, to conclude the treatment of dynamic tabu list strategies, we propose two alternative procedures that rely on the organization and processing techniques of standard tabu list implementations, but which introduce frequency criteria for determining the current tabu status of attributes on the list. In spirit, these procedures are related to those of probabilistic tabu search described in Section 9 of Part I, and the second procedure provides a natural accompaniment to a probabilistic tabu search approach.

To describe the first frequency-based method, consider the list $\overline{\text{ATL}}$, which consists of the complements of the elements of ATL (hence consists of those elements that are potentially tabu in the C-Sequence Method). We may either consider the full set of such complements, or more restrictively, by the use of multiple tabu lists, allow $\overline{\text{ATL}}$ to represent a subset that corresponds to attributes of a particular classification. In the latter case, more than one such $\overline{\text{ATL}}$ may be handled simultaneously, selecting appropriate parameters for each. The basic idea is to partition $\overline{\text{ATL}}$ into groups, bracketed according to the ages of their members, where tabu status for younger groups is maintained more stringently than for older groups. For example, the first group may consist of the 3 youngest elements of $\overline{\text{ATL}}$, which are required to maintain their tabu status without exception, the second group may consist of the next 5 youngest elements of $\overline{\text{ATL}}$, which are strongly but not invariably required to maintain their tabu status, and so on. (These divisions may be based on iterations rather than numbers of elements. For example, a group may consist of elements that were added between h and k iterations ago, and hence which may contain fewer than $k + 1 - h$ elements as cancellations occur.) A common approach for accomplishing such a progressive relaxation of tabu status is to apply successively smaller penalties to the elements in older groups, which often produces an effect similar to that of making all penalties uniform. By contrast, the approach adopted here allows elements of the groups to fully escape their tabu status, according to certain frequencies that increase with the age of the groups. Thus, rather than being subjected to diminishing penalties, elements are periodically allowed to be chosen freely, which requires the determination of an appropriate form of frequency measure and an associated means for implementing it.

The Tabu Cycle Method

The first of the two frequency-based procedures we propose is based on the use of iteration intervals called *tabu cycles*, which are made smaller for older groups than for younger groups. Specifically, if Group k has a tabu cycle of $\text{TC}(k)$ iterations, then at each occurrence of this many iterations, on average, the elements of Group k escape their tabu status and are free to be chosen. To illustrate, suppose there are three groups (each older than the preceding), whose tabu cycles are 4, 3 and 2. Then, roughly speaking, an element could be selected from the first group once in every 4 iterations, from the second group once in every 3 iterations and the third group once in every 2 iterations. (A buffer group whose elements never escape tabu status has an implicit tabu cycle of infinity.)

However, the process is not quite as simple as the illustration suggests. By selecting elements in the manner indicated, an element of $\overline{\text{ATL}}$ could escape its tabu status on virtually every iteration: e.g., by picking an element of Group 3 on iteration 2, an element of Group 2 on iteration 3, then an element either of Group 3 or Group 1 on iteration 4, and so on. Moreover, there is no clear provision of how to handle the situation where no element is chosen from Group k for a duration of several tabu cycles, given that the goal is to allow an element to be selected, on average, once every cycle.

To take care of the preceding considerations, we introduce a *cycle count*, $\text{CC}(k)$, for Group k. Initially, $\text{CC}(k)$ starts at 1 and is incremented by 1 at every iteration. Each group has three states, OFF, ON and FREE. We define Group k to be:

OFF	if $\text{CC}(k) < \text{TC}(k)$
ON	if $\text{CC}(k) \geq \text{TC}(k)$
FREE	if Group h is ON for all $h \geq k$.

An element is allowed to be chosen from Group k only if it is FREE, hence only if its cycle count equals or exceeds its tabu cycle value (qualifying the group as ON), and only if this same condition holds for all older groups. The ON and FREE states are equivalent for the oldest group. As implied by our earlier discussion, we assume $TC(k) < TC(k-1)$ for all $k > 1$.

The definition of the FREE state derives from the fact that each $CC(k)$ value should appropriately be interpreted as applying to the union of Group k with all groups younger than itself. Accordingly, once an element is selected from Group k, the cycle count $CC(k)$ is re-set by the operation

$$CC(h) := CC(h) - TC(h) \quad \text{for all } h \geq k,$$

whereupon the cycle counts are incremented again by 1 at each succeeding iteration. By this rule, if Group k becomes FREE as soon as it is ON, and if an element is chosen from Group k at that point, then $CC(k)$ is re-set to 0 (which gives it a value 1 the iteration after it receives the value $TC(k)$). On the other hand, if no element is selected from Group k (or any younger group) until $CC(k)$ is somewhere between $TC(k)$ and $2TC(k)$ iterations, the rule for re-setting $CC(k)$ assures that Group k will again become ON when the original (unadjusted) cycle count reaches $2TC(k)$ iterations. Thus, on the average, this allows the possibility of choosing an element from Group k once during every $TC(k)$ iteration.

This process is illustrated in Figure 8. An additional buffer group ("Group 0") may be assumed to be included, although not shown, whose elements are never allowed to escape tabu status, hence which is always OFF. For convenience, a sequence of iterations is shown starting from a point where all cycle counts have been re-set to 1 (iteration 61 in the illustration). Since a group is OFF until its cycle count reaches its tabu cycle value, each group begins in the OFF state. A group that is both ON and FREE is shown as FREE, and a FREE group from which an element is selected is indicated by an asterisk.

Figure 8 discloses how the choice of an element from a FREE group affects the cycle counts, and hence the states, of each group. Thus, for example, on iteration 64 the choice of an element from Group 2 reduces the cycle counts of both Group 2 and Group 3 by the rule

$$CC(2) := CC(2) - TC(2) = 4 - 3 = 1$$

$$CC(3) := CC(3) - TC(3) = 2 - 2 = 0.$$

Hence on iteration 65, where these counts are again incremented by 1, their values are shown as $CC(2) = 2$ and $CC(3) = 1$.

There are a few additional features of this process to emphasize. First the tabu cycles do not have to be integers. A value such as $TC(k) = 3.5$ can be selected

	Group 1 Tabu Cycle: 4		Group 2 Tabu Cycle: 3		Group 3 Tabu Cycle: 2	
Iteration	Cycle Count	Group State	Cycle Count	Group State	Cycle Count	Group State
61	1	OFF	1	OFF	1	OFF
62	2	OFF	2	OFF	2	FREE*
63	3	OFF	3	ON	1	OFF
64	4	FREE	4	FREE*	2	FREE
65	5	ON	2	OFF	1	OFF
66	6	FREE	3	FREE*	2	FREE
67	7	ON	1	OFF	1	OFF
68	8	ON	2	OFF	2	FREE
69	9	FREE*	3	FREE	3	FREE
70	6	ON	1	OFF	2	FREE*

Figure 8. Tabu cycle method illustrated. (*New element is selected from the indicated group.)

to allow the method, on average, to choose an element from Group k once every 3.5 iterations (hence twice every 7 iterations). The rules remain exactly as specified. A slight elaboration of the rules is required, however, to handle a situation that may occur if no element is selected from Group k or any younger group for a relatively large number of iterations. In this case, $CC(k)$ may attain a value which is several times that of $TC(k)$, causing Group k to remain continuously ON, and hence potentially FREE, until a sufficient number of its elements (or of younger groups) are chosen to bring its value back below $TC(k)$. This leads to the possibility that a series of iterations will occur where elements are repeatedly selected from Group k. (The frequency of selection will be limited however, by the cycle count values of older groups. Hence the greatest risk of inappropriate behavior occurs when elements are selected entirely outside of $\overline{ATL}$ for a fairly high number of iterations.) Such a "statistically exceptional" outcome can be guarded against by bounding the value of $CC(k)$, preventing $CC(k)$ from being incremented once it reaches a specified multiple of $TC(k)$—for example, upon reaching some fraction f of the number of elements assigned to Group k. Similarly, $CC(k)$ should initially be bounded by $(fy + 1)TC(k)$ until Group k has acquired y elements. (The multiple $fy + 1$ allows Group k to be ON, and hence permits elements to be selected from younger groups, during the initial period where $y = 0$.) It should be noted for the case of multiple tabu lists that the definition of "iteration" may need to be varied for different lists, if not all types of attributes are present in each move.

The Conditional Probability Method

The second frequency-based tabu list approach has an orientation similar to the first, but chooses elements from different groups by establishing a *probability* $P(k)$ that Group k will be FREE on a given iteration. The probability assigned to Group k may be viewed concep-

tually as the inverse of the tabu cycle value $TC(k)$, i.e., $P(k) = 1/TC(k)$. As with the treatment of cycle counts in the Tabu Cycle Method, the appropriate treatment of probabilities in the Conditional Probability Method is based on interpreting each $P(k)$ as applicable to the union of Group k with all groups younger than itself. Also, Group k can only be FREE by implicitly requiring all older groups likewise to be FREE.

Under the assumption $P(k) > P(k-1)$ for all $k > 1$, we generate a *conditional probability,* $CP(k)$, as a means of determining whether Group k can be designated as FREE. (We may again suppose the existence of a buffer, Group 0, which admits no choice of its elements, hence for which $P(0) = 0$.) The rule for generating $CP(k)$ is as follows:

$$CP(1) = P(1)$$

$$CP(k) = \frac{P(k) - P(k-1)}{1 - P(k-1)} \quad \text{for } k > 1.$$

Then at each iteration, the process for determining the state of each group starts at $k = 1$, and proceeds to larger values in succession, designating Group k to be FREE with probability $CP(k)$. If Group k is designated FREE, then all groups with larger k values are also designated FREE and the process stops. Otherwise, the next larger k is examined until all groups have been considered.

The derivation of $CP(k)$ is based on the following argument. By interpretation, $P(k)$ represents the probability that Group k or some younger group is the first FREE group, while $CP(k)$ designates the probability that Group k is the first FREE group but no younger group is FREE. The assignment $CP(1) = P(1)$ is appropriate since it is not possible for any group younger than Group 1 to be FREE. In general, for larger values of k, the event that Group k or a younger group is FREE derives from the two exclusive events (assumed independent) where either (1) Group $k-1$ or some younger group is the first FREE group, which occurs with probability $P(k-1)$, or (2) this is not the case and Group k is the first FREE group. This gives rise to the formula

$$P(k-1) + (1 - P(k-1))CP(k) = P(k),$$

and solving for $CP(k)$ gives the value specified.

Consideration of the rationale underlying the Tabu Cycle Method described earlier, however, shows that an appropriate characterization of the $CP(k)$ probabilities is not yet complete. Designating Group k to be FREE does not imply an element will be selected from this group. For some number of iterations, elements outside of $\overline{ATL}$ may be accepted regardless of the FREE state of groups of elements within $\overline{ATL}$. When this occurs, the expected number of elements per iteration chosen from groups no older than any given Group k will generally fall below $P(k)$, without any compensating increase in freedom to choose elements from these groups (which would potentially allow the average per iteration to come closer to $P(k)$). Moreover, the same result will occur if for some number of iterations no element is selected from a group as young as Group k.

To handle this, the original $P(k)$ values may be replaced by "substitute probabilities" $P^*(k)$ in the determination of $CP(k)$. These substitute probabilities make use of the same cycle count values $CC(k)$ used in the Tabu Cycle Method, invoking the relationship $TC(k) = 1/P(k)$. To begin, $P^*(k) = P(k)$ until $CC(k)$ exceeds $TC(k)$ (bounding $CC(k)$ in early iterations as specified previously). Then $P^*(k)$ is allowed to exceed $P(k)$ as an increasing function of the quantity $CC(k)/TC(k)$. In contrast to the Tabu Cycle Method, a negative value can result for $CC(k)$ in this approach, as a result of the update $CC(k) := CC(k) - TC(k)$ (which occurs whenever an element is selected from a Group h, for $h \leq k$).

A variation with an interesting interpretation re-sets the cycle count $CC(k)$ to 0 instead of decrementing it by $TC(k)$. Then $CC(k)$ counts consecutive iterations where no element is chosen from any Group h, $h \leq k$, an event which may be (loosely) construed as occurring with probability $(1 - P(k))^r$, for $r = CC(k)$. Then the substitute probability $P^*(k)$ on the following iteration may be established by setting

$$P^*(k) = 1 - (1 - P(k))^{r+1}.$$

(Note this gives $P^*(k) = P(k)$ when $r = 0$.)

Finally, to provide a valid basis for computing $CP(k)$, we require $P^*(k) \geq P^*(k-1)$ for $k > 1$. Hence, beginning with the largest k and working backward, we set

$$P^*(k-1) := \text{Minimum}(P^*(k-1), P^*(k)).$$

The substitute probability approach has the advantage of increasing the probability of choosing elements of neglected groups as long as they remain neglected. The variation that re-sets $CC(k) = 0$ on the update step also automatically avoids the creation of inappropriately high probabilities for the type of situation handled in the Tabu Cycle Method by capping $CC(k)$.

By assigning numerical values to tabu attributes, with an associated limit on the sum (or other function) of these values to determine the tabu status of a move, both the Tabu Cycle Method and the Conditional Probability Method can be applied to multiple attribute moves as readily as to single attribute moves.

2. Structured Move Sets and Staged Tabu Search

The dynamic tabu list strategies described in the preceding sections offer a variety of opportunities for tailoring methods to particular problem settings by

manipulating their parameters for escaping (or imposing) tabu status. Beyond this, however, it is possible to structure the moves treated by these processes in an adaptive manner. This offers a means of directly visiting solution states which are not contiguous by the original move definitions.

It is worth noting that procedures for solving shortest path problems already induce a structure, or an implicit progression, for examining solutions associated with nodes of a digraph, such as the digraph G in Section 5 of Part I, which is a function of the search itself. This induced structure supersedes the structure given by initially defining $S(x)$ as the set of moves to nodes y such that (x, y) is an arc of the digraph. Building on this observation, we now indicate a class of methods for creating a similar type of induced structure for moves employed by tabu search, based on generalizing a class of methods described in [15] for finding shortest paths from an origin node to all other nodes of a digraph. In its origional form the approach uses two sets, NOW and NEXT, to which we add a third set, denoted TABU.

TABU SEARCH WITH S(x) IMPLICITLY RESTRUCTURED

1. *Initialization.* Start with sets NEXT and TABU empty, and $k = 0$. Let the set NOW consist of a selected starting solution $x^* \in X$.
2. *Choose a "best" non-tabu element from NOW.* If NOW − TABU is empty, go to Step 4. Otherwise, set k: $= k + 1$ and choose $x^k =$ OPTIMUM(x: $x \in$ NOW − TABU), and remove x^k from NOW. If $c(x^k) < c(x^*)$, let x^*: $= x^k$.
3. *Apply a Screening Test for Partitioning.* For each $s(x^k)$, $s \in S(x^k)$, such that $s(x^k)$ is not dominated by an element of NOW or NEXT: if $s(x^k)$ is dominated by an element of TABU, add $s(x^k)$ to TABU; otherwise add $s(x^k)$ either to NOW or to NEXT, according to the outcome of applying a screening test to (s, x^k), and delete from NOW and NEXT any elements determined to be dominated by $s(x^k)$.
4. *Test for Termination.* If NOW and NEXT are both empty, or k exceeds an iteration cut off level, stop. Otherwise, if NOW $\neq \varnothing$, update TABU and Return to Step 2, while if NOW $= \varnothing$ proceed to Step 5.
5. *Apply a Screening Test for Repartitioning.* Choose a nonempty subset of NEXT by applying a repartitioning screening test and transfer elements of this subset from NEXT to NOW. Then return to Step 2.

Several observations apply to this procedure. The screening tests for partitioning and repartitioning are decision points of the method designed to maintain the most attractive solutions in the set NOW. The present approach maintains access to moves from different sets $S(x)$ as a means of redefining the set of moves considered currently available (those in NOW). A candidate list procedure could be superimposed on this approach to narrow the alternatives, but its function is to reduce the effort to identify good current choices as contrasted with restructuring the set currently available.

In the shortest path setting,[15] the tests to determine membership in NOW constitute evaluation thresholds which, if supplemented by those of a candidate list procedure, would represent a higher level of screening to extract the elite candidate list members for inclusion in NOW (in some instances, excluding all candidate list members, and relegating them instead to NEXT). The indicated handling of dominated solutions of course depends on procedures capable of identifying such solutions. For shortest path problems, the identification is extremely easy: a solution associated with a given node automatically is interpreted to dominate all subsequent solutions associated with the same node that have a c(x) value at least as large.

We further allow the possibility of attaching evaluation labels to elements that are added to NEXT at Step 3 as a means of biasing them against being transferred from NEXT to NOW at Step 5. This means of transmitting information from the decision rule at Step 3 to the decision rule at Step 5 encompasses the device of creating a special FLUNK set of the form specified in [15].

The sets NOW, NEXT and TABU are depicted for notational simplicity as containing only solutions $x \in X$, though in fact in most cases they will be more complex. Allowing the set TABU to correspond to the set T, interpreted as in the latter part of Section 4 of Part I, TABU may be construed to consist of pairs (s, x), recorded implicitly by reference to solution attributes from which specific solutions can be reconstructed. In general, it should be kept in mind that the foregoing procedure uses the set notation for NOW, NEXT and TABU as a way of simplifying the reference to other underlying sets, of variable specification, that may be used to generate and process solutions from a record of designated attributes.

The use of simplified references to sets occurs naturally (in a less general form) in the solution of shortest path problems. The sets NOW and NEXT in this case are sets of nodes, not solutions, where a solution consists of a node, a distance label, and a path whose identity requires processing to determine. However, the data structure simplification that results by storing nodes on NOW and NEXT nevertheless makes

it possible to access just those solutions that are not explicitly dominated by others thus far discovered. (More precisely, the current distance and predecessor labels attached to the node provide access to the essential components of the solution needed for processing. Also, strictly speaking, a solution can be dominated implicitly, and the record for recovering its associated path may be inaccurate, unless distance labels are kept sharp.)

The use of the set TABU provides one of the major departures of the procedure from the shortest path setting. As always the attributes and restrictions used to define tabu status can strongly affect the way the method functions. Recall that T (and hence TABU) may involve multiple sets of different lengths, or tenures. In contexts where a t value is allowed to grow with k, the effect can be to impart a permanent tenure to moves whose attributes are recorded on a tabu set; i.e., an element once added to such a set will not be removed. Elements permanently excluded from consideration in this manner are assumed to be extracted from NOW and NEXT in checking whether these sets are empty at Step 4.

Finally, in the simple case where NOW, NEXT and TABU are processed strictly as collections of solutions, and where the update of TABU at Step 4 consists of adding x^k (and preferably solutions dominated by solutions $s(x^k)$) to TABU with a permanent tenure, then the results of [15] provide a convergence guarantee for this setting. In particular, under the stated assumptions an optimal solution will be found in a finite number of steps, provided the associated graph is finite and contains a path from the starting solution to an optimal solution.

Staged Search Processes

In some types of search environments the sets $S(x)$ of moves associated with solutions x in X are structured in a way that naturally partitions the search into levels, or stages. When this occurs, it can be appropriate to apply tabu search in corresponding stages, with separate or nested tabu lists applicable to separate stages. In the case where the stages establish a strict hierarchy, it can further be appropriate to purge the list for a given stage when a predecessor stage is visited.

A simple example involving a strict hierarchy of stages occurs where the digraph G characterized in Section 5 of Part I corresponds to a tree, disregarding the simple cycles produced by pairs of arcs (i, j) and (j, i). We illustrate an instance of this employing a structure closely analogous to that produced by branch and bound, showing the relevance of creating staged tabu lists in parallel with stages induced by $S(x)$. Let X be the set of vectors x whose components $x_j, j < n$, take the values #, 0, 1, ..., U_j, where x_j = # has the interpretation that x_j is not assigned a value. (It is natural to map the # values of a vector into provisional "real" values, as by solving a problem relaxation.)

Then a method for defining $S(x)$ that results in a staged representation may be expressed as follows. Stage 0 occurs when all x_j = #, and is the starting point for generating the solution sequence. Stage 1 occurs when $x_1 \neq$ # but all other x_j = #, and in general, Stage p occurs when $x_j \neq$ # if $j \leq p$ and x_j = # if $j > p$. All moves can be represented as ordered pairs of the form $(x_p', \#)$ and $(\#, x_p')$, which have the interpretation that x_p changes its value, respectively, from # to x_p' and from x_p' to #. For any x' that satisfies the definition of a Stage p vector, $S(x')$ is the set of vectors x'' derived from x' either by a $(x_p', \#)$ move or by a $(\#, x_p'' + 1)$ move, where $x_p'' + 1$ is any value in the acceptable range for $x_p + 1$ other than #. (The $(x_p', \#)$ move is disallowed if $p = 0$ and the $(\#, x_p'' + 1)$ move is disallowed if $p = n$.)

A tabu list for Stage p, adopting the policy of preventing move reversals, will therefore operate in the following manner. Upon making a Stage $p - 1$ move $(\#, x_p'')$, the reverse move $(x_p'', \#)$ is entered on the tabu list for Stage p. Similarly, on making a Stage p move $(x_p', \#)$, the reverse move $(\#, x_p')$ will go on the tabu list for Stage $p - 1$. (In this latter case, having transitioned from Stage p to a lower stage, the tabu list for Stage p is purged.) By this scheme, choosing a tabu list size for Stage p greater than or equal to the number of Stage p moves, it is easy to show that tabu search will do a depth first search of the graph of solutions created by the structure of $S(x)$.

The small amount of memory and straightforward processing required by this staged tabu search scheme underscores an important point. In the absence of creating separate lists that match the stages induced by the structure of $S(x)$, a simple form of tabu search will still do a depth first search of the tree, but in this case moves and attributes generally must be characterized in a manner that involves a greater amount of memory. For example, one such scheme that assures an exhaustive search of the tree results by allowing the length of the tabu list to grow with the number of iterations and by defining tabu restrictions in a manner that prohibits exactly those moves that lead to a solution previously visited. While this is easily done in the present context, the increased overhead for memory is considerable, and hence a procedure that matches lists to stages is highly desirable.

3. Tabu Search and Mixed Integer Programming

Three principal methods for mixed integer programming (MIP) problems based on tabu search will be described in this section. The first two fall into the

framework of Sections 2–4 of Part I, while the third involves a modification of this framework which entails changing the definition of a move.

We define the MIP problem by writing x in the form $x = (x_I, x_c)$, where x_I and x_c denote the vectors of integer and continuous variables, and by identifying the set X for (P) by

$$X = \{x: A_I x_I + A_c x_c = b,\ x \geq 0 \text{ and } x_I \text{ integer}\}$$

The objective function $c(x)$ is assumed to be linear, although ways to get around this assumption will be evident from the context.

For convenience in describing the following MIP methods, we defer consideration of aspiration levels and of intermediate- and long-term memory functions until the end, where prescriptions relevant to all procedures will be indicated.

MIP Method 1

The first method is based on specifying the form of a move s to be given by

$$s(x_I', x_c') = (x_I'', x_c''),$$

where $x_I'' = s_I(x_I')$ takes the form

$$x_I'' = x_I' \pm e_j$$

and x_c'' is an optimal solution to the linear program

(LP) Minimize $c(x_I'', x_c)$: $A_c x_c = b - A_I x_I'', \quad x_c \geq 0.$

The set $S(x')$ applicable to the trial solution x' consists of those moves such that $x_I'' = x_I' \pm e_j$ does not violate a lower or upper bound for the variable x_j.

By these conventions, a tabu search method for the MIP problem can make use either of the types of tabu lists described in Part I or of the dynamic tabu lists of Section 1, applied in this case to single attribute moves. Such lists can thus be maintained by recording the index j of the integer variable whose value was changed on given iteration and whether the change was an increment or a decrement. (A vector whose components are associated with the components of x_I can be used to facilitate the checking of tabu status.)

Given a tabu list that implicitly defines the set T of tabu moves, the key to characterizing this procedure is to identify an appropriate form of the OPTIMUM function for evaluating the move to be selected on the current iteration. Suppose x' denotes the current solution, and hence the next solution x'' is given by

$$x'' = \text{OPTIMUM}(s(x'): s \in S(x') - T).$$

Then the "natural" choice for OPTIMUM, which selects x'' as the vector that minimizes $c(s(x'))$ for $s \in S(x') - T$, can be computationally expensive to implement in the MIP setting, since it requires solving (LP) to find x_c'' for each $x_I'' = x_I' \pm e_j$. Consequently, it is appropriate to use an easily computed approximation to an optimum LP solution to provide a proxy for the minimum $c(s(x'))$ value. This can derive, for example, from a "partial" dual postoptimizing pivot applied to the linear program solved on the preceding iteration of the method, or from other more complex postoptimizing penalties as standardly computed in branch and bound.[6,16,28,32,33,37]

Let $v(x_I)$ denote the function that yields such an approximation, noting that (LP) depends only on the vector x_I'', and that x_c'' is computed without reference to x_c'. Then x'' may be determined from OPTIMUM in two steps. The first examines the integer vectors $x_I'' = x_I' + e_j$ and $x_I'' = x_I' - e_j$ for appropriate choices of j, excluding alternatives ruled out by the tabu list T, and chooses the particular x_I'' that minimizes $v(x_I)$. This step determines the move s to be chosen. The second step completes the move by solving (LP) to determine x_c'' and $c(x'')$.

The choices made by such an approach can lead to surprises as a result of selecting a move that is not attractive when its consequences are identified fully (upon at last solving the linear program that discloses the true value of $c(x'')$). An improved form of OPTIMUM would therefore use $v(x_I)$ as a screening device to provide candidate moves that are then examined in greater detail before one is selected, e.g., picking the first candidate that passes a second screening test.

The final element of this approach is to endow the objective function $c(x)$ with the ability to take on values that reflect varying degrees of infeasibility, since in many MIP problems an x_I vector that yields a feasible starting solution may not be known and, in addition, subsequent moves by tabu search may lead from the feasible region into the infeasible region. By the usual convention, an infeasible solution is interpreted to yield a value of infinity for $c(x)$. Thus, if this convention is maintained, the problem should be modified to encompass a larger feasible region with penalties for lying outside the true region of interest. This can be accomplished by a goal programming formulation, or more generally by introducing bounded variables to allow increasing constraint violations at increasing cost. Specialized LP methods exist for solving such problems efficiently.

This application of tabu search to the MIP problem can take a variety of specific forms, depending on the way the original problem is altered to penalize infeasibilities and on the choice of the OPTIMUM function (via the function $v(x_I)$). An appealing feature of this approach is that once these choices are made, it is

relatively straightforward to implement. It is also able to incorporate other types of moves, such as the paired attribute moves that arise in multiple choice problems, where incrementing a given variable to the value 1 entails that all other variables in its multiple choice set receive the value 0.

MIP Method 2

The second tabu search method for the MIP problem will first be described in the context where all integer variables are zero–one variables. For this approach we assume that X does not include the stipulation that x_1 is integer, but that the constraints defining X include the bound $x_j \leq 1$ for integer variables. Correspondingly, we assume $c(x)$ penalizes points at which such variables are not 0 or 1, as by a weighted component that is 0 at $x_j = 0$ and $x_j = 1$, and that attains a unique local, hence global, maximum at $x_j = \frac{1}{2}$. (For example, the product $x_j(1 - x_j)$, raised to any positive power, provides such a component. A variety of alternative functions that can be used in this context are proposed in [5].) The function $c(x)$ therefore is transformed to be nonlinear for this case, but as will be seen this poses no difficulty. Indeed, different forms of penalty functions can be used at different stages of the search.

The approach uses the fact that an optimal zero–one MIP solution can be found at one of the extreme points of X, and hence the search for such a solution can be undertaken by a method that pivots from one extreme point to the next, like the simplex algorithm. (Such pivots include the operation of moving a nonbasic zero–one variable x_j from one of its bounds to another, or equivalently of replacing x_j by $1 - x_j$. Consequently, one of the better known heuristics in this setting has been called the "pivot and complement" method.[2])

The moves of S, therefore, need only to be defined on the extreme points of X, and for each of these, $s(x)$ is an adjacent extreme point of x for each s in $S(x)$. The implementation of tabu search in this setting compares favorably with that of MIP Method 1, and is simpler in some respects, because the moves do not require solving an associated linear program and OPTIMUM does not require reference to an approximating function $v(x_I)$. Thus, in particular, for any extreme point x' in X, we may stipulate that the choice

$$x'' = \text{OPTIMUM}(s(x'): s \in S(x') - T)$$

selects x'' to be an extreme point adjacent to x' such that

$$c(x'') = \text{Min}(c(s(x')): s \in S(x') - T).$$

Allowing $c(x)$ and $S(x)$ to vary according to availability of certain types of improving moves, such an evaluation includes that of the pivot and complement heuristic. However, it is important in the present context not to exclude a class of moves from $S(x)$ simply because they are nonimproving, but to let the tabu search framework decide an appropriate move from the larger set.

To complete the specification of the present MIP method, we need only to identify the form of T. By reference to the goal of preventing a move reversal, or more broadly, of preventing a return to an extreme point from which a move was initiated, T may be defined in the simpler types of tabu list approaches to consist of moves that would make $s(x')$ an extreme point reached on one of the preceding t iterations. An attribute of previous moves that is easily identified and recorded is the pair of nonbasic and basic variables whose exchange led from one previous extreme point to the next—i.e., this attribute is a "composite" of the two attributes that arise in a paired attribute representation of the exchange move. Thus, employing such a composite attribute, T could be maintained in the form of a related list

$$T' = \{(p, q): \text{Nonbasic variable } x_p \text{ exchanged with basic variable } x_q \text{ on iteration } h > k - t\}.$$

Then a move would be classed tabu if it involved an exchange, in reverse, of nonbasic x_q for basic x_p for some $(p, q) \in T'$.

It is interesting to note that such a characterization of T, while seeming to be similar to a paired attribute representation, is in fact quite different, and exemplifies a case where preventing a move reversal may not prevent returning to a preceding solution. For example, a sequence of pivots yielding $T' = \{(1, 2), (2, 3), (3, 1)\}$ identifies a cycle that the use of T' is unable to prevent. (If T' were organized to prevent the execution of original moves as well as move reversals, as by specifying T to be the set described in Section 2 of Part I that consists of the union of two sets T_1 and T_2, then traversing such a cycle would not be followed by a repetition of the same moves.) Among the simpler types of tabu list approaches, one that is appropriately effective, and easier to process, is to designate the set T' by

$$T' = \{q: \text{basic variable } x_q \text{ became nonbasic on iteration } h > k - t\}.$$

The tabu classification applies in this case to those moves that allow x_q to enter the basis for some $q \in T'$. Such a list T' can be maintained for the bounded variable simplex method by interpreting a variable which is nonbasic at its upper bound to be basic, with its complementary variable nonbasic.

A possible alternative for T′, is given by

$T'' = \{p$: nonbasic variable x_p became basic on iteration $h > k - t\}$.

However, T″ generally is more restrictive (and probably less desirable) than T′ since the number of nonbasic variables typically is larger than the number of basic variables and, moreover, the use of T″ to prevent a basic variable from becoming nonbasic could prevent more than one nonbasic variable from becoming basic. An appropriate value of t, in any event, would be different for T″ than for T′. The relative performance of these simpler tabu list approaches by comparison to the C-Sequence and Reverse Elimination Methods in this MIP setting provides an interesting area for investigation.

MIP Method 2 for zero–one problems can be extended to the general MIP problem by reference to *pseudo-extreme points* which are reached by *truncated pivots*. We define a pseudo-extreme point, recursively, to be an extreme point, or to be any point that can be reached from another pseudo-extreme point by incrementing or decrementing the value of a nonbasic variable, maintaining feasibility, to a new value at which some integer variable receives an integer value. Thus, a pseudo-extreme point can arise by assigning nonbasic variables values other than upper and lower bounds. It is not difficult to prove that an optimal MIP solution lies at one of these pseudo-extreme points.

A truncated pivot occurs in this setting when a change in a nonbasic variable stops short of making the variable equal to one of its bounds, and before causing any basic variable to equal one of its bounds, but drives some basic integer variable x_j to an integer value u. The basis exchange is executed in the standard manner for such a pivot, and the basic variable x_j becomes nonbasic at the value u.

From these observations, the zero–one approach may be extended to a procedure that transitions among adjacent pseudo-extreme points (or jumps beyond to nonadjacent pseudo-extreme points), while creating tabu status in a manner analogous to that previously indicated.

MIP Method 3

The final variant of tabu search for MIP problems may be characterized by reference to moves that consist of imposing (and relaxing) constraints, rather than moves defined as mappings. The constraints underlying such moves arise from disjunctions of the form "$x_j \leqslant u$ or $x_j \geqslant u + 1$," as u and $u + 1$ range over admissible integer values for an integer variable x_j. Such disjunctions are the building blocks for many branch and bound methods, and underly the derivation of a variety of cutting planes for the MIP problem.[1,3,9,16,18]

Tabu search acquires features in this setting that are similar in concept, but different in detail, from those based on defining moves as mappings. In particular, we define moves that derive from disjunctions to consist of three types: restriction moves, relaxation moves and complement moves.

A *restriction move* takes the form exemplified by the Dakin branching scheme for branch and bound which imposes one of the two constraints $x_j \leqslant [x_j']$ or $x_j \geqslant [x_j'] + 1$, relative to a given trial solution x', where x_j is an integer variable, x_j' is not an integer, and $[x_j']$ denotes the greatest integer $\leqslant x_j'$. The motivation for such moves, as in branch and bound and in cutting methods, is to take X to be the feasible continuous region, disregarding the stipulation that x_1 is integer, and to impose constraints progressively until obtaining a subset of X whose optimal continuous extreme point solution, minimizing $c(x)$, yields integer values for the components of x_1.

Specifically, let the constraints associated with a given stage of such a process be summarized by reference to regions R_i, $i = 1, \ldots, r$, where each R_i has the form $\{x: x_j \leqslant u\}$ or $\{x: x_j \geqslant u + 1\}$ for a given integer variable x_j and integer u. Then the current feasible region X', created by imposing the associated constraints, is given by

$$X' = X \cap R \quad \text{where } R = \cap R_i\text{: } i = 1, \ldots, r,$$

(By convention, $X' = X$ when $r = 0$.) The trial solution x', associated with X' is an optimal solution to the linear program

(LP′) Minimize $c(x)$: $x \in X'$

Thus, assuming (LP′) has a feasible (and bounded optimal) solution x' with some component of x_1' non-integer, a restriction move consists of selecting a constraint whose corresponding region R_{r+1} creates a new feasible region X'' given by

$$X'' = X' \cap R_{r+1}.$$

Then upon increasing r by 1, X'' becomes the current X'.

A *relaxation move* is the reverse of a restriction move and consists of discarding a previously imposed constraint.

A *complementation move* consists of replacing one of the two constraints $x_j \leqslant u$ and $x_j \geqslant u + 1$ by the other, where the constraint that is replaced corresponds to one of the regions R_i.

Based on the foregoing definitions, we present an outline of the third MIP procedure, temporarily omitting reference to how the choices of the method are

made, and to the way in which particular moves acquire and lose tabu status.

OUTLINE FOR MIP METHOD 3

1. *Initialization.* Start with $X' = X$ and $r = 0$.
2. *Solve (LP').* If an optimal solution x' for (LP') yields x_1' integer, and if $c(x') < c(x^*)$ for the current best feasible MIP solution x^*, then record x' as x^* and go to Step 5. Otherwise, if this step has been executed more than a specified number of times since obtaining an improved solution x^* (or more than a specified number of times overall), stop.
3. If x_1' is integer, go to Step 5 and otherwise go to Step 4.
4. *Make a restriction move.* Select an integer variable x_j such that x_j' is not integer, and make a non-tabu restriction move involving x_j. If no such move exists for all noninteger x_j' in x_1' go to Step 5. Otherwise, update R and X' relative to the selected move and return to Step 2.
5. *Make a complementation or relaxation move.* Select some R_i, $i = 1, \ldots, r$, and make a non-tabu complementation or relaxation move. If $r = 0$ or if no such move exists for all R_i, stop. Otherwise update x' relative to the selected move and return to Step 2.

To give this outline substance, it is necessary to identify how choices are made at Steps 4 and 5, and how tabu moves are defined. These elements have the same role as specifying the form of OPTIMUM and the composition of T in the framework of preceding sections.

The choice of restriction moves at Step 4, and of relaxation and complementation moves at Step 5, can be based on the postoptimality information available from solving (LP') at Step 2, employing considerations analogous to those described in connection with the use of the evaluation function $v(x_1)$ for MIP Method 1. Generally speaking, the types of criteria used to select branches in branch and bound[9,23,32,33,37] can also be applied to choosing moves in the present setting.

However, there are differences in the analyses relevant to branch and bound, and those relevant to tabu search, that stem from differences in the underlying organization of the two procedures. In tabu search the postoptimality information required to evaluate potential moves is always available from the most recent solution of (LP'), or from analysis that takes this solution as a starting point. In branch and bound, by contrast, information required to evaluate (or execute) a new branch which is reached by a backtracking or "sidetracking" step, must be based on some process of recovering or regenerating the tableau information from solving some linear program in the past.

Branch and bound, moreover, lacks the option of a complementation step, except where it is possible to complement a branch that meets one of the leaves (current end nodes) of the tree, since all other complementations require an enforced discard of intervening choices. There is no adequate way to compare the effect of remote complementations to those at the leaves of the tree, even after recovering or regenerating relevant information at the antecedent nodes, since bounds and penalty calculations applicable to these earlier nodes are less accurate than at their descendants.

Viewed from a tree perspective, tabu search always works at the leaf level, currently maintaining only a single path of branches from the root. Instead of having a rigidly inherited sequence, the branches on the path can have their order reshuffled to bring any branch whose alternative is not tabu to the end of the tree, thus enabling a single string of connections to yield a variety of different possibilities for the next step. By this means, there is greater latitude of choice to restructure the search than in branch and bound. The presence of additional choice opportunities on a single sequence of branches, however, can entail more evaluative effort per iteration if these opportunities are to be exploited fully.

Another characteristic of tabu search to be noted in this setting is that each option to be evaluated resides at the same tree depth, i.e., may be considered as a leaf of the tree. All such options therefore have access to information of comparable quality, in contrast to the inferior quality of information available at the earlier nodes of a branch and bound tree. Moreover, once enough branches have been created to make all components of x_1 integer, thereby reaching a depth where information concerning consequences of moves approaches its highest quality, the current depth can be maintained on subsequent iterations (allowing increases or decreases according to where new integer solutions are found and the imposed constraints that are currently binding). By thus maintaining a close proximity to integer solutions, the search tends to increase the number of feasible MIP solutions generated as candidates for x^* in relation to the total number of iterations.

In the process of building to an ideal depth, an alternative is to allow the algorithm to proceed from Step 3 to Step 5 when x_1' is not integer, provided $r \neq 0$. By this approach, the method can continue to explore a given depth, accepting only those complementation moves that the solution of (LP') identifies as improving, until a local optimum is obtained as a foundation for going to the next level. Such a local optimum may

appropriately be defined relative to a function that penalizes deviations of x_I components from integer values. (In this variation, a limit may be placed on the number of iterations allowed at a given level.)

Finally, the relaxation move for tabu search offers an additional possibility for increased latitude of choice. A complementation move can be viewed as a composite move consisting of a relaxation move followed by a restriction move, where both moves are defined relative to the same variable. After making a relaxation move, if a restriction move involving a different variable becomes preferable, then such an alternative is available to be selected by the method if the two types of moves are made in sequence rather than in combination. This type of relaxation-restriction sequence has no precise counterpart in branch and bound.

Defining Tabu Status

The manner of determining tabu status involves several considerations beyond those involved in creating the set T (or associated list T′) indicated for the two preceding MIP procedures. In particular, the three types of moves employed by MIP Method 3 suggests a natural implementation involving the creation of three tabu lists T_1, T_2 and T_3, with associated parameters t_1, t_2, t_3.

The first list, T_1, is updated whenever a relaxation move is made. If the relaxation move discards the constraint $x_j \leq u$ (or $x_j \geq u + 1$), then T_1 is used to prevent a restriction move from reimposing this same constraint until t_1 restriction or complementation moves have been made. Note that t_1 is not the size of T_1, which will generally vary. Since it is reasonable to make no more relaxation moves than restriction moves, however, T_1 typically will have no more than t_1 elements.

The list T_2 is updated when a restriction move is made. Only the identity of the variable x_j involved in the restriction, and not the form of the bound imposed, is relevant in this case. The role of T_2 is to prevent the occurrence of any relaxation move that involves x_j, for t_2 restriction or complementation moves. Here too, t_2 does not identify the size of T_2 but provides an upper limit, in this case without exception.

Finally, the list T_3 is updated when a complementation move is made. T_3 is used both to prevent the reverse of this complementation and to prevent the restriction move that reimposes the same constraint discarded by the complementation move, for a period of t_3 restriction or complementation moves.

In applying the parameters t_1, t_2 and t_3, particularly in the variant that seeks to generate local optima defined relative to a given depth before proceeding to the next, an alternative is to defer activation of the tabu lists until a deterioration in some evaluation function cannot be avoided (e.g., employing a function that combines $c(x)$ with a penalty for integer infeasibility). The opportunity to apply the types of dynamic tabu list strategies of Section 2 in this setting creates additional avenues for exploration, and poses a research challenge of determining the best way to handle moves that may be viewed as "partial complements" of others.

MIP Aspiration Level Functions

Two types of aspiration level functions are relevant to all of the preceding MIP methods. The first is based on stratifying possible values for $c(x)$ into intervals I_1, $I_2, \ldots, I_u$. Let $A(h)$ denote the aspiration level for I_h, where $A(h)$ is large initially. When a move results in replacing solution x' by a next solution x'', identify I_p and I_q such that $c(x') \in I_p$ and $c(x'') \in I_q$, and let

$$A(q) = \text{Min}(A(q), c(x'))$$

and

$$A(p) = \text{Min}(A(p), c(x'')).$$

Then tabu status can be disregarded for a move that leads from a subsequent x' to a subsequent x'' if

$$c(x'') < A(h) \quad \text{where } c(x') \in I_h.$$

To apply this criterion at the point where moves are evaluated, $c(x')$ may not be known accurately, but only approximated by the evaluation criteria. In such cases, the updating of $A(q)$ and $A(p)$ also may be based on approximate evaluations of $c(x'')$ or a refined approximation may be used for the aspiration level test. (The updating of $A(p)$ also may optionally be omitted.) For MIP Method 3, $c(x)$ should be replaced for the operations of checking and updating aspiration levels by a function which incorporates a penalty for noninteger x_I vectors.

The second type of aspiration level function is keyed to the variable and type of move employed. Let $A(j, v)$ represent the aspiration value of $c(x)$ for a "Type v" move involving variable x_j. For MIP Method 1, x_j is the variable whose value is changed on a given iteration and $v = 1$ or 2 according to whether the variable is incremented or decremented. In the adjacent extreme point procedure of MIP Method 2, x_j can be a variable that enters (or leaves) the basis, and v takes only the value 1. For MIP Method 3, x_j is the variable associated with the constraint(s) currently relaxed or (and) imposed, and v takes on values to code the possibilities previously identified as relevant to creating T1, T2 and T3. In all these cases, $A(j, v)$ can be initialized and updated in a manner analogous to that indicated for $A(h)$.

Finally, we note that these aspiration level functions can be integrated with the tabu restrictions in the manner indicated in Section 4 of Part I.

Additional MIP Memory Functions

Strategies for intensifying and diversifying search by means of intermediate- and long-term memory functions can vary widely in sophistication in the MIP setting, but it is worth noting that simple approaches are available that are easy to implement.

An intermediate-term memory function for the three MIP methods can be based on identifying variables that consistently receive certain values or bounds, or consistently appear in or out of the basis, in selected subsets of the best solutions (e.g., identified by cluster analysis that groups solutions of similar types). These variables may then be compelled to adopt the restrictions (values, bounds or basis classifications) thus associated with them for an additional period of search.

A simple long-term memory function can be based on recording the frequency that variables, or subsets of variables, take on distinguishing characteristics (i.e., appear at certain values, bounds, or basis states) in the trial solutions generated to date. Then the solution process is restarted or continued using a modified evaluation criterion that avoids particular classes of moves in relation to their recorded frequency, until reaching a trial solution that is locally optimal by this modified evaluation.

4. Tabu Search Applications

This section briefly highlights some of the applications of tabu search which have occurred (or have become more generally known) in the interval since the applications described in Part I.

A variety of tabu search implementations have been developed for problems containing a central feature that can be expressed by means of a graph theory or network flow representation. An example is a university course scheduling problem whose underlying structure has been expressed by Hertz and de Werra[17] as a weighted graph partitioning problem. In this approach, courses are represented as nodes and incompatibilities between courses are represented as edges. A different weight w(e) is assigned to each edge e according to the importance of the associated incompatibility. The goal is to partition the set of nodes N into k subsets, $N_1, \ldots, N_k$, in order to minimize $w(E_1) + \cdots + w(E_k)$, where $w(E_t) = \sum (w(e): e \in E_t)$ and E_t is the set containing the edges with both endpoints in N_t.

As in most real world applications, the resulting graph theory formulation does not encompass all constraints of interest, and the method is therefore designed to handle additional geographical constraints (involving different buildings in the university), compactness requirements and classroom capacity requirements. The approach has been successfully applied to schedule 300 courses for the Faculty of Economics at the University of Geneva. Two variants of this procedure, similarly based on an underlying graph theory model, have been developed by Hertz and de Werra[17] and by Benke[4] for scheduling courses, respectively, at the Swiss Federal Institute of Technology and at a technical school in Austria. In each case, additional restrictions which complicate the basic model framework are handled directly by incorporating appropriate infeasibility checks into the tabu search procedure.

Telecommunications problems involving minimum cost installation and call routing can be given natural formulations as network flow problems with discrete (all-or-none) conditions. The first phase of a study applying a variety of approaches for handling these formulations is described in a volume edited by J. Ryan.[31] A parallel processing implementation of tabu search for a path assignment problem (Oliviera and Stroud[27]) and a partitioned tabu search approach for a platform location and sizing problem (Lee[24]) emerged as significant developments of the volume, yielding highly effective solutions for their respective problems, while encompassing more of the real world attributes of these problems than accommodated by the other approaches studied.

A more classical graph theory application has been developed by Friden, Hertz and de Werra[8] to find large stable node sets in a graph. (A node set is stable if no pair of its elements is joined by an edge.) An approach for accelerating the method was devised by employing three different tabu lists which operate hierarchically. The study examines random graphs containing up to 1500 nodes, and in 60% of the cases obtains stable sets with cardinality equal to the probabilistic estimate of the maximum. In the remaining cases, the cardinality is only one unit below this estimate.

An extensive study applying tabu search to flow shop sequencing problems has been carried out by Widmer and Hertz.[36] Their implementation of tabu search succeeded in obtaining solutions superior to the best previously found (by applying a range of methods proposed in the literature) in about 90% of the cases.

A study by Laguna, Barnes and Glover[21] examined a machine scheduling problem that requires minimizing a weighted combination of delay penalties and sequence dependent setup costs. The method easily generated solutions to 20-job problems that could not be solved by several branch and bound procedures within 150 CPU seconds on a mainframe computer. Within a comparable time span (averaging 155 seconds) on a microcomputer, rather than a mainframe, the tabu search approach completed a set of 12 solution trials per problem and succeeded in obtaining an optimal solution to each. Moreover, the *worst* solutions obtained by tabu search over the 12 trials averaged within 99.8% of optimality, with a worst case of 99.6% of

optimality for all problems and parameter settings. The method also quickly obtained high quality solutions to larger problems which the optimizing methods could not handle either within reasonable time limits or storage requirements.

A sequel to this study by Glover and Laguna[14] focused on the harder problems of the first study and examined larger problems ranging up to 100 jobs. The goal was to seek improvements by exploiting the "best move" orientation of tabu search, incorporating an additional class of moves and using the learning procedure of target analysis.[10,13] The learning process was designed to identify the possibility of improved decision criteria for evaluating moves. As suggested by intuition, the inclusion of additional moves (introducing job transfers in conjunction with the job swaps previously studied), produced improvements in average solution quality and processing time. More significantly, with less evident intuitive support, the study found that a better criterion existed for evaluating moves than the objective function values produced by these moves. Decisions based on objective function values were shown by target analysis to exhibit a regional dependency, reducing their quality in situations where no admissible improving moves existed. This dependency was exploited by biasing evaluations in "bad regions" to reflect evaluations previously made in "good regions," and by creating an event-dependent tabu list which relaxed the tabu search requirements of admissibility—compensating for this relaxation by a strategy of penalizing repetitions as well as reversals of moves. The outcome nearly halved the number of iterations required to obtain optimal solutions to the 20 job problems (whose optimal solutions are known), and additionally improved the quality of the best solutions found for the larger problems.

An application of tabu search to the quadratic assignment problem has been developed by Skorin-Kapov,[34] utilizing tabu lists whose lengths vary both in relation to problem size and in relation to the stage of solution. A notable feature of the approach is its effective use of a long-term memory process for diversifying the search along the lines suggested in Part I. Tested on problems taken from the literature, the method yielded best known solutions in all cases while requiring less CPU time than previously reported. The method also succeeded in finding a better solution than the best previously known for Steinberg's problem,[35] and obtained solutions whose quality was always as good or better than that of solutions obtained by a study of the quadratic assignment problem using simulated annealing.

An unconventional tabu search application to the traveling salesman problem by Malek et al.[26] uses a parallel processing approach which incorporates a fault tolerant design, pursuing the goal of allowing recovery from a processor failure without having to restart the entire program. The approach makes use of Karp's procedure[19] of subdividing the problem nodes into clusters, seeking a good tour on each cluster, and then progressively merging pairs of clusters, using tabu search to guide the tour generation process at each stage. In addition to developing results concerning fault tolerant design, the approach provided a means for obtaining good trade-offs in solution time versus solution quality. For example, applied to a 532-city problem that has required 60 hours of run time to achieve optimality,[29] the method succeeded in obtaining a 95% optimal solution in only 38 seconds.

Another traveling salesman study by Mirek et al.[25] compares tabu search to simulated annealing, finding that tabu search performs uniformly more effectively and achieves additional gains by the incorporation of a long-term diversification strategy, as in the study of Skorin-Kapov.[34] This study also discloses that a hybrid approach, which trades solutions between tabu search and a modified form of simulated annealing, works well in a parallel processing environment. The authors suggest that their modification of simulated annealing can be interpreted as a relaxed version of probabilistic tabu search, hence motivating a study which investigates the probabilistic framework more thoroughly.

Another study of traveling salesman problems by Knox and Glover[20] tests the short-term memory component of tabu search against a variant of simulated annealing called the ELS method, due to Lam.[22] The ELS method departs from simulated annealing in several ways similar to those incorporated in the hybrid approach by Malek et al.,[25] and additionally makes use of special data registers to improve computational efficiency. Applied to classical test problems ranging from 25 to 105 cities, both the ELS method and tabu search succeeded in obtaining best known solutions at least once for each problem, out of 25 trial runs with different starting solutions. However, tabu search obtained best known solutions with somewhat greater frequency. Treating the frequencies of finding these solutions with each method as marginal probabilities, and applying a binomial probability distribution, the study identified the joint probability of finding a best known solution at least once in two runs to be 0.32 for tabu search and 0.04 for the ELS method. Across a series of five runs, these probabilities were 0.86 for tabu search and 0.36 for the ELS method. The study also underscored the relevance of employing candidate list strategies to reduce the number of moves examined in larger applications, as treated in [12].

A somewhat different type of application involves

the solution of a character recognition problem expressed as a minimum cardinality set covering problem by Hertz.[17] Each character is viewed as a set of black pixels on a grid. A pair of characters (C_1, C_2) is discriminated by a pixel if the pixel is contained in exactly one of the two members of the pair. The goal is to determine a smallest number of pixels that will discriminate all pairs of characters. Applied to a problem of discriminating 62 characters (1891 pairs) on a grid of 600 pixels, the tabu search procedure found a set of 16 pixels in about 10 minutes. Prior to this, the best known solution contained 17 pixels and was obtained by an integer programming method that was stopped after consuming several hours of computation.

In another study of a "recognition problem," de Werra and Hertz[7] describe an application of tabu search to neural networks. The objective is to create a transformation matrix defining synaptic weights for a visual pattern recognition problem, enabling the neural network to learn prototype patterns, or states. More precisely, the goal is to be able to identify inaccurate copies of the prototype states by a process that corrects any errors that may occur, up to d in number, where the synaptic weights successively transform an initial state containing these errors into a final stable state which corresponds to the associated prototype. Barring the ability to correct all possible states that may contain errors, a secondary goal is to minimize the number of parasite states—i.e., stable states, reached by the transformation process that do not correspond to the specified prototypes.

Using Hamming distances to evaluate the difference between a given state and its prototype, the objective function was formulated as that of minimizing the discrepancy of all initial states from their associated prototypes after a single application of the transformation matrix. The matrix that produced the greatest single step reduction in the number of errors, employing a weighted sum across all states containing at most d errors, was therefore assigned the highest evaluation. Starting with an arbitrary matrix, the moves used by the tabu search procedure consisted of changing exactly one element of the matrix, considering the smallest increase and decrease in the weight of this element such that the resulting matrix produced a different state than the unchanged matrix (where both were applied to the given current state).

The method was tested on a visual pattern recognition problem from [30] using two tabu lists, one to prevent reversing an "increase move" and the other to prevent reversing a "decrease move." The goal was to learn two prototype states in a network of 25 neurons containing up to 6 errors. The tabu search method resulted in an 80% reduction (from 21 to 5) in the number of parasite states produced by the previous learning approach. Moreover, instead of employing learning trials across all relevant associations of initial states and prototypes, which numbered approximately half a million, the approach was able to achieve its results after learning trials involving only 50 associations.

Conclusion

The foregoing applications of tabu search demonstrate the potential usefulness of the approach, and the fertile opportunity for innovation in adapting the method to alternative settings. As the number of applications of tabu search continues to grow, more is being learned about the best ways to apply these methods, and in time we may expect to see a more thorough determination of the types of data structures, tabu list procedures, aspiration criteria and other component processes that work best for particular types of problems. The aim of this paper has been to expose some of the future directions for extending and applying tabu search, and to document some of the findings of those who have contributed to its present practical success.

ACKNOWLEDGMENT

This research was supported in part by the Center for Space Construction of the University of Colorado under NASA Grant NAGQ-1388.

REFERENCES

1. E. Balas, 1979. Disjunctive Programming, in P.L. Hammer, E.L. Johnson and B. Korte (eds.), *Discrete Optimization II*, North Holland, Amsterdam, pp. 3–52.
2. E. Balas and C. Martin, Pivot and Complement—A Heuristic for 0–1 Programming, *Management Science 26*, 86–96.
3. M.S. Bazaraa and C.M. Shetty, 1976. *Foundations of Optimization*, Springer Verlag, Berlin.
4. Ch. Benke, 1988. Die Tabu-Search Method als moglicher Losungsansatz fur das Stundenplanproblem, Institut fur Hohere Studien, Vienna, Austria (August).
5. V.J. Bowman and F. Glover, 1972. A Note on Zero-One Integer and Concave Programming, *Operations Research 20: 1*, 182–183.
6. H. Crowder, E. Johnson and M. Padberg, 1983. Solving Large Scale 0–1 Linear Programming Problems, *Operations Research 31. 4*, 803–934.
7. D. de Werra and A. Hertz, 1989. Tabu Search Techniques: A Tutorial and an Application to Neural Networks, *OR Spectrum 11*, 131–141.
8. C. Friden, A. Hertz and D. de Werra, Stabulus: A Technique for Finding Stable Sets in Large Graphs with Tabu Search, to appear in *Computing*.
9. R.S. Garfinkel and G.L. Nemhauser, 1972. *Integer Programming*, John Wiley & Sons, New York.
10. F. Glover, 1986. Future Paths for Integer Programming and Links to Artificial Intelligence, *Computers and Operations Research 13: 5*, 533–549.
11. F. Glover, 1989. Tabu Search, Part I, *ORSA Journal on Computing 1: 3*, 190–206.

12. F. GLOVER, 1989. Candidate List Strategies and Tabu Search, CAAI Research Report, University of Colorado, Boulder (July).
13. F. GLOVER and H.J. GREENBERG, 1989. New Approaches for Heuristic Search: A Bilateral Linkage with Artificial Intelligence, *European Journal of Operational Research* 39: 2, 119–130.
14. F. GLOVER and M. LAGUNA, 1989. Target Analysis to Improve a Tabu Search Method for Machine Scheduling, Technical Report, Advanced Knowledge Research Group, US West Advanced Technologies, Boulder, CO (September).
15. F. GLOVER, D. KLINGMAN, N. PHILLIPS AND R. SCHNEIDER, 1985. New Polynomial Shortest Path Algorithms and their Computational Attributes, *Management Science 31*, 1106–1128.
16. P.L. HAMMER, E.L. JOHNSON, B.H. KORTE and G.L. NEMHAUSER, 1977. *Studies in Integer Programming*, North-Holland, Amsterdam.
17. A. HERTZ and D. DE WERRA, The Tabu Search Metaheuristic: How We Used It, to appear in *Annals of Mathematics and Artificial Intelligence.*
18. R.G. JEROSLOW, 1977. Cutting-Plane Theory: Disjunctive Methods, *Annals of Discrete Mathematics I*, 293–330.
19. R.M. KARP, 1977. Probabilistic Analysis of Partitioning Algorithms for the Traveling Salesman Problem in the Plane, *Mathematics of Operations Research 2: 3*, 209–224.
20. J. KNOX and F. GLOVER, 1989. Comparative Testing of Traveling Salesman Heuristics Derived from Tabu Search, Genetic Algorithms and Simulated Annealing, Center for Applied Artificial Intelligence, University of Colorado (September).
21. M. LAGUNA, J.W. BARNES and F. GLOVER, Scheduling Jobs with Linear Delay Penalties and Sequence Dependent Setup Costs Using Tabu Search, Research Report, Department of Mechanical Engineering, The University of Texas-Austin, April 1989.
22. J. LAM, 1988. An Efficient Simulated Annealing Schedule, Ph.D. Dissertation, Report 8818, Department of Computer Science, Yale University (September).
23. E.L. LAWLER, J.K. LENSTRA and A.H.G. RINNOOY KAN (eds.), 1985. *The Traveling Salesman Problem*, North-Holland, Amsterdam.
24. M. LEE, 1989. Least-Cost Network Topology Design for a New Service Using Tabu Search, *Heuristics for Combinatorial Optimization Sect. 6*, 1–18.
25. M. MALEK, M. GURUSWAMY, H. OWENS and M. PANDYA, 1989. Serial and Parallel Search Techniques for the Traveling Salesman Problem, *Annals of OR: Linkages with Artificial Intelligence.*
26. M. MALEK, M. HEAP, R. KAPUR and A. MOURAD, 1989. A Fault Tolerant Implementation of the Traveling Salesman Problem, Research Report, Department of Electrical and Computer Engineering, University of Texas-Austin, (May).
27. S. OLIVIERA and G. STROUD, 1989. A Parallel Version of Tabu Search and the Path Assignment Problem, *Heuristics for Combinatorial Optimization Sect. 4*, 1–24.
28. G. NEMHAUSER, and L. WOLSEY, 1988. *Integer and Combinatorial Optimization*, Wiley, New York.
29. M. PADBERG and G. RINALDI, 1987. Optimization of a 532-City Symmetric Traveling Salesman Problem by Branch and Cut, *Operations Research Letters 6: 1*, 1–7.
30. L. PERSONNAZ, I. GUYON and G. DREYFUS, 1986. Collective Computational Properties of Neural Networks: New Learning Mechanisms, *Physical Review A34*, 4217–4227.
31. J. RYAN (ed.), 1989. Final Report of Mathematics Clinic, *Heuristics for Combinatorial Optimization* (June).
32. H.M. SALKIN, 1975. *Integer Programming*, Addison-Wesley, Reading, Mass.
33. A. SCHRIJVER, 1986. *Theory of Linear and Integer Programming*, Wiley Interscience Series, New York.
34. J. SKORIN-KAPOV, 1989. Tabu Search Applied to the Quadratic Assignment Problem, *ORSA Journal in Computing 2: 1*, 33–45.
35. L. STEINBERG, 1961. The Backboard Wiring Problem: A Placement Algorithm, *SIAM Review 3*, 37–50.
36. M. WIDMER and A. HERTZ, A New Approach for Solving the Flowshop Sequencing Problem, to appear in *European Journal of Operational Research.*
37. S. ZIONTS, 1974. *Linear and Integer Programming*, Prentice-Hall, Englewood Cliffs, NJ.

SUPPLEMENTARY BIBLIOGRAPHY

J. BOVET, C. CONSTANTIN and D. DE WERRA, 1987. *A Convoy Scheduling Problem*, Research Report ORWP 87/22, Swiss Federal Institute of Technology in Lausanne (December).
M. GRONALT, 1988. *Die Verwendung der Tabu-Methode zur Losung eines Loading Problems*, Project Report, Institut fur Hohere Studien, Vienna, Austria (June).
P. HANSEN and B. JAUMARD, 1987. *Algorithms for the Maximum Satisfiability Problem,* RUTCOR Research Report RR#43-87, Rutgers, New Brunswick, NJ.
A. HERTZ and D. DE WERRA, 1987. Using Tabu Search Techniques for Graph Coloring, *Computing 29*, 345–351.
J. KNOX, 1989. *The Application of Tabu Search to the Symmetric Traveling Salesman Problem*, Ph.D. thesis, Graduate School of Business, University of Colorado (July).
CH. WENDELIN, 1988. *Graph Partitioning with the Aid of the Tabu Method*, Project report, Institut fur Hohere Studien, Vienna, Austria (June).