CONVERGENT TABU SEARCH FOR OPTIMAL PARTITIONING

Fred Glover

OptTek Systems Inc., 2241 17th Street, Boulder, CO 80302, USA

glover@opttek.com

Saïd Hanafi

LAMIH - UMR CNRS *r*° 8530
Unité de Recherche Opérationnelle et d'Aide à la Décision
Université de Valenciennes et du Hainaut-Cambrésis
Le Mont Houy - B.*P*. 311 - 59304 Valenciennes Cedex – France

Said.Hanafi@univ-valenciennes.fr

July, 2011

Abstract

We consider a specialization of tabu search for *optimal partitioning problems*, which can be used to model a wide variety of binary discrete optimization problems. Our method constitutes a form of tree search that applies under more general assumptions than standard tree search approaches. Moreover, the method allows combinations of problem restriction and problem relaxation strategies that are not available in ordinary branch and bound.

Keywords: Tabu search; Convergence; Recency memory; Partitioning problem.

1. Introduction.

We consider a specialization of tabu search (TS) for combinatorial optimization problems that belong to the class of *optimal partitioning problems*. These problems encompass a wide range of classical applications, and include pure 0-1 integer programming problems by incorporating penalty functions for violating problem constraints. Our TS specialization for these problems constitutes a form of tree search that applies under more general assumptions than standard tree search approaches, with the ability to generate precisely the set of feasible solutions to all problems satisfying these assumptions. Moreover, the method can use bounding information and other structural considerations to bypass examination of dominated solutions, as in customary branch and bound. However, its organization allows combinations of problem restriction and problem relaxation strategies that are not available in ordinary branch and bound.

Our approach is based on the application of *two-attribute* moves for transitioning from one solution to another, which may be viewed as exchanging values between two variables or exchanging elements between two sets. In the special case of formulations where these moves collapse to single attribute moves (such as changing the value of a single variable), and where variables are allowed to take dummy values that correspond to assigning them *no* values, the method reduces to the type of branch and bound approach commonly applied to solving integer programming problems. In this instance feasible solutions for a particular structure are effectively treated as unimportant by standard methods; *i*.e., feasibility relative to the tree search is customarily defined by constraining integer vectors to be restricted by upper and lower bounds on their components, and problem relaxations are used to weed out infeasibilities that derive from consideration of other problem constraints. By contrast, our tabu search specialization directly generates solutions that satisfy more limiting conditions, before introducing information provided from problem relaxations. The outcome yields a pliable and effective form of tree search, with a very economical memory structure. In addition, we introduce a dynamic procedure that defers the identification of forbidden (tabu) elements, yielding a search method with expanded flexibility. This dynamic approach allows further exploitation of restriction and relaxation strategies.

2. Problem Formulation.

The Optimal Partitioning (OP) Problem may be defined as follows. We seek a partition of a set $R = \{1, \ldots, r\}$ into two sets P and Q, where we require $|P| = v$ for a specified constant v, to minimize an objective function $f(P, Q)$. The set R may be conceived as an index set for a collection of elements, or for a vector of zero-one variables, where choosing an index to belong to P corresponds to assigning the associated element to a specified class or setting the associated variable equal to 1. In addition, to requiring $|P| = v$ (or equivalently $|Q| = r - v$), for increased generality we incorporate a feasibility structure that makes reference to *nested* subsets R_k of R, for $k = 1, \ldots, K$, where the nesting condition specifies that any pair of these subsets must either be disjoint or one of the pair must be contained in the other. Then we stipulate that between u_k and l_k elements of R_k must belong to P, for each k, where u_k and l_k are specified constants satisfying $u_k \geq l_k$. (If $K = 0$, we understand this condition to be irrelevant. Equivalently we may take $K = 1$ and specify $R_1 = R$ and $u_1 = l_1 = v$.)

In addition, relative to the minimal subsets R_k of R, which contain no other subsets within them, we allow consideration of special sets of ordered pairs $O_k \subseteq \{(i, j) : i, j \in R_k\}$. (For the purpose of creating the sets O_k we may introduce new subsets R_k that include any elements of a given subset that belong to no subset within it, thus allowing each such new R_k to qualify as minimal. Values of u_k and l_k for these new subsets may be chosen to be unrestrictive.) The elements (i, j) of O_k are assumed to define a partial ordering, where such an ordering is established under the condition where (i, j) represents the logical precedence relationship "$i \in P$ implies $j \in P$". (Note if the subsets R_k are pairwise disjoint, then each R_k provides a basis for such a partial ordering.) Thus, by these conventions, we obtain the following formulation :

Problem OP.

Minimize $f(P,Q)$

subject to

(1) P and Q partition R

(2) $|P| = v$

(3) $l_k \leq |R_k \cap P| \leq u_k$, $k = 1, \ldots, K$, for nested $R_k \subseteq R$

(4) $i \in P$ implies $j \in P$, for each $(i, j) \in O_k$, and for k ranging over minimal R_k.

A variety of different types of problems result by considering only subsets of the constraints (1) – (4), or by introducing restrictive assumptions about the nature of the sets

R_k or O_k. For example, classical graph partitioning problems arise as an instance of the case where (3) and (4) are entirely disregarded. The inclusion of the nesting inequalities of (3) encompasses problems arising in optimal inventory handling, and the inclusion of the logical implications of (4) is relevant to problems from the domain of logical inference. We will conceive constraints (1) and (2) as fundamental to the OP formulation, though it is easily possible by introducing dummy elements and related artifices to render them redundant, so that they are not a necessary limitation on constraints (3) and (4), as we will show shortly.

Other restrictions will also later be allowed to apply to the sets P and Q. At the moment, however, we will suppose any additional restrictions are reflected in the values taken by $f(P, Q)$, handling these constraints in a *penalty function* formulation. In general, it is important to identify conditions that need not to be embedded in penalty function evaluations. The ability to generate precisely the set of feasible solutions under such conditions can involve a much smaller set of elements when feasibility is not relegated solely to the control of penalty functions. We will see, for example, that it is possible to incorporate conditions that permit the set P to range over the set of bases of a matroid, which encompass a wide variety of problem structures. We also include conditions that permit the OP problem to model multidimensional knapsack problems and generalized covering problems.

It is useful to consider how the OP problem can be related to alternative problem formulations by the device of coding variables to represent set membership. We give an example as follows.

Zero-One IP Problems.

Pure 0-1 IP problems fall in the OP problem classification, by means of the penalty function representation, using a straightforward coding of variables. Let x_k, $k = 1,\ldots, K$ denote a set of 0-1 variables, and let the two possible value assignments, $x_k = 0$ and $x_k = 1$, for a given k, be coded to correspond to two indexes of an associated set R_k; e.g., $R_k = \{k, K + k\}$. The second element $K + k$ of R_k will be conceived to correspond to a new variable x_{K+k} that represents the complement of x_k, *i*.e., $x_{K+k} = 1 - x_k$. Hence in the 0-1 IP problem exactly one of these two variables must receive the value 1 and the other must receive the value 0. To express this condition in the OP formulation, we consider the operation of choosing an element j of R_k to belong to P as being equivalent to specifying that $x_j = 1$, hence we want to assure that exactly one element $j = k$ ou $K + k$ from R_k is selected. This corresponds to choosing exactly one of $x_k = 1$ or $x_{K+k} = 1$ (hence, in the latter case, $x_k = 0$). Stated in terms of the constraints of (3), we must specify that $|R_k \cap P| = 1$, which results by stipulating $u_k = l_k = 1$. We note in this case that $R = \{1, \ldots, 2K\}$ and the sets R_k are pairwise disjoint.

The logical implication constraints of (4) are disregarded, in this instance, and the complete formulation results by expressing (2) as $|P| = K$ (*i*.e. setting $v = K$), which in fact is redundant since exactly one element of each R_k is chosen to belong to P by the indicated form of (3).

Alternatively, the OP problem can be reexpressed in the form of a 0-1 problem. To show this, for each $j \in R$ we similarly introduce a 0-1 variable x_j where $x_j = 1$ if $j \in P$ and $x_j = 0$ if $j \in Q$. Let x denote the vector $(x_j : j \in R)$, and let $g(x) = f(P, Q)$. Then we may write the OP problem as the following 0-1 integer program :

Minimize $g(x)$

Subject to

(1') $\quad x_j = 0$ or $1, \qquad j \in R$

(2') $\quad \sum_{j \in R} x_j = v$

(3') $\quad l_k \leq \sum_{j \in R_k} x_j \leq u_k, \qquad k = 1, \ldots, K$

(4') $\quad x_i \leq x_j, \qquad (i, j) \in O_k$ for k over minimal R_k.

The constraints (1') to (4') correspond directly to the constraints (1) to (4). While we consider the constraints (1) and (2), or equivalently (1') and (2'), to be fundamental to our formulation, the Appendix shows the preceding formulation can represent a problem in which v is replaced by upper and lower bounds. We will subsequently consider additional kinds of constraints that can be explicitly included in this formulation without having to be incorporated into the function $g(x)$.

3. Convergent Tabu Search

Many optimization techniques (both heuristic and exact) for solving combinatorial and nonlinear problems are iterative neighborhood search procedures – i.e., they start with an initial solution (feasible or infeasible) and repeatedly construct new solutions from current solutions by moves defined by reference to a neighborhood structure. The process continues to generate a trajectory of "neighboring solutions" until a certain stopping criterion is satisfied. We assume the reader has a rudimentary acquaintance with tabu search as a basis for motivating the key steps of our approach. (For background see, for example, Glover and Laguna (1997).) However, the statement of the method and its associated properties can be understood directly, without requiring knowledge of tabu search to establish their validity.

The adaptive memory approach of Tabu Search generates a neighborhood trajectory by including a mechanism that forbids the search to revisit solutions already encountered – unless the intervening trajectory is modified (see Glover (1990)). The main goal of memory structures in TS is not simply to forbid cycling, however. In fact, the choice of a given neighborhood and a decision criterion for selecting moves with TS can force some solutions to be revisited before exploring other new ones. Within this context, a proposal of Glover (1990) identifies a simple rule for revisiting solutions that is conjectured to have implications for finiteness in zero-one integer programming and optimal set membership problems. Hanafi (2000) proves Glover's conjecture under the assumption that the graph of the neighborhood space is connected and symmetric. Glover and Hanafi (2002) provided new proofs that yield specific bounds establishing the finite convergence of tabu search, specifically for certain TS algorithms based on recency memory or frequency memory.

The results distinguish between symmetric and asymmetric neighborhood structures and provide insights into the sequences of solutions generated by the search. The outcomes disclose interesting contrasts between TS trajectories and the those generated by the more rigid rules underlying tree search methods. Based on these findings, we also give designs for more efficient forms of convergent tabu search, and provide special rules that create a new type of tree search. The finiteness of these methods suggests an important distinction between their underlying ideas and the rationale that gives rise to "infinite time" convergence results for certain randomized procedures such as annealing.

Let X be the set of feasible solution, each solution $x \in X$ has an associated neighborhood $N(x)$, and let $Time(x)$ = the most recent time (iteration) that solution x was visited by a search process, whose form is determined as follows.

Convergent Tabu Search (CTS)

Step 0: Initialization, the values $Time(x)$, $x \in X$, begin as arbitrary nonnegative integers, and the starting solution x^* for the search is assigned a value so that $Time(x^*) >$

Time(x) for all x other than x^*. (This includes the case where we begin with *Time*(x) = 0 for all $x \in X$ except x^*. Select a starting solution $x \in X$ and set *Iteration* = 0;

Step 1: Set *Time*(x) = *Iteration*. Select an unvisited neighbor $x' \in N(x)$ such that *Time*(x') = 0, if one exists, and otherwise choose to visit a solution x' = argmin{*Time*(y) : $y \in N(x)$}; *Time*(x) = *Iteration*.

Step 2: Stop if all solutions in X are visited i.e., *Time*(x) > 0, $\forall\, x \in X$. Otherwise set *Iteration* = *Iteration* + 1; $x = x'$ (move from x to x'); go to Step 1.

We now state some key properties of our method, which follow from the analysis of Glover and Hanafi (2002).

Property A. Denote the cardinality of X by $n = |X|$, and consider a value U_n for $n \geq 2$ which is given recursively by $U_1 = 0$ and define $U_{n+1} = 2U_n + 1$, for $n \geq 1$. Beginning with any solution $x^* \in X$, the CTS method will visit every solution in X in at most U_n steps if X is finite and there exists a neighborhood path from every solution in X to every other solution in X.

The *"*min{*Time*(x)} *rule"* is the one called the Aspiration by Default rule in the TS literature. Since frequency-based memory is also useful in TS, it is natural to speculate that a "frequency version" of Property A is valid. We apply the natural definition, *Frequency*(x) = the number of times x has been visited, and replace *Time*(x) by *Frequency*(x) and set *Frequency*(x) = *Frequency*(x) + 1 after visiting solution x. The conclusion of Property A holds when CTS is based on frequency memory.

Additional enhancements that are possible using this TS methodology include the use of an associated Reverse Elimination Memory for streamlining the new tree search procedure associated with the TS process (which differs from the convergent TS approach that uses more flexible memory). We also propose an approach for accelerating the classical tabu search Aspiration by Default rule in this setting, which may transform an exponential search into a much faster polynomial search. Finally, we give designs for more efficient forms of convergent tabu search in general.

In contrast to our previous use of the label *Time*(x) for each solution x, however, we add the stipulation that as soon as *Time*(x) is assigned a value (i.e., as soon as x is visited), we do not permit its value to be further changed. Accompanying this, we now reverse the Aspiration by Default rule, to require that, whenever all elements of $N(x)$ have previously been visited, the method moves from x to the node $x' \in N(x)$ that has the largest (rather than smallest) value of $Time(x')$, subject to the limitation that this value must be smaller than that of $Time(x)$ itself. The resulting method is as follows.

Tabu Tree Search (TTS)

1. From a given solution x, move to an unvisited neighbor $x' \in N(x)$ whenever possible, and stop if the label thus assigned to x' is $Time(x') = |X|$. Otherwise,

2. Move to the visited neighbor x' with the largest value of $Time(x') \leq Time(x)$.

We establish the relevant properties of the method as follows, under the assumption that the graph of the neighborhood space is connected.

Property B. The TTS method generates a tree, rooted at the initial solution, that spans the nodes of the neighborhood graph. Each edge of the tree is crossed exactly once in the direction away from the root, and at most once in the direction toward the root. (No edges outside of the tree are crossed.) In addition:

(a) The unique path from any solution to the root is generated by repeatedly executing the rule of Step 2 of the TTS method.

(b) Each time any solution x is visited, each labeled neighbor x' of x is either an ancestor or descendant of x in the tree currently constructed (i.e., either x' lies on the path from the root to x, or else lies on the path from the root to x').

(c) Each time step 2 is executed to reach a visited node x', all nodes of the graph that are neighbors of visited nodes x'', where $Time(x'') > Time(x')$, are also visited nodes.

(d) Each time step 1 successfully identifies an unvisited neighbor of x, then node x satisfies the condition $x = \text{Argmax}\{Time(y) : y$ is a node of the current tree and y has an unvisited neighbor$\}$.

In common with the Aspiration by Default rule, the TTS approach in some cases may visit all solutions by only visiting each solution a single time, hence effectively generating a Hamiltonian path through the neighborhood space, in contrast to the type of trajectory created by usual forms of tree search. However, more importantly, the TTS approach allows substantially greater flexibility of choice than customary types of tree search, as embodied in branch and bound approaches. On the other hand, the TTS structure differs according to the choices made – that is, different choices may produce different numbers of revisited solutions (and, as previously remarked, some may produce no revisited solutions), thus producing trees of different topologies.

4. Tabu Search Specialization for OP.

Our specialization of tabu search to the OP problem begins with arbitrary sets *P* and *Q* satisfying the constraining conditions of the problem. Each move consists of identifying an element *p* of *P* and an element *q* of *Q* and exchanging them, thus redefining $P = P – p + q$ and $Q = Q – q + p$ (applying the natural convention to the meaning of the symbols + and –). For greater heuristic effectiveness, we suppose that candidate list strategies and evaluation criteria are employed to choose a “best” current move from those available at each step, but we will not bother to refer to these aspects of the method in its description.

The elements *p* and *q*, and the ordered pair (*p, q*) (which implicitly associates *p* with *P* and *q* with *Q*), are taken to be the move attributes that will be used to define tabu status. Elements chosen as attributes in tabu search, for the purpose of determining tabu status, are often treated differently from each other according to their role in defining a move -- hence in this case, for example, according to whether such an element is transferred from *P* to *Q*, or vice versa. However, in the present specialization we treat all elements *p* and *q* in the same way. In particular we maintain a single tabu list, which we denote by *tabu_list*, to record each element that becomes tabu.

A move (in contrast to an element) will be classified tabu, and hence will be forbidden to be executed, if any of its elements is tabu. To apply this classification, we select only one of the two elements *p* and *q* associated with a given move to receive a tabu status, and allow this choice to be made arbitrarily at each step. (To compare these conditions with other common options, we note that assigning a tabu status to both elements *p* and *q* would create a stronger tabu restriction rendering a larger number of moves tabu. On the other hand, these alternatives would be weakened by specifying a move to be tabu only if all its elements are tabu.)

The move attribute represented by the ordered pair (*p*, *q*) gives rise to the weakest type of tabu restriction, by specifying the reversed pair (*q*, *p*) to be tabu (hence forbidding a move that simultaneously puts *p* back in *P* and *q* back in *Q*). This restriction is dominated by the restriction that classifies a move tabu if it contains a (selected) member element *p* or *q* of the pair (*p*, *q*). Hence the attribute pair (*q*, *p*) is relevant only in the situation where all current moves are tabu. In this case we apply the standard criterion of choosing a move according to *aspiration by default*, which selects the weakest tabu move as the one to execute.

These simple conventions determine the basis of our TS specialization, and we disregard other components of tabu search, such as special short and long term strategies for intensification and diversification. (These additional components can be included to provide a more flexible search structure, provided the specialized form is relied upon as an underlying process that is recovered to continue the search after intervening departures.)

Memory Structure.

A memory structure for handling the preceding conventions is provided by maintaining *tabu_list* as an ordered list, and by specifying that each new element is added at the end, as frequently done in classical tabu search developments. We let *last*(*tabu_list*) denote the current last element of the list. Hence the operation of adding an element *e* (= *p* or *q*) may be denoted by stipulating *tabu_list* = *tabu_list* + *e*, where *e* becomes the new element identified as *last*(*tabu_list*). (We do not bother to give the list a more advanced structure, or to treat it in an implicitly circularized form, since the indicated organization suffices for the current specialization. Also, for simplicity of description, we will not make reference to auxiliary pointer arrays that identify locations of elements on *tabu_list*, though we will understand such arrays to be used for efficient implementation.)

Ordered pairs (*q*, *p*) that identify the weakly tabu moves are similarly recorded on a list denoted *weak_tabu_list*. Upon executing a move associated with the ordered pair attribute (*p*, *q*), we therefore add the tabu attribute (*q*, *p*) to the end of *weak_tabu_list* by the operation *weak_tabu_list* = *weak_tabu_list* + (*q*, *p*).

These operations have several implications. We discuss these briefly to motivate the precise rules adopted in our specialization. Readers interested only in the form of these rules, without reference to their motivation, may skip the next subsection.

Consequences of the Specialized Organization.

When a (weak) tabu move identified by a pair (*q*, *p*) is executed, the organization of the procedure always results in (*q*, *p*) = *last*(*weak_tabu_list*). It is evidently appropriate to remove the pair (*q*, *p*) from *weak_tabu_list* at this point, since the tabu condition represented by this pair has been countermanded. (This follows the usual design of the tabu search, where executing a tabu move is accompanied by cancelling the condition that made it tabu.) Similarly, tabu conditions that were generated after the point when (*q*, *p*) was added to *weak_tabu_list* (hence which were created under the assumption that the move associated with (*q*, *p*) is forbidden) no longer remain relevant. Thus, we likewise discard these conditions when the tabu move associated with (*q*, *p*) is executed.

It will happen that some of these conditions are discarded automatically without having to change the membership of elements on *tabu_list*. To see this, note the element *p* or *q* that was added to *tabu_list*, at the same time that (*q*, *p*) was added to *weak_tabu_list*, is an attribute both of (*q*, *p*) and of the reverse pair (*p*, *q*). When the tabu move identified by (*q*, *p*) is executed, the TS approach causes its reverse move (*p*, *q*) in turn to become tabu. Consequently, we can manage this simply by leaving the previously chosen element *p* or *q* on *tabu_list*. (The element (*p*, *q*) is not put on *weak_tabu_list* at this point, because the operation of executing a tabu move precludes a weak status for its reversal.)

The remaining tabu conditions that need to be discarded, as previously observed, are the strong tabu conditions that arise when no elements are placed on *weak_tabu_list*. That is, when an element *e* = *p* or *q* is allowed to remain on *tabu_list* (upon executing an associated tabu move identified by (*q*, *p*)), we may properly view this as being an operation that first removes *e* from *tabu_list* and then places it back, but with a different status -- because upon placing it back, no associated attribute (*p*, *q*) is added to *weak_tabu_list*. We *underline e* to differentiate the stronger status that *e* gains when it is thus “re-placed” on *tabu_list*. This convention gives a convenient device for identifying and discarding tabu elements that are not updated automatically, noting that such elements are precisely the members of *tabu_list* that are underlined. Similarly, elements *p* and *q*, and ordered pairs (*p*, *q*), that receive an implied tabu status as a result of elements already on *tabu_list* (that is, which cannot take part in an exchange, while maintaining feasibility and satisfying the goal of obtaining an improved solution), are underlined and added to *tabu_list*. Such a use of underlining represents an extension of the approach introduced in (Glover, 1965), which has likewise proved useful in a more restricted implicit enumeration context (Glover, 1965).

Structure of the Method.

An appropriate rule for underlining elements of *tabu_list*, as subsequently identified, we define *underline_end* to be the sublist consisting of all underlined elements at the "end" of *tabu_list*, that is, all underlined elements that follow the last non-underlined element. This sublist is empty if *last*(*tabu_list*) itself is not underlined. (If all elements of *tabu_list* are underlined, then *underline_end* = *tabu_list*.) Then upon executing a tabu move, it follows that the tabu elements to be discarded are those that belong to *underline_end*, and we delete them simply by redefining *tabu_list* = *tabu_list* – *underline_end*.

These observations are embodied in the following description of the specialized TS approach for the OP problem, which we call the TS-OP method. For notational convenience, we introduce a parameter d that refers to the depth of the search, and maintain the convention of designating an element p or q, or an ordered pair (p, q), to be tabu if it lies on *tabu_list*. Actually, the *tabu_list* can contain pairs as well as single elements. But *tabu_list* only contains pairs only if they are underlined. The term *feasible* refers to solutions (sets P and Q) that satisfy the constraints (1) – (4) of the OP problem.

The Specialized TS-OP Method.

0. Start with an initial feasible pair P and Q. Let *tabu_list* and *weak_tabu_list* both being empty, and let the depth $d = 0$. Also let $P^* = P$ and $Q^* = Q$, identifying the best current known solution.

1. Given the restrictions on exchanges embodied in *tabu_list*, if feasibility and the goal of improving the best known solution prevent elements or ordered pairs from taking part in an exchange, then underline such elements and pairs and add them to the end of *tabu_list*. (These additions may lead to other additions.)

2. Choose a pair (p, q) satisfying $p \in P$ and $q \in Q$, where p, q and the pair (p, q) are not tabu, such that exchanging p and q between P and Q will create a new feasible solution. Then, if such a pair is found, choose $e = p$ or $e = q$ and set

   $P = P - p + q$

   $Q = Q - q + p$

   tabu_list = *tabu_list* + e

   weak_tabu_list = *weak_tabu_list* + (q, p)

   $d = d + 1$.

   If now $f(P, Q) < f(P^*, Q^*)$, set $P^* = P$ and $Q^* = Q$.

3. If no pair (p, q) can be found to satisfy the conditions in Step 2: terminate if $d = 0$ (or equivalently if *weak_tabu_list* is empty); otherwise let (q, p) = *last*(*weak_tabu_list*), and set

   $P = P + p - q$

   $Q = Q + q - p$

   tabu_list = *tabu_list* – *underline_end*

   weak_tabu_list = *weak_tabu_list* - (q, p)

   $d = d - 1$.

   Underline the current element *last*(*tabu_list*) (which is either p or q).

4. After executing Step 2 or Step 3, return to the start of Step 1.

It is possible to update only P (and to record only $P^* = P$) in the foregoing procedure, since Q is automatically known from P. Relative to the 0-1 IP problem representation of Section 2, this update corresponds to changing the value of x_p from 1 to 0 and of x_q from 0 to 1 in Step 2, and changing the value of x_p from 0 to 1 and of x_q from 1 to 0 in step 3. Note also that the depth d is equal to the size of *weak_tabu_list.*

The effort of checking whether the exchange of p and q in Step 1 is feasible is trivial where logical precedence constraints are not included and where the sets R_k are pairwise disjoint. Fast procedures to identify feasible exchanges, without having to check each one independently of the others, are given by the results of Glover (1965) in the case of the nested inequality constraints and of Glover and Greenberg (1989) in the case of the logical precedence constraints.

To illustrate how the foregoing TS-OP method works, we will consider an instance of OP problem with r = 5 and v = 2 where only constraints (2) and (3) are present and the goal is to enumerate all feasible solutions.

iteration	*x*	*Move*	*TL*	*|TL|*	*WTL*	*|WTL|*	*Depth*
1	11000	(1,3)	{}	0	{}	0	0
2	01100	(2,4)	{1}	1	{(3,1)}	1	1
3	00110	(3,5)	{1,2}	2	{(3,1);(4,2)}	2	2
4	00011	**(5,3)**	{1,2,3}	3	{(3,1);(4,2);(5,3)}	3	3
5	00110	(4,5)	{1,2,**3**}	3	{(3,1);(4,2)}	2	2
6	00101	**(5,4)**	{1,2,**3**,4}	4	{(3,1);(4,2);(5,4)}	3	3
7	00110	**(4,2)**	{1,2,**3**,**4**}	4	{(3,1);(4,2)}	2	2
8	01100	(3,4)	{1,**2**}	2	{(3,1)}	1	1
9	01010	(4,5)	{1,**2**,3}	3	{(3,1);(4,3)}	2	2
10	01001	**(5,4)**	{1,**2**,3,4}	4	{(3,1);(4,3);(5,4)}	3	3
11	01010	**(4,3)**	{1,**2**,3,**4**}	4	{(3,1);(4,3)}	2	2
12	01100	**(3,1)**	{1,**2**,**3**}	3	{(3,1)}	1	1
13	11000	(2,3)	{**1**}	1	{}	0	0
14	10100	(3,4)	{**1**,2}	2	{(3,2)}	1	1
15	10010	(4,5)	{**1**,2,3}	3	{(3,2);(4,3)}	2	2
16	10001	**(5,4)**	{**1**,2,3,4}	4	{(3,2);(4,3);(5,4)}	3	3
17	10010	**(4,3)**	{**1**,2,3,**4**}	4	{(3,2);(4,3)}	2	2
18	10100	**(3,2)**	{**1**,2,**3**}	3	{(3,2)}	1	1
19	11000		{**1**,**2**}	2	{}	0	0

Solutions depicted in blue are 3-times revisited and solutions in green are revisited 2-times, finally the remain solutions are revisited exactly once. Moves in red and bold are tabu moves.

The main results about the foregoing procedure may be stated as follows.

Property C. The TS-OP method generates each feasible solution to the OP problem formulated with constraints (3) and (4) redundant with a maximum depth $d \le r - 1$. In addition, $d \le v$ if e is always selected to be q in Step 1, and $d \le r - v$ if e is always selected to be p in Step 1.

Let $F(k)$ = the number of distinct solutions visited exactly k times by the TSOP algorithm.

Corollary. After running the TSOP algorithm each feasible solution of the OP problem is visited at least once and at most v+1 times. More precisely the number of distinct solutions visited exactly k times by the TSOP algorithm can be generated by the following formula:

$F(1) = C(r\text{-}1, v\text{-}1)$ $\qquad$ $k = 1$

$F(k) = C(r\text{-}k, v\text{-}k\text{+}2)$ $\qquad$ for $1 < k < v+1$;

$F(k) = C(r\text{-}k, v\text{-}k\text{+}2)+1$ $\qquad$ for $k = v+1$;

For $v = r - 1$, $F(1) = C(r\text{-}1, v\text{-}1)$; $F(k) = 0$ for $1 < k < v+1$; $F(r) = 1$.

where $C(r, v) = r! \,/\, p!(r\text{-}p)!$ with the convention that $C(r, v) = 0$ if $v > r$.

Note the dimension of F is equal to v+1. The total number of solutions visted by TSOP algorithm is equal to $2{*}C(r, v) - 1$, that is two times the size of the feasible set but one.

Property D. Property C is also true when the OP problem is instead formulated with constraints (3) and (4) redundant, but with the additional constraint that each P is the index set for a base of matroid.

Property E. Property A is also true when (4) is redundant and (3) takes the form for a general 0-1 IP problem identified in Section 2, with the addition of constraints that define a multidimensional knapsack problem or that define a generalized covering problem.

5. A Dynamic Tabu List Version

The most effective forms of tabu search use dynamic tabu lists. (See, e.g., Taillard (1990); Chakrapani and Skorin-Karpov (1993); Gendreau, Soriano and Salvail (1993); Hansen and Jaumard (1990).) One of the forms of dynamic tabu lists that has proved particularly effective defers the assignment (Hübscher and Glover, 1994), as applied in the studies of Dammeyer and Voss (1993); Voss (1996) and Hanafi and Fréville (2001). We show how this idea can be used to create a more advanced and flexible version of the specialized approach of Section 4.

To handle this change, we allow the tabu list to include pairs of elements (*p*, *q*) instead of a succession of single elements (selected as one of *p* or *q* at each step). The appearance of a paired element (*p, q*) on *tabu_list* means that the choice of the element *p* or *q* to be tabu has been deferred, and we will thus refer to components of such pairs as *deferred tabu elements*. An element *p* or *q* that becomes a singleton on *tabu_list* (by deleting the other element according to rules subsequently identified), is considered tabu in the usual sense and is called an *active tabu element*.

When a pair (*p*, *q*) is on *tabu_list*, with both *p* and *q* deferred tabu elements, it is possible to choose a move in the future that contains either *p* or *q* (but not both), *i*.e., to choose a move represented by a pair of the form (*z*, *p*) or (*q*, *z*), excluding *z* = *q* or *p*, respectively. By contrast, in the earlier procedure where only *p* or *q* is placed on the tabu list, one of these two types of future moves is automatically eliminated and there is no opportunity to choose between them.

The operation of deferred tabu status applies likewise to underlined elements. The appearance of underlined pair of elements on tabu_list signals that each member of the pair has a deferred tabu status, and that it will become an ordinary (single) underlined element once its status becomes active. (It is entirely possible, however, that an element will never change from a deferred status to an active status.)

When a move (*p*, *q*) is selected, the pair (*q*, *p*) (rather than the pair (*p*, *q*) is added to *tabu_list*, just as it is added to *weak_tabu_list*. The operation of underlining such a pair (*q*, *p*) on *tabu_list* is accompanied by replacing it on *tabu_list* in reverse as (*p*, *q*). (This convention is not strictly necessary, because the current composition of *P* and *Q* always identifies which member of a pair belongs to *P* and which belongs to *Q*, without bothering to order these elements. However, to avoid ambiguity in our description, we keep the ordering explicit.) Single elements on *tabu_list* are treated exactly as before.

By these conventions, *last*(*tabu_list*) can either be an order pair or a singleton, and similarly *underline_end* can include both ordered pairs and singletons. An ordered pair (*p*, *q*) on *tabu_list* becomes a singleton *p* or *q* by executing a move of the form (*z*, *q*) or (*p*, *z*), respectively. The resulting p or q is underlined according to whether (p, q) was underlined.

These prescriptions lead to the following modified form of the TS-OP method.

Dynamic TS-OP Method.

0. Start with an initial feasible pair P and Q. Let *tabu_list* and *weak_tabu_list* both being empty, and let the depth $d = 0$. Also let $P^* = P$ and $Q^* = Q$, identifying the best current known solution.

1. Choose a pair (p, q) satisfying $p \in P$ and $q \in Q$, where p, q and the pair (p, q) are not tabu, such that exchanging p and q between P and Q will create a new feasible solution. Then, if such a pair is found, set

   $P = P - p + q$

   $Q = Q - q + p$

   tabu_list = *tabu_list* + (q, p)

   weak_tabu_list = *weak_tabu_list* + (q, p)

   $d = d + 1$.

If there is a pair (p, y) on tabu_list, replace (p, y) by y, and if there is a pair (z, q) on tabu_list, replace (z, q) by z (where y and / or z is underlined if the pair that contained it was underlined). Finally, if now $f(P, Q) < f(P^*, Q^*)$, set $P^* = P$ and $Q^* = Q$.

2. If no pair (p, q) can be found to satisfy the conditions in Step 1: terminate if $d = 0$ (or equivalently if *weak_tabu_list* is empty); otherwise let (q, p) = *last*(*weak_tabu_list*), and set

   $P = P + p - q$

   $Q = Q + q - p$

   tabu_list = *tabu_list* – *underline_end*

   weak_tabu_list = *weak_tabu_list* - (q, p)

   $d = d - 1$.

If *last*(*tabu_list*) is the ordered pair (q, p) (rather than a singleton) : set *tabu_list* = *tabu_list* - (q, p) + (p, q) underline the current element *last*(*tabu_list*) (which is either p or q or the pair (p, q)).

3. After executing Step 1 or Step 2, return to the start of Step 1.

We observe that *weak_tabu_list* can be absorbed into *tabu_list* in the foregoing procedure, if we do not eliminate elements of ordered pairs to produce singletons, but instead simply “mark” elements that would correspond to such singletons to establish their tabu status. Then the pair (*q*, *p*) taken from *weak_tabu_list* in step 2 may be identified as the pair that appears as *last*(*tabu_list*), after removing *underline_end* in Step 2 (where one of *q* or *p* may be marked element).

The operation of identifying a pair (*p*, *y*) or (*z*, *q*) in Step 1 is greatly simplified, and can be executed as a “look up” rather that a search, due to the fact that, for any element *p* and any element *q*, at most one pair (*y*, *p*) and at most one pair (*z*, *q*) can be on *tabu_list*. This useful relationship is a direct consequence of the organization of the procedure and the results established in Section 3.

References

Chakrapani, J. and J. Skorin-Kapov (1993). Massively Parallel Tabu Search for the Quadratic Assignment Problem. Annals of Operations Research, 41, 327–341.

Dammeyer F., S. Voss, (1993), "Dynamic Tabu List Management using the Reverse Elimination Method", Annals of Operations Research, 41, 31-46.

Gendreau M., P. Soriano and L. Salvail, (1993). Solving the maximum clique problem using a tabu search approach. Annals of Operations Research, 41, 385-404.

Glover F., (1965). A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem. Operations Research, Vol. 13, No. 6, 879-919.

Glover F., (1990). Tabu Search, Part 2. *ORSA Journal on Computing* 2, 4-32.

Glover F. and Greenberg, H.J., (1989). New approaches for heuristic search: A bilateral linkage with artificial intelligence. European Journal of Operational Research, Volume 39, Issue 2, 24 March, 119-130.

Glover F., S. Hanafi, (2002). Tabu Search and Finite Convergence. Special Issue on “Foundations of heuristics in Combinatorial Optimization”. Discrete Applied Mathematics, 119, 3-36.

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

Hanafi S., (2000). On the Convergence of Tabu Search. Journal Of Heuristics, 7, 47-58.

Hanafi S., A. Fréville, (2001). Extension of Reverse Elimination Method Through a Dynamic Management of the Tabu List. RAIRO Oper. Res. 35, 251-267.

Hansen P. and B. Jaumard, (1990). Algorithms for the maximum satisfiability problem. Computing 44, 279-303.

Hübscher R., F. Glover, (1994). Applying Tabu Search with influential diversification to multiprocessor scheduling. Computer and Operations Research, 13, 877-884.

Taillard E., (1990). Some efficient heuristic methods for the flowshop sequencing problem. European Journal of Operational Research, 47, 65-74.

Voss S., (1996). Dynamic tabu search strategies for the traveling purchaser problem. Annals of Operations Research 63, 253-275.

Appendix : A bounded Formulation

We show how v can be replaced by upper and lower bounds, u_0 and l_0, so that (2) and (2') of Section 2 translate respectively into the more general requirements $u_0 \geq |P| \geq l_0$ and $u_0 \geq \sum_{j \in R} x_j \geq l_0$. (First, note these constraints take the form of (3) and (3') by defining $R_0 = R$, replacing P by P inter R_0 and replacing R by R_0. More precisely, to encompass the inequalities in which u_0 and l_0 replace v, let $R' = \{1, \ldots, r'\}$ denote the "original R" over which such inequalities apply. Then we obtain an initial "corresponding" OP formulation by setting $v = u_0$ and by making reference to a different R, $R = \{1, \ldots, r\}$ where $r = r' + (u_0 - l_0)$.

Without further change, this gives a loose equivalence, where feasible solutions of the original problem are included multiple times among feasible solutions of the new problem. That is, every solution of the new problem that includes within P a given number of the elements from the set $\{r' + 1, \ldots, r\}$ corresponds to just one solution to the original problem defined over R'. To eliminate this multiple counting and give a one-one correspondence between solutions of the two problems, we make use of the constraints of (4) by defining an additional set $R(K + 1) = \{r' + 1, \ldots, r\}$, and an associated set of ordered pairs $O(K + 1) = (j, j + 1)$ where j and $j + 1$ range over the elements of $R(K + 1)$. In the 0-1 IP formulation this corresponds to specifying $x_{j+1} \geq x_j$ for each j, j+1 over this set. Hence if exactly one element of this set is chosen to belong to P (setting $x_j = 1$ for this element) it must uniquely be the elements r and $r - 1$, etc. This gives the desired one-one mapping between solutions of the original and the new formulations. (In case $u_0 = l_0 + 1$, we note there are no precedence constraints to include in the new formulation.)

The device for formulating the OP problem as a 0-1 IP problem can be extended directly to formulate a problem in which each x_k takes any one of a set values, letting x_j be a 0-1 variable associated with x_k where j ranges over the elements of $P_k = \{k, k+K, \ldots.\}$. if we include a dummy value for x_k represented by $x_k = *$, which is interpreted as the case where x_k is not assigned a value, then the rules of the OP method in Section 3 give precisely the type of tree search produced by a branch and bound algorithm for a standard IP problem. (Thus implicitly it generates mores solutions than when the value $x_k = *$ is excluded, though in terms of overall computational complexity this is a trivial difference.)