Technical Aspects of Tabu Search

Annals of Operations Research 41(1993)31–46

Dynamic tabu list management using the reverse elimination method

Frank Dammeyer and Stefan Voß

Technische Hochschule Darmstadt, FB1/FG Operations Research, Hochschulstrasse 1, D-6100 Darmstadt, Germany

Abstract

Tabu search is a metastrategy for guiding known heuristics to overcome local optimality. Successful applications of this kind of metaheuristic to a great variety of problems have been reported in the literature. However, up to now mainly static tabu list management ideas have been applied. In this paper we describe a dynamic strategy, the reverse elimination method, and give directions on improving its computational effort. The impact of the method will be shown with respect to a multiconstraint version of the zero–one knapsack problem. Numerical results are presented comparing it with a simulated annealing approach.

1. Introduction

Due to the complexity of a great variety of combinatorial optimization problems, heuristic algorithms are especially relevant for dealing with large scale problems. The main drawback of algorithms such as deterministic exchange procedures is their inability to continue the search upon becoming trapped in a local optimum. This invites consideration of recent techniques for guiding known heuristics to overcome local optimality. Following this theme, we investigate the application of the *tabu search* metastrategy for solving a multiconstraint version of the zero–one knapsack problem.

Many solution approaches are characterized by identifying a neighbourhood of a given solution which contains other (*transformed*) solutions that can be reached in a single iteration. In the following a transition from a feasible solution to a transformed feasible solution will be referred to as a *move*, which may be described by a set of one or more *attributes*. For example, in a zero–one integer programming context these attributes may be the set of all possible value assignments (or changes in such assignments) for the binary variables. Then two attributes e and $\bar{e}$, which denote that a certain binary variable is set to 1 or 0, may be called *complementary* to each other. Following a steepest ascent/mildest descent approach, a move may either result in a best possible improvement or a least deterioration of the objective function value. Without additional control, however, such a process can cause a locally optimal solution to be re-visited immediately after moving to a neighbour.

J.C. Baltzer AG, Science Publishers

To prevent the search from endlessly cycling between the same solutions, we consider a version of tabu search that may be described as follows. We may imagine the attributes of all moves are first stored in a *running list*, i.e. a list representing the trajectory of solutions encountered. Then a list derived as a sublist of the running list may be defined. Based on certain restrictions, this so-called *tabu list* keeps some moves, consisting of attributes complementary to those of the running list, which will be forbidden in at least one subsequent iteration because they might lead back to a previously visited solution. Thus, the tabu list restricts the search to a subset of admissible moves (consisting of admissible attributes or combinations of attributes). This hopefully leads to "good" moves at each iteration without revisiting solutions already encountered. A general outline of a tabu search procedure (for solving a maximization problem) may be described as follows:

TABU SEARCH

Given: A feasible solution x^* with objective function value z^*.
Start: Let $x := x^*$ with $z(x) = z^*$.
Iteration:
while stopping criterion is not fulfilled **do begin**
(1) select best admissible move that transforms x into x' with objective function value $z(x')$ and add it to the running list;
(2) perform tabu list management: compute moves to be set tabu, i.e. update the tabu list;
(3) perform exchanges: $x := x'$, $z(x) = z(x')$
if $z(x) > z^*$ **then** $z^* := z(x)$, $x^* := x$ **endif**
endwhile
Result: x^* is the best of all determined solutions with objective function value z^*.

For a background on tabu search and a number of references on successful applications of this metaheuristic see Glover [9, 10] and Hertz and de Werra [12, 13]. Further applications can be found, for instance, in [2, 3, 14, 15].

In the next section we shall describe a specific dynamic method for step (2) and give some comments on and corrections to related implementational issues presented in the literature. In section 3 we present the application of our procedure to the multiconstraint zero–one knapsack problem. Finally we draw some conclusions.

2. The reverse elimination method

Tabu list management means the update of the tabu list, i.e. deciding on how many and which moves have to be set tabu within any iteration of the search. Up to now most implementations of tabu search use static tabu list management. In

such a method, moves are set tabu as soon as their complements have been selected. These moves stay tabu for a given number of iterations. More precisely, selected move attributes are assigned a tabu status which in turn determines whether moves containing these attributes are tabu. The efficiency of the algorithm depends on the choice of the tabu status duration or, equivalently, on the length of the tabu list. In the literature, often a "magic" tabu list length of 7 is proposed. Though successful in some applications, this seems to be a rather limited approach. More recently, dynamic tabu list management which assigns a status duration that varies according to the attributes considered, has found increasing favor (see e.g. [3, 11]). Some of these methods determine a tabu status based on sequential relationships between the selected moves, in a manner that rigorously excludes certain types of cycling behaviour. Examples in this respect are the *cancellation sequence method* (cf. [10] and [1]) and the *reverse elimination method* (REM) proposed by Glover [10].

The remainder of this paper will deal with the latter method. For ease of description we assume that a move consists of exactly one attribute (i.e. we consider so-called *single-attribute moves* instead of *multi-attribute moves*).

2.1. OVERVIEW OF REM

REM follows the idea that any solution can only be re-visited in the next iteration if it is a neighbour of the current solution. Therefore, in each iteration the running list will be traced back to determine all moves which have to be set tabu, since they would lead to an already explored solution. For this purpose a *residual cancellation sequence* (RCS) is built up stepwise by a trace. (Tracing back the running list will be called a *trace*.) In each step of a trace, i.e. in each so-called *tracing step*, exactly one attribute is processed, starting with the last (most recent) back to the first (earliest). Beginning with an empty RCS, only those attributes are added whose complements are not in the sequence. Otherwise their complements in the RCS are eliminated (i.e. cancelled). Then at each tracing step it is known which attributes have to be reversed in order to turn the current solution back into one examined at an earlier iteration of the search.

If the remaining attributes in the RCS can be reversed by exactly one move then this move is tabu in the next iteration. Otherwise this move will reproduce an earlier solution. Let a *position* denote an attribute's location in the running list. In general, if a move consists of more than one attribute, the position of an attribute and the iteration number of the corresponding performed move differ from each other. Figure 1 gives an example for building residual cancellation sequences. As every move consists of exactly one attribute the length of an RCS has to become equal to one to enforce a tabu move.

The tabu status assigned by REM represents a necessary and sufficient criterion to prevent re-visiting known solutions. The effort required by REM clearly grows as the number of iterations increases, and thus ideas for reducing the number of computations must be developed.

Running list: $\bar{1}\ \bar{6}\ 7\ 3\ 1\ \bar{5}\ 6\ 4\ \bar{6}\ 5$ (latest move: 5) iteration: k = 10

position	tracing step	residual cancellation sequence	length	tabu move
10	1	5	1	$\bar{5}$
9	2	$\bar{6}$ 5	2	-
8	3	4 $\bar{6}$ 5	3	-
7	4	4 5	2	-
6	5	4	1	$\bar{4}$
5	6	1 4	2	-
4	7	3 1 4	3	-
3	8	7 3 1 4	4	-
2	9	$\bar{6}$ 7 3 1 4	5	-
1	10	$\bar{6}$ 7 3 4	4	-

Fig. 1. Example for RCS-development.

2.2. REDUCING THE NUMBER OF TRACING STEPS

Assume that in any iteration k of the search procedure the tth tracing step leads to an RCS of length r. Then in iteration $k+1$ the length of the RCS in tracing step $t+1$ is not smaller than $r-1$. More specifically, it is equal to either $r-1$ or $r+1$. Therefore, following [10], a vector *Least* may be defined as follows.

> *Least*(i) is the smallest position *pos_n* such that backtracing up to *pos_n* leads to an RCS of length $r \leq i+1$.

If *Least*(1), . . . , *Least*(p) have been determined in iteration k (with $p \leq k$), in iteration $k+j$ ($j \leq p$) backtracing is necessary up to the position number stored in *Least*(j) only. Any additional step cannot lead to an RCS of length 1 and therefore to a move that has to be set tabu. In iteration $k+p+1$ the vector *Least* no longer provides usable information. Therefore, *Least* must be updated not later than in iteration $k+p+1$. Nevertheless, an update of *Least* may be performed whenever *Least*(j) = 1 in any iteration $k+j$. This procedure can easily be generalized for the case of multi-attribute moves.

Consider the example given in fig. 1. For $p = 3$ we calculate

Least(1) = 5, *Least*(2) = 4, *Least*(3) = 1.

Figures 2 and 3 show the tracing steps necessary in iterations $k+1$ and $k+2$. Figure 3 also shows a worst-case example for the calculation of *Least*, i.e. an update of *Least* in iteration 12 would lead to *Least*(1) = 1.

As the number of stored attributes increases it may become too burdensome for large iteration numbers to trace back the entire running list (even when using *Least* since this vector has to be updated). Therefore, a parameter *at_n* (attribute number) is introduced to limit the number of tracing steps per iteration although

Running list: $\bar{1}\ \bar{6}\ 7\ 3\ 1\ \bar{5}\ 6\ 4\ \bar{6}\ 5\ \bar{3}$ (latest move: $\bar{3}$) iteration: k+1 = 11

position	tracing step	residual cancellation sequence	length	tabu move	comment
11	1	$\bar{3}$	1	3	
10	2	$5\ \bar{3}$	2	-	
9	3	$\bar{6}\ 5\ \bar{3}$	3	-	
8	4	$4\ \bar{6}\ 5\ \bar{3}$	4	-	
7	5	$4\ 5\ \bar{3}$	3	-	
6	6	$4\ \bar{3}$	2	-	
5	7	$1\ 4\ \bar{3}$	3	-	termination

Fig. 2. Example for the use of *Least*(1) = 5 (7 necessary tracing steps).

Running list: $\bar{1}\ \bar{6}\ 7\ 3\ 1\ \bar{5}\ 6\ 4\ \bar{6}\ 5\ \bar{3}\ \bar{4}$ (latest move: $\bar{4}$) iteration: k+2 = 12

position	tracing step	residual cancellation sequence	length	tabu move	comment
12	1	$\bar{4}$	1	4	
11	2	$\bar{3}\ \bar{4}$	2	-	
10	3	$5\ \bar{3}\ \bar{4}$	3	-	
9	4	$\bar{6}\ 5\ \bar{3}\ \bar{4}$	4	-	
8	5	$\bar{6}\ 5\ \bar{3}$	3	-	
7	6	$5\ \bar{3}$	2	-	
6	7	$\bar{3}$	1	3	
5	8	$1\ \bar{3}$	2	-	
4	9	1	1	$\bar{1}$	termination
3	10	$7\ 1$	2	-	in case of updating Least
2	11	$\bar{6}\ 7\ 1$	3	-	
1	12	$\bar{6}\ 7$	2	-	

Fig. 3. Example of the use of *Least*(2) = 4 (9 necessary tracing steps).

accuracy may suffer: the complement of an attribute that is no longer traced might be part of a move leading back to a previously explored solution.

2.3. TERMINATION OF THE BACKTRACING

Obviously it is not necessary to trace back the running list if an attribute of a current move does not find its complement in the running list. To demonstrate this we add element 2 as next move in the example given in fig. 1. Then the length of every RCS is increased by 1 with respect to the previous iteration. Only the complement of the current move has to become tabu.

More generally, Glover [10] remarks that a trace need not be continued after adding an attribute within a specific tracing step that does not have its complement among earlier entries of the running list. With respect to necessity and sufficiency this is not correct: Assume that, due to the chosen definition of *at_n* or *Least*, the running list has to be traced back p steps. Then a direct analogue to Glover's condition to terminate the backtracing would be that an attribute e to be added to the current RCS at step i does not find its complement $\bar{e}$ on the running list within backtracing steps $i, \ldots, p$. (In the case of single-attribute moves $\bar{e}$ has not been added to the running list during iterations $k-p+1, \ldots, i-1$). We call such an attribute a *solo attribute*. For $i > 1$ this condition need not give a suitable termination criterion as can be seen in the example given in fig. 1. In the third tracing step the attribute $e = 4$ is added without having its complement among all earlier entries of the running list. Therefore, the trace will not be continued. Figure 4 shows the same example with a duplicated solution after adding move $\bar{4}$. This should have been a tabu move according to the fifth tracing step of fig. 1.

Running list: ... $1\ \bar{5}\ 6\ 4\ \bar{6}\ 5\ \bar{4}$ (latest move: $\bar{4}$)

position	tracing step	residual cancellation sequence	length	tabu move	comment
11	1	$\bar{4}$	1	4	
10	2	$5\ \bar{4}$	2	-	
9	3	$\bar{6}\ 5\ \bar{4}$	3	-	
8	4	$\bar{6}\ 5$	2	-	
7	5	5	1	$\bar{5}$	
6	6		0	-	duplicated solution

Fig. 4. Example for termination of the backtracing and choosing move $\bar{4}$.

In the following we discuss how and when the condition has to be modified to give a correct termination criterion.

A modification that leads to a suitable criterion is to terminate the procedure if, in addition to a solo attribute e, a second solo attribute $e' \neq e$ is found. Figure 5 gives an example of the improved termination criterion. With two solo attributes e and e' identified as described above every additional tracing step will lead to an RCS-length of at least 2. Therefore, this modified criterion is correct since any RCS-length will be reduced by at most 1 in the next iteration. When multi-attribute moves are considered the number of solo attributes found must be increased appropriately to terminate a trace.

The latter criterion can be extended as follows. The trace is continued after detection of the first solo attribute. Whenever the complementary move to this solo attribute becomes tabu within a subsequent tracing step the backtracing may be terminated.

a) iteration k with move 5

Running list: $\bar{1}\ \bar{6}\ 7\ 1\ \bar{5}\ 3\ 6\ 4\ \bar{6}\ 5$ (latest move: 5)

position	tracing step	residual cancellation sequence	length	tabu move	comment
10	1	5	1	$\bar{5}$	
9	2	$\bar{6}$ 5	2	-	
8	3	4 $\bar{6}$ 5	3	-	e = 4
7	4	4 5	2	-	
6	5	3 4 5	3	-	e' = 3 , termination
5	6	3 4	2	-	
4	7	1 3 4	3	-	
...					

b) iteration k+1

Running list: $\bar{1}\ \bar{6}\ 7\ 1\ \bar{5}\ 3\ 6\ 4\ \bar{6}\ 5\ \bar{4}$ (latest move: $\bar{4}$)

position	tracing step	residual cancellation sequence	length	tabu move
11	1	$\bar{4}$	1	4
10	2	5 $\bar{4}$	2	-
9	3	$\bar{6}$ 5 $\bar{4}$	3	-
8	4	$\bar{6}$ 5	2	-
7	5	5	1	$\bar{5}$
6	6	3 5	2	-
5	7	3	1	$\bar{3}$
4	8	1 3	2	-
...				

Fig. 5. Example for termination of the backtracing.

Although there are two suitable termination criteria, another computationally more attractive modification is to retain the termination approach of Glover [10]. However, the move consisting of the attribute whose complement is the first found solo attribute is set tabu and the trace is terminated. An example may again be drawn from fig.1, where the backtracing is terminated after three steps with the tabu list consisting of the moves $\bar{5}$ and $\bar{4}$. This is a sufficient but not a necessary criterion, as the last entry of the tabu list might not lead to an already visited solution (cf. fig. 2 where $\bar{4}$ would be set tabu after step 4, although this is not necessary). Additional related strategic considerations are sketched by Glover [10].

2.4. SEARCH INTENSIFICATION

A general idea for reducing the computational effort in a tabu search algorithm is that of search intensification using a so-called *short term memory* (STM, compare the expression *intermediate term memory* in [9]). The basic idea is to observe the

attributes of all performed moves and to eliminate those from further consideration that have not been part of any solution generated during a given number of iterations. This results in an obvious concentration of the search. The number of neighbourhood solutions in each iteration, and consequently the computational effort, decreases. Obviously the cost of this reduction may be a loss of accuracy.

Correspondingly, a search diversification may be defined as a *long term memory* (however, resulting in an increased computation time). Here we choose a combined version (L + STM) as follows. Whenever a certain number of iterations has been performed (including at least one application of STM) a new feasible solution consisting of attributes that have been eliminated through STM is generated and used as a new starting solution.

2.5. MULTI-ATTRIBUTE MOVES

In the preceding subsections we explained REM for single-attribute moves. The generalization to multi-attribute moves is straightforward. Still, a different definition of neighbourhood search is necessary. We distinguish two kinds of multi-attribute moves: *general* and *successive multi-attribute moves*. In the first case a move may consist of more than one attribute and a trace will be performed whenever a move is realized, i.e. the number of traces is equal to the number of iterations. This kind of move seems to be necessary if a mathematical formulation of a problem may be given incorporating very restrictive equality constraints (e.g. the traveling salesman problem or the quadratic semi-assignment problem).

If a multi-attribute move may be completely decomposed into single-attribute moves, where feasibility of a solution is maintained after performing each part of the move, we use the notation of a successive multi-attribute move. This kind of move may be useful if an underlying mathematical formulation of a problem consists of inequality constraints (compare section 3). In the case of successive multi-attribute moves REM may be applied regarding every move as a number of separate single-attribute moves. In that case in any iteration the number of traces to be performed is equal to the number of attributes of the corresponding move.

Another classification of multi-attribute moves distinguishes *static* and *dynamic moves* depending on whether the number of attributes in a move is constant for all iterations or not. For static moves a straightforward generalization of REM that may reduce the computational effort is the processing of all attributes of a complete move within a single step of the trace.

3. An application

We consider the *multiconstraint zero–one knapsack problem* (MCKP):

$$\text{Maximize} \quad Z(x) = \sum_{j=1}^{n} c_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij}x_j \leq b_i \qquad i = 1, \ldots, m, \tag{2}$$

$$x_j \in \{0,1\} \qquad j = 1, \ldots, n. \tag{3}$$

Without loss of generality all a_{ij} are assumed to be nonnegative and all c_j and b_i are assumed to be positive. MCKP is a special case of general zero–one programming with a great variety of applications in the areas of, e.g. resource allocation and capital budgeting. Various algorithms for solving this NP-hard problem have been proposed in the literature (see, e.g. [4,5,7,8,16–18]). Here we refer to Drexl [4], who developed an efficient simulated annealing algorithm which may be used as a benchmark for an implementation of REM with respect to MCKP.

A trivial feasible solution for MCKP with an objective function value of $Z(x) = 0$ consists of $x_j = 0$ for all $j = 1, \ldots, n$. Any other feasible solution may be obtained by adding or including appropriate elements into this solution as long as no constraint of (2) is violated. In a similar fashion every infeasible combination of binary variables may become feasible by dropping or excluding appropriate elements. Given a feasible solution $x = (x_1, \ldots, x_n)$, Drexl [4] applies a random neighbourhood search in his simulated annealing approach by repetitively adding and/or dropping randomly chosen elements. Feasibility is maintained in all intermediate steps of the algorithm.

For the purpose of finding a neighbourhood solution within tabu search we define the following transformation:

DROP/ADD MOVE

Given: A feasible solution $x = (x_1, \ldots, x_n)$.

Choose $j^* = \arg\max \{a_{i^*j}/c_j \mid x_j = 1,\ j = 1, \ldots, n\}$ with i^* being a bottleneck resource with

$$i^* := \arg\max\left\{ \sum_{j=1}^{n} a_{ij} \cdot x_j / b_i \mid i = 1, \ldots, m \right\},$$

$x_{j^*} := 0$
Feasible := **true**
while Feasible
choose $k^* = \arg\max \{c_j \mid x_j = 0\}$ with

$$a_{ik^*} + \sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i \quad \forall i = 1, \ldots, m$$

if k^* exists **then** $x_{k^*} := 1$ **else** Feasible := **false endif**
endwhile

The DROP/ADD move as described above may be considered as a multi-attribute move with variable length depending on a specific instance of MCKP. Any such move consists of exactly one *DROP-attribute* j^* and a variable number of *ADD-attributes* according to the choice of elements k^* given in the while-loop*. We will now use an implementation of REM considering these moves as successive multi-attribute moves. Therefore, the tabu list management is the same as described in section 2.

Remark

In most cases we will have $j^* \neq k^*$ due to the different selection criteria. Furthermore, j^* leads to an RCS of length 1 before choosing the first ADD-attribute. Due to the notion of successive multi-attribute moves a trace is performed before entering the while-loop of the DROP/ADD move.

Both approaches, simulated annealing as described in Drexl [4] and tabu search using REM, have been applied to MCKP. To have a comparative study, 57 test problems with known optimal solutions from the literature were solved with both algorithms. The data are, among others, from [17] and [18] and are fully reproduced in [5] and [6]. In the first three columns of table 2 the problem data are identified by their authors' names, the number of variables n, and the number of constraints m. All programs are implemented in PASCAL and were run on an IBM PS2/70, 386 personal computer. (Our re-implementation of the simulated annealing algorithm turned out to be at least as good as the original implementation of [4]).

To thoroughly study each test problem, 20 initial feasible solutions have been generated. Starting with $x = (0, \ldots, 0)$ for the first feasible solution we added elements according to the ADD-criterion given in the description of the DROP/ADD move as long as possible. Nine further solutions have been gained from $x = (0, \ldots, 0)$ by adding randomly chosen elements as long as possible. In the same way we proceeded with an initialization of $x = (1, \ldots, 1)$. Nine solutions have been gained by randomly dropping elements until feasibility is achieved, and the tenth solution was obtained by applying a DROP-criterion inverse to the ADD-criterion mentioned above.

In the sequel numerical results are presented in order to compare simulated annealing to tabu search with the basic version of REM as described in section 2 and with the STM- and the L+STM-version, respectively.

The basic parameter setting of both simulated annealing and tabu search may be found in table 1. Considering the framework of tabu search given in section 1, we need to describe a termination criterion. We refer to a *stride* as a certain number

*In some pathological cases (e.g. when dropping any element from a given solution is tabu) we allow for a dummy attribute (not to be added to the running list) to preserve the structure of the DROP/ADD move.

of iterations. Given one of the 20 initial feasible solutions, the algorithms (simulated annealing or tabu search) will terminate whenever there is no improvement of the best feasible solution within all iterations of a stride. The length of a stride starting from an initialization is increased over time by an increase factor as given in table 1. Additionally, an overall termination criterion may be applied. In our case we choose a stopping criterion of at most 500 n iterations for simulated annealing and of at

Table 1

Parameters of the algorithms.

Simulated annealing (cf. [4])	*Tabu search*
Temperature reduction factor: 0.6	Maximum length of *Least*: 2
Initial temperature: 0.5 $(\max\{c_j\} - \min\{c_j\})$	Attribute number: $at\_n = 4n$
Initial number of iterations (stride): n	Initial number of iterations (stride): n
Increase factor (number of iterations): 1.3	Increase factor (number of iterations): 1.1

most 10 n iterations for the tabu search strategies. The overall stopping criterion is checked after each stride. For our examples, this criterion did not affect the termination of tabu search.

In our STM strategy, an element is eliminated from further consideration at the end of a stride if it has not been included in any solution examined during the stride. For L+STM we have implemented the following modification. Whenever REM with STM is stopped a new starting solution is obtained by choosing from those elements that have previously been eliminated according to STM, and REM with STM is restarted. This procedure is repeated no more than ten times as long as a new starting solution can be found.

Table 2 gives a detailed description of our numerical results. For each combination of problem instance and algorithm, *opt* gives the number of optimal solutions found. The value *dev* gives the relative deviation from optimality of the objective function value of the best found feasible solution (both values gained from the sample of 20 instances). In addition, columns *iter* (rounded to integer values) and *CPU* give the average number of iterations, i.e. the number of performed neighbourhood exchanges, and the CPU-time in seconds averaged over all 20 instances, respectively. The value it^* shows the least number of iterations needed to calculate the best found feasible solution in any of the 20 instances.

Analyzing the results shows advantages of tabu search over simulated annealing. All tabu search implementations find a larger number of optimal solutions than simulated annealing, indicating a higher flexibility of tabu search. Computation times seem to be comparable for all methods except REM with L+STM. This method needs more time due to the restarts with new starting solutions. It should be noted that whenever $it^* = 0$, at least one initial feasible solution corresponds to the best found solution using the improvement procedure at hand, i.e. either no improvement was possible ($dev = 0$) or none was found.

Table 2
Numerical results.

			Simulated annealing					Tabu search/basic REM					Tabu search/REM with STM					Tabu search/REM with L+STM				
	n	*m*	*opt*	*dev*	*iter*	CPU	it^*	*opt*	*dev*	*iter*	CPU	it^*	*opt*	*dev*	*iter*	CPU	it^*	*opt*	*dev*	*iter*	CPU	it^*
Fleisher	20	10	1	0	1556	1.50	143	0	0.795	49	0.40	4	0	0.795	47	0.35	4	0	0.795	365	2.89	4
Hansen/Plateau 1	28	4	0	1.2	1542	1.84	0	0	0.497	85	0.56	7	0	0.497	85	0.54	7	0	0.497	188	1.32	7
Hansen/Plateau 2	35	4	0	1.287	4097	4.57	684	0	1.193	81	0.72	8	0	1.193	81	0.67	8	0	1.193	159	1.43	8
Freville/Plateau 1	27	4	0	1.553	1527	1.89	260	1	0	79	0.53	6	1	0	70	0.44	6	5	0	207	1.37	6
Freville/Plateau 2	34	4	0	0.596	2613	3.07	1728	0	0.534	86	0.73	4	0	0.534	86	0.68	4	0	0.534	271	2.31	4
Freville/Plateau 3	19	2	1	0	3918	3.90	0	2	0	43	0.31	0	2	0	43	0.25	0	20	0	132	0.86	0
Freville/Plateau 4	29	2	3	0	1550	2.00	313	0	0.743	85	0.68	1	0	0.743	66	0.47	1	0	0.743	130	1.05	1
Freville/Plateau 5	20	10	1	0	1562	1.51	143	0	0.795	50	0.41	4	0	0.795	48	0.35	4	0	0.795	326	2.55	4
Freville/Plateau 6	40	30	2	0	1218	2.12	886	7	0	117	4.45	10	6	0	109	2.75	10	17	0	511	13.47	10
Freville/Plateau 7	37	30	0	0.193	1023	1.75	390	5	0	109	2.52	59	3	0	99	1.77	68	8	0	1049	19.68	68
Petersen 1	6	10	18	0	129	0.23	0	20	0	11	0.03	0	20	0	11	0.04	0	20	0	24	0.09	0
Petersen 2	10	10	2	0	1209	1.35	8	4	0	21	0.08	1	4	0	20	0.07	1	19	0	40	0.17	1
Petersen 3	15	10	2	0	542	0.82	15	16	0	43	0.21	3	12	0	39	0.19	3	12	0	46	0.24	3
Petersen 4	20	10	0	0.327	1159	1.59	7	1	0	54	0.37	1	1	0	43	0.27	1	10	0	136	0.93	1
Petersen 5	28	10	0	1.774	4600	4.55	0	0	0.242	91	0.78	8	0	0.242	70	0.56	8	0	0.161	249	2.17	30
Petersen 6	39	5	0	1.092	7422	8.54	265	1	0	90	0.76	8	1	0	90	0.74	8	11	0	283	2.81	8
Petersen 7	50	5	0	0.683	10867	12.37	1637	0	0.447	131	1.58	44	0	0.447	121	1.35	44	0	0.447	512	6.37	44
Senju/Toyoda 1	60	30	3	0	5952	8.69	3113	0	0.142	264	14.63	30	0	0.142	156	5.56	30	4	0	1664	59.81	298
Senju/Toyoda 2	60	30	1	0	7990	11.95	3127	4	0	244	8.60	168	0	0.229	160	4.25	7	2	0	1663	47.51	316
Weingartner 1	28	2	0	0.567	3904	4.27	1072	2	0	74	0.57	1	2	0	60	0.40	1	19	0	456	3.39	1
Weingartner 2	28	2	1	0	4874	5.03	0	2	0	64	0.55	0	2	0	65	0.49	0	5	0	457	3.67	0
Weingartner 3	28	2	3	0	5838	5.78	0	1	0	67	0.85	0	1	0	64	0.58	0	14	0	350	3.50	0
Weingartner 4	28	2	3	0	2759	3.09	559	2	0	64	0.48	2	2	0	60	0.39	2	7	0	200	1.39	2
Weingartner 5	28	2	0	3.046	5566	5.84	0	2	0	79	0.96	2	2	0	58	0.53	2	6	0	557	5.61	2
Weingartner 6	28	2	0	0.299	4813	5.07	0	0	0.299	88	0.83	0	0	0.299	62	0.46	0	1	0	542	4.40	144
Weingartner 7	105	2	0	0.260	28753	34.87	6213	0	0.006	329	6.95	19	0	0.006	281	5.65	19	0	0.006	701	14.65	19
Weingartner 8	105	2	0	0.441	12608	16.66	3013	0	0.311	260	32.23	5	0	0.311	241	16.66	5	0	0.311	1306	81.79	5

Wei Shih 1	30	5	5	0	1408	1.75	327	3	0	96	1.27	1	3	0	80	0.84	1	15	0	842	9.48	1
Wei Shih 2	30	5	5	0	983	1.26	419	5	0	68	0.75	1	4	0	68	0.64	1	18	0	686	6.90	1
Wei Shih 3	30	5	12	0	1559	1.92	305	12	0	81	1.10	1	11	0	83	0.84	1	20	0	636	6.72	1
Wei Shih 4	30	5	13	0	726	0.94	142	20	0	64	0.91	1	20	0	64	0.69	1	20	0	127	1.43	1
Wei Shih 5	30	5	11	0	713	0.93	124	20	0	64	0.90	1	20	0	64	0.69	1	20	0	136	1.56	1
Wei Shih 6	40	5	2	0	2092	2.60	485	1	0	111	1.90	5	1	0	102	1.35	5	4	0	919	12.89	5
Wei Shih 7	40	5	5	0	1903	2.38	643	4	0	90	1.54	9	4	0	100	1.40	9	20	0	609	9.19	9
Wei Shih 8	40	5	1	0	2115	2.61	526	0	0.196	92	1.43	5	0	0.196	85	1.08	5	0	0.196	442	6.20	5
Wei Shih 9	40	5	4	0	1734	2.30	423	17	0	90	1.91	1	15	0	85	1.31	1	20	0	257	4.24	1
Wei Shih 10	50	5	5	0	2308	3.15	756	2	0	124	3.48	1	2	0	118	2.32	1	2	0	475	9.36	1
Wei Shih 11	50	5	3	0	1690	2.37	673	9	0	130	4.32	7	9	0	130	2.72	7	20	0	302	6.26	7
Wei Shih 12	50	5	5	0	1968	2.73	662	2	0	115	3.23	1	2	0	109	2.20	1	2	0	322	6.82	1
Wei Shih 13	50	5	9	0	2056	2.87	944	8	0	112	3.23	5	8	0	109	2.17	5	15	0	288	6.02	5
Wei Shih 14	60	5	9	0	2898	4.00	1108	18	0	131	5.05	6	18	0	127	3.40	6	20	0	411	11.58	6
Wei Shih 15	60	5	6	0	2461	3.26	1322	15	0	178	6.89	6	18	0	189	4.51	6	20	0	498	12.11	6
Wei Shih 16	60	5	0	0.027	3928	5.08	1667	1	0	127	4.35	10	1	0	127	3.11	10	20	0	574	14.86	10
Wei Shih 17	60	5	5	0	4745	6.27	829	20	0	127	2.47	9	20	0	127	2.21	9	20	0	303	5.92	9
Wei Shih 18	70	5	6	0	3987	5.19	1447	13	0	169	5.57	6	14	0	174	4.40	6	20	0	1045	28.73	6
Wei Shih 19	70	5	1	0	4036	5.51	1125	9	0	152	8.03	6	9	0	152	5.23	6	19	0	549	19.42	6
Wei Shih 20	70	5	0	0.095	3636	4.76	1416	10	0	174	6.62	10	10	0	169	4.48	10	20	0	658	16.92	10
Wei Shih 21	70	5	0	0.430	7420	7.99	1068	1	0	148	6.06	7	1	0	148	4.28	7	1	0	471	14.26	7
Wei Shih 22	80	5	0	0.727	4661	6.47	2462	0	0.599	174	10.31	9	0	0.559	169	6.87	9	19	0	723	30.51	178
Wei Shih 23	80	5	0	0.036	4799	6.78	1764	3	0	189	12.71	8	3	0	169	7.18	8	4	0	379	16.63	8
Wei Shih 24	80	5	0	0.157	5007	6.62	2330	0	0.049	266	11.20	92	0	0.049	273	7.72	81	0	0.049	1411	44.52	81
Wei Shih 25	80	5	0	0.221	5392	7.02	1904	0	0.030	194	9.29	10	0	0.030	179	5.87	10	0	0.030	939	33.10	10
Wei Shih 26	90	5	0	0.543	5777	8.15	4766	0	0.334	224	16.92	11	0	0.334	201	9.68	11	3	0	560	28.72	197
Wei Shih 27	90	5	0	0.998	5441	7.87	2192	2	0	240	18.19	10	2	0	201	9.51	10	19	0	1885	91.16	10
Wei Shih 28	90	5	0	0.917	5491	7.76	1870	3	0	212	16.21	4	3	0	195	9.67	4	18	0	564	28.69	4
Wei Shih 29	90	5	0	0.978	6158	8.56	1313	4	0	218	17.11	1	4	0	195	9.95	1	12	0	611	32.55	1
Wei Shih 30	90	5	0	0.268	5754	8.03	2802	11	0	234	11.70	14	7	0	212	7.43	14	20	0	828	29.54	14

Table 3 gives a summary of our results. Considering the number of problems solved to optimality, all tabu search strategies clearly outperform simulated annealing. Referring to the best found solution, the average deviation from the optimal objective function value shows differences worth noting. Calculating the average deviation over all problems and all 20 initial solutions (for each problem), tabu search with REM and STM performs less effectively than the other two tabu search procedures, and behaves similarly to simulated annealing as far as average deviations are concerned (in contrast to best deviations). REM with L+STM gives improved solutions with higher CPU-times. The most astonishing entries in table 3 are the average number of moves needed to find the best solution, again referring to the most successful out of the sample of 20 trials.

Table 3

Summarized numerical results.

	Simulated annealing	Tabu search basic REM	Tabu search REM/STM	Tabu search REM/L+STM
Number of optimal solutions found				
with respect to all 57 test problems	31	40	39	44
referring to all instances (i.e. from $20 \cdot 57 = 1140$)	148	283	268	591
Average deviation from optimality of best solutions found				
with respect to all 57 test problems	0.328	0.126	0.130	0.101
referring to all 1140 instances	4.043	3.483	4.207	0.558
Average number of neighbourhood exchanges needed				
with respect to the best found solution over all 57 test problems	1077	11	9	28
Average CPU-time				
referring to all 1140 instances	5.12	4.85	2.99	14.59

Both tabu search methods, REM in its basic form as well as incorporating STM, behave in nearly the same way. Concerning solution quality, the inclusion of STM into REM gives only slightly worse results but with a remarkable decrease in computation times. REM with L+STM leads to improved results, but it needs at least twice as much CPU-time as REM with STM. If REM is applied with long term memory only instead of L+STM the solution quality is slightly affected in some test problems with mostly increased CPU-times.

To conclude our numerical investigation some results will be reported that cannot be drawn from tables 1–3. The difference in CPU-times when using *Least* as described above (with maximum length of 2) or omitting that vector is less than

or equal to 5%. Other values for the maximum length of *Least* did not yield a further reduction in CPU-times. The size of *at_n* should not be chosen too small. Therefore, it should depend on the length of the data of the underlying problem instance. Usually values of *at_n* larger than $4n$ did not give considerable changes of the results while increasing the computation time. Some reasonable exceptions are summarized in table 4 (each time over a sample of 4 instances) where REM is applied with no restriction on *at_n*. For ease of implementation our results were obtained by setting the complement of a solo attribute tabu only if it is included in the current move. This is reasonable since our proposals for a corrected version for terminating the backtracing did not give improvements in running times.

Table 4

Numerical results for tabu search with basic REM.

	opt	*dev*	*iter*	CPU	*it**
Fleisher	4	0	4001	568.9	232
Freville/Plateau 5	4	0	4001	568.7	425
Petersen 5	4	0	5601	869.5	172
Wei Shih 8	4	0	8001	1296.6	761
Wei Shih 25	4	0	16001	2942.7	265

4. Conclusions

In this paper we have studied implementations of tabu search with a specific dynamic tabu list management, and have disclosed its superiority to an implementation of simulated annealing, designed to perform effectively on the multiconstraint knapsack problem. Our work is based on a limited number of neighbourhood search strategies as well as starting solutions. Considering our outcomes in conjunction with those of additional studies using alternative strategies (see e.g. Domschke et al. [3] for a quadratic optimization problem as well as Dammeyer and Voß [2] for MCKP) the following conclusions may be drawn.

Tabu search using REM seems to be more robust with respect to varying starting solutions than simulated annealing. While using simulated annealing, a "good" choice of the control parameters (temperature, annealing schedule, etc.) greatly influences the solution quality. By contrast, REM in its basic version does not require such sensitive parameters. Even the refinements of REM given in this paper may only influence its computation time. However, the average solution quality remains about the same for all approaches of section 2 provided the number of attributes in the running list is not chosen too small.

Theoretically, one drawback of tabu search is that we may get stuck in a feasible solution. If REM applies a tabu status that forbids all immediately accessible moves, i.e. the residual cancellation sequences may prohibit all possible neighbourhood

solutions, although some regions of the solution space might not yet have been investigated, additional criteria have to be invoked. With respect to the problem dealt with in this paper, however, this was not necessary.

Additional versions of tabu search for MCKP still have to be investigated (cf. [2]). That may also include the idea of what we call *tabu tunneling*, i.e. a method that allows for infeasibilities throughout the neighbourhood search, as in tabu search strategies described in [2] as well as in [11, 14, 15].

References

[1] F. Dammeyer, P. Forst and S. Voß, On the cancellation sequence method of tabu search, ORSA J. Comput. 3(1991)262–265.

[2] F. Dammeyer and S. Voß, Application of tabu search strategies for solving multiconstraint zero–one knapsack problems, Working Paper, TH Darmstadt (1991).

[3] W. Domschke, P. Forst and S. Voß, Tabu search techniques for the quadratic semi-assignment problem, in: *New Directions for Operations Research in Manufacturing*, ed. G. Fandel, T. Gulledge and A. Jones (Springer, Berlin, 1992) pp. 389–405.

[4] A. Drexl, A simulated annealing approach to the multiconstraint zero–one knapsack problem, Computing 40(1988)1–8.

[5] A. Freville and G. Plateau, Méthodes heuristiques performantes pour les problèmes en variables 0–1 à plusieurs constraintes en inégalité, Publication ANO-91, Université des Sciences et Techniques de Lille (1982).

[6] A. Freville and G. Plateau, Hard 0–1 multiknapsack test problems for size reduction methods, Investigacion Operativa 1(1990)251–270.

[7] A. Freville and G. Plateau, An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem, *Conf. on Viewpoints on Optimization*, Grimentz (1990).

[8] B. Gavish and H. Pirkul, Efficient algorithms for solving multiconstraint zero–one knapsack problems to optimality, Math. Progr. 31(1985)78–105.

[9] F. Glover, Tabu search. Part I, ORSA J. Comput. 1(1989)190–206.

[10] F. Glover, Tabu search. Part II, ORSA J. Comput. 2(1990)4–32.

[11] F. Glover, E. Taillard and D. de Werra, A user's guide to tabu search, Working Paper, University of Colorado and EPFL Lausanne (1991), to appear.

[12] A. Hertz and D. de Werra, The tabu search metaheuristic: how we used it, Ann. Math. Art. Int. 1(1990)111–121.

[13] A. Hertz and D. de Werra, Tabu search techniques: a tutorial and an application to neural networks, OR Spektrum 11(1989)131–141.

[14] J.P. Kelly, B.L. Golden and A. Assad, Large-scale controlled rounding using tabu search with strategic oscillation, Working Paper, University of Colorado and University of Maryland (1992), to appear.

[15] M. Laguna, J.P. Kelly, J.L. Gonzalez-Velarde and F. Glover, Tabu search for the multilevel generalized assignment problem, Working Paper, University of Colorado (1991).

[16] R. Loulou and E. Michaelides, New greedy-like heuristics for the multidimensional 0–1 knapsack problem, Oper. Res. 27(1979)1101–1114.

[17] S. Senju and Y. Toyoda, An approach to linear programming with 0–1 variables, Manag. Sci. 15(1968) B196–B207.

[18] H.M. Weingartner and D.N. Ness, Methods for the solution of the multi-dimensional 0/1 knapsack problem, Oper. Res. 15(1967)83–103.