Contents lists available at ScienceDirect

EURO Journal on Computational Optimization

www.elsevier.com/locate/ejco

Exponential extrapolation memory for tabu search

Håkon Bentsen, Arild Hoff, Lars Magnus Hvattum [*]

Faculty of Logistics, Molde University College, Norway

ARTICLE INFO

Keywords:
0-1 Integer programming
Binary optimization
Adaptive memory
Tabu search

ABSTRACT

Tabu search is a well-established metaheuristic framework for solving hard combinatorial optimization problems. At its core, the method uses different forms of memory to guide a local search through the solution space so as to identify high-quality local optima while avoiding getting stuck in the vicinity of any particular local optimum. This paper examines characteristics of moves that can be exploited to make good decisions about steps that lead away from recently visited local optima and towards a new local optimum. Our approach uses a new type of adaptive memory based on a construction called exponential extrapolation. The memory operates by means of threshold inequalities that ensure selected moves will not lead to a specified number of most recently encountered local optima. Computational experiments on a set of one hundred different benchmark instances for the binary integer programming problem suggest that exponential extrapolation is a useful type of memory to incorporate into a tabu search.

* Corresponding author.
E-mail addresses: hakon.bentsen@himolde.no (H. Bentsen), arild.hoff@himolde.no (A. Hoff), hvattum@himolde.no (L.M. Hvattum).

https://doi.org/10.1016/j.ejco.2022.100028

1. Introduction

Tabu search is a metaheuristic framework associated with the use of memory to guide a search consisting of iteratively making small changes to a current solution. Glover and Laguna [11] referred to four dimensions relevant to memory structures in tabu search: recency, frequency, quality, and influence. The two first dimensions complement each other, in that they roughly correspond to short-term and long-term memory. The quality dimension is essential, since one would like to learn how to reach good solutions while avoiding bad solutions. Thus, actions leading to good solutions should be reinforced, and actions leading to bad solutions should be discouraged. Finally, the dimension of influence considers how actions change the structure of solution elements.

Laguna and Glover [16] used target analysis to evaluate a tabu search for a class of sequencing problems. The target analysis works by identifying high quality solutions for given test instances, and then evaluating the moves made during the search when applying alternative decision rules. The authors discovered that improving moves were more likely to introduce attributes of optimal solutions than non-improving moves. That is, moves made when approaching a local optimum were more likely to make the current solution more similar to optimal solutions, whereas moves made when retreating from a local optimum were not. Further details of target analysis were provided by Glover and Laguna [11].

We are motivated by the study of Laguna and Glover [16] to change the rules customarily used in tabu search to select the next move. In doing so we evaluate a form of adaptive memory based on a function called exponential extrapolation, introduced by Glover [10], which makes it possible to track the number of times that variables receive their current values in any selected number of most recent local optima. When selecting non-improving moves, the idea is that the tabu search should select among those moves that lead away from recently encountered local optima, thus inducing the search to diversify and seek other promising solutions. This is achieved by using the new form of memory and enforcing a threshold value for moves, such that the allowed moves must be among those that best avoid moving towards recent local optima.

Talbi [20] presented a summary of different types of memory that have been used in tabu search. Short-term memory is used to store the recent history of the search. The main example is the use of tabu lists representing information about recently visited solutions or recently performed moves, with the primary function to prevent the search from cycling. Medium-term memory goes beyond the most recent search history, while not encompassing the entire search. One example of such memory is the storing of good solutions encountered during the search, so called elite solutions, that can be used to restart the search by creating a new solution consisting of elements derived from the elite solutions. Medium-term memory also encompasses recency-based features, as in storing for each potential component of a solution, the number of iterations since the status of the component has changed. Medium-term memory is primarily used for intensification purposes [20]. Long-term memory is often used to support diversification mechanisms, and includes the use of frequency calculations such as calculating for each component the number of iterations it has been present or the number of times its status has changed throughout the entire search. Exponential extrapolation forms a medium-term memory whose primary function is to diversify the search, or perhaps more accurately to avoid excessive intensification.

To test the exponential extrapolation memory, we present two basic variants of tabu search for solving the general *binary integer programming* (BIP) problem. This problem can be stated mathematically as

$$\max Z = \sum_{j=1}^{n} c_j x_j,$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \qquad i = 1, \ldots, m,$$

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

where all a_{ij}, b_i, and c_j are assumed to be integers (not necessarily positive). The use of only $\leq$-constraints is not a limitation, as equality constraints can be converted into a pair of $\leq$- and $\geq$-constraints, whereas $\geq$-constraints can be transformed into $\leq$-constraints after multiplication by -1.

Once we have presented two basic tabu search variants for the BIP, we show how they can be extended by the use of exponential extrapolation and perform a computational study to evaluate the effectiveness of including the new form of adaptive memory. The computational study is performed using benchmark instances from four different optimization problems that are special cases of the BIP: the *optimum satisfiability problem* (OptSAT) [7], the *multiple-choice multidimensional knapsack problem* (MMKP) [14], the *multidemand multidimensional knapsack problem* (MDMKP) [4], and the *weighted max-cut problem* (MaxCut) [9].

To summarize, the goal of this work is to analyze and evaluate exponential extrapolation, a new form of memory to be used within tabu search. The hypothesis is that adding exponential extrapolation can help the search to avoid recently visited local optima, which should help the search to be more efficient and thus find better solutions within a given computational budget. The remainder of this paper is thus structured as follows. Section 2 contains a description of two basic tabu search variants for the BIP. Then, the use of exponential extrapolation is explained in Section 3. The computational study is presented in Section 4, followed by concluding remarks in Section 5.

2. Tabu search

In the following we describe two basic versions of tabu search for solving general BIPs. They are both based on the same type of neighborhood structure, and both use a standard attribute-based tabu criterion with a new best aspiration criterion. A simple restart mechanism is used to improve diversification. The two versions differ in the move evaluation: one uses a static move evaluation, while the other uses a dynamic move evaluation that induces a strategic oscillation between feasible and infeasible solutions. The latter is based on previous implementations of tabu search as made by Arntzen et al. [1], Cordeau et al. [5], and Hvattum et al. [13]. We refer to the tabu search with a static move evaluation as S-TS, and the tabu search with a dynamic move evaluation as D-TS.

The internal representation of the BIP, based on using ranged rows to represent constraints, is

$$\max Z = \sum_{j=1}^{n} c_j x_j,$$

subject to

$$\underline{b}_i \leq \sum_{j=1}^{n} a_{ij} x_j \leq \overline{b}_i, \qquad i = 1, \ldots, m,$$

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

where $\underline{b}_i = -\infty$ for $\leq$-constraints, $\overline{b}_i = \infty$ for $\geq$-constrains, and $\underline{b}_i = \overline{b}_i$ for equality constraints.

2.1. Neighborhood structure

To search for improvements of a solution we define a simple flip-neighborhood, consisting of changing the value of a single variable from 0 to 1 or from 1 to 0. This implies that the Hamming distance between a solution x and a neighboring solution x' is exactly 1. That is, for a solution x we define the neighborhood $N(x)$ as

$$N(x) = \{x' : \sum_{j=1}^{n} |x_j - x'_j| = 1\}.$$

2.2. Tabu and aspiration criteria

When a move is made and a variable is flipped, the variable becomes tabu for a number of iterations. This is called the tabu tenure, and is drawn at random from a uniform distribution $\{T^{MIN}, T^{MIN}+1, \dots, T^{MAX}\}$, based on two parameters T^{MIN} and T^{MAX}. The tabu status of a variable expires once a number of iterations corresponding to the tabu tenure has been made, but until that happens the search does not allow the variable to be flipped.

As this attribute-based tabu tenure is inexact, it may prevent good solutions from being examined even though they have not been visited before. Therefore, a new best aspiration criterion is also used: the tabu status of a variable is overridden if flipping the variable leads to a better solution than the best solution visited by the search so far.

2.3. Initial solution and restarts

The initial solution of the search is a random solution, where each variable is assigned the value 0 or the value 1 with equal probability. A basic tabu search naturally focuses on intensification. To ensure that the search has a minimum level of diversification, a restart mechanism is employed. The idea is to restart from a new randomly generated solution if the search has performed a large number of iterations without finding a new best solution. This means we introduce another parameter I^R which is the number of iterations without improvements that is allowed before a random restart takes place.

2.4. Static move evaluation

For the static move evaluation, we first define the sum of activities in constraint i as $L_i(x) = \sum_{j=1}^{n} a_{ij}x_j$. Then, we define $\underline{M}(x) = \{i : \underline{b}_i > L_i(x)\}$ as the set of constraints whose lower bound is violated, and $\overline{M}(x) = \{i : \overline{b}_i < L_i(x)\}$ as the set of constraints whose upper bound is violated. To calculate the level of infeasibility, we consider normalized constraints. Let

$$\tilde{a}_i = \frac{\sum_{j=1}^{n} |a_{ij}|}{|\{j : a_{ij} \neq 0\}|}.$$

Then, the normalized constraints become

$$\underline{b}_i/\tilde{a}_i \leq L_i(x)/\tilde{a}_i \leq \overline{b}_i/\tilde{a}_i,\ i = 1, \dots, m.$$

Now, a solution is considered to be better than another either if it has a better level of infeasibility, or it has the same level of infeasibility but a better objective function value. Formally, for a given solution x, the objective function value is $Z(x) = \sum_{j=1}^{n} c_j x_j$. The sum of constraint violations is

$$V(x) = \sum_{i \in \underline{M}(x)} (\underline{b}_i - \sum_{j=1}^{n} a_{ij}x_j)/\tilde{a}_i + \sum_{i \in \overline{M}(x)} (\sum_{j=1}^{n} a_{ij}x_j - \overline{b}_i)/\tilde{a}_i,$$

and the number of violated constraints is $W(x) = |\underline{M}(x)| + |\overline{M}(x)|$. Then, the overall measure of infeasibility is taken as a linear combination of $V(x)$ and $W(x)$, $A(x) = V(x) + \lambda W(x)$, where $A(x) = 0$ implies that $W(x) = 0$ and that the solution is feasible. This means that x is a better solution than x' if either 1) $A(x) < A(x')$, or 2) $A(x) = A(x')$ and $Z(x) > Z(x')$. Bentsen and Hvattum [2] found that a weight of $\lambda = 1$ is suitable, and the same value is used here.

A move can now be evaluated in terms of the change in $Z(x)$ and in $A(x)$. Two moves can be compared by determining which of the resulting solutions are better. Defining $\Delta_j^A(x)$ as the improvement of $A(x)$ when flipping variable j and $\Delta_j^Z(x)$ as the improvement in $Z(x)$ when flipping variable j, we say that a move j is better than another move k if either 1) $\Delta_j^A(x) > \Delta_k^A(x)$, or 2) $\Delta_j^A(x) = \Delta_k^A(x)$ and $\Delta_j^Z(x) > \Delta_k^Z(x)$.

When performing a move, by flipping the value of x_j, the sets $\underline{M}(x)$ and $\overline{M}(x)$ will in general change. To efficiently calculate evaluations for the new neighbors after flipping x_j, stored values of $L_i(x)$ and $L_i(x)/\tilde{a}_i$ must be updated only for constraints i such that $a_{ij} \neq 0$. Furthermore, the evaluation of $\Delta_k^A(x)$, for $k \in \{1, \dots, n\}$, must be updated only for these constraints.

2.5. Dynamic move evaluation

A dynamic move evaluation can encourage the search to move through infeasible intermediate solutions to find better regions of feasible solutions. An additional weight w is introduced to balance the importance of feasibility and the objective function value. The evaluation of a move can then be stated as

$$\Delta_j(x) = \Delta_j^Z(x) + w\Delta_j^A(x),$$

where a move j is better than another move k if $\Delta_j(x) > \Delta_k(x)$. Depending on the value of w, a move j can be considered better than a move k even if it leads to a worse solution.

The weight is initialized based on the largest objective function coefficient. Let $c^{MAX} = \max_{j=1}^n |c_j|$ and $c^{MIN} = \min_{j=1}^n |c_j|$. Then, the initial weight is $w = (c^{MAX} + c^{MIN})/2$. The weight is updated dynamically after each move based on the status of the current solution. If the current solution is feasible, the weight is decreased to put more emphasis on improving the objective function value. That is $w \leftarrow w - w^{DEC}(c^{MAX} + c^{MIN})/2$. If the current solution is infeasible, the weight is increased to put more emphasis on improving the level of infeasibility, setting $w \leftarrow w + w^{INC}(c^{MAX} + c^{MIN})/2$. Thus, compared to the static move evaluation, we need two additional parameters: w^{INC} and w^{DEC}.

3. Exponential extrapolation

The two variants of tabu search described in Section 2 are relatively simple. For S-TS, memory is only used to store the tabu status of variables, to remember when the last update was made to the best found solution, and to remember the best found solution itself. In the D-TS, the only additional type of memory is the weight w, which is a type of short-term memory that indicates how often feasible solutions have been encountered during the most recent iterations. We now present a new type of memory, referred to as exponential extrapolation memory. This was recently introduced by Glover [10] who described the memory as a main component of a new type of metaheuristic framework called alternating ascent. Here, the exponential extrapolation memory is incorporated into the two tabu search variants, resulting in two new variants, which we refer to as S-TS-E and D-TS-E, respectively.

Using the notation from the static move evaluation, we start with the following definition: A solution x is a local optimum if for all $j \in \{1, \dots, n\}$ we have either $\Delta_j^A(x) < 0$ or both $\Delta_j^A(x) = 0$ and $\Delta_j^Z(x) \leq 0$. This definition of a local optimum is also used when referring to the tabu search using a dynamic move evaluation.

3.1. Definition of memory structure

Let the Q most recent local optima be denoted by $x^q = (x_1^q, x_2^q, \dots, x_n^q)$ for $q = 1, \dots, Q$. To measure how frequently and recently local optima have included the assignment $x_j = 1$, we define

$$E_1(j) = \sum_{q=1}^{Q} \alpha^{q-1} x_j^q.$$

Similarly, for $x_j = 0$ we have

$$E_0(j) = \sum_{q=1}^{Q} \alpha^{q-1} (1 - x_j^q).$$

Using $\alpha = 2$ allows the values of $E_1(j)$ and $E_0(j)$ to be stored in a single integer variable that can easily be updated using right shift and addition operators. That is, when the search encounters a new local optimum, the values of $E_1(j)$ and $E_0(j)$ are updated, given that the current solution now becomes the new most recent local optimum, x^Q. This is accomplished efficiently by exploiting that $\alpha = 2$ and by storing the values of $E_1(j)$ and $E_0(j)$ as integers. Then, considering the newest local optimum x^Q, the updates are performed by setting $E_1(j) \leftarrow \lfloor E_1(j)/\alpha \rfloor + \alpha^{Q-1} x_j^Q$ and $E_0(j) \leftarrow \lfloor E_0(j)/\alpha \rfloor + \alpha^{Q-1}(1 - x_j^Q)$.

For short, we define $E(j)$ to measure how frequently and recently local optima have included the same value as the current assignment to x_j:

$$E(j) = E_1(j) x_j + E_0(j)(1 - x_j),$$

and we can also define $\overline{E}(j)$ to measure how frequently and recently local optimal have included the opposite value as the current assignment:

$$\overline{E}(j) = E_0(j)x_j + E_1(j)(1 - x_j).$$

If we define $T(r) = \sum_{q=Q-r}^{Q} \alpha^{q-1}$, and if $\alpha \geq 2$, then it follows that variable x_j has a different value than in the r most recent local optima if and only if $E(j) < T(r)$.

The exponential extrapolation values represent the Q most recently visited local optima, thus forming a medium-term memory structure. The calculations can be illustrated by the following simplified example. If we take $Q = 4$ (larger values would be used in practice), use $\alpha = 2$, consider variable j, and the four most recent local optima had values $x_j^1 = 1$, $x_j^2 = 1$, $x_j^3 = 0$, and $x_j^4 = 1$, we would get

$$E_1(j) = \alpha^0 x_j^1 + \alpha^1 x_j^2 + \alpha^2 x_j^3 + \alpha^3 x_j^4 = 1 + 2 + 0 + 8 = 11,$$

while similarly, $E_0(j) = 4$. If the next local optimum has $x_j = 0$, the new situation would be characterized by $x_j^1 = 1$, $x_j^2 = 0$, $x_j^3 = 1$, and $x_j^4 = 0$. Then we could calculate $E_1(j)$ and $E_0(j)$ from scratch, or use the simplified calculations $E_1(j) = \lfloor 11/2 \rfloor + 0 = 5$ and $E_0(j) = \lfloor 4/2 \rfloor + 8 = 10$.

3.2. Initialization

The initial solution of the search is generated randomly, with each x_j being assigned a value of 1 or 0 with equal probability. At this point no previous local optima have been visited, and hence the values of $E_1(j)$ and $E_0(j)$ are simply initialized as if the previous Q local optima are identical to the random initial solution.

3.3. Modified move selection

The purpose of using exponential extrapolation is to modify the move selection, so as to improve the search. Tabu search is inherently greedy, in that in each iteration, the best move is performed subject only to the tabu criterion. We differentiate between improving and non-improving moves: For the static move evaluation, an improving move involves flipping x_j such that either 1) $\Delta_j^A(x) > 0$, or 2) $\Delta_j^A(x) = 0$ and $\Delta_j^Z(x) > 0$. For the dynamic move evaluation, a move is considered to be improving if $\Delta_j(x) > 0$.

When applying exponential extrapolation, we attempt to add a filter that affects only non-improving moves, since these are the ones that were observed to be less effective in the study by Laguna and Glover [16]. That is, all improving moves are allowed, but some non-improving moves are removed from consideration based on a filter that is calculated using the values of $E(j)$.

We first calculate the maximum, minimum, and average values of $E(j)$, considering all variables j except those that are tabu. Refer to these values as E^{MAX}, E^{MIN}, and E^{AVG}. A cut-off value for $E(j)$ is then calculated based on a parameter $\beta \in (0, 1]$:

$$E^{CUT} = \begin{cases} E^{AVG} + 2(\beta - 0.5)(E^{MAX} - E^{AVG}) & \text{if } \beta \geq 0.5, \\ E^{MIN} + 2\beta(E^{AVG} - E^{MIN}) & \text{if } \beta < 0.5. \end{cases}$$

In this formula, β is used to approximate the $(100\beta)^{th}$ percentile of the $E(j)$ values. To decide which move to execute, the search considers all variables j such that $E(j) \geq E^{CUT}$ and such that j is not tabu, and then selects the best move satisfying these conditions according to the move evaluation used.

The consequence of this filter is that when making non-improving moves, the search avoids those moves that lead back to variable values most similar to those encountered in the most recent local optimum. This should lead to an increased level of diversification in the search.

3.4. Additional remarks

There are now four variants of tabu search to consider: S-TS, S-TS-E, D-TS, and D-TS-E. To analyze the behavior of these variants, in some runs we gather information about the number of local optima visited, and the number of revisited local optima. To approximately detect revisited local optima, we adopt a strategy used in solution-based tabu search, as implemented by Lai et al. [17].

Consider a number of hash functions that each map the solution space to $\{0, 1, \ldots, L-1\}$. This is accomplished by defining weights $\omega_{ju} = \lfloor j^{\gamma_u} \rfloor$ for each combination of variable j and hash function u, based on a function-specific parameter γ_u. Then, for a solution x, hash values are calculated as

$$h_u(x) = \left(\sum_{j=1}^{n} \omega_{ju} x_j \right) \mod L.$$

Now, associate with each hash function h_u an array H_u of L binary values initialized as zeros. Once a local optimum has been reached, H_u is updated by setting $H_u(h_u(x)) = 1$. Let there be U hash functions in total. Before these are updated, if at least one of $H_1(h_1(x)), \ldots, H_U(h_U(x))$ has the value 0, one can know with certainty that the local optimum x has not been previously visited. On the other hand, if all of $H_u(h_u(x)) = 1$, it is very likely that the local optimum has been visited before, although not guaranteed. In our implementation, we use $L = 10^8$, $U = 3$, with $\gamma_1 = 1.9$, $\gamma_2 = 2.1$, and $\gamma_3 = 2.3$, as proposed by Lai et al. [17].

4. Computational study

Computational tests are carried out using a standard PC with a 64 bit 3.5 GHz i5-4690 CPU and 16 GB RAM, running Windows 10. In the following, we describe the test instances used, the results of parameter tuning, and then the results, focusing first on search performance and then on search behavior.

Table 1
Characteristics of instances in the training set, averaged per instance type.

	n	m	non-zeros	density
OptSat	440	1,600	37,000	0.0500
MMKP	1,475	121	18,156	0.2197
MDMKP	290	25	6,420	1.0000
MaxCut	12,724	22,887	68,662	0.0005

4.1. Instances

Many different optimization problems can be formulated as a BIP. Appendix A defines the four different optimization problems that are considered in this work: OptSAT, MMKP, MDMKP, and MaxCut. A total of 120 selected instances are considered, split into a training set and a test set. The training set consists of five instances from each problem class, for a total of 20 instances, while the test set consists of 25 instances from each problem class. The instances have all been used in the existing literature, and we refer to Davoine et al. [7] and da Silva et al. [6] for OptSAT instances, to Helmberg and Rendl [12] and Martí et al. [18] for MaxCut instances, to Cappanera and Trubian [4] for MDMKP instances, and to Khan et al. [15] and Shojaei et al. [19] for instances of the MMKP. Table 1 provides selected characteristics of the 20 instances in the training set.

4.2. Parameter tuning

Parameters were tuned for each of the four tabu search variants separately, using the 20 instances from the training set. The tuning was performed manually, by iteratively fixing some parameters and varying the others. Instances were solved for five minutes, and the solution quality at the end of the runs were used to evaluate which parameter settings worked better.

For the S-TS we ended up with the parameters $T^{MIN} = 10$, $T^{MAX} = 40$, and $I^R = 40,000$. In the D-TS, the dynamic changes of the move evaluation helps to diversify the search and the method can more easily escape local optima. This is reflected in the final parameter settings, where we have $T^{MIN} = 7$, $T^{MAX} = 32$, and $I^R = 400,000$. That is, smaller tabu tenures and longer periods between restarts. The parameters specific for the dynamic move evaluation were set to $w^{INC} = 10^{-6}$ and $w^{DEC} = 0.0125$.

In the variants with exponential extrapolation, we used $\alpha = 2$ without tuning. We also set $Q = 50$, which is close to as high as possible without needing special care to avoid overflow in calculations of E^{CUT}. We expected to see that the addition of exponential extrapolation would also benefit diversification in general, allowing the search to use shorter tabu tenures or longer restart intervals in compensation. For S-TS-E we found the best settings to be $T^{MIN} = 5$, $T^{MAX} = 25$, and $I^R = 20,000$, while using $\beta = 0.25$. For D-TS-E the parameters ended up as $T^{MIN} = 7$, $T^{MAX} = 22$, $I^R = 400,000$, and $\beta = 0.2$.

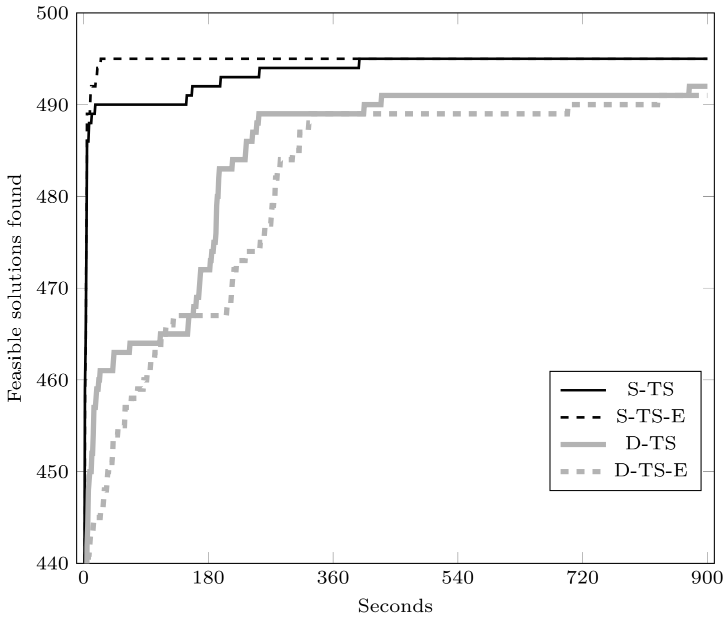

Fig. 1. Number of feasible solutions found as a function of time.

4.3. Search performance

The main comparison of the four variants of tabu search is based on runs for the test set. Each tabu search variant is executed on each of the 100 instances using a time limit of 15 minutes. Given that the starting solution is random, and that tabu tenures are randomly drawn, each instance is solved five times with different random seeds.

The performance of the methods is first analyzed by looking at their ability to find feasible solutions. Fig. 1 shows the development of the number of runs where feasible solutions have been found as a function of running time. The best method with respect to finding feasible solutions quickly is S-TS-E, which after 25 seconds has found feasible solutions in 495 out of 500 runs. The tabu search with a static move evaluation function and not using exponential extrapolation memory also finds feasible solutions in 495 runs, but only after 398 seconds. Variants with a dynamic move evaluation are not as good at finding feasible solutions, and after 900 seconds, D-TS has found feasible solutions in 492 runs whereas D-TS-E has found feasible solutions in 491 runs. There is one MMKP instance (INST24) where none of the methods find any feasible solutions in any of the five runs.

In terms of evaluating the effect of adding the exponential extrapolation memory, the results regarding feasibility are mixed. The tabu search with a static move evaluation is improved, and the version with exponential extrapolation has the best performance in terms of finding feasible solutions. However, for the variants with a dynamic move evaluation, it may seem that adding the new memory makes it harder to find feasible solutions quickly.

The performance of the methods is next evaluated using the final primal gaps and the primal integral of Berthold [3]. Consider a solution x with the associated objective function value Z, and assume that an optimal or best known solution is given with the objective function value Z^{OPT}. The primal gap of the solution is then defined as

$$\gamma(x) = \begin{cases} 0, & \text{if } |Z^{OPT}| = |Z| = 0, \\ 1, & \text{if } Z^{OPT} Z < 0, \\ |Z^{OPT} - Z| / \max\{|Z^{OPT}|, |Z|\}, & \text{otherwise.} \end{cases}$$

A primal gap function $p(t)$ is now defined on $[0, T]$, where T is the running time of the solution method, such that

$$p(t) = \begin{cases} 1, & \text{if no feasible solution has been found at time } t, \\ \gamma(x(t)), & \text{otherwise, with } x(t) \text{ being the best solution found at time } t. \end{cases}$$

The function $p(t)$ measures the quality of the solution found over time for a given run. The primal integral condenses this information into a single number, by taking the integral of the primal gap function. In particular, let $\{t_1 = 0, t_2, \ldots, t_\tau = T\}$ be a set containing the points in time that are of interest, i.e., the starting point, any time when a new best solution is found, and the final time of the run. The primal integral is then defined as

$$P(T) = \sum_{s=1}^{\tau} p(t_{s-1}) \cdot (t_s - t_{s-1}).$$

When considering many runs for a given solution method, the average value of the primal integrals provides an indication of the performance of the solution method, with lower values being better. In the following tests, we take the best found solutions within the tests to obtain a proxy for Z^{OPT}. This means that the primal gaps and primal integrals presented are not relative to the optimal solutions, but rather to the best solutions found by the methods compared. Furthermore, we divide the primal integrals by the total running time, resulting in a number between 0 and 1 representing the average solution quality during the search process.

Fig. 2 shows the primal gaps as a function of running time. The value of using exponential extrapolation is clear in the case of using a static move evaluation function: S-TS-E has smaller gaps than S-TS from a very early stage, and remains lower for the duration of the runs. When using a dynamic move evaluation, the results are in general much better than when using a static move evaluation. This is in contrast to the results for feasibility, where the static move evaluation performed better. Comparing D-TS and D-TS-E, the lead goes back and forth during the first five minutes of the runs. Then,

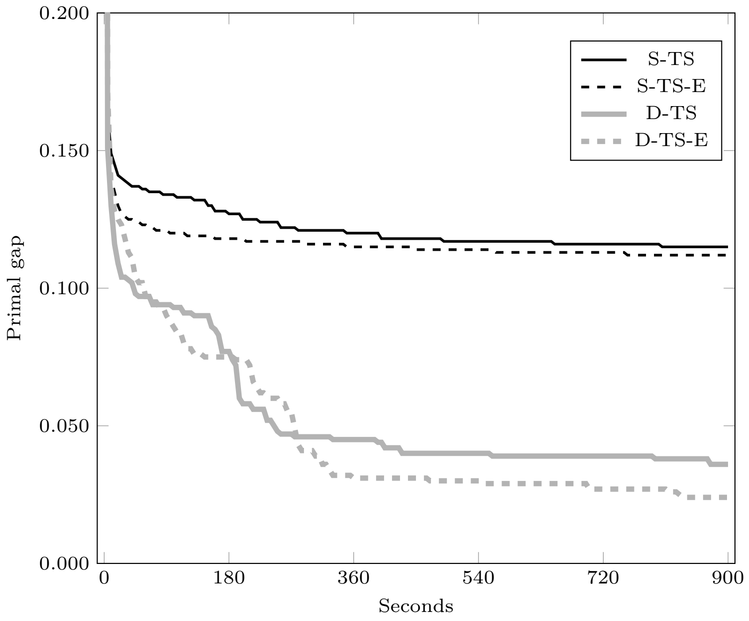

Fig. 2. Primal gaps as a function of time.

Table 2
Primal gaps after 15 minutes for each type of instance separately and in total.

	S-TS	S-TS-E	D-TS	D-TS-E
OptSat	0.029	0.025	**0.002**	0.002
MMKP	0.179	0.175	0.084	**0.082**
MDMKP	0.046	0.040	0.006	**0.005**
MaxCut	0.206	0.207	0.054	**0.009**
All	0.115	0.112	0.036	**0.024**

from 280 seconds and onwards, the variant with exponential extrapolation retains the best primal gap.

Looking in more detail at the final average primal gaps after 15 minutes of running time provides further support for the value of adding exponential extrapolation. Table 2 shows the final average gaps, and all the best results belong to variants with the dynamic move evaluation. For most types of instances, and overall, the variants with exponential extrapolation memory perform better than the corresponding variant without exponential extrapolation.

Table 3 shows the variability of performance of different methods on different sets of instances. This is done through two measures. First, we consider the standard deviation of the final primal gap across all runs of a given method. This shows how much the performance varies overall between runs, in the columns labeled "Overall". Second, we calculate the standard deviations of final primal gaps by each method for each instances

Table 3
Standard deviations for the final primal gaps, considering both variations between all runs ("Overall") and variations within runs for each instance individually ("Instance").

	S-TS		S-TS-E		D-TS		D-TS-E	
	Overall	Instance	Overall	Instance	Overall	Instance	Overall	Instance
OptSat	0.045	0.002	0.043	0.002	0.004	0.001	0.004	0.001
MMKP	0.209	0.004	0.211	0.004	0.243	0.024	0.257	0.023
MDMKP	0.029	0.005	0.026	0.003	0.011	0.003	0.009	0.002
MaxCut	0.152	0.005	0.151	0.007	0.068	0.004	0.027	0.004
All	0.153	0.004	0.154	0.004	0.131	0.008	0.133	0.007

individually, and then we take the average value of this across instances. This indicates how much the performance of a given method varies between runs on the same instance, and is presented in the columns labeled "Instance". The standard deviations for runs on the same instance is in general small, with one outlier when it comes to D-TS and D-TS-E on MMKP instances. Standard deviations across instances vary more, indicating that some problem types are harder to solve than others. The performances of D-TS and D-TS-E are in general more stable than for S-TS and S-TS-E, except when solving the MMKP instances. Adding exponential extrapolation memory does not seem to influence the performance stability to any large extent, except that D-TS-E is more stable than D-TS on MaxCut instances.

The above discussion indicates that the final primal gaps are, overall, improved when adding exponential extrapolation. Next, we analyze whether S-TS-E and D-TS-E are more likely than the corresponding methods without exponential extrapolation to produce the best result on a given run. To this end we perform pair-wise comparisons between the 500 final primal gaps, and Table 4 summarizes how many times the method with exponential extrapolation has the better gap (columns "Win"), the same gap (columns "Draw"), and the worse gap (columns "Loss"). To test whether the differences are statistically significant, we use a non-parametric sign-test [8] with the null-hypothesis that the methods compared are equally likely to produce the better final primal gap in a given run. This avoids the need to deal with assumptions regarding independence, normality, and homoscedasticity, that are commonly made in parametric tests.

Comparing S-TS-E and S-TS, the former has 319 runs with a better gap, 159 runs with a worse gap, and 22 runs with identical gaps. This difference is statistically significant with a P-value less than 10^{-12}. Comparing D-TS-E and D-TS, the former has 204 runs with better gaps, 174 with worse gaps, and 122 with equal gaps. The difference here is not statistically different: the P-value is 0.195 if assuming that draws are supporting a null-hypothesis that the methods are equally likely to produce the best gap and 0.136 if instead draws are removed from consideration.

The sign-tests ignore the magnitude of the differences between methods. This means that the difference between S-TS-E and S-TS is statistically significantly, even if the difference in the average primal gaps is not very large, as seen in Fig. 2. On the other hand, the difference between D-TS-E and D-TS is not statistically significant, even though the

Table 4
Pair-wise comparisons of final gaps with and without exponential extrapolation.

	S-TS-E vs. S-TS				D-TS-E vs. D-TS			
	Win	Draw	Loss	P-Value	Win	Draw	Loss	P-Value
OptSat	95	11	19	0.000	24	69	32	0.530
MMKP	68	5	52	0.178	37	19	69	0.005
MDMKP	97	0	28	0.000	69	16	40	0.012
MaxCut	59	6	60	1.000	74	18	33	0.000
All	319	22	159	0.000	204	122	174	0.195

Table 5
Primal integrals calculated for each type of instance separately and in total, with the best values highlighted in bold.

	S-TS	S-TS-E	D-TS	D-TS-E
OptSat	0.032	0.028	**0.004**	0.005
MMKP	0.184	0.181	**0.141**	0.164
MDMKP	0.061	0.045	0.012	**0.008**
MaxCut	0.214	0.215	0.061	**0.015**
All	0.123	0.117	0.055	**0.048**

difference in average final primal gaps is larger. This can be explained by the fact that relative results differ between types of instances, as observed from Table 2. For both MDMKP and MaxCut-instances, the final primal gap of D-TS-E is significantly better than the gap of D-TS, with P-values of 0.01 and 0.0003, respectively. However, for MMKP-instances, the final gaps of D-TS are considered to be better than for D-TS-E, with a P-value of 0.005, despite the fact that the average gap is better for D-TS-E than for D-TS as seen in Table 2.

The overall primal integrals scaled by total running time are shown in Table 5, which also shows primal integrals calculated for each type of instance. For MMKP-instances and OptSAT-instances the D-TS has the best primal integral, but for MDMKP, MaxCut, and in total, D-TS-E has the best performance. Overall, the results indicate that in terms of providing high-quality solutions, the dynamic move evaluation is better than the static move evaluation, and in both cases adding exponential extrapolation has a positive effect. However, given that D-TS has a better primal integral than D-TS-E on two problem classes, the effect of adding exponential extrapolation is not as convincing for the tabu search with a dynamic move evaluation as it is for the tabu search with a static move evaluation.

4.4. Search behavior

Having established that taking into account exponential extrapolation memory has a potentially positive effect within a tabu search, some analysis was performed to see how the use of exponential extrapolation changes the search behavior. To this end we consider the instances in the training set, which are run for five minutes each while

Table 6
Average number of iterations between local optima, evaluated on the training set.

	S-TS	S-TS-E	D-TS	D-TS-E
OptSAT	250.9	83.1	18,228.7	18,964.9
MMKP	8.2	4.2	14.6	11.7
MDMKP	2.6	2.5	19.2	17.6
MaxCut	1,916.8	885.0	12,500.7	12,525.7

Table 7
Proportion of revisited local optima, evaluated on the training set.

	S-TS	S-TS-E	D-TS	D-TS-E
OptSAT	0.006	0.018	0.037	0.020
MMKP	0.799	0.882	0.676	0.731
MDMKP	0.777	0.776	0.538	0.550
MaxCut	0.013	0.133	0.112	0.172

gathering some additional statistics regarding visits to local optima. In particular, we used the hash functions described in Section 3.4 to determine whether each local optimum had been previously visited during the search. Recall that a solution x is considered as a local optimum if for all $j \in \{1, \dots, n\}$ we have either $\Delta_j^A(x) < 0$ or both $\Delta_j^A(x) = 0$ and $\Delta_j^Z(x) \leq 0$.

Since instances of the four optimization problems considered are quite different in structure, we present results separately for each type. Table 6 shows, for each type of problem and each variant of tabu search, the average number of iterations between visiting a local optimum. The most striking difference is between the methods with a static move evaluation and methods with a dynamic move evaluation. For the former, the distance between local optima is much smaller. A possible explanation is that the dynamic move evaluation does not necessarily favor moves that lead towards a local optimum, but may in fact prefer moves that make the current solution worse in terms of feasibility.

There is also a big difference between the four types of optimization problems, with MMKP and MDMKP leading to short distances between local optima, and OptSAT and MaxCut having much longer distances. However, the effect of adding exponential extrapolation is less clear: S-TS-E seems to have fewer iterations between local optima than S-TS, whereas D-TS-E has either fewer and more iterations compared to D-TS, depending on the type of problem solved.

Table 7 shows the proportion of revisited local optima, or in other words the ratio of total minus unique local optima to the total number of local optima. For most situations, the variants with exponential extrapolation has a higher proportion of revisited local optima. This may seem counter-intuitive, as we expected the mechanism to favor more diversification and thus fewer revisited local optima. However, often the combinations with exponential extrapolation also have fewer iterations between local optima, and are thus visiting more local optima in total. Thus, even though the number of revisited

Table 8
Average number of unique local optima visited, evaluated on the training set.

	S-TS	S-TS-E	D-TS	D-TS-E
OptSat	318,324	941,116	4,218	3,909
MMKP	1,050,681	1,132,623	934,607	936,862
MDMKP	7,407,072	7,279,117	2,580,630	2,536,445
MaxCut	13,867	17,273	1,730	1,105

Table 9
Final gaps after 15 minutes using original and D-TS-E parameters.

	Original	D-TS-E
S-TS	0.116	0.120
S-TS-E	0.112	0.118
D-TS	0.037	0.041
D-TS-E	0.025	—

local optima is higher, in some cases the number of unique local optima visited is also higher. Table 8 shows the average number of unique local optima visited for the different combinations of problem classes and methods.

Summarizing the results on search behavior, we can say that there is a large difference in behavior between different types of instances. Referring to the characteristics of instances in the training set described in Table 1, it appears that the density of constraint coefficients is an important factor for the number of iterations between local optima and the proportion of revisited local optima. Furthermore, there is a significant difference in the behavior of the search depending on whether a static or a dynamic move evaluation is used. Although it is clear that adding exponential extrapolation memory also changes the search, the nature of the change is not uniform across methods or instance types.

4.5. Parameter sensitivity

While the main results revealed some differences in the performance of the four methods investigated, it must be acknowledged that the four methods also had different parameter settings. In an attempt to highlight the effect of the parameter settings, as opposed to the effects of the different search components used, an additional experiment was conducted. Here, we ran each of the three methods S-TS, S-TS-E, and D-TS using the parameter settings from D-TS-E.

Table 9 shows the final primal gaps and Table 10 shows the primal integrals for both the original parameters settings, as described in Section 4.2, and for the parameters settings mimicking D-TS-E. Results are obtained by five runs on each of the 100 instances in the test set. As can be seen from the tables, the original parameter settings perform better for all the methods, and we conclude that the good performance of D-TS-E is so due to the particular combination of search components used, rather than the particular parameter settings applied.

Table 10
Primal gaps after 15 minutes using original and D-TS-E parameters.

	Original	D-TS-E
S-TS	0.123	0.126
S-TS-E	0.118	0.124
D-TS	0.055	0.058
D-TS-E	0.048	—

5. Concluding remarks

The underlying idea of the metaheuristic called tabu search is to use different forms of adaptive memory to guide a search trajectory with the goal of finding near-optimal solutions to combinatorial optimization problems. This paper analyses a new type of memory based on a concept called exponential extrapolation. The new type of memory is added to two different tabu search implementations for solving *binary integer programming* problems.

We find that using the new type of memory improves the performance of the tabu search implementations, as measured by their primal gaps and the corresponding primal integral. However, based on the computational study, it is not entirely clear how the mechanism influences the behavior of the tabu search, as it seems the effect depends on the structure of the instances solved as well as the type of move evaluation used in the tabu search. Furthermore, for the best type of tabu search, using dynamic move evaluations, the improvements from adding exponential extrapolation memory are not statistically significant when considering results from runs on four different sets of varied benchmark test instances. This suggests that additional research is required to better understand the search behavior and how different tabu search mechanisms interact under different circumstances.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors thank the editors and anonymous reviewers that provided valuable comments and suggestions for how to improve an initial version of the paper.

Appendix A. Test problems

In the following we describe the combinatorial optimization problems that have been used in the computational experiments to evaluate the exponential extrapolation memory.

A.1. Optimum satisfiability problem

The *optimum satisfiability problem* (OptSAT) was suggested by Davoine et al. [7], who referred to it as the *Boolean optimization problem*. The mathematical formulation provided here was described by da Silva et al. [6]. Consider a set of m clauses forming a Boolean expression that must be satisfied. Let A_i be the set of non-negated variables appearing in clause i, B_i the set of negated variables, and c_j the profit obtained by setting variable x_j to true (equivalently to 1). The problem is then written as

$$\begin{aligned} \max\ z = \sum_{j=1}^{n} c_j x_j, & \\ \sum_{j \in A_i} x_j - \sum_{j \in B_i} x_j \leq |A_i| - 1, & \qquad i \in \{1, \dots, m\}, \\ x_j \in \{0, 1\}, & \qquad j \in \{1, \dots, n\}. \end{aligned}$$

A.2. Multiple-choice multidimensional knapsack problem

In the *multiple-choice multidimensional knapsack problem* (MMKP), n disjoint groups of items, $G_1, G_2, \dots, G_n$, are given. Exactly one item j from each group G_i should be selected. The profit from the selected item is c_{ij}. However, the selection is limited by m knapsack constraints. For knapsack k, a capacity of b_k is enforced, and the weight of item j from group i is a_{ijk}. This leads to the following formulation, based on [14]:

$$\begin{aligned} \max\ z = \sum_{i=1}^{n} \sum_{j \in G_i} c_{ij} x_{ij}, & \\ \sum_{i=1}^{n} \sum_{j \in G_i} a_{ijk} x_{ij} \leq b_k, & \qquad k \in \{1, \dots, m\}, \\ \sum_{j \in G_i} x_{ij} = 1, & \qquad i \in \{1, \dots, n\}, \\ x_{ij} \in \{0, 1\}, & \qquad i \in \{1, \dots, n\}, j \in G_i. \end{aligned}$$

A.3. Multidemand multidimensional knapsack problem

Cappanera and Trubian [4] presented the most comprehensive early work on the *multidemand multidimensional knapsack problem* (MDMKP). The MDMKP combines knapsack constraints and covering constraints, and can be formulated as

$$\max\ z = \sum_{j=1}^{n} c_j x_j,$$

$$\begin{aligned}
\sum_{j=1}^{n} a_{ij}x_j &\leq b_i, && i \in \{1,\dots,m\}, \\
\sum_{j=1}^{n} a_{ij}x_j &\geq b_i, && i \in \{m+1,\dots,m+q\}, \\
x_j &\in \{0,1\}, && j \in \{1,\dots,n\},
\end{aligned}$$

where $b_i > 0$ and $a_{ij} \geq 0$. The MDMKP includes as a special case the *multidimensional knapsack problem* when $q = 0$.

A.4. The weighted max-cut problem

The *weighted max-cut problem* (MaxCut) is to find a partition of the nodes N of a weighted, undirected graph $G(N, E)$ into two sets $\{S \subset N, N \setminus S\}$ such that the sum of the weights of the edges between the sets is maximized [9]. When allowing non-positive edge weights, this can be formulated as a binary integer programming problem as follows.

$$\max \ z = \sum_{(u,v)\in E} w_{uv}y_{uv}$$

$$\begin{aligned}
y_{uv} - x_u - x_v &\leq 0, && (u,v) \in E : w_{uv} \geq 0, \\
y_{uv} + x_u + x_v &\leq 2, && (u,v) \in E : w_{uv} \geq 0, \\
-y_{uv} - x_u + x_v &\leq 0, && (u,v) \in E : w_{uv} < 0, \\
-y_{uv} + x_u - x_v &\leq 0, && (u,v) \in E : w_{uv} < 0, \\
x_u &\in \{0,1\}, && u \in N, \\
y_{uv} &\in \{0,1\}, && (u,v) \in E,
\end{aligned}$$

where x_u indicates which of the two sets to which a given node $u \in N$ belongs, y_{uv} equals 1 in the optimal solution only if u and v are in different sets, and w_{uv} is the weight of the edge between node u and node v. The number of variables in this formulation equals $|N| + |E|$, whereas the number of constraints equals $2|E|$. The formulation is not very strong, and it is primarily put forth so that solution methods for general binary integer programming problems can be compared, rather than to compare with the best available problem specific solvers.

References

[1] H. Arntzen, L.M. Hvattum, A. Løkketangen, Adaptive memory search for multidemand multidimensional knapsack problems, Comput. Oper. Res. 33 (2006) 2508–2525.
[2] H. Bentsen, L.M. Hvattum, Variable neighborhood search for binary integer programming problems, Int. J. Metaheuristics (2022), forthcoming.
[3] T. Berthold, Measuring the impact of primal heuristics, Oper. Res. Lett. 41 (2013) 611–614.

[4] P. Cappanera, M. Trubian, A local-search-based heuristic for the demand-constrained multidimensional knapsack problem, INFORMS J. Comput. 17 (2005) 82–98.
[5] J.-F. Cordeau, G. Laporte, A. Mercier, A unified tabu search heuristic for vehicle routing problems with time windows, J. Oper. Res. Soc. 52 (2001) 928–936.
[6] R.F. da Silva, L.M. Hvattum, F. Glover, Combining solutions of the optimum satisfiability problem using evolutionary tunneling, MENDEL 26 (2020) 23–29.
[7] T. Davoine, P.L. Hammer, B. Vizvári, A heuristic for Boolean optimization problems, J. Heuristics 9 (2003) 229–247.
[8] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm Evol. Comput. 1 (2011) 3–18.
[9] P. Festa, P.M. Pardalos, M.G.C. Resende, C.C. Ribeiro, Randomized heuristics for the max-cut problem, Optim. Methods Softw. 7 (2002) 1033–1058.
[10] F. Glover, Exploiting local optimality in metaheuristic search, arXiv:2010.05394, 2020.
[11] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publisher, Boston, Dordrecht, London, 1997.
[12] C. Helmberg, F. Rendl, A spectral bundle method for semidefinite programming, SIAM J. Optim. 10 (2000) 673–696.
[13] L.M. Hvattum, A. Løkketangen, F. Glover, Adaptive memory search for Boolean optimization problems, Discrete Appl. Math. 142 (2004) 99–109.
[14] H. Kellerer, U. Pferschy, D. Pisinger, The multiple-choice knapsack problem, in: Knapsack Problems, Springer, Berlin, Heidelberg, 2004, pp. 317–347.
[15] S. Khan, K.F. Li, E.G. Manning, M.M. Akbar, Solving the knapsack problem for adaptive multimedia system, Studia Inform. Universalis 2 (1) (2002) 157–178.
[16] M. Laguna, F. Glover, Integrating target analysis and tabu search for improved scheduling systems, Expert Syst. Appl. 6 (1993) 287–297.
[17] X. Lai, J.-K. Hao, D. Yue, Two-stage solution-based tabu search for the multidemand multidimensional knapsack problem, Eur. J. Oper. Res. 274 (2019) 35–48.
[18] R. Martí, A. Duarte, M. Laguna, Advanced scatter search for the max-cut problem, INFORMS J. Comput. 21 (2009) 26–38.
[19] H. Shojaei, T. Basten, M.C.W. Geilen, A. Davoodi, A fast and scalable multi-dimensional multiple-choice knapsack heuristic, ACM Trans. Des. Autom. Electron. Syst. 18 (4) (October 2013) 51.
[20] E.-G. Talbi, Metaheuristics – From Design to Implementation, John Wiley & Sons, Hoboken, New Jersey, USA, 2009.