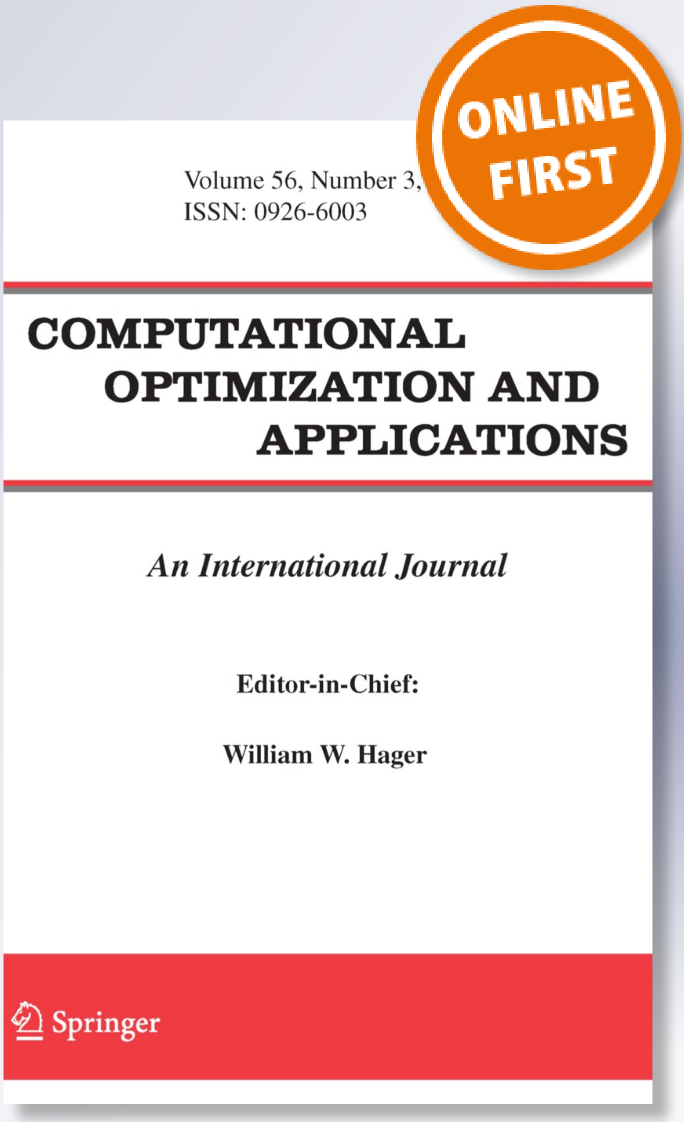
Strategic oscillation for the quadratic multiple knapsack problem

Carlos García-Martínez, Fred Glover, Francisco J. Rodriguez, Manuel Lozano & Rafael Martí

Computational Optimization and Applications
An International Journal

ISSN 0926-6003

Comput Optim Appl
DOI 10.1007/s10589-013-9623-y



Comput Optim Appl
DOI 10.1007/s10589-013-9623-y

Strategic oscillation for the quadratic multiple knapsack problem

Carlos García-Martínez · Fred Glover · Francisco J. Rodriguez · Manuel Lozano · Rafael Martí

Received: 19 June 2013

Abstract The quadratic multiple knapsack problem (QMKP) consists in assigning a set of objects, which interact through paired profit values, exclusively to different capacity-constrained knapsacks with the aim of maximising total profit. Its many applications include the assignment of workmen to different tasks when their ability to cooperate may affect the results.

Strategic oscillation (SO) is a search strategy that operates in relation to a critical boundary associated with important solution features (such as feasibility). Originally proposed in the context of tabu search, it has become widely applied as an efficient memory-based methodology. We apply strategic oscillation to the quadratic multiple knapsack problem, disclosing that SO effectively exploits domain-specific knowl-

C. García-Martínez (✉)
Department of Computing and Numerical Analysis, University of Córdoba, Córdoba 14071, Spain
e-mail: cgarcia@uco.es

F. Glover
OptTek Systems, Boulder, CO, USA
e-mail: glover@opttek.com

F.J. Rodriguez · M. Lozano
Department of Computer Sciences and Artificial Intelligence, University of Granada, Granada, 18071, Spain

F.J. Rodriguez
e-mail: fjrodriguez@decsai.ugr.es

M. Lozano
e-mail: lozano@decsai.ugr.es

R. Martí
Department of Statistics and Operations Research, University of Valencia, Valencia, Spain
e-mail: rafael.marti@uv.es

Published online: 28 November 2013

edge, and obtains solutions of particularly high quality compared to those obtained by current state-of-the-art algorithms.

Keywords Strategic oscillation · Tabu search · Quadratic multiple knapsack problem · Empirical study

1 Introduction

The quadratic multiple knapsack problem (QMKP) [1] consists in assigning a set of objects disjunctively to a set of capacity-constrained knapsacks for the goal of maximising the total sum of profits. Profit values are assigned not only to individual objects but also to pairs of them, which renders traditional approaches (that do not consider pairwise linkages between objects) unable to effectively address the problem [2]. The QMKP arises commonly in contexts where resources have to be assigned to different tasks and the success measure depends on interactions between these resources. Previous efforts to solve QMKP by metaheuristic approaches include local search methods [1, 3], genetic algorithms [1, 4, 5], swarm intelligence methods [3] and iterated greedy methods [6].

Strategic oscillation (SO) [7] is a search strategy that operates by moves defined in relation to a *critical boundary* that identifies regions of the search space that are expected to contain solutions of particular interest. Originally proposed in the context of tabu search as a long term strategy, it now constitutes a well-established method within the family of memory-based methodologies known as *adaptive memory programming* and has produced competitive results in applications fields such as binary quadratic programs [8], the multi-resource generalized assignment problem [9], the maximally diverse grouping problem [10] and the linear ordering problem with cumulative costs [11], among others.

In this paper, we analyse the application of the SO framework to effectively solve the QMKP. In particular, we exploit *capacity* and *object-individuality* constraints of the QMKP as a source of critical boundaries that SO may explore. Our approach iteratively applies three stages:

1. An oscillation process explores the feasible and infeasible regions around a current solution and returns a new candidate;
2. A local optimization operator is applied to every new candidate to get an associated improved solution from the nearby area of the search space; and
3. An acceptance criterion decides which improved solution is chosen to continue the search. In addition, we employ a constructor operator to build initial solutions at the beginning of the run and when stagnation is detected.

We include tests of a variety of heuristics for each operator of our SO framework, such as completely random operators, procedures biased by the objective function, and tabu search strategies taking advantage of memory structures. The experiments show that the final algorithm, SO-QMKP, is able to effectively exploit domain-knowledge associated with the problem requirements and outperforms current state-of-the-art approaches.

The rest of the paper is structured as follows. In Sect. 2, we describe the QMKP model and current state-of-the-art approaches for its solution. In Sect. 3, we introduce our SO framework for the QMKP and detail several strategic variations that can be used at each stage. Section 4 presents the findings from our empirical study, which is designed to: (1) analyse the influence of the supporting strategies, parameters and settings associated with our method, (2) compare the results of our SO-QMKP algorithm with those of other approaches from the literature, and (3) identify the algorithmic components that provide the greatest contribution to exploiting the domain knowledge associated with the QMKP. Finally, Sect. 5 discusses conclusions and further work.

2 The quadratic multiple knapsack problem

The QMKP seeks an optimal assignment of n objects and K knapsacks. Each object $i \in \{1, 2, \dots, n\}$ has a profit p_i and a weight w_i, each pair of objects (i and j) has a profit p_{ij}, and each knapsack $k \in \{1, 2, \dots, K\}$ has a capacity C_k. To simplify the notation, we refer to objects and knapsacks by their index positions. The profit p_{ij} associated with the pair of objects i and j is added to the total profit if both i and j belong to the same knapsack. The objective is to allocate each object to at most one knapsack so that the total weight of the objects in each knapsack k does not exceed its capacity C_k and the total profit of all the objects included into the knapsacks is maximised. Formally, given the binary variables x_{ik}, which indicate whether object i is included in knapsack k, the QMKP can be formulated as:

$$\text{Maximise} \quad \sum_{i=1}^{n}\sum_{k=1}^{K} x_{ik} p_i + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{k=1}^{K} x_{ik} x_{jk} p_{ij}, \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} x_{ik} w_i \le C_k, \quad \forall k \in \{1, 2, \dots, K\}, \tag{2}$$

$$\text{and} \quad \sum_{i=1}^{K} x_{ik} \le 1, \quad \forall i \in \{1, 2, \dots, n\}. \tag{3}$$

The QMKP has many real-world applications in situations where resources with different levels of interaction have to be distributed among different tasks, for instance, assigning team members to different projects where member contributions are considered both individually and in pairs. The QMKP is an extension of two well-known combinatorial optimisation problems, the multiple knapsack problem and the quadratic knapsack problem, and as in the case of its special instances, the QMKP is NP-hard [1].

Hiley and Julstrom [1] were the first to study this problem and proposed three approaches for its solution: a greedy heuristic (discussed in Sect. 3.1), which provides feasible solutions from scratch, a hill-climbing method that consists in removing some random objects from their corresponding knapsacks and applying the greedy heuristic to refill the solution, and a generational genetic algorithm. This latter comprises a population of solutions that are initialised by assigning objects iteratively to random knapsacks that can accommodate them. Afterwards, binary tournament selection is employed, and mutation and crossover operators are executed iteratively according to a crossover probability parameter. The tournament selection removes two random objects from their knapsacks and applies the greedy heuristic, and the crossover operators first copy into each offspring all object assignments common to its two parents and then considers all remaining objects in random order for the knapsacks that can accommodate them, preserving the best solution found.

Singh and Baghel [5] addressed the QMKP with a grouping genetic algorithm, implementing a steady-state model where crossover and mutation produce a single child at each iteration (in a mutually exclusive manner), which replaces the least fit member of the population. Solutions are encoded as a set of knapsacks and multiple copies of the same solution are avoided by checking new ones against the existing population members. Initial solutions are created by the aforementioned greedy heuristic [1], but initially assigning a random object to an arbitrary knapsack. Binary tournament selection is applied. The crossover operator iteratively selects one of the two parents, assigns the knapsack with largest profit value to the child, and updates the remaining knapsacks in both parents. Unassigned objects are then included randomly in the knapsacks where this is possible without violating the capacity constraints. Mutation removes some objects from knapsacks and refills them randomly.

Saraç and Sipahioglu [4] proposed another genetic algorithm in which initial solutions are generated by assigning random objects whose weights are smaller than the remaining capacity of the current knapsack, proceeding from the knapsack with the smallest capacity to the largest. Binary tournament selection is applied to allow solutions to be copied to the next generation. The crossover operator interchanges the object assignments between two randomly selected parents. When a knapsack capacity is exceeded because of this interchange, the newly added object causing the violation is removed from the knapsack. Subsequently, unassigned objects are considered according to the heuristic rule of [1]. Two mutation operators are included, the first one removing four objects from their knapsacks and refilling the solution, and the second one interchanging objects assigned to different knapsacks. Elitism is applied to maintain the best solution from one generation to another.

Sundar and Singh [3] proposed an artificial bee colony algorithm (SS-ABC) combined with local search. In the ABC terminology, bees are classified as workers, scouts and onlookers, which exploit food sources representing solutions of the problem at hand. Initially, each worker bee is associated with a randomly generated food source (candidate solution) generated in a way similar to [5]. Then, during each iteration, each worker bee determines a new food source in the neighbourhood of its currently associated food source and computes the nectar amount (fitness or solution value) of this new one to apply or discard the associated move. Subsequently, onlookers choose one of the food sources associated with worker bees by means of binary tournament selection, and determine a new food source in the neighbourhood that replaces the selected one if it is better. When a food source is not improved for a predetermined number of iterations, then, the associated worker bee abandons it to become a scout. This scout draws a new random solution and becomes a worker bee associated with it. An improvement method based on swapping unassigned objects with those already in a knapsack is also applied. The authors report computational experiments to assess the superiority of this procedure with respect to the previous approaches described above.

Finally, iterated greedy has recently been applied to the QMKP [6]. This method, TIG-QMKP, alternates constructive and destructive phases linked by an improvement process. Specifically, after an initial construction, a destruction mechanism (enhanced by a short-term tabu search memory) removes different objects from the knapsacks, and then reconstructs the partial solution with a greedy method. This latter method is partially based on a previous heuristic [5], plus a local optimization. The authors perform extensive experimentation and show that this method and the previous ABC reported above are able to obtain the best known solutions for the QMKP. We therefore include both methods, SS-ABC and TIG-QMKP, in our computational comparison reported in Sect. 4.

3 Strategic oscillation for the QMKP

The SO methodology, first introduced in [12], operates by orienting moves in relation to a critical level, as identified by a stage of construction or a chosen interval of functional values. As summarized in [7], such a critical level or oscillation boundary often represents a point where the method would normally stop. Instead of stopping when this boundary is reached, however, the rules for selecting moves are modified, either to retreat some distance in the reverse direction or to permit the region defined by the critical level to be crossed.

In this study, we analyse the combination of different strategies, leading the search process to traverse the oscillation boundary defined by the constraints in QMKP:

- *Maximal knapsack capacities*: The sum of the weights of the objects in each knapsack must be inferior or equal to its capacity (Eq. (2)). On the other hand, when profit and weight values are positive, which is the general case, then moves are selected so as not to miss the opportunity of including another available object when there is room for it. Knapsack capacities become critical levels where effective solutions are distributed and strategic oscillation are performed that approach the critical levels from both sides.
- *Indivisibility*: Objects must be assigned to no more than one knapsack (Eq. (3)). When this constraint is relaxed, the algorithm is allowed to include the same object in different knapsacks. Then, the presence of more than one knapsack offers another opportunity to perform oscillations around the aforementioned critical levels.

We consider the SO framework shown in Fig. 1. Initially, a complete solution is built by the constructive procedure (described in Sect. 3.1), which is subsequently refined by the improvement method operator (described in Sect. 3.2). Afterwards, the algorithm iteratively applies first, the oscillation method (described in Sect. 3.3), which performs moves around the critical levels defined by previous constraints and

```
Input:
  $max_{fails}$: Number of failed internal iterations before
  reinitiating
  $max_K$: Number of span bounds
  $\sigma_1, \ldots, \sigma_{max}$: Span bounds for the oscillation method
Output:
  $S_b$: Best solution generated
1  while (stop-condition is not reached) do
2      $S' \leftarrow$ constructive procedure();
3      $S \leftarrow$ improvement method($S'$);
4      $S_b \leftarrow S$, $k \leftarrow 1$, $num_{fails} \leftarrow 0$;
5      while ($num_{fails} < max_{fails}$ and stopping-condition is not
       reached) do
6          $\{S'\} \leftarrow$ oscillation method($S, \sigma_k$);
7          $S'' \leftarrow$ improvement method($S'$);
8          if ($S''$ is better than $S_b$) then
9              $S_b \leftarrow S''$, $k \leftarrow 1$, $num_{fails} \leftarrow 0$;
10         else
11             if ($k == max_K$) then
12                 $k \leftarrow 1$, $num_{fails} \leftarrow num_{fails} + 1$;
13             else
14                 $k \leftarrow k + 1$;
15             end
16         end
17         $S \leftarrow$ acceptance-criterion($S, S''$);
18     end
19 end
20 return $S_b$;
```

Fig. 1 Pseudocode of the SO framework

the current solution according to a span parameter σ. The oscillation method also applies an improvement method to refine the new solution and an acceptance-criterion (described in Sect. 3.4) that decides which configuration becomes the new current solution, until a maximum number of fails (iterations without improving the best solution) is reached. Note that the parameter k for the oscillation method is strategically updated according to whether the best overall solution is improved or not (lines 8–16 in Fig. 1). The entire procedure is repeated until a stopping condition is reached and the best solution generated is returned at the end. Additionally, Sect. 3.5 describes several enhanced code implementations to reduce the computational effort associated with evaluating solutions.

3.1 Constructive procedure

In the context of our SO framework, the constructive procedure is in charge of providing an initial solution to attain the critical levels associated with the QMKP constraints, i.e., configurations where several objects have been assigned to one knapsack and there is no room for another object in any of them. From the different alternatives for pursuing this goal, we have considered the following five constructive methods, which iteratively introduce objects in knapsacks until there is no room in any knapsack for another unassigned object:

- *Random procedure*: Introduces a randomly chosen object, currently not assigned to any knapsack, into a randomly chosen knapsack with sufficient room for it.
- *Greedy procedure*: Hiley and Julstrom [1] describe a greedy constructive algorithm, which may be used to build feasible solutions for the QMKP. In order to do this, we firstly define the contribution ($\Delta(i, k)$) and density ($D(i, k)$) of an object i with regards to knapsack $k > 0$, as the sum of profit values associated to the object i and those already in the knapsack k, and its division by the weight of object i, respectively:

$$\Delta(i, k) = p_i + \sum_{j \in k} p_{ij}, \tag{4}$$

$$D(i, k) = \Delta(i, k)/w_i. \tag{5}$$

  We note that $D(i, k)$ is the classic bang-for-buck ratio used to evaluate assignments in a simple knapsack problem setting, where the numerator is adjusted here to reflect the full profit of the assignment in view of the quadratic objective. The greedy construction algorithm consists in performing the iterative assignment of the still unassigned object i to knapsack k that maximizes the value $D(i, k)$, subject to $w_i + \sum_{j \in k} w_j \leq C_k$.
- *Infrequent solution sampling*: A common element of tabu search and SO is the application of memory structures that record statistics of the visited solutions in order to promote either intensification or diversification in future samplings [7]. Employing this idea, the infrequent solution sampling operator uses a long-term memory, updated after each iteration of the improvement method, that registers the number of times that two objects were assigned to the same knapsack. Then, when this method is applied to a new complete solution, the long-term memory structure is accessed in the search for the assignment of an unselected object i into a knapsack k such that the sum of frequencies of object i and those already in k is minimal.
- *Frequent non-tabu assignments*: In contrast to the preceding heuristic, this method exploits long-term memory to build solutions similar to those already visited, by selecting assignments that make the sum of frequencies maximal. However, a short-term memory is included as well in order to enhance the diversity of generated solutions. In particular, the short-term memory keeps track of the assignments (objects and knapsacks) of recent improved solutions, which are avoided when building a new complete solution. These assignments remain tabu during a number of SO iterations that is randomly chosen from the interval $[0, max_T]$, where max_T is a parameter of the operator.
- *Greedy non-tabu assignments*: This operator looks for assignments that maximise the aforementioned heuristic value $D(i, k)$, but exploits as well a short-term memory in order to enhance the diversity of generated solutions. This memory is updated and employed as described above.

3.2 Refining solutions: improvement method

The improvement method of our SO operates on solutions passed to it by exploring local modifications that lead to increased objective values. We analyse two alternatives, one from the literature and another proposed in this work:

- *Swap Moves*: García-Martínez et al. [6] introduce a local search operator that seeks profitable exchanges of two assigned objects from different knapsacks, or of one assigned object with an unassigned object. Their results showed that this strategy provides better results than one that considers only swaps of the second kind (as proposed in [3]).
- *Reallocations + swaps*: The swap moves previously discussed may result in a situation where some knapsack winds up with sufficient room to receive new objects or objects from other knapsacks. Therefore, we propose a multiple neighbourhood strategy, as commonly employed with SO and tabu search methods [13–15]. In our present design we first seek profitable allocations of objects to knapsacks by drawing on objects already assigned to other knapsacks or on objects currently unassigned. When no profitable allocation of this type exists, we then examine swaps until none remains that increases the objective value. At this point we return to start again by examining the first type of move. As soon as two consecutive stages fail to find an improved solution, the process stops. This approach constitutes a special instance of the multiple neighbourhood SO approach of [13] and hence we call it the strategic oscillation multi-neighbourhood (SOM) method. (For a comparative analysis of several types of multiple neighbourhood strategies, see [15].)

Two choice rules are analysed in combination with the previous strategies: a first-improvement rule, which selects the first move that improves the objective value, and the best-improvement rule, which selects the move that produces the largest improvement in the objective value.

3.3 Exploiting critical levels: oscillation method

For the QMKP, our oscillation method explores the critical levels associated with the problem constraints (Sect. 3) to generate effective solutions by paths that are launched from different points of departure and that traverse different regions. We analyse several oscillation heuristics based on applying two specific steps every time they are invoked:

1. *Divergent step*: The solution is induced to diverge (move away) from the critical level defined by its current state and the constraints defining feasibility (Sect. 3). In this stage, operations are performed until reaching a parametrized span bound ($\sigma \in [0, 1]$) described below.
2. *Convergent step*: The solution is induced to converge to (move toward) the critical level where constraints are satisfied and there is no room for another unassigned object in any knapsack.

Accordingly, two basic types of operations are used to execute these steps. The conditions for employing these operations depend on whether either the divergent step or convergent step is being applied:

- *Insertion*: An object i is inserted into a knapsack k. For a divergent step, knapsacks are permitted to be overloaded by a maximum factor equal to σ, and objects are allowed to be inserted into more than one knapsack until a maximum number of over-assignments is reached, computed as σ times the average number of objects than can be allocated to the knapsacks (as specified in Eq. (6)). For a convergent step, unassigned objects are allowed to be inserted into knapsacks with sufficient space for them.

$$max_{\text{overassignments}} = \sigma \cdot n \cdot \min\left(1, \frac{\sum_{i=1}^{n} w_i}{\sum_{k=1}^{K} C_k}\right). \tag{6}$$

- *Extraction*: An object i is removed from knapsack k. A divergent step allows a percentage (σ) of the assigned objects to be removed from their knapsacks. On the other hand, during a convergent step the successive application of this operation is used to produce feasible solutions from infeasible ones. When performing extractions during a convergent step, objects assigned to more than one knapsack are dropped first. Then, objects from overloaded knapsacks are selected. These operations are repeated until the problem constraints are satisfied.

Note that operation types are paired in the sense that a divergent step on the feasible side of the critical boundary employs extractions (to make the solution feasible by an increasing margin) and is then followed by a convergent step that employs insertions to move back toward the boundary. By contrast, a divergent step that is carried out on the infeasible side of the critical boundary employs insertions (to make the solution infeasible by an increasing margin), and is then followed by a convergent step that employs extractions to move back toward (and cross) the boundary. In particular, our oscillation method randomly decides which set of operations to apply at every invocation, launched upon reaching a critical level, selecting divergent step insertions before proceeding to convergent step extractions with probability p_{I-E} and selecting divergent step extractions before proceeding to convergent step insertions with probability ($p_{E-I} = 1 - p_{I-E}$).

Considering the four possibilities, resulting from the combination of type of step (divergent step and convergent step) and basic operation (insertion and extraction), different heuristics can be implemented to select the object to be inserted in, or extracted from a knapsack. In this study, we analyse the following options:

- *Random heuristic*: The object to be inserted or extracted is selected at random from the different possibilities.
- *Greedy heuristic*: The best (or the least bad) operation is chosen from the different possibilities available, according to the density values (Eq. (5)).
- *Tabu heuristic*: A short-term memory structure is implemented, which keeps track of the objects recently inserted into or extracted from a knapsack. This heuristic looks for the best non-tabu operation, according to the density values (Eq. (5)). Then, recently affected objects remain tabu for a number of iterations ahead which is computed randomly in the set $\{0, \dots, n \cdot tenure_{max}\}$, where $tenure_{max}$ is a parameter of the method. This general description requires particular adaptations for two special cases:

Table 1 SO analysed combinations

		Convergent step			
		Random	Greedy	Tabu	Short-term
Divergent step	Random	–	RGRG	RTRT	–
	Greedy	GRGR	–	GTGT	GMGM
	Tabu	TRTR	TGTG	TTTT	TMTM
	Short-term	–	MGMG	MTMT	MMMM

  - For convergent step insertions, a short-term memory tracking objects could potentially make the heuristic produce solutions falling short of the critical level when there was room in the knapsacks for more objects, but the possibilities were tabu. Therefore, in this case the implemented short-term memory keeps track of the object and knapsack pair that composes the proposed operation. This way, tabu objects are prevented from being inserted in the knapsacks they belonged to in the recent past, but can still be associated with other knapsacks.
  - For convergent step extractions, the short-term memory could potentially prevent the heuristic from reaching a feasible solution if extractions either of overassigned objects or from overloaded knapsacks are tabu. When this situation appears, the greedy (best) option is chosen regardless of its tabu condition (this is a form of aspiration criterion for overriding tabu status).
- *Random short-term memory heuristic*: This alternative implements the previously described short-term memory and strategies with the exception that random non-tabu moves, instead of best non-tabu moves, are selected.

Therefore, a wide range of possibilities is brought into the scene. In order to reduce the set of alternatives, we analyse the combinations that apply the same heuristic for each type of step (the divergent step and the convergent step). These combinations are presented in Table 1 in which the methods are denoted with a four letter name that stands for <divergent step insertions> <convergent step extractions> <divergent step extractions> <convergent step insertions> according to 'R' for the random heuristic, 'G' for greedy, 'T' for tabu, and 'M' for random short-term memory. For example, the cell labelled 'RTRT' represents an oscillation method that applies the random heuristic for divergent step operations (whether insertions or extractions), and the tabu heuristic for convergent step operations.

Empty cells represent combinations that have not been analysed because they introduced too much randomness (RRRR, RMRM, and MRMR), or not enough (GGGG), in the search process.

3.4 The acceptance criterion

The acceptance-criterion of our SO establishes the rules by which the algorithm wanders over the regions of the search space in the quest for better solutions. We analyse the following criteria from the literature:

- *Replace if better*: The new solution is accepted only if it attains a better objective function value [8, 10].

– *Random walk*: Several authors have pointed out that the foregoing acceptance criterion may lead the method to stagnation, because of getting trapped in a local optimum [11, 16]. By contrast, the random walk criterion always selects the new solution, regardless of its objective function value, which prevents the method from being confined in the area of one local optimum. This approach is a special case of applying tabu search with a tabu tenure of 0, proposed as an option for probabilistic tabu search [7].

3.5 Move evaluations

In this section, we describe several improvements that allow an efficient evaluation of new solutions, which were generated by means of insertion and extraction operations (Sects. 3.1 and 3.3) or reallocations and swaps (Sect. 3.2), for the QMKP. These enhancements accelerate the execution of our algorithm without altering its outcomes.

Define the *raw objective value* of a solution to be the sum of individual and paired profits of the objects included in the knapsacks (Eq. (1)) regardless of whether constraints (2) and (3) are satisfied. Then, given a solution S, feasible or not, the insertion of object i into the knapsack k, which does not contain i, produces a new solution whose raw objective value is that of S plus the corresponding contribution value $\Delta(i,k)$ (defined by Eq. (4)):

$$f\left(S'\right) = f(S) + \Delta(i,k). \tag{7}$$

Conversely, the extraction of object i from knapsack k, which contains i, produces another solution with a raw objective value reduced by the same quantity. Therefore, new solutions generated by insertion and extraction operations need not be completely evaluated by the objective function $f(\cdot)$, which saves computational effort.

To achieve this savings in effort, we maintain the current $\Delta(i,k)$ values for a given solution S in a memory structure which is updated every time an insertion or extraction operation is performed. These values in the memory structure will be referred to as $\Delta_S^{mem}(i,k)$ to differentiate them from the mathematical definition of the contribution of object j to knapsack k (Eq. 4). In particular, given a null solution S, where no object is assigned to any knapsack, the contribution of any object for any knapsack is initialised to its profit value ($\Delta_S^{mem}(i,k) \leftarrow p_i$, $k \in \{1,\dots,K\}$). Thereafter, inserting object i into knapsack k, which does not contain i, increments the contribution of each other object j, whether or not included in k, with the quadratic profit for the corresponding object j and i (Eq. (8)). Similarly, extracting object i from knapsack k, which contains i, reduces the contribution of any object j, whether or not included in k, by the same quantity (Eq. (9)).

$$\Delta_S^{mem}(j,k) \leftarrow \Delta_S^{mem}(j,k) + p_{ij}, \quad \forall j \neq i,\ j \in \{1,\dots,n\}, \tag{8}$$

$$\Delta_S^{mem}(j,k) \leftarrow \Delta_S^{mem}(j,k) - p_{ij}, \quad \forall j \neq i,\ j \in \{1,\dots,n\}. \tag{9}$$

Reallocation and swap operations (Sect. 3.2) are applied only to feasible solutions, employing the following equations according to whether one object i is transferred from knapsack k to knapsack k' (10), two objects i and j are exchanged between their respective knapsacks k_i and k_j (11), or an object i assigned to knapsack k_i

is exchanged with an unassigned object j (12). Notice that the use of the memory structure in (10), (11), (12) allows their computational complexities to be $O(1)$.

$$f\left(S:k \xrightarrow{i} k'\right) = f(S) - \Delta_S^{mem}(i,k) + \Delta_S^{mem}\left(i,k'\right), \tag{10}$$

$$\begin{aligned} f(S:i_{ki} \leftrightarrow j_{kj}) = f(S) &- \Delta_S^{mem}\left(i,k^i\right) + \Delta_S^{mem}\left(i,k^j\right) \\ &- \Delta_S^{mem}\left(j,k^j\right) + \Delta_S^{mem}\left(j,k^i\right) - 2p_{ij}, \end{aligned} \tag{11}$$

$$f(S:i_{ki} \leftrightarrow j_0) = f(S) - \Delta_S^{mem}\left(i,k^i\right) + \Delta_S^{mem}\left(j,k^i\right) - p_{ij}. \tag{12}$$

To our knowledge, the exploitation of (8) through (12) to efficiently update the objective function contribution of assigning objects to knapsacks has not been exploited for the QMKP in previous work.

Finally, we should mention that we have implemented another memory structure that stores the sum of the weights of the objects in each knapsack, which is accordingly updated after applying any operation, in order to check capacity constraints efficiently.

4 Experiments

In this section, we present the experiments carried out in order to analyse the performance of our SO framework. The experimental setup and comparison methodology are described in Sect. 4.1. The remaining sections report our computational experiments to analyse the influence of parameters and settings in order to obtain a tuned and robust instance of our SO algorithm (Sect. 4.2), compare the results obtained with our algorithm with those of previous proposals (Sect. 4.3), and analyse the exploitation of domain-knowledge that allows our strategies to deal effectively with the QMKP (Sect. 4.4).

4.1 Experimental framework

All algorithms were implemented in C++ and compiled with gcc 4.6.31. The experiments were conducted on a computer with a 2.8 GHz Intel(R) Core(TM) i7-930 processor (8 MB cache, 4 cores and 8 threads) with 12 GB of RAM running Fedora(TM) Linux 15. Each execution of an algorithm is performed sequentially, using a unique thread. We have developed experiments on the same 60 QMKP instances used in [1, 3–6][1], which are characterised by the density d (proportion of non-zero profits $p_{ij} \in \{0.25, 0.75\}$), the number of objects $n \in \{100, 200\}$, and the number of knapsacks $K \in \{3, 5, 10\}$ (5 instances per combination). Knapsack capacities are set to 80 % of the sum of the instances objects weights divided by the number of knapsacks, as in the aforementioned studies.

For parameter tuning, our SO algorithm variants were executed 10 times only on two out of the five QMKP instances per combination of characteristics (Sect. 4.2), in

[1] They were obtained by adapting those at http://cedric.cnam.fr/~soutif/QKP/ according to the guidelines in [1].

order to avoid overtuning and to reduce the computational cost of the study. Each run consumed a maximum of 5 seconds for problems with 100 objects and 30 seconds for problems with 200 objects, which are similar to the computational time values provided in [3] and [6]. For the comparison study, all the problem instances and 40 runs were used (Sect. 4.3), which is in line with the previous QMKP studies in [3] and [6]. Additionally, the same computational time values provided in [3] and [6] were assigned to these SO runs.

Non-parametric tests have been used to compare the results of different search algorithms [17–19]. It is customarily recommended that non-parametric tests be performed by applying the same criterion for sampling results to each method tested. Consequently, we compute the same aggregation (based on the average of the best objective function value in each run) over the same number of runs for each algorithm and problem. More specifically, we employ two alternative non-parametric tests to analyse the experimental results:

– The *Iman and Davenport* test [20] with *Holm's* method [21] employed as a post hoc procedure. The first test may be used to see whether there are significant statistical differences among the results of a certain group of algorithms. If differences are detected, then Holm's test is employed to compare the best algorithm (control algorithm) against the remaining methods. These two methods are based on the *Friedman ranking procedure* [18] of the algorithms on the considered problems. This measure computes, for each problem, the ranking r_j of the observed result for algorithm j, assigning to the best of them the ranking 1, and to the worst the ranking J (J is the number of algorithms). Then, an average measure is obtained from the rankings of this method for all the test problems.
– The *Wilcoxon matched-pairs signed-ranks* test [22]. With this test, the results of two algorithms may be directly compared. In order to avoid the bias introduced by the ranges of the different problems, we have normalised the results of every algorithm on each test function by mapping them into the interval [0, 1] taking into consideration the highest and the lowest values achieved by the set of algorithms considered in this type of statistical analysis.

4.2 Parameter Study and Analysis

The goal of this section is to investigate the effect of the parameters and strategies applied in our proposal, in order to provide a tuned SO algorithm. We have undertaken the study to find the most effective combination(s), under a full factorial design, summarized in Table 2, with the following elements:

– the set of values for the span bound parameter σ (Sect. 3), consisting of a variable set $V = \{0.025, 0.05, 0.1, 0.15, 0.2\}$, or a constant set $C = \{0.1, 0.1, 0.1, 0.1, 0.1\}$;
– the maximum number of fails to generate a new solution ($max_{fails} \in \{1, 20, \infty\}$);
– the constructor heuristic (Sect. 3.1) consisting of random (R), greedy (G), infrequent (I), frequent (F) or greedy non-tabu (GT);
– the improvement heuristic (Sect. 3.2) consisting of first-improvement swaps (FS), best-improvement swaps (BS), first-improvement SOM (FSOM), best-improvement SOM (BSOM);

Table 2 SO parameters and strategies tested

Elements	Alternatives
σ	V, C
max_{fails}	1, 20, ∞
constructive procedure	R, G, I, F, GT
improvement method	FS, BS, FSOM, BSOM
oscillation method	Table 1 (12 designs)
p_{I-E}	0.5
$tenure_{max}$	0.1
acceptance criterion	RB, RW

Table 3 Rankings of the best ten SO variants

	Algorithm	Ranking
1	SO-SV-F20-CR-IFVND-OMTMT-RW	101.46
2	SO-SV-F20-CR-IFVND-ORBRB-RW	112.5
3	SO-SV-F20-CI-IFVND-ORBRB-RW	115.1
4	SO-SC-F20-CR-IFVND-OMBMB-RW	116.17
5	SO-SV-F20-CI-IFVND-ORTRT-RW	117.19
6	SO-SC-F20-CR-IFVND-OMTMT-RW	117.81
7	SO-SV-F20-CI-IFVND-OMBMB-RW	119.08
8	SO-SV-F20-CR-IFVND-OMTMT-RW	120.02
9	SO-SV-F20-CR-IFVND-OMBMB-RW	121.81
10	SO-SC-F20-CI-IFVND-OMTMT-RW	123.31
...	...	...

– oscillation methods (see Table 1) with $p_{I-E} = 0.5$ and $tenure_{max} = 0.1$; and
– the acceptance-criterion (Sect. 3.4) consisting of Replace if better (RB), and Random walk (RW).

Table 3 shows the Friedman rankings of the best ten SO algorithm variants (out of 1728 tested), which are denoted as SO-S<span set type>-F< max_{fails} >-C<constructor>-I<improvement>-O<oscillation>-<acceptance>. For instance, using the indicated order, the best SO algorithm applies the variable set of span bounds, $max_{fails} = 20$, the random constructor, the SOM optimiser with the first-improvement strategy, the random short-term memory alternative for divergent step oscillation operations and the tabu alternative for convergent step operations and random walk as the acceptance criterion. Iman and Davenport's and Holm's analyses are not reported because they do not find significant differences between these leading ten algorithms (which rank in the upper 0.5 % of the algorithmic variants tested).

From the results in Table 3 we may remark that:

– All of the ten best SO algorithm variants have the max_{fails} parameter set to 20, apply the SOM improvement method with the first-improvement strategy, and employ the random walk (0 tabu tenure) acceptance criterion.
– Most of the ten best variants apply the variable set of span bounds.

Table 4 Best SO variants in the second tuning phase

	Algorithm	Ranking	R+	R−	Wilcoxon
1	SO-F50-P0.25-T0.2	7.67			
2	SO-F50-P0.25-T0.1	7.85	162	138	$\sim$
3	SO-F50-P0.25-T0.05	7.85	150.5	149.5	$\sim$
4	SO-F50-P0.5-T0.2	9.15	249	51	$+$
5	SO-F20-P0.25-T0.1	9.17	205	95	$\sim$
6	SO-F20-P0.25-T0.05	9.17	207	93	$\sim$
7	SO-F20-P0.25-T0.2	9.23	208	92	$\sim$
8	SO-F50-P0.5-T0.05	9.60	253	47	$+$
9	SO-F50-P0.5-T0.1	9.88	252	48	$+$
10	SO-F20-P0.5-T0.05	11.92	248	52	$+$
...	...	...	...	...	...

– There is a very slight preference for the random constructor heuristic over the infrequent sampling heuristic. In any case, these two approaches yield better results than the other alternatives for this component of the algorithm (i.e., the greedy constructor, the frequent non-tabu assignment, and the greedy non-tabu assignment).
– While there is a modest variety of combinations for the oscillation heuristic that produce the best results, very interestingly all of them apply a divergent step that is not biased by the objective function (the random and random short-term memory approaches) and a convergent step that, on the contrary, is biased by the objective (the greedy and tabu approaches). In particular, the divergent step applying the short-term memory is the one used most commonly by the best algorithms, and is most often paired with the tabu convergent step.

With the aim of further tuning our proposal, we analyse some other settings under a full factorial design for three parameters of the best SO variant (SO-SV-F20-CR-IFSOM-OMTMT-RW, which is in accordance with previous remarks for the best results): the maximum number of fails ($max_{fails} \in \{10, 20, 50\}$); the probability for applying the first divergent step insertion and second convergent step extraction operations $p_{I-E} \in \{0.25, 0.5, 0.75\}$; and the maximum tenure value ($tenure_{max} \in \{0.05, 0.1, 0.2\}$). Table 4 shows the ten best combinations (out of 27) with the notation SO-F$< max_{fails} >$-P$< p_{I-E} >$-T$< tenure_{max} >$. Iman and Davenport's and Holm's analysis are not reported because they do not find significant differences between these ten algorithms. In contrast, the R+ and R− values of Wilcoxon's test (respectively associated with the best ranked algorithm and an alternative algorithm being compared with it) are presented together with the test's result: when the R-value is inferior to the critical value at significance factor 0.05 (81), this test finds significant differences in favour of the best ranked algorithm (+); if neither R+ nor R− is inferior to the critical value, it does not find significant differences ($\sim$).

From the results in Table 4, we observe that, though these algorithms perform almost equally well, there is a slight preference for $max_{fails} = 50$ and $p_{I-E} = 0.25$. We do not find any indication of a preferred $tenure_{max}$ value (from among those tested) that yields better results. From now on, the experiments are carried out only

Table 5 Comparison of best methods on 60 instances

Type	SS-ABC			TIG-QMKP			SO-QMKP		
	Avg.	Dev.	#Best	Avg.	Dev.	#Best	Avg.	Dev.	#Best
100, 25, 3	27705.006	2.03	0	28063.925	0.76	2	28232.775	0.16	5
100, 25, 5	21351.048	2.27	0	21602.395	1.11	1	21716.3	0.59	5
100, 25, 10	14854.166	4.48	0	15385.86	1.05	5	15369.845	1.15	0
200, 25, 3	102281.2	0.72	2	102429.105	0.6	0	102640.735	0.39	3
200, 25, 5	75667.804	0.72	3	75760.675	0.71	1	75895.72	0.53	1
200, 25, 10	50372.774	3.6	0	51779.055	0.9	4	51772.69	0.91	1
100, 75, 3	69039.64	0.82	0	69569.635	0.05	4	69569.86	0.05	3
100, 75, 5	48829.54	0.82	0	49022.54	0.43	3	49060.195	0.35	4
100, 75, 10	30013.22	1.58	1	30229.925	0.91	1	30289.635	0.71	3
200, 75, 3	261867.42	1.13	0	264638.745	0.09	4	264692.07	0.07	4
200, 75, 5	178872.298	1.27	0	180756.55	0.22	3	180776.27	0.21	3
200, 75, 10	106641.482	2.38	0	108746.83	0.44	2	108788.665	0.4	3
Summary	82291.3	1.82	6	83165.44	0.61	30	83233.73	0.46	35

with SO-F50-P0.25-T0.2, the best ranked algorithm, which in the rest of the paper we simply denote as SO-QMKP.

4.3 Comparison with other metaheuristics for the QMKP

In this section, we compare SO-QMKP with the two current state-of-the-art methods, SS-ABC [3] and TIG-QMKP [6], described in Sect. 2. We include in this study the five instances per problem type described in Sect. 4.1, totalling 60 instances. The three methods were run 40 times per problem instance and for the same CPU time, as reported in [3, 6]. Table 5 summarizes the results per type of instance characterized by three values: n, d, and K. We compute in each run the value obtained with each method on each instance. Then we compute the average value, Avg, across the 40 runs, and the average percentage deviation, Dev, of theses values with respect to the best known values on each instance. Table 5 reports, for each type of instance (in each row), the average of the Avg. and Dev. values of the 5 instances. It also reports the number of instances, out of the 5 in each type, in which the method is able to find the best solution known, #Best. The last row summarizes the results over the entire set of 60 instances, reporting the average of the Avg. and Dev. values and the sum of the number of best founds.

Tables 7 and 8 in the appendix show the individual results for each instance in this experiment. Specifically, Table 7 reports the Avg. values of the three methods on the 30 instances with $d = 0.25$, the best solution found on the 40 runs, Best, and the standard deviation of the values in the 40 runs, SD. Table 8 reports these statistics for the 30 instances with $d = 0.75$.

Table 5 shows that SO-QMKP presents a 0.46 % average deviation from the best known solution and is able to obtain 35 out of 60 best solutions, which compares

Table 6 Wilcoxon's test for SO-QMKP vs. state-of-the-art algorithms

	R+	R−	Wilcoxon
SO-QMKP vs SS-ABC	1830	0	+
SO-QMKP vs TIG-QMKP	1437	393	+

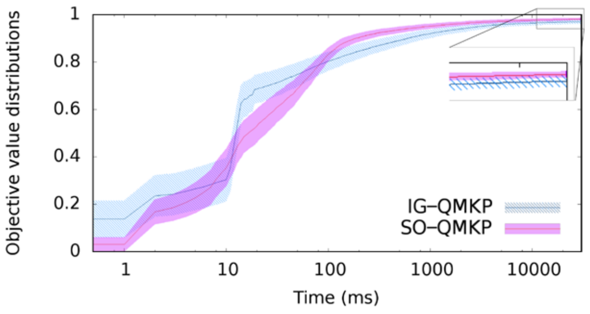


Fig. 2 Search profile

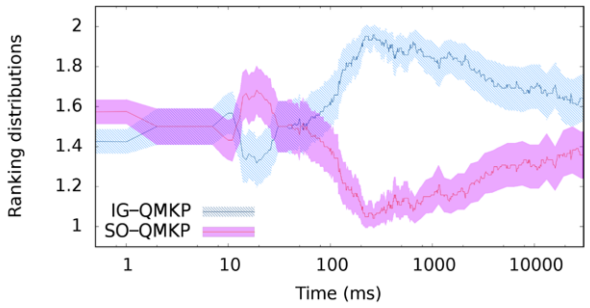


Fig. 3 Evolution of the Friedman ranking distributions

favourably with the 0.61 % average deviation and 30 best solutions obtained with TIG-QMKP and with the 1.82 % average deviation and 6 best solutions obtained with SS-ABC. In line with this, Tables 7 and 8 show that SO-QMKP often obtains better average results than the other algorithms (42 out of 60 problem instances) and its standard deviations are usually the smallest ones. Table 6 summarises the results of Wilcoxon's test between our proposal and the other two algorithms with 0.05 as the significance level. This test finds significant differences between SO-QMKP and the other two algorithms, confirming the superiority of our method.

Additionally, we compare in Figs. 2 and 3 the evolution of the objective function and Friedman ranking distributions respectively, of the two leading methods, SO-QMKP and TIG-QMKP, according to the previous experiments. In Fig. 2 all objective values have been normalised to lie in the interval [0, 1]. Averaged results of the algorithms over the different problem instances are indicated by lines and the associated 95 % confidence intervals (normal estimation) of their results are indicated by the shaded areas. Notice that the ranking distribution graphs, unlike the convergence graphs, are not monotonic. That is because the Friedman ranking is a relative performance measure, and thus, if one algorithm improves its results, then, the ranking of the other may deteriorate. In addition, while variations in convergence graphs

Table 7 State-of-the-art results on the QMKP with $d = 0.25$ (truncated executions for SO-QMKP and TIG-QMKP)

n	Instance				SS-ABC			TIG-QMKP			SO-QMKP		
	K	I	C	Time	Best	Avg	SD	Best	Avg	SD	Best	Avg	SD
100	3	1	688	3.8	29139	28753	209.68	29138	28894.6	120.74	**29234**	**29159.25**	**48.43**
100	3	2	738	3.69	28443	28004	285.91	28473	28224.35	60.87	**28491**	**28467**	**32.27**
100	3	3	663	2.87	26901	26585.33	192.9	27013	26905.4	77.47	**27179**	**27155.35**	**31.79**
100	3	4	804	3.74	28568	28109.03	342.19	**28593**	**28573.6**	**10.56**	**28593**	28570.4	13.50
100	3	5	723	3.57	27849	27073.67	274.77	**27892**	27721.68	174.68	**27892**	**27811.53**	**20.56**
100	5	1	413	2.7	22390	22117.12	141.17	22264	22126	91.03	**22509**	**22337.43**	**84.13**
100	5	2	442	2.93	21584	21224.03	188.87	21580	21430.38	**84.20**	**21678**	**21540.35**	87.18
100	5	3	398	2.4	21093	20771.12	215.23	21100	21015.03	**34.45**	**21188**	**21104.75**	57.49
100	5	4	482	3.26	22178	21767.5	194.69	22180	22043	98.19	**22181**	**22136**	**56.43**
100	5	5	434	3.18	21301	20875.47	199.51	**21669**	21397.58	126.71	**21669**	**21462.88**	**70.51**
100	10	1	206	1.42	15953	15573.65	170.84	**16118**	15863	120.50	16065	**15886.58**	**92.51**
100	10	2	221	1.84	15487	14896.35	192.44	**15525**	**15398.43**	**64.66**	15510	15359.95	65.19
100	10	3	199	1.58	14339	14027.83	191.24	**14773**	14554	106.90	14663	**14568.38**	**63.75**
100	10	4	241	2.1	15807	15397	244.28	**16181**	**16089.95**	**71.61**	16159	16013	77.94
100	10	5	217	1.82	14719	14376.8	177.96	**15150**	**15023.45**	84.78	15130	15021.33	**64.63**
200	3	1	1381	23.99	100662	100103.02	283.79	100218	100056.23	**92.62**	**101100**	**100653.5**	181.44
200	3	2	1246	18.61	**107958**	107545.2	240.77	107787	**107644.98**	**61.42**	107805	107607.15	86.29
200	3	3	1335	29.85	104521	104006.98	311	104479	104251.5	**79.93**	**104538**	**104271.68**	91.71
200	3	4	1413	38.93	98791	98344.32	268.1	98896	98557.4	**142.53**	**99559**	**99003.63**	223.48
200	3	5	1358	29.22	**102049**	101406.48	344.51	101973	101635.43	**93.83**	102041	**101667.73**	160.54

Table 7 (*Continued*)

n	Instance				SS-ABC			TIG-QMKP			SO-QMKP		
	K	I	C	Time	Best	Avg	SD	Best	Avg	SD	Best	Avg	SD
200	5	1	828	19.88	**74922**	74132.95	519.19	74239	73977.78	**118.20**	74559	**74237.4**	169.65
200	5	2	747	16.75	**79506**	79073.32	278.65	79480	**79234.28**	107.32	79400	79153.55	**100.12**
200	5	3	801	22.86	77607	77069.52	244.68	**77700**	77420.5	179.10	77632	**77452.25**	**150.44**
200	5	4	848	28.07	73181	72607.25	372.38	73173	72477.65	247.03	**73327**	**72884.03**	**182.63**
200	5	5	815	20.74	**76022**	75455.98	248.11	75884	75693.18	**116.06**	75996	**75751.38**	125.09
200	10	1	414	10.37	49883	49079.47	425.35	**51413**	50845.78	193.36	51323	**50862.7**	**156.42**
200	10	2	373	8.48	53298	51831.55	459.79	**54116**	53608.45	169.08	53975	**53649.03**	**139.02**
200	10	3	400	11.15	52281	51324.28	359.07	52735	**52456.28**	143.13	**52841**	52337.73	**140.19**
200	10	4	424	12.83	49210	48190.6	466.33	**50221**	49656.4	204.40	50190	**49802.43**	**163.80**
200	10	5	407	10.99	51921	51437.97	296.41	**52651**	**52328.38**	164.68	52470	52211.58	**126.16**

Table 8 State-of-the-art results on the QMKP with $d = 0.75$ (truncated executions for SO-QMKP and TIG-QMKP)

n	Instance				SS-ABC			TIG-QMKP			SO-QMKP		
	K	I	C	Time	Best	Avg	SD	Best	Avg	SD	Best	Avg	SD
100	3	1	669	2.07	69721	69373	231.72	**69977**	**69936.05**	**6.64**	69935	69925.73	37.77
100	3	2	714	1.86	69462	69041	236.38	**69504**	**69442.78**	**39.75**	**69504**	69442.48	49.40
100	3	3	686	1.86	68635	67960.05	406.74	68811	68811	**0**	**68832**	**68812.58**	5.6
100	3	4	666	1.88	69986	69687.68	217.28	**70028**	70019.88	32.91	**70028**	**70028**	**0**
100	3	5	668	2.12	69679	69136.4	246.88	**69692**	69638.48	24.32	69653	**69640.53**	**12.95**
100	5	1	401	2.07	4922	48937.47	196.48	49345	**49218.1**	**29.81**	**49363**	49197.53	55.7
100	5	2	428	1.96	49313	48908.05	202.63	**49316**	49081.53	**81.88**	49305	**49137.38**	82.1
100	5	3	411	1.71	48472	47874.5	380.3	**48495**	**48327.48**	85.51	**48495**	48287.88	**77.08**
100	5	4	400	1.83	50199	50017.93	197.12	49866	49866	**0**	**50246**	**50025.03**	97.69
100	5	5	400	2.01	48710	48409.75	133.12	**48752**	48619.6	65.78	**48752**	**48653.18**	**62.88**
100	10	1	200	1.22	29875	29429.2	208.12	29877	29548.68	102.80	**29931**	**29788.48**	**54.53**
100	10	2	214	1.45	30939	30697.8	134.28	**30980**	**30832.15**	83.73	30973	30829.05	**70.25**
100	10	3	205	1.3	29465	28983.78	246.93	29695	29439.95	142.78	**29730**	**29519.48**	**112.25**
100	10	4	200	1.4	**31663**	31218.85	176.33	31550	31333.45	**67.95**	31587	**31392.48**	71.15
100	10	5	200	1.42	30219	29736.47	213.33	30096	29895.4	**75.08**	**30229**	**29918.7**	94.01
200	3	1	1311	14.11	269736	267117.92	1070.76	**270718**	270525.9	204.46	**270718**	**270617.48**	**166.63**
200	3	2	1414	16.27	256195	253916.75	896.46	**257090**	256764.98	95.68	257026	**256852.3**	**88.06**
200	3	3	1342	11.87	268890	267079.03	1124.57	**270069**	**269974.03**	**85.82**	**270069**	269955.03	91.51
200	3	4	1565	30.64	246205	244618.4	1022.02	246704	246356.53	**101.17**	**246882**	**246473.13**	157.1
200	3	5	1336	10.46	279490	276605	1443.72	**279598**	**279572.3**	34.82	**279598**	279562.43	**33.9**

Table 8 (*Continued*)

n	Instance				SS-ABC			TIG-QMKP			SO-QMKP		
	K	I	C	Time	Best	Avg	SD	Best	Avg	SD	Best	Avg	SD
200	5	1	786	12.34	184448	183046.65	735.6	**184909**	184500.8	150.78	184822	**184529**	**107.67**
200	5	2	848	12.34	173575	171738.85	735.6	**174682**	174239.48	245.75	**174682**	**174267**	**139.36**
200	5	3	805	12.1	185107	185059.52	469.38	186443	186170.68	172.71	**186526**	**186216.75**	**123.93**
200	5	4	939	27.03	165273	164042.2	777.57	166358	166159.55	**97.56**	**166584**	**166165.38**	137.84
200	5	5	801	14.06	192764	190474.27	1021.33	**193084**	**192712.25**	118.40	193053	192702.25	**104.88**
200	10	1	393	7.64	111000	109624.73	714.45	112262	111889.75	169.32	**112354**	**112043.68**	**150.05**
200	10	2	424	9.96	103540	102603.18	522.59	105092	104669.83	169.32	**105151**	**104781.5**	**162.83**
200	10	3	402	8.04	112509	111388.2	509.42	113868	113510.55	194.64	**113869**	**113563.08**	**140.04**
200	10	4	469	14.81	96859	95681.7	545.54	**98252**	**97807.73**	155.00	98028	97747.55	**106.18**
200	10	5	400	8.21	115125	113909.6	590.96	**116513**	**115856.3**	240.19	116298	115807.53	**168.32**

may be strongly affected by distant objective values on few instances, rankings are robust with regard to distant values but susceptible to small relative changes on many functions.

Figure 2 shows that SO-QMKP is able to get good results faster than TIG-QMKP, because its mean values are superior for running times lower than 10 milliseconds. Moreover, it also supersedes TIG-QMKP on a lover term horizon (for running times larger than 0.1 seconds). It is particularly significant that the distribution of SO-QMKP is narrower than that one of TIG-QMKP, which means that SO-QMKP performs more robustly over the different problem instances. The abrupt temporary improvement in the TIG-QMKP performance that starts just after the first 0.01 seconds can be partially explained by the fact the first call to the local optimizers occurs at this time. Then, since the SOM process of SO-QMKP is slower than the local search in TIG-QMKP (because of the use of more than one neighbourhood structure), an extra time is required for the contribution of this process. Once this happens, the performance of the SO-QMKP approach overtakes that of the TIG-QMKP approach to produce superior outcomes.

Figure 3 confirms the performance described above, and the differences between both methods become now more evident and explained by the fact that SO-QMKP overcomes TIG-QMKP on most of the problem instances (objective value differences are not considered in the ranking distributions, where the lower the ranking values the better the method). The greatest performance difference is attained at 0.2 seconds of the run approximately. After that point, TIG-QMKP tends to obtain results similar to those of SO-QMKP in quality.

4.4 Knowledge exploitation

We analyse in this section whether the algorithmic components of SO-QMKP adequately suit the QMKP characteristics, showing profitable problem-knowledge exploitation with regards to other heuristics and general-purpose approaches that do not combine all of them. Concretely, the following characteristics are studied along the optimisation process:

– *Both sides of the search space exploration*: We want to analyse the benefits of addressing the QMKP from the two fronts of action considered by SO-QMKP, feasible and infeasible regions of the search space. To carry out this goal, we include in this study two SO variants that tackle the problem from only one of these fronts. Concretely, SO-feasible will explore only the feasible region by means of divergent step extractions and convergent step insertions, whereas SO-infeasible will wander through the infeasible side with divergent step insertions and convergent step extractions. In this latter approach, the last convergent step always generates a feasible solution.
– *Solution optimisation by FSOM*: The application of local search methods usually consumes a considerable portion of the computational resources allotted to the complete method [23, 24]. Practitioners often have to evaluate the trade-off between improvement and consumption imposed by these methods. Thus, we compare the performance of SO-QMKP with an SO variant that does not apply any

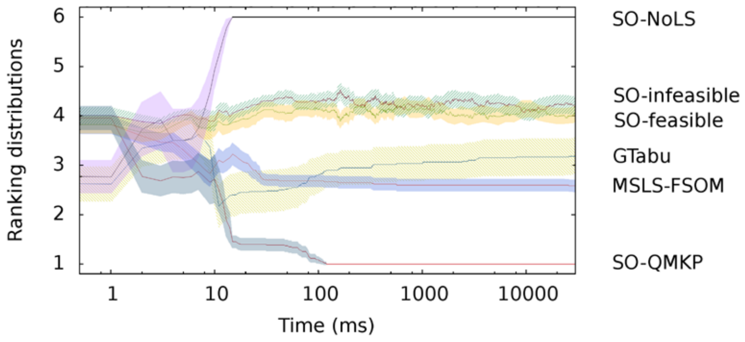


Fig. 4 Friedman ranking distributions of the algorithms

local search mechanism (SO-NoLS). In Sect. 4.2 we noticed that the SO algorithm that applied FSOM was better than the one that did not at the end of the run. In this occasion we analyse the effect along the whole run.

– *Iterative generation of candidate solutions from low and overloaded configurations*: In contrast to our SO strategy, many metaheuristics explore the search space by sampling complete solutions, i.e., feasible configurations where no or little room would be available in knapsacks for more objects. Most general-purpose approaches employ such a strategy. In this study, we will consider as well two (almost-)general-purpose algorithms for the QMKP. As observed in [25], our SO algorithm specifically designed for this problem should perform better than that of other algorithms not explicitly configured for the QMKP. In this case, we consider a tabu search method that starts from a greedy solution (GTabu) (which obtained the best results in [6] among the analysed general-purpose algorithms), and a multi-start approach that repeatedly applies our FSOM procedure to random initial solutions.

Figure 4 shows the Friedman ranking distributions of these five algorithms and our final variant, SO-QMKP, over the runtime. Lines represent the averaged ranking values of the algorithms over the 60 problem instances and areas depict the associated 95 % confidence intervals.

From the graphics shown in Fig. 4, we can conclude the following:

– SO-QMKP is the algorithm that attains the best (lowest) ranking values from 10 ms. Therefore, the combination of its mechanisms clearly produces a profitable synergy.
– Regarding the use of the optimiser FSOM, SO-NoLS gets the worst ranking values from 10 ms and MSLS-FSOM achieves the second position along most of the run. Therefore, we conclude that the improvement procedure makes a clear contribution to compensate for the computational resources it requires. Remarkably, SO-NoLS gets the best ranking values at the beginning of the run. This is due to the fact that the absence of the improvement procedure allows the algorithm begin the discovery of new solutions earlier in the computational process, and hence initially to obtain better results.

- SO-infeasible and SO-feasible appear to perform similarly but much worse than SO-QMKP. This result shows the benefits of addressing the problem from both fronts of action.
- Finally, the comparison between the almost general-purpose approaches, GTabu and MSLS-FSOM, and the SO variants shows that the iterative generation of candidate solutions from incomplete or infeasible configurations is only advantageous when both sides of the search space are considered. This suggests that the operations we have chosen to embed in the SO framework achieve insufficient diversification, when only one side of the search space is considered (SO-feasible and SO-infeasible). However, the progress when both sides are visited (SO-QMKP) is much more advantageous compared to the progress of the general-purpose algorithms.

5 Conclusions

In this work, we have addressed the QMKP with the SO methodology. We have defined critical levels for this problem and have designed specific strategies to exploit the constraint structure by means of effective explorations of solutions in feasible and infeasible regions close to these levels. In addition, we make use of a strategic oscillation multi-neighbourhood (SOM) component that has proved more advantageous than previously proposed local search methods for this problem, and have described code implementations that allow a fast evaluation of new solutions built from insertion, extraction, reallocation and swap operations.

We have empirically analysed a wide variety of heuristics and parameter settings for each of the stages of our SO framework (initial construction, improvement, oscillation, and acceptance), obtaining an algorithm that outperforms the state-of-the-art approaches for this problem. Finally, we have analysed the algorithmic components of the proposal, examining other SO variants and competitive general-purpose methods, to establish that SO-QMKP effectively exploits the domain-knowledge associated with QMKP.

Three interesting avenues present themselves for future research: (1) incorporating tabu search strategies to guide the decisions currently handled by choice rules designed for search paths that terminate at local optimality; (2) adapting our SO framework and methodology to other challenging combinatorial problems, such as the demand-constrained multi-dimensional knapsack problem, and (3) building hybrid metaheuristics combining the proposed SO framework with other approaches that have not previously been joined with strategic oscillation (e.g., artificial bee colony algorithms).

Acknowledgements This work was supported by the research projects TIN2009-07516, TIN2012-35632-C02, TIN2012-37930-C02-01, and P08-TIC-4173.

References

1. Hiley, A., Julstrom, B.: The quadratic multiple knapsack problem and three heuristic approaches to it. In: Proc. of the Genetic and Evolutionary Computation Conference (GECCO'06), pp. 547–552 (2006)

2. Julstrom, B.A.: Greedy, genetic, and greedy genetic algorithms for the quadratic knapsack problem. In: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO'05), pp. 607–614. ACM, New York (2005)
3. Sundar, S., Singh, A.: A swarm intelligence approach to the quadratic multiple knapsack problem. In: ICONIP. LNCS, vol. 6443, pp. 626–633 (2010)
4. Saraç, T., Sipahioglu, A.: A genetic algorithm for the quadratic multiple knapsack problem. In: BVAI. LNCS, vol. 4729, pp. 490–498 (2007)
5. Singh, A., Baghel, A.: A new grouping genetic algorithm for the quadratic multiple knapsack problem. In: EvoCOP. LNCS, vol. 4446, pp. 210–218 (2007)
6. García-Martínez, C., Rodríguez-Díaz, F.J., Lozano, M.: A tabu-enhanced iterated greedy for the quadratic multiple knapsack problem. Eur. J. Oper. Res. **232**(3), 454–463 (2014)
7. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic, Dordrecht (1997)
8. Glover, F., Kochenberger, G., Alidaee, B.: Adaptive memory tabu search for binary quadratic programs. Manag. Sci. **44**(3), 336–345 (1998)
9. Yagiura, M., Iwasaki, S., Ibaraki, T., Glover, F.: A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem. Discrete Optim. **1**(1), 87–98 (2004)
10. Gallego, M., Laguna, M., Martí, R., Duarte, A.: Tabu search with strategic oscillation for the maximally diverse grouping problem. J. Oper. Res. Soc. **64**, 724–734 (2013)
11. Duarte, A., Martí, R., Álvarez, A., Ángel-Bello, F.: Metaheuristics for the linear ordering problem with cumulative costs. Eur. J. Oper. Res. **216**, 270–277 (2012)
12. Glover, F.: Heuristics for integer programming using surrogate constraints. Decis. Sci. **8**(1), 156–166 (1977)
13. Glover, F., Glover, R., McMillan, C.: A heuristic programming approach to the employee scheduling problem and some thoughts on managerial robots. J. Oper. Manag. **4**(2), 113–128 (1984)
14. Glover, F.: Tabu search and adaptive memory programming—advances, applications and challenges. In: Barr, Helgason, Kennington (eds.) Interfaces in Computer Science and Operations Research, pp. 1–75. Kluwer Academic, Dordrecht (1996)
15. Lu, Z., Hao, J.K., Glover, F.: Neighborhood analysis: a case study on curriculum-based course timetabling. J. Heuristics **17**(2), 97–119 (2011)
16. Lozano, M., Glover, F., García-Martínez, C., Rodriguez, F., Martí, R.: Tabu search with strategic oscillation for the quadratic minimum spanning tree. IEE Trans. (2013). doi:10.1080/0740817X.2013.768785
17. Garcia, S., Molina, D., Lozano, M., Herrera, F.: A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. J. Heuristics **15**(6), 617–644 (2009)
18. Zar, J.H.: Biostatistical Analysis. Prentice Hall, New York (1999)
19. Sheskin, D.J.: The Handbook of Parametric and Nonparametric Statistical Procedures. Chapman & Hall/CRC, New York (2003)
20. Iman, R., Davenport, J.: Approximations of the critical region of the Friedman statistic. In: Commun. Stat., pp. 571–595 (1980)
21. Holm, S.: A simple sequentially rejective multiple test procedure. Scand. J. Stat. **6**, 65–70 (1979)
22. Wilcoxon, F.: Individual comparisons by ranking methods. Biomaterials **1**, 80–83 (1945)
23. García-Martínez, C., Lozano, M.: Evaluating a local genetic algorithm as context-independent local search operator for metaheuristics. Soft Comput. **14**(10), 1117–1139 (2010)
24. Molina, D., Lozano, M., García-Martínez, C., Herrera, F.: Memetic algorithms for continuous optimization based on local search chains. Evol. Comput. **18**(1), 27–63 (2010)
25. García-Martínez, C., Lozano, M., Rodriguez, F.J.: Arbitrary function optimization. No free lunch and real-world problems. Soft Comput. **16**(12), 2115–2133 (2012)