Noname manuscript No.
(will be inserted by the editor)

Iterated Responsive Threshold Search for the Quadratic Multiple Knapsack Problem

Yuning Chen · Jin-Kao Hao*

Accepted to **Annals of Operations Research** on 5 Sept. 2014; DOI:10.1007/s10479-014-1720-5

Abstract The quadratic multiple knapsack problem (QMKP) consists in assigning objects with both individual and pairwise profits to a set of limited knapsacks in order to maximize the total profit. QMKP is a NP-hard combinatorial optimization problem with a number of applications. In this paper, we present an iterated responsive threshold search (IRTS) approach for solving the QMKP. Based on a combined use of three neighborhoods, the algorithm alternates between a threshold-based exploration phase where solution transitions are allowed among those satisfying a responsive threshold and a descent-based improvement phase where only improving solutions are accepted. A dedicated perturbation strategy is utilized to ensure a global diversification of the search procedure. Extensive experiments performed on a set of 60 benchmark instances in the literature show that the proposed approach competes very favorably with the current state-of-the-art methods for the QMKP. In particular, it discovers 41 improved lower bounds and attains all the best known results for the remaining instances. The key components of IRTS are analyzed to shed light on their impact on the performance of the algorithm.

Keywords Quadratic multiple knapsack problem · Constrained quadratic optimization · Responsive threshold search · Multiple neighborhood · Heuristics

Yuning Chen
LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
E-mail: yuning@info.univ-angers.fr

Jin-Kao Hao* *(Corresponding author)*
LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
E-mail: hao@info.univ-angers.fr

1 Introduction

Let $N = \{1, 2, ..., n\}$ be a set of n objects and $M = \{1, 2, ..., m\}$ a set of m knapsacks. Each object i ($i \in N$) is associated with a profit p_i and a weight w_i. Moreover, each pair of objects i and j ($1 \leq i \neq j \leq n$) generates a joint profit p_{ij} when they are allocated to the same knapsack. Each knapsack k ($k \in M$) has a weight capacity C_k. The quadratic multiple knapsack problem (QMKP) is to assign objects of N to the knapsacks such that the total profit of the allocated objects is maximized while the weight sum of the objects allocated to each knapsack does not exceed the capacity of the knapsack.

Let x be a $n \times m$ binary matrix such that $x_{ik} = 1$ if object i is allocated to knapsack k, $x_{ik} = 0$ otherwise. Then the QMKP can be formulated as a 0-1 quadratic program:

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

subject to:

$$\sum_{i=1}^{n} x_{ik} w_i \leq C_k, \qquad \forall k \in M \tag{2}$$

$$\sum_{k=1}^{m} x_{ik} \leq 1, \qquad \forall i \in N \tag{3}$$

As a generalization and combination of the well-known multiple knapsack problem [13, 17, 19] and the quadratic knapsack problem [1, 2, 7, 18], the QMKP is known to be NP-hard [12]. Like many knapsack problems [14], the QMKP can be used to formulate a number of real-world problems where resources with different levels of interaction have to be distributed among a set of different tasks, for example, allocating team members to different projects in which member contributions are calculated both personally and in pairs. The QMKP is not to be confused with the quadratic knapsack problem with multiple constraints [24].

Compared with its two composing problems (multiple knapsack problem and quadratic knapsack problem), the QMKP is somewhat less studied in the literature. Yet, given both its theoretical and practical relevance, the QMKP is receiving increasing attention in recent years. In particular, a number of heuristics has been proposed to solve this difficult problem. To our surprise, no exact algorithm can be found in the literature.

The authors of [12] present one of the first studies on the QMKP and proposed three different heuristics: a greedy heuristic which is based on an object value density criterion, a stochastic hill-climbing method which uses a local operator to remove objects from knapsacks and refill them greedily, and a genetic algorithm which employs both specific crossover and heuristic mutation operators. This paper also introduced a set of 60 benchmark instances from 20 existing instances of the quadratic (single-)knapsack problem defined in [1].

These 60 instances were largely employed in the subsequent studies and will also be used in this work for algorithm evaluations and comparisons.

In [20], the authors propose a genetic algorithm where initial solutions are generated randomly and binary tournament is applied for selection. The algorithm uses a dedicated crossover operator which maintains the feasibility of solutions and two mutation operators with different improvement techniques. Experimental evaluation showed the effectiveness of the proposed algorithm.

A steady-state grouping genetic algorithm is presented in [21] where a single child is produced at each generation and replaces the least fit solution in the population. Solutions are encoded as a set of knapsacks. The algorithm uses the greedy heuristic of [12] for population initialization and the binary tournament for selection. Specialized crossover and mutation operators are proposed for solution recombination and diversity maintenance. This algorithm is experimentally compared with the algorithms in [12].

An artificial bee colony algorithm (SS-ABC) is introduced in [23] in which a local search is integrated. The initial food sources are randomly generated. Onlooker bees choose one of the neighbor food sources using binary tournament selection. A scout bee is reconstructed randomly when the associated food source is not improved for a predetermined number of iterations. A local search based on swapping an unassigned object with an assigned object is employed to further improve the solution quality. Computational results showed that the SS-ABC algorithm outperformed the previous approaches.

A memetic algorithm called ALA-EA is proposed in [22] for the QMKP under the name "quadratic multiple container packing problem". ALA-EA uses a network random key encoding scheme and initializes the gene values with a greedy heuristic based on the object density. The chromosomes are decoded using the best-fit heuristic and the resulting solutions are improved by a simple exchange heuristic. Using the real world tournament selection for reproduction, the algorithm generates offspring solutions by the uniform crossover and swap mutation. We show in Section 3.5 a study of this algorithm.

A tabu-enhanced iterated greedy algorithm (TIG-QMKP) is presented very recently for the QMKP in [11]. This algorithm makes alterations between a constructive phase and a destructive phase. The constructive phase reconstructs a partial solution with a greedy method which is followed by a local improvement to ameliorate the solution quality. The destruction mechanism removes a set of objects from the knapsacks by making use of a short-term tabu memory. TIG-QMKP is one of the current best heuristics for the QMKP since it discovered many best known results for the instances in the literature. We will use this heuristic as one of our reference algorithms in the comparative study.

In another very recent paper [10], strategic oscillation is applied to solve the QMKP (SO-QMKP). This approach iteratively applies three stages. In the first stage, an oscillation process explores both the feasible and infeasible regions around a current solution and returns a new solution, after which a local optimization procedure is applied to each new candidate solution to try to get an improved solution in the second stage, and the last stage decides which solution is to be chosen to continue the search with an acceptance criterion. Like TIG-QMKP, this algorithm obtains many best known solutions for the benchmark instances and will be used as another reference for comparison.

In this paper, we present an iterated responsive threshold search algorithm (IRTS) for solving the QMKP with the following main contributions.

- From the perspective of algorithm design, we propose a special responsive mechanism for guiding a threshold process. By incorporating this mechanism, the algorithm makes an original combination between a threshold-based exploration phase and a descent-based improvement phase. The threshold-based exploration phase examines three neighborhoods and accepts new solutions (including deteriorating solutions) as long as their quality satisfies a responsive threshold while the descent-based improvement phase accepts only improving solutions identified in two neighborhoods. When these two phases are found to be stagnating, a specialized perturbation is applied to displace the search to a distant new region.
- From the point of view of computational performance, we assess the proposed algorithm on a set of 60 benchmark instances commonly used in the literature. Computational results show that the proposed approach competes very favorably with the state-of-the-art methods and is able to find an improved lower bound for 41 instances and match the best known solution for the remaining 19 instances.

The rest of the paper is organized as follows. Section 2 describes the details of the proposed algorithm. Section 3 presents a parameter sensitivity analysis, experimental results and comparisons with the state-of-the-art algorithms in the literature. Section 4 studies some key components of our proposed algorithm. Conclusions are drawn in Section 5.

2 An iterated responsive threshold search algorithm for the QMKP

2.1 Main scheme

Basically, our iterated responsive threshold search (IRTS) algorithm alternates between a threshold-based exploration phase (Exploration for short) and a descent-based improvement phase (Improvement for short). Based on three different neighborhoods, the threshold-based exploration phase prospects for good solutions in a large area of the search space. At each iteration of this phase, the algorithm accepts any encountered neighboring solution which may or may not improve over the current solution but must satisfy a quality threshold which is dynamically determined by a special responsive mechanism that is adjusted as a function of the best local optimum found so far. As a complement of this exploratory search phase, the descent-based improvement phase ensures a more focused and directed search such that only improving neighboring solutions are accepted. These two phases are repeated to seek increasingly better local optima until a search stagnation is encountered. The algorithm then triggers a dedicated perturbation phase to displace the search process in a distant zone of the search space from where a new round of Exploration-Improvement cycles is launched.

As shown in Algorithm 1, IRTS starts from an initial solution s (Line 3) generated by the procedure given in Section 2.3. After initializing some global variables like the best solution s^* found so far, the objective value of the best local optimum f_p and the first threshold coefficient r which is used to define the responsive threshold (Lines 4-7), the search enters into the main loop. For each loop, IRTS realizes a threshold-based exploration phase which is composed of L (L is a parameter) calls of the $ThresholdBasedExploration(s, r, N_D, CN_R, CN_E)$ procedure (Lines 10-12, see Section 2.6). For each call, the search iteratively examines three neighborhoods N_D, CN_R, CN_E which are induced by three move operators (i.e., removing an allocated object, reallocating an object and exchanging two objects, see Section 2.4 and 2.5). Any encountered neighboring solution s' is accepted to replace the incumbent solution s as long as the quality (i.e., the objective value) of s' is not worse than a given threshold. This threshold is dynamically tuned and depends on the objective value of the best local optimum (f_p) found so far (as well as a parameter r called threshold ratio). Thanks to this responsive threshold, the search process is expected to explore various zones of the search space without being easily trapped in local optima. At the end of this exploration phase, the search switches to the descent-based improvement phase to intensify the search (Lines 13-21).

During the Improvement phase (see Section 2.7), the search exploits two neighborhoods CN_R, CN_E (i.e., those generated by reallocating an object and exchanging two objects) and replaces the current solution by a first met improving neighboring solution. This phase stops when no improving solutions can be found in the two neighborhoods meaning that a local optimum is reached. If this local optimum has a better objective value than the recorded best objective value f_p, the algorithm updates f_p as well as the threshold ratio r (Lines 15-21) before resuming a new round of Exploration-Improvement phases. Otherwise, if the recorded best local optimum objective value f_p can not be updated for a consecutive W of Exploration-Improvement phases, the search is considered to be trapped in a deep local optimum. In this case, the algorithm switches to a dedicated perturbation phase to make some important changes to the current solution (Lines 23-28) and restarts a new Exploration-Improvement phase with the perturbed solution as its initial solution.

The whole procedure thus iterates the Exploration-Improvement phases, punctuated with the perturbation phase until a prefixed stop condition is verified. This is typically a time cutoff, a maximum number of allowed iterations, or still a maximum number of allowed Exploration-Improvement phases.

Algorithm 1 Pseudo-code of the IRTS algorithm for the QMKP

```
Input: P: a QMKP instance; L: exploration strength; ρ: perturbation strength coefficient;
  W: the number of non-improving attractors visited before strong perturbation;
Output: the best solution s* found so far;
s ← InitialSolution();
s* ← s;            /* s* records the global best solution found during the search */
f_p ← f(s*);       /* f_p records the objective value of the best local optimum */
r ← CalculateRatio(f_p); /* r denotes the threshold ratio, Sect. 2.6 */
w ← 0;        /* set counter for consecutive non-improving local optima */
while stopping condition not reached do
   /* Threshold-based exploration phase using neighborhoods N_D, CN_R and CN_E to ex-
   plore the search space, Sect. 2.6 */
   for i ← 1 to L do
      (s, s*) ← ThresholdBasedExploration(s, r, N_D, CN_R, CN_E) and update the best
      solution found s*;
   end for
   /* Descent-based improvement phase using neighborhoods CN_R and CN_E to improve
   the solution, Sect. 2.7 */
   (s, s*) ← DescentBasedImprovement(s, CN_R, CN_E);
   if f(s) > f_p then
      f_p ← f(s);
      r ← CalculateRatio(f_p);
      w ← 0;
   else
      w ← w + 1;
   end if
   /* Density-based perturbation phase with neighborhood union N_D ∪ CN_R ∪ CN_E, Sect.
   2.8 */
   if w = W then
      s ← DensityBasedPerturbation(s, ρ, N_D ∪ CN_R ∪ CN_E);
      f_p ← f(s);
      r ← CalculateRatio(f_p);
      w ← 0;
   end if
end while
```

2.2 Search space, evaluation function and solution representation

Before presenting the ingredients of the IRTS algorithm, we first define the search space Ω explored by the algorithm, the evaluation function f to measure the quality of a candidate solution and the solution representation.

For a given QMKP instance with its object set $N = \{1, 2, ..., n\}$ and its knapsacks $M = \{1, 2, ..., m\}$, the search space Ω visited by IRTS is composed of all allocations of objects to the knapsacks such that the total weight of the objects allocated to each knapsack does not surpass the capacity of the knapsack. In other words, the IRTS algorithm visits only feasible solutions.

Formally, let $A : N \rightarrow \{0\} \cup M$ be an allocation function of objects to knapsacks ($A(i) = 0$ indicates that object i is not allocated to any knapsack). For each knapsack $k \in M$ with weight w_k and capacity C_k, let $I_k = \{i \in N : A(i) = k\}$ (i.e., I_k is the set of objects allocated to knapsack k). Then the search space is given by:

$$\Omega = \{A : \forall k \in M, \sum_{i \in I_k} w_i \leq C_k\}.$$

For any candidate solution (i.e., an allocation function A) $s \in \Omega$, its quality is evaluated directly by the objective function f of the QMKP. Let p_i be the profit of object i and p_{ij} be the joint profit of two objects i and j. The objective value $f(s)$ is given by the following sum of profits:

$$f(s) = \sum_{k \in M} \sum_{i \in I_k} p_i + \sum_{k \in M} \sum_{i \neq j \in I_k} p_{ij} \tag{4}$$

This function introduces a total order over Ω. Given two solution s_1 and s_2, s_2 is better than s_1 if $f(s_2) > f(s_1)$.

To encode a feasible solution of Ω, we adopt an integer vector $s \in \{0, 1, ..., m\}^n$ where n is the number of objects and m is the number of knapsacks. In this representation, value $s(i) = k$ $(k \in M)$ indicates that object i is allocated to knapsack k while $s(i) = 0$ means that object i is not allocated to any knapsack. This representation was also used in [12].

Notice that a solution $s \in \Omega$ can also be considered as a partition of the set of n objects into $m+1$ groups $\{I_0, I_1, ..., I_m\}$ such that each I_k $(k \in M)$ is the set of objects allocated to knapsack k while I_0 contains the unallocated objects. Hereafter, we will use H to represent the set of allocated objects of N (i.e., $H = \bigcup_{k=1}^{m} I_k$) and $|H|$ the number of allocated objects (i.e., $|H| = n - |I_0|$).

2.3 Initial solution

IRTS constructs an initial solution according to the greedy constructive heuristic, which was first proposed in [12], and subsequently used in several studies [10, 11, 20, 21, 23]. To present this greedy heuristic, we first introduce two basic definitions: *contribution* and *density* of an object.

Definition 1 Given a solution $s = \{I_0, I_1, ..., I_m\}$, the *contribution* of object i $(i \in N)$ to knapsack k $(k \in M)$ with respect to s is given by:

$$VC(s, i, k) = p_i + \sum_{j \in I_k, j \neq i} p_{ij} \tag{5}$$

Definition 2 Given the contribution $VC(s, i, k)$, the *density* of object i $(i \in N)$ with respect to knapsack k $(k \in M)$ is given by:

$$D(s, i, k) = VC(s, i, k)/w_i \tag{6}$$

Given these definitions, at each iteration of the greedy construction algorithm, an unallocated object i $(i \in I_0)$ with the highest density $D(s, i, k)$ and satisfying $w_i + \sum_{j \in I_k} w_j \leq C_k$ is selected and assigned to knapsack k $(k \in M)$. This process is repeated until no more object with $VC(s, i, k) \geq 0$ $(i \in I_0, k \in M)$ can be assigned to any knapsack without violating a capacity constraint.

After each object allocation, updating VC can be realized in $O(n)$ since allocating (inserting) object i into knapsack k increases the contribution of any other object j $(j \in N, j \neq i)$ to insert into knapsack k with the pairwise profit p_{ij}. Given that the initialization procedure can repeat at most n times, the initialization procedure can be realized in time $O(n^2)$ in the worst case.

2.4 Basic move operators and unconstrained neighborhoods

One of the most critical features of local search is the definition of its neighborhood. Typically, a neighborhood is defined by a move operator mv, which transforms a current solution s to generate a neighboring solution s' by some local changes of s. The transition from s to s' is denoted as $s' = s \oplus mv$. Let $MV(s)$ be the set of all possible moves that can be applied to s, then the neighborhood of s induced by mv is given by: $N(s) = \{s' : s' = s \oplus mv, mv \in MV(s)\}$.

Our IRTS algorithm jointly employs three neighborhoods N_R, N_E and N_D which are defined by three basic move operators: *DROP*, *REALLOCATE* and *EXCHANGE*. Operators *REALLOCATE* and *EXCHANGE* have been used for local improvement in other approaches like [10,11], while operator *DROP* is first introduced in this work.

Let $s = \{I_0, I_1, ..., I_m\}$ be a solution. For an object i, let $k_i \in \{0, 1, ..., m\}$ be the knapsack to which the object is allocated. Our move operators are described as follows:

- $DROP(i)$: This move operator removes an already assigned object i from its associated knapsack. The neighborhood defined by this operator is given by:

  $N_D(s) = \{s' : s' = s \oplus DROP(i), i \in N, k_i \in M\}$

  The objective value of the new solution after a *DROP* move can be efficiently calculated as:

$$f(s') = f(s) - VC(s, i, k_i), \quad k_i \in M \tag{7}$$

- $REALLOCATE(i, k)$: This move operator displaces an object i from its current knapsack $k_i \in \{0, 1, ..., m\}$ to another knapsack k ($k \neq k_i, k \neq 0$). In fact, this move operator includes two cases: to allocate an unassigned object i ($k_i = 0$) to a knapsack k ($k \in M$) and to reallocate an assigned object i ($k_i \neq 0$) to a different knapsack k ($k \neq k_i, k \in M$). The (unconstrained) neighborhood N_R induced by this move operator is given by:

  $N_R(s) = \{s' : s' = s \oplus REALLOCATE(i, k), i \in N, k \in M \setminus \{k_i\}\}$

  Given the objective value of solution s, the objective value $f(s')$ of a neighboring solution s' after applying *REALLOCATE* to s can be efficiently calculated as:

$$f(s') = \begin{cases} f(s) + VC(s, i, k) - VC(s, i, k_i), & \text{if} \quad k_i \neq 0 \\ f(s) + VC(s, i, k), & \text{otherwise} \end{cases} \tag{8}$$

- $EXCHANGE(i, j)$: This move operator exchanges a pair of objects (i, j) where 1) one of them is an assigned object and the other is not assigned, or 2) both of them are assigned but belong to different knapsacks. Notice that exchanging objects that are both unassigned or both included in the same knapsack does not change the objective value and thus will not be considered by our exchange move operator. We define $Y = \{Y_1, Y_2, ..., Y_n\}$ be the set of all possible pairs of exchange objects where $Y_i = \{(i,j) : j \in N, i \neq j, k_i \neq k_j\}$ contains all pairs related to object i. The unconstrained neighborhood N_E induced by this move operator is given by:

$$N_E(s) = \{s' : s' = s \oplus EXCHANGE(i,j), (i,j) \in Y\}$$

The objective value of a new solution s' after an *EXCHANGE* move can be conveniently calculated as:

$$f(s') = \begin{cases} f(s) + VC(s,i,k_j) + VC(s,j,k_i) \\ -VC(s,i,k_i) - VC(s,j,k_j) - 2 * p_{ij}, & \text{if } k_i, k_j \in M \\ f(s) + VC(s,i,k_j) - VC(s,j,k_i) - p_{ij}, & \text{if } k_i = 0, k_j \in M \\ f(s) + VC(s,j,k_i) - VC(s,i,k_j) - p_{ij}, & \text{if } k_j = 0, k_i \in M \end{cases} \quad (9)$$

One observes that both *REALLOCATE* and *EXCHANGE* can either improve or deteriorate the quality of the current solution, while *DROP* always decreases the objective value.

A neighboring solution s' can be generated by applying one of the above three move operators. After a solution transition, the algorithm updates the contribution table VC with a time complexity of $O(n)$ since each of the three move operators can be decomposed into one or several object insertion or extraction operations and like object insertion (see Section 2.3), updating VC after an object extraction requires $O(n)$ time at most.

2.5 Constrained neighborhoods and generation of neighboring solutions

The move operators (and their neighborhoods) introduced in the last section do not consider the knapsack constraints. Consequently, the associated neighborhoods may contain unfeasible solutions (violating some knapsack constraints). Since our IRTS algorithm explores only the feasible search space Ω defined in Section 2.2, we introduce two constrained neighborhoods associated with the *REALLOCATE* and *EXCHANGE* operators. These constrained neighborhoods are both more focused and smaller-sized. By using the constrained neighborhoods, the IRTS algorithm tries to avoid generating irrelevant unfeasible neighboring solutions, consequently saves its computing time needed to examine the neighboring solutions and improves its computational efficiency. Notice that when the *DROP* operator is applied to a feasible solution (this is our case), it always generates a feasible solution.

Recall that the *REALLOCATE* operator allocates an unassigned object to a knapsack or reallocates an assigned object to another knapsack. In the general case, its associated neighborhood N_R is of size $|N| \times |M| - |H|$. However, for a given object and a solution, if the weight of the object exceeds the residual capacity of each knapsack of the solution, it is useless to examine any neighboring solution relative to this object. The idea of our constrained reallocate neighborhood is to limit the objects to be considered to a specifically identified subset $X \subseteq N$ such that $|X|$ is as small as possible, and the resulting neighborhood still contains all feasible solutions of the unconstrained neighborhood.

Given a solution $s = \{I_0, I_1, ..., I_m\}$, let $SW = \{SW_1, SW_2, ..., SW_m\}$ be a vector where each $SW_k = \sum_{i \in I_k} w_i$ is the weight sum of the objects in knapsack k of the solution s. The maximum spare or residual space $maxSlack$ among all the knapsacks is given by:

$$maxSlack = \max_{k \in M}\{C_k - SW_k\} \tag{10}$$

Then by defining subset X as $X = \{i \in N : w_i < maxSlack\}$, our constrained neighborhood CN_R induced by $REALLOCATE$ is given by:

$CN_R(s) = \{s' : s' = s \oplus REALLOCATE(i, k), i \in X, k \in M \setminus \{k_i\}\}$.

Obviously, the size of CN_R is $|X| \times |M| - |H|$ which is typically smaller than the size of unconstrained neighborhood N_R ($|N| \times |M| - |H|$). In most cases $maxSlack$ can be updated in $O(1)$ since only two knapsacks are impacted by a 'reallocate' move. One exception is when a move yields a smaller space than $maxSlack$ for the two knapsacks concerned by the operation and one of them originally holds $maxSlack$. In this case, we need to traverse all knapsacks to update $maxSlack$.

Similarly, a constrained neighborhood can be devised for the $EXCHANGE$ operator by exploring the following idea. Given an object $i (i \in I_{k_i})$, if it can not be accommodated by a knapsack k ($k \neq k_i$) even by removing the object with the highest weight, then it is useless to try to exchange this object i with this knapsack. For this purpose, we define a vector $MW = \{MW_1, MW_2, ..., MW_m\}$ where each $MW_k = max_{i \in I_k}\{w_i\}$ stores the maximum weight among all objects in each knapsack. Then we can exclude any neighboring solution which is obtained by exchanging object i ($i \in N$) with any object of knapsack k ($k \in M$) if the following condition holds:

$$w_i > MW_k + (C_k - SW_k) \tag{11}$$

We define all possible pairs of exchangeable objects as $Z = \{Z_1, Z_2, .., Z_n\}$ where $Z_i = \{(i, j) : j \in N, j \neq i, k_j \neq k_i, MW_{k_j} + (C_{k_j} - SW_{k_j}) \geq w_i\}$ contains all pairs related to object i. Our constrained exchange neighborhood $CN_R(s)$ is defined as:

$CN_E(s) = \{s' : s' = s \oplus EXCHANGE(i, j), (i, j) \in Z\}$.

Obviously, the neighborhood CN_E typically has a smaller size than the unconstrained neighborhood N_E since $\forall i \in N, |Z_i| \leq |Y_i|$.

Usually, MW can be updated in $O(1)$ since an 'exchange' move only concerns two knapsacks. One special case happens when object i is to be exchanged with object j in another knapsack k such that j holds the maximum weight MW_k of knapsack k. In this case, we need to traverse the objects of the knapsack k.

2.6 Threshold-based exploration using neighborhoods N_D, CN_R and CN_E

In the context of multiple neighborhood search, there are several ways to combine different neighborhoods, such as *neighborhood union* and *token-ring search* [6,8,15]. One main motivation for considering combination of diverse neighborhoods is to allow the search procedure to continue its search without being easily trapped in local optima. Given that a local minimum for a neighborhood is not necessarily a local optimum for another neighborhood, an algorithm that explores multiple neighborhoods is expected to have more chances to locate better solutions. As it is shown in Section 2.1, our IRTS algorithm alternates between the threshold-based exploration phase and the descent-based improvement phase. These two phases both employ a *neighborhood composition* strategy but differ in the number of neighborhoods they use and their acceptance criterion for solution transitions.

In the threshold-based exploration phase, both improving and deteriorating solutions are allowed in order to favor a large exploration of the search space. This phase is based on the three constrained neighborhoods N_D, CN_R and CN_E and adapts the Threshold Accepting heuristic ([4,5]) to define its solution accepting criterion. Specifically, the three neighborhoods are examined iteratively in a token-ring search $N_D \rightarrow CN_R \rightarrow CN_E \rightarrow N_D \rightarrow CN_R ...$ (see Algorithm 2). For each neighborhood under consideration, neighboring solutions are examined in a random order. The acceptance of each sampled solution is subject to a responsive threshold denoted by T. Precisely, a neighboring solution s' is accepted to replace the incumbent solution if its objective value is no worse than the given threshold T, i.e., $f(s') \geq T$.

Our responsive threshold is critical to the performance of the search algorithm. A too small threshold (i.e., $f_p - T$ is large) makes the search similar to a pure random search while a too large threshold (i.e., $f_p - T$ is small) weakens the capability of the search to escape local optima. In our case, the responsive threshold T is determined according to the recorded best local optimum objective value f_p and a threshold ratio r: $T = (1 - r) \times f_p$. Thus T increases when f_p increases and r decreases (r itself is inversely proportional to f_p). Since f_p is updated during the search, the threshold T is dynamically evolving too.

Notice that we do not give r a fixed value since f_p changes (with increasing values) during the search course and moreover, the range of f_p can be quite different and large for different problem instances. To determine an appropriate value of r, we design an inverse proportional function with respect to f_p as follows (a, b and c are empirically fixed coefficients):

$$r = 1/(a * (f_p/10000) + b) + c \tag{12}$$

This function is monotonically decreasing which means r strictly decreases when f_p enlarges. Empirically, we set $a = 16.73$, $b = 76.56$ and $c = 0.0021$ for the instances we used. These values are identified in the following manner. We selected three different problem instances whose best objective values range from small to large and we tuned manually for each of these three instances a value of r which is kept unchanged during the search course. We then use the best objective value as the f_p and we obtain three pairs of (f_p, r) values. The value of a, b and c are then decided by solving the simultaneous equations. In our IRTS algorithm, r is recalculated each time f_p is updated.

Algorithm 2 Pseudo-code of the threshold-based exploration phase

```
Input:
    s: A feasible solution ;
    L: Exploration strength;
    f_p: The objective value of the best local optimum;
    r: The threshold ratio;
Output: A new solution s and the best solution s*;
for i ← 1 to L do
    (s, s*) ← ThresholdSearch(s, f_p, r, N_D);
    (s, s*) ← ThresholdSearch(s, f_p, r, CN_R);
    (s, s*) ← ThresholdSearch(s, f_p, r, CN_E);
end for
Return s and s*;
```

2.7 Descent-based improvement using neighborhoods CN_R and CN_E

After each threshold-based exploration phase, the IRTS algorithm continues with a descent-based improvement phase which is based on the constrained 're-allocate' neighborhood CN_R and constrained 'exchange' neighborhood CN_E. The aim of this phase is to attain a local optimum (as good as possible) in both CN_R and CN_E starting from the solution s returned by the previous threshold-based exploration phase. To this end, the algorithm iteratively explores CN_R and CN_E in a token-ring way $CN_R \rightarrow CN_E \rightarrow CN_R \rightarrow CN_E ...$. For each iteration, a neighboring solution s' is picked at random from the neighborhood under consideration and replaces the incumbent solution s if s' is better than s (i.e., $f(s') > f(s)$). Since this search phase aims to find solutions of increasing quality, the *DROP* neighborhood N_D is not applicable (since the *DROP* operator always decreases the objective value).

This process stops when no more improving neighboring solution can be found in CN_R and CN_E to update s which corresponds to a local optimum. At this point, s is checked to verify whether its objective value is better than the best recorded objective value f_p. If this is the case, we update f_p and reset to 0 the counter for consecutive non-improving Exploration-Improvement phases w (see Algorithm 1, Sect. 2.1). Otherwise, w is increased by 1 and the search switches back to the next threshold-based exploration phase. When w reaches its maximum value W, the search is considered to be stagnating.

To unblock the situation, the algorithm calls for a strong perturbation which changes significantly the incumbent solution before restarting a new round of threshold-based exploration and descent-based improvement phases.

2.8 Density-based perturbation phase using N_D, CN_R and CN_E

As described previously, the threshold-based exploration phase may accept deteriorating solutions (subject to the responsive threshold) and thus provides a form of controlled diversification which allows the search to escape some local optima. However, this mechanism may not be sufficient to escape deep local optima due to the quality limit fixed by the threshold. To establish a more global form of diversification, and thereby reinforce the capacity of the algorithm to explore unexplored areas in the search space, we introduce a dedicated perturbation strategy (based on object density) which is triggered when the search is stagnating (i.e., when the best local optimum has not been improved for W consecutive Exploration-Improvement phases).

Recall that the density of an object is defined as its contribution divided by its weight (see Section 2.3). Intuitively, high density objects are preferred to be included in the solution since their contributions are high while their weights are relatively low. The general idea of our perturbation strategy is to force a set of objects with the least densities to change their status (which means to drop them, to reallocate them or to exchange them against other objects) with the perspective of including objects with higher densities. One notices that the perturbation may be strong in the sense that the deterioration of the current solution after perturbation is not controlled.

Specifically, we first sort all the allocated objects according to the ascending order of their densities $D(s,i,k)$ $(i \in H, k \in M)$ and then force the first PL objects to change their status. PL is called perturbation strength and is calculated by:

$$PL = \rho * |H| \tag{13}$$

where ρ (a parameter) is the perturbation strength coefficient and $|H|$ represents the number of allocated objects in s. Precisely, for each of the PL objects, we examine the *union* of the three neighborhoods $N_D \cup CN_R \cup CN_E$ relative to this object and select the best neighboring solutions to replace the current solution. This transition may lead to an improving or deteriorating solution. The status of the displaced objects are prohibited to be changed again during the perturbation phase. The rationale of using the combined neighborhood is that there is no absolute dominance of one neighborhood over another when the best-improvement strategy is used for solution transitions. Though $DROP$ moves always decrease the objective value, it could generate the most profitable move when no improving move exists for the other two neighborhoods.

Table 1 Parameter settings of the IRTS algorithm

Parameters	Description	Value	Section
L	diversification strength	30	2.6
ρ	perturbation strength coefficient	0.1	2.7
W	max number of consecutive non-improving Exploration-Improvement phases	20	2.8

3 Computational Experiments

In this section, we carry out extensive experiments on a set of 60 well-known QMKP benchmark instances, in order to evaluate the performance of the proposed algorithm. These instances are characterized by their number of objects $n \in \{100, 200\}$, density $d \in \{0.25, 0.75\}$ (i.e. number of non-zero coefficients of the objective function divided by $n(n+1)/2$), and the number of knapsacks $m \in \{3, 5, 10\}$. For each instance, the capacities of the knapsacks are set to 80% of the sum of object weights divided by the number of knapsacks. These instances are built from the quadratic knapsack instances introduced in [1] which can be download from: http://cedric.cnam.fr/ soutif/QKP/QKP.html.

Our IRTS algorithm is coded in C++[1] and compiled using GNU g++ on an Intel Xeon E5440 processor (2.83GHz and 2GB RAM) with -O3 flag. When solving the DIMACS machine benchmarks[2] without compiler optimization flag, the run time on our machine is 0.44, 2.63 and 9.85 seconds respectively for graphs r300.5, r400.5 and r500.5. All computational results were obtained with the parameter values shown in Table 1 which are identified with the analysis of Section 3.1. The exact experimental conditions are provided in the corresponding sections below when we present the computational results and comparisons.

For our experimental studies, we apply a number of statistical tests including Friedman test, Post-hoc test and Wilcoxon signed-rank test [3]. The Friedman test is a non-parametric statistical test which aims to detect statistical differences in treatments across multiple test attempts. When a difference is detected by this test, a Post-hoc test can be additionally applied to decide which groups are significantly different from each other. Contrary to the Friedman test, the Wilcoxon signed-rank test is a paired difference test which is useful to compare two related samples, matched samples or repeated measurements on a single sample.

3.1 Analysis of IRTS parameters

IRTS requires three parameters: L (exploration strength), W (the number of consecutive non-improving attractors) and ρ (perturbation strength coefficient). In this section, we show a statistical analysis of these parameters. We

[1] Our best results are available at `http://www.info.univ-angers.fr/pub/hao/qmkp.html`. The source code of the IRTS algorithm will also be available.

[2] dfmax: ftp://dimacs.rutgers.edu/pub/dsj/clique/

Table 2 Post-hoc test for solution sets obtained by varying ρ

$\rho =$	0.05	0.1	0.15	0.2
0.1	0.8853			
0.15	0.9364	0.4276		
0.2	0.0128	0.0004	0.1152	
0.25	0.2497	0.0009	0.1833	0.9996

Table 3 Statistical results of varying ρ. $Tot.$ indicates the sum of average objective values over 60 instances. $\#Bests$ denotes the number of best solutions found by each parameter setting. The best result of each row is indicated in bold.

$\rho =$	0.05	0.1	0.15	0.2	0.25
$Tot.$	5018320	**5018630**	5018220	5017340	5017090
$\#Bests$	36	**52**	49	40	44

test for each IRTS parameter a set of potential values with the other parameters fixed to their default values from Table 1. We run our algorithms 10 times on each instance with a time limit of 5 seconds for small instances ($n = 100$) and 30 seconds for large instances ($n = 200$). These stopping criteria are also used in [11]. The average solution values over the 10 runs are considered for each instance and the corresponding parameter. Specifically, we test L in the range $[10, 50]$ with a step size of 10, W in the range $[5, 25]$ with a step size of 5, and ρ in the range $[0.05, 0.25]$ with a step size of 0.05.

We applied the Friedman test to check whether the performance of IRTS varies significantly in terms of its average solution values when changing the value of a single parameter. The Friedman test finds no significant difference with a *p-value* of 0.6856 and 0.8306 respectively when L and W vary in their given range, which means that the IRTS algorithm is not sensitive to these two parameters. On the contrary, a statistical difference is observed when ρ varies in the given range with a *p-value* of 0.0003461 which indicates that this is a sensitive parameter. Therefore, a Post-hoc test is performed to examine the statistical difference between each pair of parameter settings and the results are displayed in Table 2. From Table 2, we can see that there are three pairs of parameter settings presenting significant difference (with *p-value* < 0.05) where two of them are related to $\rho = 0.1$ (i.e. (0.1,0.2) and (0.1,0.25)). Table 3 shows that $\rho = 0.1$ produces the best average objective values and the highest number of best solutions.

According to this analysis, we adopt $\rho = 0.1, L = 30, W = 20$ as the default setting of the IRTS algorithm.

3.2 Computational results of the IRTS algorithm

In this section, we report the results obtained by our IRTS algorithm on the set of 60 instances under two different stopping criteria. The first stop criterion is a short time limit where for each run 5 and 30 seconds are allowed for instances with 100 and 200 objects, respectively. This stop criterion is similar to the truncating time limit used in [10, 11, 23]. The second criterion is a long time limit where we use a cutoff time of 15 and 90 seconds for instances with 100 and 200 objects, respectively. By using the second stopping criterion, we investigate the behavior of our IRTS algorithm when more time is available. For both criteria and for each instance, our algorithm is executed 40 times. Hereafter, we use IRTS_Short and IRTS_Long to denote respectively IRTS running with the short and long time limit.

Table 4 shows the results of our IRTS algorithm. Columns 1 to 5 give the characteristics of the instances, including the number of objects (n), density (d), number of knapsacks (m), instance identity number (I) and capacity of each knapsack (C). Column 6 (f_{bk}) lists the best known results which are compiled from results reported in [10, 11, 23]. Columns 7 to 16 show our own results obtained under both short time limit and long time limit: the overall best objective value (f_{best}), the average of the 40 best objective values (f_{avg}), the standard deviation of the 40 best objective values (sd), the number of times for reaching the f_{best} (hit) and the earliest CPU time in seconds over the runs when the f_{best} value is reached (CPU).

From Table 4, we observe that for *all* these 60 instances, our IRTS algorithm can reach or improve the best known results within the short time limit. Moreover, for 40 out of 60 cases, even our average results (f_{avg}) are better or equal to the best known results. Finally, IRTS_Short has a perfect successful rate ($hit = 40/40$) for 6 out of 60 instances, meaning that one run suffices for IRTS to attain its best solution for these cases.

When a long time limit is available, our IRTS algorithm reaches a still better performance. In particular, IRTS_Long finds 17 improved best lower bounds with respect to IRTS_Short (starred in Table 4). The average solution values of IRTS_Long are also better than those of IRTS_Short for 52 cases out of 60 (86.7%). Moreover, IRTS_Long decreases the average value of the standard deviations (sd) from 68.17 (of IRTS_Short) to 49.87, and increases the average number of hit from 13.8 (of IRTS_Short) to 18.53.

Additionally, we apply two statistical tests to compare both the best results and the average results of IRTS_Short, IRTS_Long as well as the best known results. The first test is the non-parametric Friedman test performed on the best results obtained by IRTS_Short, IRTS_Long and the best known results (BKR). The resulting *p-value* is 1.44e-15 which clearly indicates a significant difference. The post-hoc test results shown in Table 5 reveal that the differences lie in the pair of IRTS_Short and BKR, and the pair of IRTS_Long and BKR. Though a *p-value* of 0.0946 does not disclose a significant statistical difference for the pair of IRTS_Short and IRTS_Long when testing their best results, a Wilcoxon test performed on the average results (see Table 6) clearly shows the average results of IRTS_Long are better than those of IRTS_Short for a majority of cases. Indeed, the resulting *p-value* is 3.60e-10, and the sum of the positive ranks are significantly larger than that of the negative ranks. To provide more statistical information, we also list in Table 6 the minimum value, the first quartile, the median, the mean, the third quartile and the maximum value. We can observe that all these values of IRTS_Long are larger than or equal to those of IRTS_Short which confirms the dominance of IRTS_Long.

Table 4 Performance of IRTS on the 60 benchmark instances with short time condition (5 seconds per run for instances with 100 objects and 30 seconds per run for instances with 200 objects) and long time condition (15 seconds per run for instances with 100 objects and 90 seconds per run for instances with 200 objects). A value with an asterisk indicates an improved lower bound obtained by IRTS_Long over IRTS_Short.

Instance					f_{bk}	IRTS_Short					IRTS_Long				
n	d	m	I	C		f_{best}	f_{avg}	sd	hit	CPU(s)	f_{best}	f_{avg}	sd	hit	CPU(s)
100	25	3	1	688	29234	29286	29286.00	0.00	40	0.02	29286	29286.00	0.00	40	0.02
100	25	3	2	738	28491	28491	28491.00	0.00	40	0.02	28491	28491.00	0.00	40	0.02
100	25	3	3	663	27179	27179	27175.40	9.00	33	0.21	27179	27179.00	0.00	40	0.21
100	25	3	4	804	28593	28593	28593.00	0.00	40	0.09	28593	28593.00	0.00	40	0.09
100	25	3	5	723	27892	27892	27889.42	11.64	38	0.06	27892	27892.00	0.00	40	0.06
100	25	5	1	413	22509	22581	22489.50	39.47	3	0.27	22581	22530.68	36.22	10	0.28
100	25	5	2	442	21678	21678	21643.75	28.70	3	1.68	21704*	21667.00	10.71	1	9.78
100	25	5	3	398	21188	21239	21210.80	34.97	23	0.33	21239	21235.95	13.73	38	0.26
100	25	5	4	482	22181	22181	22178.50	12.61	31	0.26	22181	22180.90	0.44	38	0.28
100	25	5	5	434	21669	21669	21625.40	35.15	6	1.06	21669	21656.42	18.53	23	1.05
100	25	10	1	206	16118	16221	16180.00	27.01	1	3.11	16221	16200.53	13.43	4	3.25
100	25	10	2	221	15525	15700	15618.82	54.86	8	0.43	15700	15665.65	42.73	22	0.44
100	25	10	3	199	14773	14927	14820.95	33.62	1	4.97	14927	14852.00	27.88	2	4.91
100	25	10	4	241	16181	16181	16178.30	8.05	33	0.08	16181	16181.00	0.00	40	0.08
100	25	10	5	217	15150	15326	15249.35	43.35	8	0.08	15326	15293.00	38.61	22	0.08
200	25	3	1	1381	101100	101445	101414.00	32.94	1	13.77	101471*	101441.00	8.37	1	65.08
200	25	3	2	1246	107958	107958	107958.00	0.78	39	0.18	107958	107958.00	0.00	40	0.18
200	25	3	3	1335	104538	104575	104544.00	11.49	1	21.80	104589*	104559.00	16.72	5	55.50
200	25	3	4	1413	99559	100098	100098.00	0.00	40	0.98	100098	100098.00	0.00	40	1.33
200	25	3	5	1358	102049	102311	102307.00	3.36	13	2.93	102311	102310.00	2.14	27	2.81
200	25	5	1	828	74922	75596	75511.48	36.90	1	4.84	75623*	75554.10	32.47	4	33.45
200	25	5	2	747	79506	80033	79955.42	65.74	13	4.17	80033	80023.40	21.88	33	4.14
200	25	5	3	801	77700	78043	78005.90	40.60	21	5.03	78043	78028.95	28.46	32	5.02
200	25	5	4	848	73327	74111	74011.22	59.60	2	22.20	74140*	74061.29	40.70	1	57.02
200	25	5	5	815	76022	76610	76544.85	59.83	12	1.83	76610	76597.62	20.69	29	1.41
200	25	10	1	414	51413	52259	52032.70	122.34	2	20.84	52293*	52158.50	76.56	1	72.91
200	25	10	2	373	54116	54746	54582.20	87.53	1	18.72	54830*	54666.25	57.24	1	39.00
200	25	10	3	400	52841	53646	53535.57	73.44	1	8.80	53661*	53588.28	40.10	1	65.28
200	25	10	4	424	50221	51176	50951.64	85.59	1	10.88	51297*	51078.20	69.96	1	56.17
200	25	10	5	407	52651	53616	53482.40	55.98	1	26.42	53621*	53532.24	41.60	1	33.13
100	75	3	1	669	69977	69977	69975.90	6.56	39	0.14	69977	69977.00	0.00	40	0.14
100	75	3	2	714	69504	69504	69491.45	14.85	23	0.01	69504	69499.60	10.36	34	0.01
100	75	3	3	686	68832	68832	68831.50	3.28	39	0.12	68832	68832.00	0.00	40	0.12
100	75	3	4	666	70028	70028	70028.00	0.00	40	0.01	70028	70028.00	0.00	40	0.01
100	75	3	5	668	69692	69692	69687.50	12.16	35	0.01	69692	69692.00	0.00	40	0.01
100	75	5	1	401	49363	49421	49327.80	78.65	7	0.06	49421	49365.98	61.80	9	0.06
100	75	5	2	428	49316	49365	49340.80	11.30	4	0.15	49365	49350.60	10.70	13	0.16
100	75	5	3	411	48495	48495	48495.00	0.00	40	0.02	48495	48495.00	0.00	40	0.03
100	75	5	4	400	50246	50246	49934.50	144.30	7	0.80	50246	50141.50	169.68	29	0.80
100	75	5	5	400	48752	48753	48732.80	24.69	10	0.74	48753	48749.10	9.91	23	0.75
100	75	10	1	200	29931	30290	30159.38	101.14	6	0.91	30296*	30240.20	68.34	1	5.69
100	75	10	2	214	30980	31101	31030.35	42.35	2	3.84	31207*	31095.80	50.10	3	5.84
100	75	10	3	205	29730	29908	29868.53	37.36	7	0.20	29908	29894.75	19.21	20	0.19
100	75	10	4	200	31663	31762	31671.34	44.95	5	1.13	31762	31706.50	42.98	13	1.08
100	75	10	5	200	30229	30507	30408.50	76.99	3	4.27	30507	30458.50	26.71	6	4.38
200	75	3	1	1311	270718	270718	270627.00	156.93	19	2.40	270718	270685.00	81.17	29	2.38
200	75	3	2	1414	257090	257288	257232.00	111.85	22	1.59	257288	257273.00	58.87	33	1.58
200	75	3	3	1342	270069	270069	269887.00	212.10	16	0.95	270069	269926.00	189.44	21	0.96
200	75	3	4	1565	246882	246993	246651.00	389.98	6	7.48	246993	246877.00	161.74	15	7.52
200	75	3	5	1336	279598	279598	279570.00	66.56	31	1.59	279598	279570.00	66.56	31	1.59
200	75	5	1	786	184909	185353	184854.00	235.13	1	29.05	185493*	184904.00	251.61	1	36.05
200	75	5	2	848	174682	174836	174649.00	90.26	1	5.14	174836	174688.00	80.04	1	5.21
200	75	5	3	805	186526	186753	186591.00	100.00	1	16.26	186774*	186674.00	81.19	1	52.13
200	75	5	4	939	166584	166990	166619.00	199.02	1	9.35	166990	166747.00	126.66	1	9.54
200	75	5	5	801	193084	193310	193180.00	116.40	1	29.56	193310	193217.00	111.03	6	28.95
200	75	10	1	393	112354	113103	112652.00	153.91	1	16.65	113139*	112809.00	169.70	1	60.75
200	75	10	2	424	105151	105807	105393.00	185.17	1	8.79	105807	105437.00	161.37	1	8.75
200	75	10	3	402	113869	114567	114280.00	121.61	1	4.80	114596*	114367.00	96.64	1	84.14
200	75	10	4	469	98252	99075	98756.40	143.42	1	28.25	99106*	98851.55	103.44	1	74.50
200	75	10	5	400	116513	117309	116809.00	135.07	1	16.34	117309	116947.00	123.62	1	16.28

Table 5 Post-hoc test for best objective values of IRTS computational results along with best known results

	BKR	IRTS_Short
IRTS_Short	4.75e-10	
IRTS_Long	0.00	9.46e-02

Table 6 Statistical data and Wilcoxon test of IRTS computational results between IRTS_Long and IRTS_Short. The best result of each column is indicated in bold.

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	R+	R-	p-value	Diff?
IRTS_Short	14820	29720	61710	83640	104800	279600				
IRTS_Long	**14850**	**29740**	**61750**	**83680**	104800	279600	1378	0	3.60e-10	Yes

3.3 Comparative results with a truncating time limit per instance

In order to further evaluate our IRTS algorithm, we compare it with three recent QMKP algorithms in the literature.

- An artificial bee colony algorithm (SS-ABC) [23]. The tests were performed on a Linux based 3.0 GHz Core 2 duo system with 2 GB RAM. According to SPEC - Standard Performance Evaluation Corporation (www.spec.org), our computer is slightly faster than the reference machine of [23] with a factor of 1.05. The authors set 300 generations for SS-ABC and run it 40 times for each instance.
- A tabu-enhanced iterated greedy algorithm (TIG-QMKP) [11]. The evaluations were performed on a computer with a 2.8GHz Intel Core i7-930 processor with 12 GB RAM. Solving the DIMACS machine benchmarks on this computer requires 0.40, 2.40 and 9.06 seconds respectively for graphs r300.5, r400.5 and r500.5[3] without compilation optimization flag. Compared to the results achieved by our machine (see Section 3), this computer is faster than ours with a factor of around 1.1. The authors reported the results obtained by running their algorithms 40 times for each instance and truncating each run at the mean time indicated in [23] for each instance.
- A strategic oscillation algorithm (SO-QMKP) [10]. The same machine and testing protocols as those of TIG-QMKP [11] are used.

To perform a fair comparison, we run our algorithms 40 times on each of the 60 instances and stop each run at the indicated time which was first reported in [23] and adopted in [10,11] for comparisons. Table 7 summarizes the results of the IRTS algorithm along with those reported by the three reference algorithms. We list in the table the characteristics of the instances including the truncating time for each instance (column $Time$), the previous best known results (f_{bk}) and for each algorithm, the overall best objective value (f_{best}) and the average of the 40 best objective values (f_{avg}).

From Table 7, one observes that our IRTS algorithm always attains the best known results or finds improved results for all 60 instances. Specifically, IRTS

[3] We thank the authors of [11] for providing us with these values.

Table 7 Comparative results of IRTS with respect to SS-ABC, TIG-QMKP and SO-QMKP. The truncating time for each instance was first reported by SS-ABC and then used for TIG-QMKP and SO-QMKP. 'The best results of the four compared algorithms are indicated in bold.

Instance					f_{bk}	SS-ABC [23]		TIG-QMKP [11]		SO-QMKP [11]		IRTS	
n	d	K	I	Time		f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
100	25	3	1	3.80	29234	29139	28753.00	29138	28894.60	29234	29159.25	**29286**	**29276.80**
100	25	3	2	3.69	28491	28443	28004.00	28473	28224.35	**28491**	28467.00	**28491**	**28491.00**
100	25	3	3	2.87	27179	26901	26585.33	27013	26905.40	**27179**	27155.35	**27179**	**27159.15**
100	25	3	4	3.74	28593	28568	28109.03	**28593**	28573.60	**28593**	28570.40	**28593**	**28591.72**
100	25	3	5	3.57	27892	27849	27073.67	**27892**	27721.68	**27892**	27811.53	**27892**	**27880.60**
100	25	5	1	2.70	22509	22390	22117.12	22264	22126.00	22509	22337.43	**22581**	**22463.43**
100	25	5	2	2.93	21678	21584	21224.03	21580	21430.38	**21678**	21540.35	**21678**	**21614.00**
100	25	5	3	2.40	21188	21093	20771.12	21100	21015.03	21188	21104.75	**21239**	**21170.90**
100	25	5	4	3.26	22181	22178	21767.50	22180	22043.00	**22181**	22136.00	**22181**	**22168.95**
100	25	5	5	3.18	21669	21301	20875.47	**21669**	21397.58	**21669**	21462.88	**21669**	**21606.80**
100	25	10	1	1.42	16118	15953	15573.65	16118	15863.00	16065	15886.58	**16213**	**16144.13**
100	25	10	2	1.84	15525	15487	14896.35	15525	15398.43	15510	15359.95	**15700**	**15544.30**
100	25	10	3	1.58	14773	14339	14027.83	14773	14554.00	14663	14568.38	**14860**	**14757.24**
100	25	10	4	2.10	16181	15807	15397.00	**16181**	16089.95	16159	16013.00	**16181**	**16160.00**
100	25	10	5	1.82	15150	14719	14376.80	15150	15023.45	15130	15021.33	**15326**	**15187.25**
200	25	3	1	23.99	101100	100662	100103.02	100218	100056.23	101100	100653.50	**101445**	**101410.00**
200	25	3	2	18.61	107958	**107958**	107545.20	107787	107644.98	107805	107607.15	**107958**	**107957.00**
200	25	3	3	29.85	104538	104521	104006.98	104479	104251.50	104538	104271.68	**104575**	**104544.00**
200	25	3	4	38.93	99559	98791	98344.32	98896	98557.40	99559	99003.63	**100098**	**100098.00**
200	25	3	5	29.22	102049	102049	101406.48	101973	101635.43	102041	101667.73	**102311**	**102307.00**
200	25	5	1	19.88	74922	74922	74132.95	74239	73977.78	74559	74237.40	**75596**	**75491.48**
200	25	5	2	16.75	79506	79506	79073.32	79480	79234.28	79400	79153.55	**80033**	**79914.90**
200	25	5	3	22.86	77700	77607	77069.52	77700	77420.50	77632	77452.25	**78043**	**77992.93**
200	25	5	4	28.07	73327	73181	72607.25	73173	72477.65	73327	72884.03	**74111**	**73999.22**
200	25	5	5	20.74	76022	76022	75455.98	75884	75693.18	75996	75751.38	**76610**	**76532.10**
200	25	10	1	10.37	51413	49883	49079.47	51413	50845.78	51323	50862.70	**52115**	**51862.20**
200	25	10	2	8.48	54116	53298	51831.55	54116	53608.45	53975	53649.03	**54716**	**54454.75**
200	25	10	3	11.15	52841	52281	51324.28	52735	52456.28	52841	52337.73	**53646**	**53401.80**
200	25	10	4	12.83	50221	49210	48190.60	50221	49656.40	50190	49802.43	**51176**	**50832.48**
200	25	10	5	10.99	52651	51921	51437.97	52651	52328.38	52470	52211.58	**53568**	**53406.60**
100	75	3	1	2.07	69977	69721	69373.00	**69977**	69936.05	69935	69925.73	**69977**	**69970.13**
100	75	3	2	1.86	69504	69462	69041.00	**69504**	69442.78	**69504**	69442.48	**69504**	**69465.50**
100	75	3	3	1.86	68832	68635	67960.05	68811	68811.00	**68832**	68812.58	**68832**	**68823.00**
100	75	3	4	1.88	70028	69986	69687.68	**70028**	70019.88	**70028**	70028.00	**70028**	**70028.00**
100	75	3	5	2.12	69692	69679	69136.40	**69692**	69638.48	69653	69640.53	**69692**	**69674.60**
100	75	5	1	2.07	49363	4922	48937.47	49345	49218.10	49363	49197.53	**49421**	**49310.80**
100	75	5	2	1.96	49316	49313	48908.05	49316	49081.53	49305	49137.38	**49365**	**49311.80**
100	75	5	3	1.71	48495	48472	47874.50	**48495**	48327.48	**48495**	48287.88	**48495**	**48479.60**
100	75	5	4	1.83	50246	50199	50017.93	49866	49866.00	**50246**	**50025.03**	**50246**	49935.17
100	75	5	5	2.01	48752	48710	48409.75	48752	48619.60	48752	48653.18	**48753**	**48689.20**
100	75	10	1	1.22	29931	29875	29429.20	29877	29548.68	29931	29788.48	**30290**	**30023.00**
100	75	10	2	1.45	30980	30939	30697.80	30980	30832.15	30973	30829.05	**31079**	**30909.93**
100	75	10	3	1.30	29730	29465	28983.78	29695	29439.95	29730	29519.48	**29908**	**29794.85**
100	75	10	4	1.40	31663	31663	31218.85	31550	31333.45	31587	31392.48	**31682**	**31576.68**
100	75	10	5	1.42	30229	30219	29736.47	30096	29895.40	30229	29918.70	**30465**	**30211.50**
200	75	3	1	14.11	270718	269736	267117.92	**270718**	270525.90	**270718**	**270617.48**	**270718**	270518.00
200	75	3	2	16.27	257090	256195	253916.75	257090	256764.98	257026	256852.30	**257288**	**257117.00**
200	75	3	3	11.87	270069	268890	267079.03	**270069**	**269974.03**	**270069**	269955.03	**270069**	269796.00
200	75	3	4	30.64	246882	246205	244618.40	246704	246356.53	246882	246473.13	**246993**	**246658.00**
200	75	3	5	10.46	279598	279490	276605.00	**279598**	**279572.30**	**279598**	279562.43	**279598**	279548.00
200	75	5	1	12.34	184909	184448	183046.65	184909	184500.80	184822	184529.00	**185221**	**184801.00**
200	75	5	2	12.34	174682	173575	171738.85	174682	174239.48	174682	174267.00	**174836**	**174499.00**
200	75	5	3	12.10	186526	185107	185059.52	186443	186170.68	186526	186216.75	**186678**	**186510.00**
200	75	5	4	27.03	166584	165273	164042.20	166358	166159.55	166584	166165.38	**166990**	**166602.00**
200	75	5	5	14.06	193084	192764	190474.27	193084	192712.25	193053	192702.25	**193253**	**193133.00**
200	75	10	1	7.64	112354	111000	109624.73	112262	111889.75	112354	112043.68	**112834**	**112443.00**
200	75	10	2	9.96	105151	103540	102603.18	105092	104669.83	105151	104781.50	**105807**	**105241.00**
200	75	10	3	8.04	113869	112509	111388.20	113868	113510.55	113869	113563.08	**114567**	**114084.00**
200	75	10	4	14.81	98252	96859	95681.70	98252	97807.73	98028	97747.55	**99006**	**98664.60**
200	75	10	5	8.21	116513	115125	113909.60	116513	115856.30	116298	115807.53	**117092**	**116649.00**

Table 8 Wilcoxon test for results obtained with truncating time limit

Algorithm Pair	Best Results				Average Results			
	R+	R-	p-value	Diff?	R+	R-	p-value	Diff?
IRTS vs SS-ABC	1770	0	2.45e-11	Yes	1829	1	1.76e-11	Yes
IRTS vs TIG-QMKP	1176	0	1.68e-09	Yes	1803	27	6.43e-11	Yes
IRTS vs SO-QMKP	1035	0	5.36e-09	Yes	1714	56	4.01e-10	Yes
IRTS vs BKR	861	0	2.52e-08	Yes	-	-	-	-

improves the best known result for 41 out of 60 instances (68.3%) and reaches the previous best known solutions for the remaining 19 instances. Recall that the best known results (column f_{bk}) are the best objective values extracted from the columns f_{best} of the three reference algorithms. One can confirm that our IRTS algorithm competes very favorably with these reference algorithms in terms of the best solution found. Considering the average results, our IRTS algorithm are able to attain significantly better f_{avg} values compared to the reference algorithms. Compared to SS-ABC, IRTS holds better average results for all the instances except one case (100-75-5-4). Compared to TIG-QMKP, IRTS obtains better average results for 57 out of 60 instances. Compared to SO-QMKP, IRTS attains 55 better average results, 1 equal average result and 3 worse average results.

In order to assess the validity of our conclusion, we apply the Wilcoxon test with a significance factor of 0.05 for pairwise comparison between our IRTS algorithm and the three reference algorithms as well as the best known results. Table 8 summarizes the results where the left part of the table is dedicated to the statistical data with the best results as input, and the right part of the table provides the statistical data with the average results as input. From Table 8, we can observe that a statistical difference is detected for each compared case with $p-value < 0.05$. The dominance of our IRTS algorithm is confirmed by the fact that the sum of the positive ranks are significantly larger than the sum of the negative ones which corresponds to our observations on Table 7.

3.4 Comparative results with a long time limit

Given the fact that our IRTS algorithm can achieve significantly improved results when more time is available (see Section 4), this section is dedicated to a comparison of our IRTS algorithm with two best performing reference algorithms (TIG-QMKP [11] and SO-QMKP [10]) by extending the time limit to 15 seconds for instances with 100 objects and 90 seconds for those with 200 objects. For this experiment, we run the source codes of TIG-QMKP and SO-QMKP provided by the authors of [11,10] on our computing environment under exactly the same condition. Additionally, we include the state-of-the-art integer programming solver CPLEX 12.4 as another reference method based on the 0-1 quadratic model of the QMKP given by (Eq. 1-3). For CPLEX, each instance is solved once with a time limit of 1 hour which corresponds to the accumulated time limit of 40 runs used by IRTS, TIG-QMKP and SO-QMKP.

Table 9 shows the results of CPLEX, our IRTS algorithms together with the two reference algorithms TIG-QMKP and SO-QMKP. For CPLEX, we report the lower bound (LB), the upper bound (UB) and the gap (GAP). The gap is computed by using the formula: $(UB\text{-}LB)/LB \times 100$. For the heuristic algorithms, the overall best objective value (f_{best}) and the average of the 40 best objective values (f_{avg}) are listed.

From Table 9, we can see that none of the 60 instances is solved to optimality by CPLEX 12.4 with the given time limit. Across the whole instance set, CPLEX always finds lower bounds worse than the previous best known results which were obtained within a computing time of 38.93 seconds (see the column *Time* of Table 7). The resulting *p-value* of 1.67e-11 of the Wilcoxon test additionally demonstrates their statistical difference (see Table 10). For all 60 instances, CPLEX produces a large positive gap from 87.94% to 654.55%. Typically, this gap increases when the problem size enlarges, when the number of knapsacks increases or when the density grows. Compared to the current best performing QMKP heuristic algorithms, CPLEX is easily dominated (see the box and whisker plot displayed in Figure 1). We mention that we also tested CPLEX 12.4 without any time limit. Even under this condition, no optimal solution was found even for the smallest instance. From these observation, it seems that it is impractical to use CPLEX to solve these QMKP benchmarks with the basic quadratic model. More studies are needed to check whether CPLEX can achieve better results with alternative models. However, such a topic is clearly beyond the scope of this paper.

When we compare our IRTS algorithm with the two reference heuristic algorithms, we can observe that IRTS attains a better result for 41 instances and an equal result for the remaining 19 instances when comparing with TIG-QMKP. With respect to SO-QMKP, IRTS performs better for 43 instances and finds an equal result for the remaining instances. In terms of the average results, our IRTS algorithm has a better, equal and worse performance in 53, 2 and 5 cases, respectively, in comparison with TIG-QMKP. When comparing with SO-QMKP, the number of better, equal and worse average results becomes 58, 1 and 1, respectively. In Table 11, we report six summary statistics of the three compared algorithms and their Wilcoxon test outcomes performed on the average results. This test discloses a significant performance difference between IRTS_Long and each of the two reference algorithms and the six summary statistic values of IRTS_Long are entirely better than those of TIG_Long and SO_Long. This experiment demonstrates that our IRTS algorithm outperforms the two reference algorithms under the long time limit.

3.5 Comparison with a memetic algorithm

In this section, we compare our IRTS algorithm with a memetic algorithm (ALA-EA) presented in [22]. Since the source code of ALA-EA and the so-

Table 9 Comparative results of IRTS with TIG-QMKP, SO-QMKP and CPLEX 12.4 with a long time limit (15 seconds per run for instances with 100 objects and 90 seconds per run for instances with 200 objects for IRTS, TIG-QMKP and SO-QMKP, one hour for CPLEX 14.2). The best results of the three compared algorithms as well as CPLEX are indicated in bold.

Instance				f_{bk}	CPLEX			TIG-QMKP [11]		SO-QMKP [10]		IRTS	
n	d	K	I		LB	UB	GAP(%)	Best	Avg	Best	Avg	Best	Avg
100	25	3	1	29234	28077	53963.23	92.20	**29286**	29027.90	**29286**	29201.70	**29286**	**29286.00**
100	25	3	2	28491	28169	52942.22	87.94	**28491**	28470.70	**28491**	28488.32	**28491**	**28491.00**
100	25	3	3	27179	26492	51010.72	92.55	27095	27015.90	**27179**	27175.20	**27179**	**27179.00**
100	25	3	4	28593	27793	53382.60	92.07	**28593**	**28593.00**	**28593**	28580.75	**28593**	**28593.00**
100	25	3	5	27892	27058	53083.61	96.18	**27892**	27885.33	**27892**	27821.98	**27892**	**27892.00**
100	25	5	1	22509	21194	55784.57	163.21	22413	22273.98	22509	22403.50	**22581**	**22530.68**
100	25	5	2	21678	20725	54647.80	163.68	21678	21648.00	21678	21622.43	**21704**	**21667.00**
100	25	5	3	21188	19674	52614.27	167.43	21181	21099.30	21188	21153.00	**21239**	**21235.95**
100	25	5	4	22181	20644	55098.88	166.90	**22181**	22180.42	**22181**	22164.32	**22181**	**22180.90**
100	25	5	5	21669	20054	54887.58	173.70	**21669**	**21663.85**	**21669**	21567.00	**21669**	21656.42
100	25	10	1	16118	14804	57710.44	289.83	16157	16057.60	16162	15996.83	**16221**	**16200.53**
100	25	10	2	15525	14191	56294.00	296.69	**15700**	15557.68	15617	15446.40	**15700**	**15665.65**
100	25	10	3	14773	13560	54274.15	300.25	14832	14736.23	14760	14648.43	**14927**	**14852.00**
100	25	10	4	16181	14630	56871.67	288.73	**16181**	16168.50	16159	16082.68	**16181**	**16181.00**
100	25	10	5	15150	14142	56609.11	300.29	15289	15189.45	15196	15094.89	**15326**	**15293.00**
200	25	3	1	101100	94992	222051.26	133.76	100372	100207.00	101218	100776.00	**101471**	**101441.00**
200	25	3	2	107958	105978	222937.93	110.36	107927	107814.00	**107958**	107663.00	**107958**	**107958.00**
200	25	3	3	104538	98214	219794.08	123.79	104532	104445.00	104538	104365.00	**104589**	**104559.00**
200	25	3	4	99559	93693	219675.07	134.46	99000	98836.80	99559	99170.50	**100098**	**100098.00**
200	25	3	5	102049	94818	217794.90	129.70	101999	101877.00	102084	101792.00	**102311**	**102310.00**
200	25	5	1	74922	67464	226198.30	235.29	74682	74361.52	74665	74389.82	**75623**	**75554.10**
200	25	5	2	79506	71876	226791.30	215.53	79604	79459.33	79473	79244.40	**80033**	**80023.40**
200	25	5	3	77700	70259	223882.05	218.65	77795	77720.58	77695	77570.82	**78043**	**78028.95**
200	25	5	4	73327	65940	223859.76	239.49	73189	72984.40	73405	73005.00	**74140**	**74061.29**
200	25	5	5	76022	69523	222056.15	219.40	76137	75905.14	76037	75829.90	**76610**	**76597.62**
200	25	10	1	51413	43054	230387.32	435.11	51592	51298.40	51389	51043.93	**52293**	**52158.50**
200	25	10	2	54116	45774	231059.61	404.78	54290	54077.30	54102	53831.20	**54830**	**54666.25**
200	25	10	3	52841	44187	227911.74	415.79	52985	52791.49	52841	52483.48	**53661**	**53588.28**
200	25	10	4	50221	41608	228108.72	448.23	50577	50282.50	50371	50002.82	**51297**	**51078.20**
200	25	10	5	52651	43847	226045.81	415.53	53337	52856.38	52596	52395.10	**53621**	**53532.24**
100	75	3	1	69977	69010	157543.51	128.29	69935	69935.00	69935	69935.00	**69977**	**69977.00**
100	75	3	2	69504	68157	158390.17	132.39	**69504**	**69504.00**	**69504**	69497.40	**69504**	69499.60
100	75	3	3	68832	67681	157395.07	132.55	**68832**	68816.20	**68832**	68813.00	**68832**	**68832.00**
100	75	3	4	70028	69717	154667.08	121.85	**70028**	**70028.00**	**70028**	**70028.00**	**70028**	**70028.00**
100	75	3	5	69692	68638	160393.28	133.68	**69692**	69681.30	**69692**	69652.22	**69692**	**69692.00**
100	75	5	1	49363	48270	161306.19	234.17	**49421**	49295.60	49363	49238.83	**49421**	**49365.98**
100	75	5	2	49316	48643	162654.44	234.38	49360	49266.80	49320	49226.60	**49365**	**49350.60**
100	75	5	3	48495	44474	161403.85	262.92	**48495**	48474.20	**48495**	48360.85	**48495**	**48495.00**
100	75	5	4	50246	48756	159342.88	226.82	**50246**	49966.60	**50246**	50124.20	**50246**	**50141.50**
100	75	5	5	48752	47286	164701.44	248.31	48752	48735.20	48752	48718.38	**48753**	**48749.10**
100	75	10	1	29931	28124	165782.37	489.47	30138	29900.84	30018	29897.80	**30296**	**30240.20**
100	75	10	2	30980	29436	167227.44	468.11	31092	30969.15	30973	30914.00	**31207**	**31095.80**
100	75	10	3	29730	27340	165637.23	505.84	29812	29662.00	29765	29638.80	**29908**	**29894.75**
100	75	10	4	31663	27916	163759.49	486.62	31672	31491.82	31634	31481.30	**31762**	**31706.50**
100	75	10	5	30229	27003	168992.47	525.83	30188	30046.10	30348	30055.42	**30507**	**30458.50**
200	75	3	1	270718	258740	644499.87	149.09	**270718**	**270718.00**	**270718**	270697.00	**270718**	270685.00
200	75	3	2	257090	248139	645348.35	160.08	**257288**	257099.00	257277	256931.00	**257288**	**257273.00**
200	75	3	3	270069	262806	649880.92	147.29	**270069**	**270069.00**	**270069**	270028.00	**270069**	269926.00
200	75	3	4	246882	227193	633617.30	178.89	246882	246684.00	246882	246555.00	**246993**	**246877.00**
200	75	3	5	279598	270979	655098.56	141.75	**279598**	**279598.00**	**279598**	**279598.00**	**279598**	279570.00
200	75	5	1	184909	164519	654784.78	298.00	184984	184774.00	184882	184641.00	**185493**	**184904.00**
200	75	5	2	174682	160577	656046.05	308.56	174776	174642.00	174682	174445.00	**174836**	**174688.00**
200	75	5	3	186526	175943	661330.18	275.88	186674	186507.00	186619	186352.00	**186774**	**186674.00**
200	75	5	4	166584	158807	643762.33	305.37	166832	166487.00	166584	166246.00	**166990**	**166747.00**
200	75	5	5	193084	173698	665834.00	283.33	193255	193002.00	193138	192836.00	**193310**	**193217.00**
200	75	10	1	112354	95514	665339.06	596.59	112591	112330.00	112457	112258.00	**113139**	**112809.00**
200	75	10	2	105151	88469	667546.03	654.55	105297	105064.00	105260	104947.00	**105807**	**105437.00**
200	75	10	3	113869	94768	672239.84	609.35	114237	113930.00	114007	113717.00	**114596**	**114367.00**
200	75	10	4	98252	88580	653941.84	638.25	98556	98219.93	98285	97885.95	**99106**	**98851.55**
200	75	10	5	116513	98288	676442.05	588.22	116725	116266.00	116298	116031.00	**117309**	**116947.00**

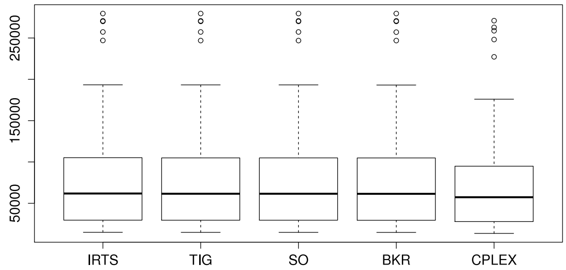

Fig. 1 Box and whisker plot of the best results obtained with long time limit

Table 10 Wilcoxon test of best results obtained with long time limit

Algorithm Pair	R+	R-	p-value	Diff?
BKR vs CPLEX	1830	0	1.67e-11	Yes
IRTS_Long vs CPLEX	1830	0	1.67e-11	Yes
IRTS_Long vs TIG_Long	861	0	2.52e-08	Yes
IRTS_Long vs SO_Long	946	0	1.16e-08	Yes

Table 11 Statistical data and Wilcoxon test of average results obtained with long time limit. The results of the Wilcoxon test are responsible for (TIG_Long vs IRTS_Long) and (SO_Long vs IRTS_Long). The best result of each column is indicated in bold.

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	R+	R-	p-value	Diff?
IRTS_Long	**14850**	**29740**	**61750**	**83680**	**104800**	279600				
TIG_Long	14740	29500	61450	83390	104600	279600	57	1654	6.48e-10	Yes
SO_Long	14650	29530	61320	83330	104500	279600	32	1738	1.24e-10	Yes

lution certificates are no longer available from the authors[4], we decided to implement their algorithm by following rigorously the description given in the paper. By adopting exactly the same experiment protocol as used in [22], ALA-EA is executed 30 times for each instance under two stop conditions which are respectively a limit of 200 generations (denoted as 200) and 1000 consecutive generations without improvement (denoted as final). Table 12 shows the comparative results between our IRTS algorithm (with the short time limit, see Section 3.2) and the ALA-EA algorithm (under its two stop conditions 200 and final). For each instance, we show the best known result (column f_{bk}), the best result of IRTS (under the short time condition, from column IRTS_Short of Table 4) and the two best results of ALA-EA with its two stop conditions. From this table, we observe that our IRTS algorithm (under the short time condition) largely dominates ALA-EA. The Wilcoxon test between our best results and those of ALA-EA leads to a *p-value* of 1.67e-11, confirming the signification of the observed differences.

Even though the table does not include the computing times, we observed that under the 'final' condition, one run of ALA-EA consumes at least 80 and

[4] This is confirmed by the authors of [22].

Table 12 Comparative results of IRTS with a memetic algorithm

Instance				f_{bk}	IRTS	ALA-EA		Instance				f_{best}	IRTS	ALA-EA	
n	d	K	I			200	$final$	n	d	K	I			200	$final$
100	25	3	1	29234	29286	27883	27883	100	75	3	1	69977	69977	69760	69760
100	25	3	2	28491	28491	25963	26023	100	75	3	2	69504	69504	68414	68414
100	25	3	3	27179	27179	25097	25097	100	75	3	3	68832	68832	66750	66750
100	25	3	4	28593	28593	25681	25681	100	75	3	4	70028	70028	69571	69571
100	25	3	5	27892	27892	24293	24293	100	75	3	5	69692	69692	69243	69243
100	25	5	1	22509	22581	21249	21262	100	75	5	1	49363	49421	48069	48069
100	25	5	2	21678	21678	19386	19394	100	75	5	2	49316	49365	48505	48505
100	25	5	3	21188	21239	18351	18351	100	75	5	3	48495	48495	47653	47653
100	25	5	4	22181	22181	19986	20027	100	75	5	4	50246	50246	48996	48996
100	25	5	5	21669	21669	18116	18216	100	75	5	5	48752	48753	48284	48284
100	25	10	1	16118	16221	13962	14007	100	75	10	1	29931	30290	28825	28873
100	25	10	2	15525	15700	13739	13739	100	75	10	2	30980	31101	30288	30288
100	25	10	3	14773	14927	12726	12832	100	75	10	3	29730	29908	28768	28816
100	25	10	4	16181	16181	13732	13870	100	75	10	4	31663	31762	30830	30830
100	25	10	5	15150	15326	12230	12269	100	75	10	5	30229	30507	29630	29651
200	25	3	1	101100	101445	95819	95819	200	75	3	1	270718	270718	265955	265955
200	25	3	2	107958	107958	105157	105196	200	75	3	2	257090	257288	250643	250643
200	25	3	3	104538	104575	101517	101517	200	75	3	3	270069	270069	268047	268094
200	25	3	4	99559	100098	95662	95680	200	75	3	4	246882	246993	245707	245707
200	25	3	5	102049	102311	97144	97144	200	75	3	5	279598	279598	276247	276247
200	25	5	1	74922	75596	68470	68502	200	75	5	1	184909	185353	180657	180771
200	25	5	2	79506	80033	74982	75080	200	75	5	2	174682	174836	167587	167641
200	25	5	3	77700	78043	73800	73887	200	75	5	3	186526	186753	184498	184498
200	25	5	4	73327	74111	68648	68872	200	75	5	4	166584	166990	163373	163413
200	25	5	5	76022	76610	72234	72234	200	75	5	5	193084	193310	187194	187194
200	25	10	1	51413	52259	45578	45578	200	75	10	1	112354	113103	107544	107662
200	25	10	2	54116	54746	49313	49322	200	75	10	2	105151	105807	101876	102016
200	25	10	3	52841	53646	48905	49058	200	75	10	3	113869	114567	110311	110346
200	25	10	4	50221	51176	44327	44404	200	75	10	4	98252	99075	95088	95088
200	25	10	5	52651	53616	48571	48748	200	75	10	5	116513	117309	114083	114116

350 seconds to solve instances with 100 and 200 objects, respectively. Even for the 200 generations limit, ALA-EA requires more than 15 and 60 seconds for instances with 100 and 200 objects, respectively.

4 Discussion

In this section, we turn our attention to an analysis of two ingredients of the proposed IRTS algorithms: the neighborhoods and the perturbation strategy.

4.1 Observations on neighborhood effectiveness

As described in section 2.4, our IRTS algorithm employs three dedicated neighborhood structures which are induced by the *DROP*, *REALLOCATE* and *EXCHANGE* operators. In this section, we investigate the influence of each neighborhood over the performance of the algorithm. For this purpose, we propose three weakened versions of IRTS such that for each IRTS variant, we disable one particular neighborhood while keeping the other components unchanged. For instance, IRTS_NoReallocate is the IRTS algorithm without the neighborhood CN_R defined by *REALLOCATE*. Along with the standard IRTS algorithm, four IRTS versions are tested on the whole instance set using the short time limit of Section 3.2 (i.e., 5 seconds for instances with $n = 100$ and 30 seconds for instances with $n = 200$). Each algorithm is run 10 times to solve each of the 60 instances. We calculate for each instance the absolute

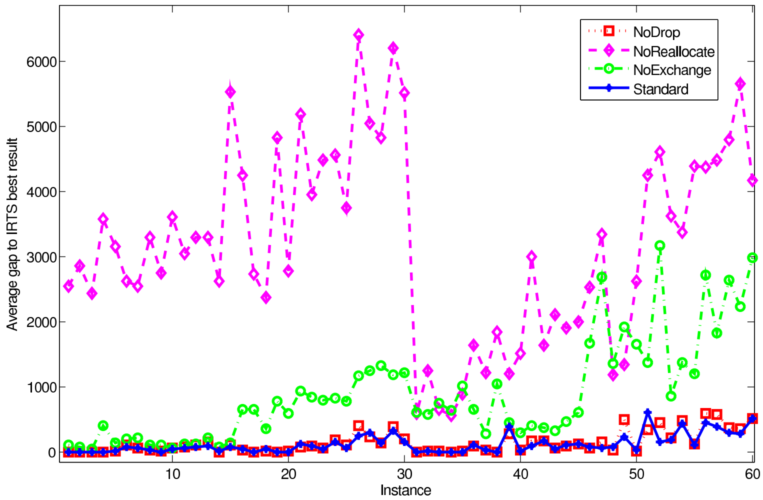

Fig. 2 Average solution gaps to the IRTS_Long best result

gap between the average results obtained by each algorithm variant and the best result obtained by IRTS_Long reported in Section 3.2. The best results of IRTS_Long are the overall best results reported until now. The experimental results are shown in Figure 2.

Figure 2 shows removing a neighborhood sacrifices the search power of IRTS. Specifically, IRTS_NoReallocate achieves the worst performance with a significant large gap between the average solution values and the best results. Moreover, for all 60 instances, the average results of IRTS_NoReallocate are worse than those obtained by the original IRTS algorithm. Though it is easily dominated by the standard IRTS algorithm, IRTS_NoExchange obtains much better results than IRTS_NoReallocate. Compared to IRTS_NoReallocate, IRTS_NoExchange decreases the gap between the average solution values and the best results for a majority of cases. In particular, IRTS_NoExchange is better than IRTS_NoReallocate for 54 out of 60 instances in the average result obtained. When it comes to IRTS_NoDrop, it is hard to see the difference between its performance and that of IRTS from the figure because of the wide range of gap from 0 to almost 7000. Statistical data demonstrates that IRTS_NoDrop has a worse average result for a large number of cases (45 out of 60 instances) and the average of its 60 gap values (8477.10) is much larger than that of IRTS (7136.70) which can be observed from Table 13.

This experiment confirms that all three neighborhoods contribute to the performance of the IRTS algorithm. Among the three neighborhoods, the most important one is *REALLOCATE*, the *EXCHANGE* neighborhood ranks second, followed by the *DROP* neighborhood. Apart from the usefulness of each individual neighborhood, their combined use within the IRTS algorithm constitutes an important feature to ensure the performance of the algorithm.

Table 13 Average value of the average solution gap to the best results for four IRTS variants

Algorithm	IRTS_NoDrop	IRTS_NoReallocate	IRTS_NoExchange	IRTS
avg.	8477.1	190716.8	53537.2	7136.7

Table 14 Wilcoxon test for different perturbation strategies

Algorithm Pair	Best Results				Average Results			
	R+	R-	p-value	Diff?	R+	R-	p-value	Diff?
Sort vs NoSort	365	70	1.48e-03	Yes	1254	231	1.08e-05	Yes
Sort vs Restart	292	114	4.39e-02	Yes	1047	384	3.39e-03	Yes

4.2 Influence of perturbation strategy

As shown in Section 2.8, our IRTS algorithm uses a specialized perturbation strategy to ensure a more global diversification. The perturbation strategy operates by sorting the objects in the knapsacks according to their densities and forcing the first PL objects to change their status. In order to assess this strategy, we compare it with a traditional restart strategy (denotes as $Restart$) and a modified perturbation strategy where the step of sorting the objects is eliminated (denoted as $NoSort$). The perturbation strategy used in IRTS is denoted by $Sort$. We run the above three algorithm variants 10 times on all 60 instances using the short time limit and use the Wilcoxon test to check the statistical difference between $Sort$ and the other two versions both in terms of the best results and average results (Table 14). This table discloses statistical differences between $Sort$ and each of the other two compared strategies for best results as well as average results. When we examine the sum of the signed ranks, it is clear that $Sort$ is better than the compared variants by always holding a higher sum of positive ranks for the two measures we used. This experiment confirms thus the interest of the proposed perturbation strategy.

5 Conclusions

In this work, we have presented an iterated responsive threshold search algorithm for solving the quadratic multiple knapsack problem. The proposed IRTS algorithm is based on a joint use of three neighborhoods induced by three types of move operators namely $DROP$,$REALLOCATE$ and $EXCHANGE$. A key innovation of our method is a special strategy for guiding a threshold process which we call responsive thresholding. Our algorithm incorporates this strategy by alternating between an exploration phase (with three neighborhoods) – where neighboring solutions are accepted as long as their quality satisfies the responsive threshold – and an improvement phase (with two neighborhoods) where only improving solutions are accepted. To escape deep local optima, IRTS integrates a guided perturbation strategy to reinforce its global diversification capacity.

We have assessed the performance of the proposed algorithm on a set of 60 well-known QMKP benchmark instances and demonstrated its effectiveness in comparison with the state-of-the-art methods in the literature. In particular, the proposed algorithm has established 41 improved lower bounds which can serve as new references for the evaluation of new QMKP algorithms. Additionally, we have provided an analysis to show the role of the employed neighborhoods and investigated the impact of the dedicated perturbation strategy over the performance of the proposed algorithm.

For future work, our responsive thresholding strategy can be integrated with other types of thresholding procedures such as those proposed in [9]. Additional future applications of our approach can make use of other types of neighborhoods and multi-neighborhood designs as in the frameworks of [15, 16].

Acknowledgment

We are grateful to the reviewers for their insightful comments which helped us improve the paper. We would like to thank Dr. García-Martínez for answering our questions and making the codes of [10,11] available to us. This work is partially supported by the RaDaPop (2009-2013) and LigeRo projects (2009-2013) from the Region of Pays de la Loire (France). Support for Yuning Chen from the China Scholarship Council is also acknowledged.

References

1. Billionnet, A., & Soutif, E. (2004). An exact method for the 0-1 quadratic knapsack problem based on Lagrangian decomposition, European Journal of Operational Research, 157(3), 565-575.
2. Caprara, A., Pisinger, D., & Toth P. (1999). Exact solution of the quadratic knapsack problem, INFORMS Journal on Computing, 11(2), 125-137.
3. Corder, G.W., & Foreman, D.I. (2014). Nonparametric statistics for non-statisticians: A step-by-step approach, John Wiley & Sons, Hoboken, New Jersey.
4. Dueck, G., & Scheuer, T. (1990). Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing, Journal of Computational Physics, 90, 161-175.
5. Dueck, G. (1993). New optimization heuristics: the great deluge algorithm and the record-to-record travel, Journal of Computational Physics, 104, 86-92.
6. Di, Gaspero L., & Schaerf, A. (2006). Neighborhood portfolio approach for local search applied to timetabling problems, Journal of Mathematical Modeling and Algorithms, 5(1), 65-89.
7. Gallo, G., Hammer, P.L., & Simeone, B. (1980). Quadratic knapsack problems, Mathematical Programming Studies, 20, 132-149.
8. Glover, F., McMillan, C., & Glover, R. (1984). A heuristic programming approach to the employee scheduling problem and some thoughts on managerial robots, Journal of Operations Management, 4(2), 113-128
9. Glover, F. (1995). Tabu thresholding: improved search by nonmonotonic trajectories, ORSA Journal on Computing, 7(4), 426-442.
10. García-Martínez, C., Glover, F., Rodríguez, F.J., Lozano, M., & Martí R. (2014). Strategic oscillation for the quadratic multiple knapsack problem, Computational Optimization and Applications, 58(1), 161-185.
11. García-Martínez, C., Rodríguez, F.J., & Lozano, M. (2014). Tabu-enhanced iterated greedy algorithm: a case study in the quadratic multiple knapsack problem, European Journal of Operational Research, 232(3), 454-463.

12. Hiley, A., & Julstrom, B. (2006). The quadratic multiple knapsack problem and three heuristic approaches to it, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), P.547-552.
13. Hung, M.S., & Fisk, J.C. (1978). An algorithm for 0-1 multiple-knapsack problems, Naval Research Logistics Quarterly, 25(3), 571-579.
14. Kellerer, H., Pferschy, U., & Pisinger D. (2004). Knapsack Problems, Springer Verlag, ISBN 3-540-40286-1.
15. Lü, Z., Hao, J.K., & Glover, F. (2011). Neighborhood analysis: a case study on curriculum-based course timetabling, Journal of Heuristics, 17(2), 97-118.
16. Lü, Z., Glover, F., & Hao, J.K. (2014). Neighborhood combination for unconstrained binary quadratic problems, To appear in M. Caserta and S. Voss (Eds.): Metaheuristics International Conference 2011 Post-Conference Book, Chapter 4, 49-61.
17. Martello, S., & Toth P. (1981). Heuristic algorithms for the multiple knapsack problem, Computing, 27(2), 93-112.
18. Pisinger, D. (2007). The quadratic knapsack problem–a survey, Discrete Applied Mathematics, 155(5), 623-648.
19. Pisinger, D. (2009). An exact algorithm for large multiple knapsack problems, European Journal of Operational Research, 114(3), 528-541.
20. Saraç, T., & Sipahioglu A. (2007). A genetic algorithm for the quadratic multiple knapsack problem, In Proceeding of Second International Symposium on Advances in Brain, Vision, and Artificial Intelligence, Lecture Notes in Computer Science, 4729, 490-498.
21. Singh, A., & Baghel, A. (2007). A new grouping genetic algorithm for the quadratic multiple knapsack problem, In Proceedings of International Conference on Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science, 4446, 210-218.
22. Soak, S., & Lee, S. (2012). A memetic algorithm for the quadratic multiple container packing problem, Applied Intelligence, 36, 119-135.
23. Sundar, S., & Singh, A. (2010). A swarm intelligence approach to the quadratic multiple knapsack problem, In Proceeding of 17the International Conference on Neural Information Processing, Lecture Notes in Computer Science, 6443, 626-633.
24. Wang, H., Kochenberger, G., & Glover, F. (2012). A computational study on the quadratic knapsack problem with multiple constraints, Computers and Operations Research, 39(1), 3-11.