Knowledge-Based Systems 92 (2016) 23–34

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

An evolutionary path relinking approach for the quadratic multiple knapsack problem

Yuning Chen[a], Jin-Kao Hao[a,b,*], Fred Glover[c]

[a] LERIA, Université d'Angers, 2 Bd Lavoisier, 49045 Angers, Cedex 01, France
[b] Institut Universitaire de France, Paris, France
[c] OptTek Systems, Inc., 2241 17th Street Boulder, Colorado 80302, USA

ARTICLE INFO

Article history:
Received 4 March 2015
Revised 26 September 2015
Accepted 4 October 2015
Available online 21 October 2015

Keywords:
Quadratic multiple knapsack
Path relinking
Constrained optimization
Responsive threshold search
Evolutionary computing

ABSTRACT

The quadratic multiple knapsack problem (QMKP) is a challenging combinatorial optimization problem with numerous applications. In this paper, we propose the first evolutionary path relinking approach (EPR) for solving the QMKP approximately. This approach combines advanced features both from the path relinking (PR) method and the responsive threshold search algorithm. Thanks to the tunneling property which allows a controlled exploration of infeasible regions, the proposed EPR algorithm is able to identify very high quality solutions. Experimental studies on the set of 60 well-known benchmarks and a new set of 30 large-sized instances show that EPR outperforms several state-of-the-art algorithms. In particular, for the 60 conventional benchmarks, it discovers 10 improved results (new lower bounds) and matches the best known result for the remaining 50 cases. More significantly, EPR demonstrates remarkable efficacy on the 30 new larger instances by easily dominating the current best performing algorithms across the whole instance set. Key components of the algorithm are analyzed to shed lights on their impact on the proposed approach.

1. Introduction

The quadratic multiple knapsack problem (QMKP) is a well-known combinatorial optimization problem [1]. Given a set of knapsacks of limited capacity and a set of objects (or items), each object is associated with a weight, an individual profit and a pairwise profit with any other object. The QMKP aims to determine a max-profit assignment (packing) of objects to the knapsacks subject to the capacity constraint of each knapsack. The QMKP has a number of relevant applications where resources with different levels of interaction have to be distributed among different tasks [1].

The QMKP belongs to a large family of the knapsack problems. Among these, the basic linear 0–1 knapsack problem (KP) is probably the least difficult case since exact solution methods based on dynamic programming and branch-and-bound can be easily applied [2]. The KP can be encountered in many other settings [2] (see [3] for an example in knowledge based systems—collaborative filtering). The linear multiple knapsack problem (MKP) [4] and the quadratic knapsack problem (QKP) [5] generalize the basic KP. Between them, the MKP where multiple knapsacks are available for packing objects is a relatively easier problem for which state-of-the-art MKP solvers can find optimal solutions for very large instances (with up to 100,000 items) in less than 1 s [4]. However, the QKP where pairwise profits are defined is more computationally intractable and the most powerful exact solver can only deal with instances with no more than 1500 items [6]. The MKP is not to be confused with the very popular multidimensional knapsack problem (MDKP) [7,8] which has a single knapsack, but multiple linear constraints. Finally, the QMKP is also related to another variant of the basic KP—the discounted 0–1 knapsack problem (DKP) where discounted profits are defined for a pair of items but the problem remains linear [9].

The QMKP considered in this work generalizes both the QKP and the MKP by allowing multiple knapsacks and pairwise profits between objects. On the other hand, the QMKP also belongs to the still larger family of quadratic optimization problems whose representative members include the quadratic assignment problem [10], the quadratic knapsack problem [5], the unconstrained binary quadratic programming problem [11] and specific non-linear programming problems [12,13]. Due to their quadratic nature, these problems are known to be extremely difficult. For this reason, approximate algorithms based on metaheuristics like evolutionary algorithms, constitute a very popular approach for tackling these problems [1,10,11,14–16].

The QMKP is highly combinatorial with a solution space of order $O((m+1)^n)$ for n objects and m knapsacks. Given the high

* Corresponding author at: LERIA, Université d'Angers, 2 Bd Lavoisier, 49045 Angers, Cedex 01, France. Tel.: +33 2 41 73 50 76.
E-mail addresses: yuning@info.univ-angers.fr (Y. Chen), hao@info.univ-angers.fr, jin-kao.hao@univ-angers.fr (J.-K. Hao), glover@opttek.com (F. Glover).

http://dx.doi.org/10.1016/j.knosys.2015.10.004
0950-7051/

computational complexity of the problem, no exact approach for the QMKP has been published in the literature to the best of our knowledge. On the other hand, some effort has been devoted to develop heuristics which aim to provide satisfactory sub-optimal solutions in acceptable computing time, but without provable optimal guarantee of the attained solutions. Among these heuristics, neighborhood search and constructive/destructive search approaches are very popular, including hill-climbing [1], tabu-enhanced iterated greedy search [17], strategic oscillation [18] and iterated responsive threshold search [19]. Population-based algorithms constitute another class of popular tools for addressing the QMKP, such as genetic algorithms [1,14], memetic algorithms [15,16] and artificial bee colony algorithms [20].

In this work, we are interested in developing improved solution methods for the QMKP. For this purpose, we propose a new algorithm based on the general evolutionary path-relinking (EPR) metaheuristic [22]. Our work is mainly motivated by two key observations.

- From the perspective of solution methods, the EPR framework provides a variety of appealing features which have been demonstrated to be useful for designing effective heuristic algorithms. Indeed, EPR has been successfully applied to solve a number of challenging problems encountered in diverse settings. Representative examples include location routing [23], vehicle routing [24], max–min diversity [25], permutation flowshop [21], warehouse layout [26], unconstrained quadratic programming [11], and quadratic assignment [27]. Often, local search/constructive heuristics such as tabu search [11,27,28] and GRASP [25] are combined with the path relinking method. Fundamentally, EPR operates with a population of (elite) solutions and employs path relinking procedures to generate (many) intermediate solutions [22]. The process of path relinking can help to discover better solutions along the paths or around some intermediate solutions (typically identified by local refinement). More generally, EPR provides a unique framework for designing effective search algorithms with a suitable balance of diversification and intensification.
- From the perspective of the problem under investigation, the QMKP is a strongly constrained combinatorial optimization problem in which the feasible region often consists of components which are separated from each other by infeasible regions in the search space, especially when the capacity constraints are tight. In this case, imposing solution feasibility during the search, like most of the existing QMKP methods, can make it difficult for the search process to locate global optima or high quality solutions. On the other hand, methods that can tunnel through feasible and infeasible regions are particularly attractive as a means to cope with such a situation [29,30]. In this regard, EPR has the useful property of generating new solution paths where both feasible and infeasible solutions are eligible, thus providing an algorithm with the desired tunneling property.

As a general metaheuristic (or algorithmic framework) [31], EPR can theoretically be applied to any optimization problem. Still in order to obtain an effective solution algorithm for the problem at hand, it is indispensable to carefully adapt EPR to the problem setting by considering a set of algorithmic design issues such as the procedure for generating initial solutions, the way of building solution paths from an initiating solution to a guiding solution, the procedure for local refinement, the criterion for selecting one or more solutions on a path for local optimization as well as the way of managing the reference set of elite solutions. Within the context of the QMKP, as shown in Section 3, our proposed EPR algorithm distinguishes itself from existing studies by introducing specific strategies to address all these issues and by devising dedicated techniques to handle infeasible solutions (see Section 3.8 for a summary of the main characteristics of the proposed approach). In addition, as demonstrated in Section 4, the proposed EPR algorithm attains a remarkable performance when it is assessed on well-known and hard QMKP benchmark instances. We undertake to shed light on what makes the proposed algorithm successful by providing an in-depth algorithmic analysis (see Section 5).

The following identifies the main contributions of the present work:

- The proposed EPR algorithm is the first adaptation of the general EPR method tailored to the QMKP. The algorithm integrates a set of original features including a probabilistic greedy construction procedure to generate initial solutions of the reference set, a double-neighborhood path relinking procedure to build diversified solution paths, an effective responsive threshold search algorithm for local refinement, and a fitness-based updating strategy to maintain a reference set with a healthy diversity.
- Of particular interest is the ability of the proposed EPR algorithm to make a controlled exploration of infeasible solutions during the path relinking process. By allowing the search to oscillate between feasible and infeasible solutions, this strategy promotes exploration of large search zones and helps to identify high quality solutions.
- The computational assessment on two sets of 90 QMKP benchmark instances indicates a remarkable performance of the proposed algorithm. For the first set of 60 well-known QMKP instances, the algorithm discovers 10 improved best solutions (i.e., new lower bounds) and matches all the remaining 50 best known results. For the 30 new large-sized instances, the algorithm always dominates the state-of-the-art algorithms.
- Ideas of the proposed approach can readily be adapted to design effective algorithms for other knapsack and constrained optimization problems.

The rest of the paper is organized as follows. Section 2 introduces the problem definition and the mathematical formulation. Section 3 describes the proposed EPR approach in detail. Section 4 presents experimental results of our algorithm, including comparisons with the state-of-the-art algorithms in the literature. Section 5 analyzes several essential components of our proposed algorithm, followed by a discussion of conclusions in Section 6.

2. Modeling the quadratic multiple knapsack problem

Let $N = \{1, 2, \ldots, n\}$ be a set of objects (or items) and $M = \{1, 2, \ldots, m\}$ a set of knapsacks. Each object i ($i \in N$) has a profit p_i and a weight w_i. Each pair of objects i and j ($1 \leq i \neq j \leq n$) has a joint profit p_{ij} when both objects i and j are allocated to the same knapsack. Each knapsack k ($k \in M$) has a capacity C_k. The purpose of the QMKP is to assign the n objects to the m knapsacks (some objects can remain unassigned) such that the overall profit of the assigned objects is maximized subject to the following two constraints:

- Each object i ($i \in N$) can be allocated to at most one knapsack.
- The total weight of the objects assigned to each knapsack k ($k \in M$) cannot exceed its capacity C_k.

Let $S = \{I_0, I_1, \ldots, I_m\}$ be an allocation of the n objects to the m knapsacks where each $I_k \subset N$ ($k \in M$) represents the set of objects assigned to knapsack k and I_0 is the set of unassigned objects. Then the QMKP can be formalized as follows:

$$\max \ f(S) = \sum_{k \in M} \sum_{i \in I_k} p_i + \sum_{k \in M} \sum_{i \neq j \in I_k} p_{ij} \tag{1}$$

subject to:

$$\sum_{i \in I_k} w_i \leq C_k, \forall k \in M \tag{2}$$

$$S \in \{0, \ldots, m\}^n \tag{3}$$

Note that the QMKP can be conveniently formulated as a 0–1 quadratic program [19]. However, as shown in [19], the computational results of solving this quadratic model with the CPLEX 12.4 solver are quite disappointing since it cannot find an optimal solution even for some of the smallest benchmark instances with only 100 objects.

3. An evolutionary path relinking algorithm for the QMKP

The general PR framework typically starts from a collection of diverse elite solutions which are contained in a population called a reference set (*RefSet*). In the most common version, pairs of solutions in *RefSet* are created and recorded in a so-called pair set (*PairSet*). A Path Relinking method is then applied to each pair of solutions in *PairSet*, to generate a path of intermediate solutions, where the solution starting the path is called *initiating solution*, and the one ending the path is called *guiding solution*. One or several solutions are picked from the path to be submitted to a Local Refinement method in order to identify enhanced solutions. One iteration (or generation) of PR terminates by updating *RefSet* and *PairSet* with the improved solutions from Local Refinement. This process is iterated until all pairs in *PairSet* are examined.

In what follows, we introduce our evolutionary path relinking (EPR) method and provide an explanation of its ingredients.

3.1. Main scheme

The main scheme of our EPR algorithm is described in Algorithm 1 whose operations are presented below.

At the beginning, an initial *RefSet* of elite solutions $\{S_1, S_2, \ldots S_p\}$ are generated by a probabilistic greedy construction method (Line 4, see Section 3.2) and are further improved by the Local Refinement procedure (Section 3.6). *PairSet* is then formed to contain all possible index pairs (i, g), $(i, g \in \{1, 2, \ldots p\})$ of solutions of *RefSet* according to two conditions. First, the initiating solution S_i is worse than the guiding solution S_g (i.e., $f(S_i) < f(S_g)$). Second, these index pairs are ranked in descending order of the objective value $f(S_g)$ of the guiding solution S_g, and for solutions with the same $f(S_g)$, the index pairs are ranked in ascending order of the objective value $f(S_i)$ of the initiating solution S_i.

EPR then enters a while loop of the path relinking phase until *PairSet* becomes empty. At the beginning of each loop, from an empty offspring solution S_0, the algorithm picks the first index pair (i_0, g_0) from the ranked *PairSet*, and applies the Relinking Method to the two selected solutions (S_{i_0}, S_{g_0}) to generate a *Sequence* of solutions (Line 11, see Section 3.3). This *Sequence* of solutions forms a path connecting S_{i_0} (initiating solution) and S_{g_0} (guiding solution). From this *Sequence*, we select one solution and apply operations which differ according to whether the selected solution is feasible or infeasible.

EPR first checks if *Sequence* contains a feasible solution S^{bf} whose quality is better than the best solution S^* found so far (Lines 12–13). If this is the case, EPR records S^{bf} as the offspring solution S_0 of the current EPR generation and then updates *RefSet* and *PairSet* accordingly (Line 29, see Section 3.7).

Then, if no new best solution exists in *Sequence*, EPR selects, with the path solution selection method of Section 3.4, a solution S^r from *Sequence* which can be either feasible or infeasible (Line 17). If the selected solution S^r is infeasible, EPR repairs it with the Repair Method of Section 3.5 (Lines 18–19). If the repaired solution S^r is a new best solution, EPR records S^r as the offspring solution S_0 (Lines 21–22) and then updates *RefSet* and *PairSet* accordingly (Line 29, see Section 3.7).

If the offspring solution S_0 is not updated by the repaired solution S^r since it does not improve the best solution S^* (i.e., S_0 remains empty), EPR applies the Local Refinement procedure (Line 25, see Section 3.6) to further improve S^r and assigns the improved S^r to S_0.

Algorithm 1 Pseudo-code of the EPR algorithm for the QMKP.

1: **Input**:
   P: an instance of the $QMKP$, p: reference set size;
2: **Output**: the best solution S^* found so far;
3: **repeat**
4: $RefSet = \{S_1, S_2, \ldots, S_p\} \leftarrow Initial\_RefSet(p)$ /* Sect. 3.2 */
5: $S^* \leftarrow Best(RefSet)$ /* S^* records the best solution found so far */
6: $PairSet \leftarrow \{(i, g) : S_i, S_g \in RefSet, f(S_i) < f(S_g)\}$
7: Rank $PairSet$ in descending order of $f(S_g)$, for solutions with the same $f(S_g)$, rank $PairSet$ in ascending order of $f(S_i)$
8: **while** $PairSet \neq \emptyset$ **do**
9: $S_0 \leftarrow \{0\}^n$
10: Pick the first solution pair $(i_0, g_0) \in PairSet$ /* Sect. 3.3 */
11: $Sequence = \{S^1, S^2, \ldots, S^t\} \leftarrow path\_relink(S_{i_0}, S_{g_0})$
{Check if $Sequence$ contains a solution better than the current best solution S^*}
12: $S^{bf} \leftarrow BestFeasible(Sequence)$ /* Identify the best feasible solution of Sequence */
13: **if** $f(S^{bf}) > f(S^*)$ **then**
14: $S_0 \leftarrow S^{bf}$
15: **end if**
{The best of $Sequence$ is not better than the best solution S^* found so far}
16: **if** $f(S_0) = 0$ **then**
17: Select $S^r \in Sequence$ /* Sect. 3.4 */
18: **if** S^r is infeasible **then**
19: $S^r \leftarrow repair(S^r)$ /* Repair infeasible S^r to be feasible, Sect. 3.5 */
20: **end if**
21: **if** $f(S^r) > f(S^*)$ **then**
22: $S_0 \leftarrow S^r$
23: **end if**
24: **if** $f(S_0) = 0$ **then**
25: $S_0 \leftarrow local\_refine(S^r)$ /* Improve S^r by local refinement, Sect. 3.6 */
26: **end if**
27: **end if**
{Offspring S_0 is better than initiating solution S_{i_0}, use S_0 to update $RefSet$ and $PairSet$}
28: **if** $f(S_0) > f(S_{i_0})$ **then**
29: $(RefSet, PairSet) \leftarrow pool\_update(RefSet, PairSet)$ /* Sect. 3.7 */
30: **if** $f(S_0) > f(S^*)$ **then**
31: $S^* \leftarrow S_0$
32: **end if**
33: **else**
34: $PairSet \leftarrow PairSet \backslash (i_0, g_0)$
35: **end if**
36: **end while**
37: **until** stopping condition is reached
38: **return** S^*

EPR finishes the iteration at the pool updating phase where it updates *RefSet, PairSet* and the best solution.

EPR applies the above operations to each paired solutions of *RefSet*. Upon the completion of the path relinking process on all pairs of *RefSet*, EPR rebuilds a new *RefSet* which includes the best solution found so far S^*, and the process repeats until a stopping condition is verified (e.g., a time cutoff, a maximum number of allowed path relinking iterations).

3.2. RefSet *initialization*

In the reference set initialization phase, EPR uses a probabilistic greedy construction method (PGCM) to build an initial solution which is further improved by the Local Refinement procedure. Unlike deterministic greedy construction methods such as those used in [17–19], PGCM employs a probabilistic choice rule which bears some resemblance to the popular GRASP method [32] although proposed earlier in [33,34]. PGCM relies on the notion of *conditional attractiveness* which extends the familiar bang-for-buck ratio of ordinary linear knapsack problems to the present setting.

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

$$CA(S, i, k) = (p_i + \sum_{j \in I_k, j \neq i} p_{ij})/w_i \tag{4}$$

Starting from an empty solution S and the first knapsack (i.e., $k = 1$), PGCM iteratively and probabilistically selects an unallocated object i from a restricted candidate list $RCL(S, k)$ and assigns i to knapsack k. Let $R(S, k)$ denote the set of unselected objects such that $\forall i \in R(S, k)$, $w_i + \sum_{j \in I_k} w_j \leq C_k$. To build $RCL(S, k)$, we first sort all objects in $R(S, k)$ in descending order of their conditional attractiveness values (calculated by Eq. (4)), and then we put the first $\min\{rcl, |R(S, k)|\}$ (rcl is a parameter) objects into $RCL(S, k)$. The rth object in $RCL(S, k)$ is associated with a bias $b_r = 1/e^r$ and is selected with a probability $p(r)$ which is calculated as:

$$p(r) = b_r / \sum_{j=1}^{|RCL(S,k)|} b_j \tag{5}$$

PGCM skips to the next knapsack each time $RCL(S, k)$ ($\forall k \in M$) becomes empty and this is repeated until the last knapsack (i.e., $k = m$) is examined.

Algorithm 2 summarizes this *RefSet* initialization procedure. *RefSet* is set to empty at first. Then a solution is generated using PGCM (Lines 6–10) and then improved by a Local Refinement procedure. The ameliorated solution is either inserted to *RefSet* if it is not a clone of any solution of *RefSet*, or discarded if it appears already in *RefSet*. The initialization procedure is iterated until the population is filled with p (reference set size) nonclone individuals or the number of iterations reaches the *maxTrial* value (set to $3 \times p$). Finally, p is reset to the number of solutions eventually obtained.

Algorithm 2 Pseudo-code of the *RefSet* initialization procedure.

```
1:  Input: P: An instance of the QMKP, p: reference set size;
2:  Output: The reference set RefSet
3:  RefSet ← ∅
4:  maxTrial ← 3 * p, nTrial ← 0, nIndi ← 0
5:  while nTrial < maxTrial do
6:    S ← {0, ..., 0}
7:    for k = 1 → m do
8:      while RCL(S, k) ≠ ∅ do
9:        Select an object i ∈ RCL(S, k) according to the biased probability function of Equation 5.
10:       S(i) = k
11:       Update RCL(S, k)
12:     end while
13:   end for
14:   S ← local_refine(S)
15:   if S is not a clone of any solution in RefSet then
16:     RefSet ← RefSet ∪ S
17:     nIndi ← nIndi + 1
18:     if nIndi = p then
19:       break;
20:     end if
21:   end if
22:   nTrial ← nTrial + 1
23: end while
24: p ← |RefSet|
25: return RefSet
```

3.3. *The relinking method*

The purpose of the relinking method is to generate a sequence of new solutions by exploring trajectories that connect high quality solutions. Starting from an initiating solution, the relinking method generates a solution path towards the guiding solution by progressively introducing attributes into the initiating solution that are contained in the guiding solution.

To ensure the efficacy of the relinking method, we consider two issues. First, we need a distance measure to identify the difference between two solutions. Second, we need a means to explore the trajectories connecting the initiating solution and the guiding solution with the help of some neighborhoods and an evaluation function. These issues are elaborated in the next two subsections.

3.3.1. *Measuring the distance between two solutions*

The QMKP can be viewed as a grouping problem in the sense that the n objects are to be distributed into different knapsacks (unassigned objects are kept in knapsack zero). Therefore, the well-known set-theoretic partition distance [35] appears to be appropriate for measuring the differences between two solutions. Given solutions S_1 and S_2, we identify their distance $Dist(S_1, S_2)$ by resorting to a complementary measure $Sim(S_1, S_2)$ called *similarity*, i.e., $Dist(S_1, S_2) = n - Sim(S_1, S_2)$. The similarity $Sim(S_1, S_2)$ defines the size of the identical part of two solutions which represents the maximum number of elements of S_1 that do not need to be displaced to obtain S_2. Now we explain how to identify the similarity $Sim(S_1, S_2)$.

In the context of the QMKP, knapsack i of the first solution might correspond to knapsack j ($j \neq i$) of the second solution where the two knapsacks may have many objects in common. To find the *similarity* of two solutions, we first match the knapsack zero of the first solution with the knapsack zero of the second solution. Then we create a complete bipartite graph $G = (V_1, V_2, E)$ where V_1 and V_2 represent respectively the m knapsacks of solutions S_1 and S_2. Each edge $(k_i^1, k_j^2) \in E$ is associated with a weight $w_{k_i^1 k_j^2}$, which is defined as the number of shared objects in knapsack k_i^1 of solution S_1 and knapsack k_j^2 of solution S_2. From this bipartite graph, we find a maximum weight matching with the well-known Hungarian algorithm [36] which can be accomplished in $O(m^3)$. After the knapsack matching procedure, we adjust the knapsack numbering of the two solutions according to the matching outcome. The similarity $Sim(S_1, S_2)$ can then be identified by simply summing up the number of common objects of each matched knapsack pair of solutions S_1 and S_2.

3.3.2. *Building the path*

The performance of the relinking method highly depends on the neighborhoods and evaluation function to move from the initiating solution towards the guiding solution. A suitable design of the relinking method seeks to create inducements to favor solution transitions that lead to good attribute compositions of the initiating and guiding solutions. For this purpose, we propose a double-neighborhood relinking method (denoted as DNRM).

DNRM jointly employs two restricted neighborhoods RN_R and RN_E induced by two basic move operators: REALLOCATE (REAL for short) and EXCHANGE (EXC for short). At each relinking step, the best neighboring solution among both neighborhoods is selected for the transition. $REAL(u, k)$ displaces an object u from its current knapsack $S(u)$

to knapsack k ($k \neq S(u)$, $k, S(u) \in \{0, 1, \ldots, m\}$) while $EXC(u, v)$ exchanges a pair of objects that are from two different knapsacks (i.e., $S(u) \neq S(v)$, $S(u), S(v) \in \{0, 1, \ldots, m\}$). Let $S \oplus OP$ denote the operation of applying move operator OP to S. Let S_i and S_g be the initiating and guiding solutions respectively. The two restricted neighborhoods induced by the above two move operators are described as follows:

- $RN_R(S) = \{S' : S' = S \oplus REAL(u, k), S_i(u) \neq S_g(u), S_g(u) = k\}$. $RN_R(S)$ contains a set of solutions that are closer to the guiding solution by exactly one unit compared to the current solution S. These solutions are obtained by moving an object u from its current knapsack $S_i(u)$ to another knapsack $S_g(u)$ to which it belongs in the guiding solution.
- $RN_E(S) = \{S' : S' = S \oplus EXC(u, v), S_i(v) = S_g(u), S_i(v) \neq S_g(v)\}$. $RN_E(S)$ contains a set of solutions that are closer to the guiding solution by one or two units compared to the current solution S. These solutions are obtained by moving an object u from its current knapsack $S_i(u)$ to another knapsack $S_g(u)$ to which it belongs in the guiding solution, and moving an object v from its current knapsack $S_i(v)$ to another knapsack $S_i(u)$ provided that $S_i(v) \neq S_g(v)$. If $S_i(u) = S_g(v)$, the solution obtained is two units closer to the guiding solution, otherwise it is only one unit closer.

It is known that allowing a controlled exploration of infeasible solutions may enhance the performance of neighborhood-based search, which may facilitate transitioning between structurally different feasible solutions [29]. We thus allow our DNRM to tunnel through infeasible regions of the solution space. Given a solution $S = \{I_0, I_1, \ldots, I_m\}$, its total violation of the knapsack capacity is computed as: $V(S) = \sum_{k=1}^{m} (\max\{0, \sum_{i \in I_k} w_i - C_k\}$, and its total weight of the allocated objects is given by: $W(S) = \sum_{k=1}^{m} \sum_{i \in I_k} w_i$. The quality of the (infeasible) solution S is then assessed by considering both the violation of the capacity constraint and the total profit:

$$\phi(S) = f(S) - \alpha * V(S) \tag{6}$$

where α is a penalty parameter that balances the tradeoff between total profit and capacity violation. A larger value of $\phi(S)$ indicates a better solution. It would be possible to dynamically change the value of α during the search. However, according to our empirical outcomes, we observe that the proposed algorithm performs well by fixing α to the following value throughout the relinking process: $\alpha = f(S_i)/W(S_i)$, where S_i is the initiating solution.

Algorithm 3 shows the pseudo-code of our double-neighborhood relinking method (DNRM). Initially, DNRM performs a matching of the knapsacks of the initiating solution S_i and guiding solution S_g (see Section 3.3.1), and then adjusts the knapsack numbering of the guiding solution S_g according to the matching result. When the knapsacks of the two solutions are matched, the distance $Dist(S_i, S_g)$ can be identified easily by simply counting the number of different assignments of the two matched solutions. Before entering the path building loop, the penalty parameter α and counters are initialized, the *Sequence* is set to an empty set, and the current solution is set to the initiating solution S_i. At each step towards the guiding solution, DNRM selects the best solution, according to the penalized evaluation function (Eq. (6)), from the union of the restricted reallocate neighborhood $RN_R(S)$ and the restricted exchange neighborhood $RN_E(S)$. If the selected solution is two units closer to the guiding solution compared to the current solution, the counter *count* is incremented by two, otherwise it is incremented by one. The current solution is then set to the selected best solution, and it is included into the *Sequence* as an intermediate solution of the path. The number of steps needed by DNRM to accomplish the path building process from the initiating solution to the guiding solution is bounded by $Dist(S_i, S_g)$.

Algorithm 3 Pseudo-code of the double-neighborhood relinking method.

```
Input: An initiating solution S_i and a guiding solution S_g;
Output: A Sequence of intermediate solutions;
Perform a matching of the knapsacks of S_i and S_g, and adjust the
knapsack numbering of S_g according the matching result.
Compute the distance Dist(S_i, S_g).
α ← f(S_i)/W(S_i)
count ← 0
Sequence ← ∅
S ← S_i
while count < Dist(S_i, S_g) do
   Select a solution S' ∈ RN_R(S) ∪ RN_E(S) that maximizes φ(S').
   if Dist(S, S_g) − Dist(S', S_g) = 2 then
      count ← count + 2
   else
      count ← count + 1
   end if
   S ← S'
   Sequence ∪ S'
end while
return Sequence
```

3.4. Path solution selection

Following [22], once a relinking path is built from an initiating solution S_i to the guiding solution S_g, some intermediate solutions within the path are selected and further improved by local refinement. Note that in our case, two consecutive solutions on the relinking path differ only in the attribute that was just introduced. Consequently, applying local optimization to solutions close to S_i or S_g would very probably lead to the same local optimum. For this reason, we select the path solution by considering both its quality and distance to S_i or S_g according to the following two rules:

- **Rule 1:** We first try to select a best feasible solution from the middle three fifths of the path. To do so, we first identify a subset of solutions in *Sequence* denoted as *subSeq* (*subSeq* ⊂ *Sequence*) such that $\forall S \in subSeq$, $Dist(S_i, S) \geq Dist(S_i, S_g)/5$ and $Dist(S_g, S) \geq Dist(S_i, S_g)/5$. We then pick the best feasible solution S from *subSeq* with the largest objective value $f(S)$.
- **Rule 2:** If there is no feasible solution in *subSeq*, we select the infeasible solution S in the middle of the path (i.e., $Dist(S_i, S) = Dist(S_i, S_g)/2$ or $Dist(S_i, S) = Dist(S_i, S_g)/2 + 1$ given that the move step can be either one unit or two units). We then apply a Repair Method (see Section 3.5) to bring the infeasible solution back to the feasible region before submitting it to the Local Refinement procedure.

3.5. The repair method

If the selected solution in the relinking path is infeasible, we apply an aggressive neighborhood search method (ANSM) to repair this solution. Basically, ANSM jointly employs two well-known operators *REAL* and *EXC* to carry out a best improvement local search. At each iteration, all solutions that can be reached by these two operators from the current solution (without confining to any constraint) are examined according to the penalty-based evaluation function (Eq. (6)), and the best one (i.e., the one with the largest evaluation value) is chosen as the output solution of this stage. The current solution will only be replaced by the output solution if the latter is better than the current solution. The output of the whole repairing procedure is assured to be a local optimum since ANSM conducts a thorough search within the neighborhoods defined by the two operators. We can thus expect

that the repaired feasible solution could be able to improve the best solution found so far as well.

To redirect the search towards feasible region, we initialize the penalty parameter α of Eq. (6) with a relatively large value: $\alpha = 10^i * f(S)/W(S)$, where S is the current solution and i is set to 1 at the beginning. The value of α is kept unchanged during the search. If one round of aggressive neighborhood search is not enough to repair the input solution, i is incremented by one and α is recalculated. ANSM then restarts a new round of its search. This is repeated until ANSM has been launched three consecutive times. If the resulting solution is still infeasible (this happens rarely), a greedy repair procedure is triggered. The greedy repair procedure consists in simply removing the least conditionally attractive objects from the capacity-violated knapsacks until this constraint becomes satisfied. The pseudo-code of the aggressive neighborhood search repair method is presented in Algorithm 4.

Algorithm 4 Pseudo-code of the repair method.

```
Input: An infeasible solution S_inf;
Output: A feasible solution S_f;
S ← S_inf
i ← 1
while i <= 3 do
    α ← 10^i * f(S)/W(S)
    S ← agressiveNeighSearch(S, α).
    if S is feasible then
        break
    else
        i ← i + 1
    end if
end while
if S is infeasible then
    S ← greedyRepair(S)
end if
return S
```

3.6. The local refinement method

The Local Refinement component has a significant impact to the overall performance of the proposed EPR algorithm. For our purpose, we adopt the responsive threshold search algorithm (RTS) introduced in [19]. RTS was originally proposed with a perturbation operation where the overall algorithm is called iterated responsive threshold search (IRTS). The purpose of the perturbation operation is to help the search escape from deep local optima. In the case of EPR, such a functionality is realized by the path relinking method coupled with the path solution selection rules in EPR. Therefore, the perturbation operation is not useful and thus excluded from EPR.

RTS basically alternates between a threshold-based exploration phase (Exploration for short) and a descent-based improvement phase (Improvement for short). Starting from an initial solution which is a feasible solution either picked in the relinking path or produced by the Repair Method, RTS realizes an Exploration phase which is composed of L (L is a parameter) calls of the *Threshold Based Exploration* procedure which is based on three move operators (i.e., *REAL, EXC* and *DROP*). Note that in RTS, the *REAL* operator excludes the case of displacing an assigned object to knapsack 0, which is actually accomplished by the *DROP* operator. At each iteration, RTS accepts any encountered neighboring solution that satisfies a responsive quality threshold T. This threshold is dynamically determined according to the recorded best local optimum value (f_p) and a threshold ratio r: $T = (1 - r) \times f_p$, where r is calculated by an inverse proportional function with respect to f_p (see [19] for details). After an Exploration phase, RTS switches to the Improvement phase for intensification purpose. In the Improvement phase, RTS continues accepting the first met improving neighboring solutions which are induced by *REAL* or *EXC* operator until no improving solutions can be found. If the local optimum eventually attained during the Improvement phase has a better objective value than the recorded best local optimum value (f_p), RTS updates f_p as well as the threshold ratio r, and then restarts a new round of Exploration–Improvement phases. RTS terminates when f_p has not been updated for a consecutive W times of Exploration–Improvement phases.

To accelerate neighborhood examination, RTS restricts its search within the feasible neighboring solutions when operators *REAL* and *EXC* are applied. No restriction is needed for the *DROP* operator since it never generates infeasible solution. The neighborhoods are explored in a token-ring way during both the Exploration phase and the Improvement phase [19].

3.7. The pool updating strategy

For each new solution S_0 which can be the best feasible solution in the relinking path, the best feasible solution obtained by the Repair Method or the best solution produced by the Local Refinement procedure, we need to decide whether S_0 should be inserted into *RefSet* or not. To make this decision, we employ a fitness-based replacement strategy (FBRS) for updating both *RefSet* and *PairSet* at each generation.

With FBRS, EPR replaces the initiating solution S_{i_0} (which is always worse than the guiding solution S_{g_0}) with S_0 if the following two conditions are simultaneously verified: (1) S_0 is better than the initiating solution S_{i_0}; (2) S_0 is not a clone of any other solution in *RefSet*. Once S_0 is inserted into *RefSet*, we update *PairSet* accordingly by inserting all the previously removed index pairs that are associated with S_0 into *PairSet*. For each inserted index pair (i, g), we ensure that the objective value of S_i is smaller than that of S_g. Then, the index pairs in *PairSet* are ranked in descending order of the objective value $f(S_g)$ of the guiding solution S_g, and for solutions with the same $f(S_g)$, they are ranked in ascending order of the objective value of the initiating solution $f(S_i)$. The pool updating procedure is shown in Algorithm 5.

Algorithm 5 Pseudo-code of the pool updating procedure.

```
Input: RefSet, PairSet, an offspring solution S_0, an initiating solution
    S_{i_0}, a guiding solution S_{g_0};
Output: RefSet, PairSet;
if f(S_0) > f(S_i) and S_0 is not a clone of any solution S ∈ RefSet
then
    RefSet ← (RefSet ∪ {S_0})\S_i;
    Update PairSet;
    Rank index pairs in PairSet;
else
    PairSet ← PairSet\(i_0, g_0);
end if
return RefSet,PairSet
```

3.8. Discussion

By properly adapting the PR method to the QMKP at hand, the proposed EPR algorithm has a number of notable characteristics. First, unlike other evolutionary frameworks where an improved offspring solution is typically provided by some variant of a local refinement procedure, our EPR algorithm uses the path relinking procedure and the repair procedure as two complementary means for generating improved offspring solutions. This is achieved thanks to the mechanisms used in these two procedures which allow a controlled exploration of infeasible solutions. We provide experimental data in

Section 5.2 to show that these two procedures help to identify high quality solutions.

Second, our EPR algorithm explores only one directional path starting from a *worse* initiating solution and ends at a *better* guiding solution. A solution in the path is picked for local refinement when suitable conditions (e.g., there is no improved solution along the path) are verified. This is different from the typical practice where two paths are usually built by reversing the initiating solution and the guiding solution. According to our observations, building an extra link seldom achieves improved solutions while consuming more computing time. Therefore, our strategy to explore one directional path proves to be a good choice when the allowed computing time is limited as in our case.

Third, we update both *RefSet* and *PairSet* at each generation when an improved offspring solution is identified. This strategy accelerates the evolution of our algorithm by dynamically transferring the improved solution pairs taken from *RefSet* into *PairSet* which helps to maintain the elitism of the initiating and guiding solution. Our strategy is different from the typical strategy where one finishes examining all solution pairs in the current *PairSet* before proceeding to examine the updated *PairSet*. We investigate the implications of these differences in Section 5.3.

Finally, when all solution pairs in the *PairSet* have been examined, our EPR algorithm starts a new round of evolutionary path relinking process. This restart mechanism provides a form of diversification and is able to displace the search to a distant unexplored region. Investigations presented in Section 5.3 show the usefulness of this mechanism.

4. Computational experiments

To evaluate the performance of the proposed EPR algorithm, we conducted extensive experiments on 90 benchmark instances. The assessment was performed by comparing our results to those of the state-of-the-art methods and the current Best Known Results (BKR) ever reported in the literature.

4.1. Benchmark instances

The 90 benchmark instances used in our experiments belong to the two sets:

- **Set I:** This set consists of 60 well-known benchmarks which are commonly used for the QMKP algorithm assessment in the literature [1,14,15,17–20]. Built from the quadratic knapsack problem (QKP) instances introduced in [37] which can be download from: http://cedric.cnam.fr/~soutif/QKP/QKP.html, these instances are characterized by their number of objects $n \in \{100, 200\}$, the density $d \in \{0.25, 0.75\}$ (i.e., the 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. The optimal solutions of these instances are unknown.
- **Set II:** The second set is composed of 30 new large instances with 300 objects (i.e., $n = 300$), with unknown optima. Like the instances of Set I, they are also characterized by their density $d \in \{0.25, 0.75\}$ and the number of knapsacks $m \in \{3, 5, 10\}$. To build these 30 QMKP instances, we first randomly generated 10 QKP instances (5 instances for each density) using the method introduced in [37]. Based on each QKP instance, we then created 3 QMKP instances, each with a different number of knapsacks, and a knapsack capacity set to 80% of the sum of object weights divided by the number of knapsacks. These new instances can be downloaded from: http://www.info.univ-angers.fr/pub/hao/qmkp.zip.

4.2. Experimental settings

The proposed EPR algorithm was programmed in C++,[1] and compiled with GNU g++ on an Intel Xeon E5440 processor (2.83 GHz and 2 GB RAM) with '-O3' flag. Without using a compiler optimization flag, it requires respectively 0.44, 2.63 and 9.85 s to solve the well-known DIMACS machine benchmark graphs[2] r300.5, r400.5 and r500.5 on our machine.

Like other QMKP algorithms, the proposed EPR algorithm has a number of parameters to be tuned. Most of the parameters are required by the RTS local refinement procedure for which we adopt the same values used in [19] (which were identified by conducting a careful statistical analysis). As suggested in [22], EPR maintains a fairly small-sized reference set of 10 elite solutions. Even if it would be possible to find a parameter setting better than the setting used in this paper, the experimental results reported in this section show that the adopted setting performs very well on the tested benchmark instances.

Note that it is a common practice in the QMKP literature to use a time cutoff as the stopping condition to evaluate the algorithm performance. In our case, we imposed a time limit of 15 s, 90 s and 180 s for instances with respectively 100 objects, 200 objects and 300 objects. The first two time limits for the instances of Set I have been used in [19]. Under these stopping conditions, we focus primarily on comparing the solution quality, the deviation and the success rate over multiple runs of the competing algorithms.

4.3. Comparative results on the instances of Set I

This section is dedicated to the computational results obtained by our EPR algorithm on the 60 instances of Set I. EPR was run 40 times for each instance and for each run the program was terminated with a time limit of 15 s for the instances with 100 objects and 90 s for those with 200 objects. We also included a comparative study of our results with respect to those of the state-of-the-art algorithms.

Table 1 shows the results of our EPR algorithm along with the best known results ever reported in the literature. The characteristics of each instance are listed from Columns 1–5: the number of objects n, density d, number of knapsacks m, instance identity number I and knapsack capacity C. Column 6 gives the best known lower bounds which are extracted from the results reported in [19]. Columns 7–11 show our results, including the overall best lower bound (f_{best}), the average value of the 40 best lower bounds (f_{avg}), the standard deviation of the 40 best lower bounds (sd), the number of runs where EPR reach f_{best} (hit) and the earliest CPU time in seconds over the 40 runs when f_{best} is first reached (CPU).

Table 1 shows that EPR achieves a very competitive performance by matching (bold entries) or improving (starred bold entries) all the best known lower bounds. Specifically, our EPR can reach the previous best known results for 50 out of 60 cases, and for 18 out of these 50 cases, EPR achieves a success rate of 100% which means one run suffices for EPR to attain the best known lower bound. More importantly, for 10 instances, EPR is able to attain an improved result over the previous best known lower bound.

To further evaluate the performance of the proposed EPR algorithm, we show a comparison of our results with those of 3 recent best performing algorithms:

- TIG: A tabu-enhanced iterated greedy algorithm [17].
- SO: A strategic oscillation algorithm [18].
- IRTS: An iterated responsive threshold search algorithm [19].

[1] The best solution certificates are available at http://www.info.univ-angers.fr/pub/hao/EPRresults.zip.

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

Table 1
Computational results of EPR on the 60 benchmark instances of Set I with 15 s per run for instances with 100 objects and 90 s per run for instances with 200 objects.

Instance					f_{bk}	EPR				
n	d	m	I	C		f_{best}	f_{avg}	sd	Hit	CPU(s)
100	25	3	1	688	29,286	**29,286**	29,286.00	0.00	40	0.10
100	25	3	2	738	28,491	**28,491**	28,491.00	0.00	40	0.08
100	25	3	3	663	27,179	**27,179**	27,179.00	0.00	40	0.25
100	25	3	4	804	28,593	**28,593**	28,593.00	0.00	40	0.11
100	25	3	5	723	27,892	**27,892**	27,892.00	0.00	40	0.16
100	25	5	1	413	22,581	**22,581**	22,568.80	25.31	31	0.38
100	25	5	2	442	21,704	**21,704**	21,671.70	8.43	1	4.78
100	25	5	3	398	21,239	**21,239**	21,239.00	0.00	40	0.18
100	25	5	4	482	22,181	**22,181**	22,180.95	0.31	39	0.16
100	25	5	5	434	21,669	**21,669**	21,660.85	14.83	27	0.92
100	25	10	1	206	16,221	**16,221**	16,205.17	8.10	4	4.27
100	25	10	2	221	15,700	**15,700**	15,683.98	33.44	29	0.41
100	25	10	3	199	14,927	**14,927**	14,866.50	28.92	3	7.36
100	25	10	4	241	16,181	**16,181**	16,181.00	0.00	40	1.12
100	25	10	5	217	15,326	**15,326**	15,297.52	36.37	24	0.92
200	25	3	1	1381	101,471	**101,471**	101,467.10	6.87	24	9.25
200	25	3	2	1246	107,958	**107,958**	107,958.00	0.00	40	0.22
200	25	3	3	1335	104,589	**104,589**	104,581.68	7.22	19	11.04
200	25	3	4	1413	100,098	**100,136***	100,136.00	0.00	40	10.81
200	25	3	5	1358	102,311	**102,311**	102,308.65	2.26	19	12.99
200	25	5	1	828	75,623	**75,623**	75,578.35	37.77	12	25.07
200	25	5	2	747	80,033	**80,033**	80,009.27	41.36	29	5.42
200	25	5	3	801	78,043	**78,043**	78,036.10	20.82	36	3.56
200	25	5	4	848	74,140	**74,140**	74,080.12	41.73	1	21.59
200	25	5	5	815	76,610	**76,610**	76,610.00	0.00	40	1.10
200	25	10	1	414	52,293	**52,293**	52,147.97	112.48	1	62.77
200	25	10	2	373	54,830	**54,830**	54,696.15	76.22	7	8.48
200	25	10	3	400	53,661	**53,678***	53,596.60	38.09	1	64.25
200	25	10	4	424	51,297	**51,302***	51,096.15	78.88	1	60.67
200	25	10	5	407	53,621	**53,621**	53,575.57	40.06	8	9.71
100	75	3	1	669	69,977	**69,977**	69,977.00	0.00	40	0.08
100	75	3	2	714	69,504	**69,504**	69,504.00	0.00	40	0.07
100	75	3	3	686	68,832	**68,832**	68,832.00	0.00	40	0.07
100	75	3	4	666	70,028	**70,028**	70,028.00	0.00	40	0.04
100	75	3	5	668	69,692	**69,692**	69,692.00	0.00	40	0.12
100	75	5	1	401	49,421	**49,421**	49,417.40	8.57	34	0.25
100	75	5	2	428	49,365	**49,400***	49,360.47	17.20	5	0.58
100	75	5	3	411	48,495	**48,495**	48,495.00	0.00	40	0.11
100	75	5	4	400	50,246	**50,246**	50,246.00	0.00	40	0.41
100	75	5	5	400	48,753	**48,753**	48,749.25	8.93	16	0.62
100	75	10	1	200	30,296	**30,296**	30,231.95	75.62	10	5.49
100	75	10	2	214	31,207	**31,207**	31,120.22	41.50	3	1.39
100	75	10	3	205	29,908	**29,908**	29,900.60	12.57	25	0.43
100	75	10	4	200	31,762	**31,762**	31,717.47	41.55	18	0.57
100	75	10	5	200	30,507	**30,507**	30,465.35	25.29	8	4.40
200	75	3	1	1311	270,718	**270,718**	270,718.00	0.00	40	1.88
200	75	3	2	1414	257,288	**257,288**	257,287.73	1.72	39	6.34
200	75	3	3	1342	270,069	**270,069**	270,069.00	0.00	40	1.79
200	75	3	4	1565	246,993	**24,6993**	246,963.23	34.73	7	38.53
200	75	3	5	1336	279,598	**279,598**	279,598.00	0.00	40	6.43
200	75	5	1	786	185,493	**185,493**	185,487.58	28.10	37	7.06
200	75	5	2	848	174,836	**174,836**	174,814.30	27.54	21	4.87
200	75	5	3	805	186,774	**186782***	186,737.12	27.72	2	39.44
200	75	5	4	939	166,990	**167,142***	166,957.67	96.46	2	24.71
200	75	5	5	801	193,310	**193,310**	193,219.40	40.89	1	37.23
200	75	10	1	393	113,139	**113,324***	112972.98	158.98	1	84.97
200	75	10	2	424	105,807	**105,966***	105554.00	143.34	1	69.09
200	75	10	3	402	114,596	**114,860***	114397.62	141.19	1	82.61
200	75	10	4	469	99,106	**99,422***	98,875.45	146.35	1	39.89
200	75	10	5	400	117,309	**117,309**	116867.85	126.49	1	77.89

We divide the 60 instances of Set I into 12 classes, each containing 5 instances, which are indicated in $n - d - m$ where n is the number of objects, d is the density and m is the number of knapsacks. Table 2 summarizes the statistical results of our algorithm as well as those of the reference algorithms. For each algorithm, we list the number of instances out of the 5 instances in each class where the corresponding algorithm achieves the best known lower bound (*#Bests*). The best known lower bounds used in this comparative study are compiled from the previous best known lower bounds (Column f_{bk} of Table 1) and the best lower bound obtained by our EPR algorithm (Column f_{best} of Table 1). We also list in Table 2 the average of the average results (Avg., for all algorithms) and the average of the standard deviation (SD, for IRTS and EPR) across the 5 instances of each instance class. The detailed results of the reference algorithms (TIG, SO, IRTS), which were obtained by running the source code of each algorithm 40 times in our computing environment under exactly the

Table 2
Comparison of EPR with three state-of-the-art methods on Set I: TIG [17], SO [18], and IRTS [19].

Inst.	TIG		SO		IRTS			EPR		
	#Bests	Avg.	#Bests	Avg.	#Bests	Avg.	SD	#Bests	Avg.	SD
100-25-3	4	28,198.56	**5**	28,253.59	**5**	**28,288.20**	**0.00**	**5**	**28,288.20**	**0.00**
100-25-5	2	21,773.08	2	21,782.05	**5**	21,854.16	15.93	**5**	**21,864.26**	**9.78**
100-25-10	2	15,541.86	0	15,453.85	**5**	15,638.42	24.53	**5**	**15,646.83**	**21.37**
200-25-3	0	102,635.96	1	102,753.30	4	103,273.20	5.45	**5**	**103,290.29**	**3.27**
200-25-5	0	76,086.16	0	76,007.99	**5**	76,853.04	28.84	**5**	**76,862.77**	**28.34**
200-25-10	0	52,261.18	0	51,951.31	3	53,004.66	**57.09**	**5**	**53,022.49**	69.15
100-75-3	4	69,592.90	4	69,585.12	**5**	69,605.72	2.07	**5**	**69,606.60**	**0.00**
100-75-5	3	49,147.68	2	49,133.77	4	49,220.42	50.42	**5**	**49,253.62**	**6.94**
100-75-10	0	30,413.96	0	30,397.46	**5**	30,679.14	41.47	**5**	**30,687.12**	**39.31**
200-75-3	4	264,833.60	3	264,761.80	**5**	264,866.20	111.56	**5**	**264,927.19**	**7.29**
200-75-5	0	181,082.40	0	180,904.00	3	181,246.00	130.11	**5**	**181,443.21**	**44.14**
200-75-10	0	109,161.98	0	108,967.79	1	109,682.30	**130.96**	**5**	**109,733.58**	143.27

same stopping criterion as the one used in this paper, were extracted directly from the results reported in [19]. In this regard, we can say that the comparison performed in this section is quite fair.

From Table 2, we observe that the proposed EPR algorithm competes very favorably with the 3 reference state-of-the-art algorithms in all listed indicators. Indeed, EPR easily dominates TIG and SO by obtaining a higher number of best known lower bounds and a better average value for almost all 12 instance classes. EPR also outperforms its competitive counterpart IRTS by attaining more best known lower bounds (60 vs. 50). Moreover, compared to IRTS, EPR always achieves a better average result and usually attains a smaller standard deviation which demonstrates that EPR performs more stably. Given that EPR uses a simplified version of IRTS as its local refinement procedure, this comparison demonstrates the usefulness of the evolutionary path-relinking procedure compared to the single trajectory counterpart.

To validate our conclusion, we applied the Wilcoxon test with a significance factor of 0.05 for pairwise comparisons between our results (both best and average) and those of the reference algorithms. Table 3 shows the statistical test outcomes where the left part is for the best result comparison and the right part is for the average result comparison. For each comparison item and for each algorithm pair, we list in Table 3 the positive rank sum ($R+$), the negative rank sum ($R-$), the resulting p-value (*p-value*) and whether they are significant different (*Diff*?). From Table 3, we can see that EPR is statistically different from any of the three reference algorithms both in terms of best result and average result. Moreover, the fact that the positive rank sum is consistently larger than the negative rank sum in both comparison items confirms the dominance of our EPR algorithm over the reference algorithms.

4.4. Comparative results on the instances of Set II

The computational results and comparative study of Section 4.3 demonstrated that the proposed EPR algorithm performs very well on the set of commonly used instances with up to 200 objects. Now we are interested to know how the proposed algorithm will perform on larger instances. For this purpose, we conduct an additional experiment on the 30 instances of Set II with 300 objects. For this experiment, we adopt a time limit of 180 s and again run our EPR algorithm 40 times for each instance. For comparative purposes, we also provide the results obtained by TIG [17], SO [18] and IRTS [19] by running their source codes on our machine under exactly the same stopping condition.

Computational results of our EPR algorithm as well as those of the 3 reference algorithms are reported in Table 4 where we list for each instance the best lower bound (f_{best}) and the average result (f_{avg}) obtained over 40 runs by each algorithm. Table 4 discloses a dominance of our EPR algorithm over the reference algorithms. Indeed, EPR attains the best lower bound for *all* instances of Set II. Such a performance is not matched by any of the 3 reference algorithms. Moreover, EPR obtains a best average result for 27 out of 30 cases (90%). The average performance is also the best among the compared algorithms. When compared with TIG and SO, even our average results are better than their best lower bounds for most cases.

In order to evaluate the statistical significance of our findings, we applied the Wicoxon test with a significance factor of 0.05 for pairwise comparison of our results with those of the reference algorithms on the instances of Set II. Table 5 summarizes the test results where the left part of the table provides the statistical data with the best results as input, and the right part shows the statistical data with the average results as input. Table 5 discloses that a statistical difference is detected for each compared case (between EPR and any reference algorithm) with a p-value smaller than 0.05. The superiority of our EPR algorithm over the best performing reference algorithms is confirmed by the fact that the sum of the positive ranks is always significantly larger than the sum of the negative ones.

Without bothering to give a detailed tabulation of additional results, we mention that, when more computing time is allowed, EPR can further improve its best results for some of the instances and the above conclusions still hold. Indeed, when we set the time limit to 360 seconds for all methods, our EPR algorithm always outperforms the reference algorithms.

5. Analysis

The computational outcomes and especially the comparisons of EPR with its counterpart IRTS [19] presented in Section 4 have shown the strength of the evolutionary path relinking framework. In this section, we conduct additional experimental analyses to gain a deeper understanding of the important ingredients of the proposed EPR algorithm.

5.1. The repair method

When an infeasible solution is picked from the relinking path, EPR adopts an aggressive neighborhood search method (ANSM) to repair this solution. A greedy procedure is triggered as a complement when the solution cannot be repaired by ANSM. The greedy procedure simply drops the least conditionally attractive objects from the capacity-violated knapsacks until each knapsack becomes satisfied. It would be interesting to determine if the greedy procedure alone is sufficient for EPR to achieve its best performance. To test this, we conduct an experiment to compare the performance of EPR with its alternative version EPR_{GR} which repairs an infeasible solution solely with the indicated greedy procedure. Each algorithm is run 40 times on the instances of Set I. Table 6 summarizes the computational results in the form of 12 instance classes. In this experiment, we set the best known

Table 3
Wilcoxon test for results of Set I.

Algorithm pair	Best result				Average result			
	R+	*R*−	*p*-value	Diff?	*R*+	*R*−	*p*-value	Diff?
EPR vs TIG	861	0	2.52e−08	Yes	1483	2	1.87e−10	Yes
EPR vs SO	946	0	1.16e−08	Yes	1711	0	3.60e−11	Yes
EPR vs IRTS	55	0	5.92e−03	Yes	1082	94	4.16e−07	Yes

Table 4
Comparative results of EPR with three state-of-the-art algorithms (TIG [17], SO [18], IRTS [19]) on the 30 large-sized instances of Set II with a time limit of 180 s. A value in bold indicates a best known lower bound or a best average result over 40 runs.

Instance					TIG		SO		IRTS		EPR	
n	d	m	I	C	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
300	25	3	1	2048	223,243	221,842.77	223,291	222,986.12	**223,661**	223,514.08	**223,661**	**223,609.52**
300	25	3	2	2058	209,202	207,786.02	209,940	209,320.70	**210,981**	210,812.08	**210,981**	**210,958.70**
300	25	3	3	2090	209,296	208,036.48	209,621	208,962.30	**210,910**	210,732.25	**210,910**	**210,886.02**
300	25	3	4	2104	214,624	212,620.15	214,773	214,348.10	215,639	215,564.62	**215,732**	**215,608.40**
300	25	3	5	2045	211,378	209,392.75	211,567	211,216.77	**212,432**	212,429.25	**212,432**	**212,432.00**
300	25	5	1	1229	162,924	160,672.77	162,952	162,529.55	163,668	163,539.35	**163,746**	**163,654.27**
300	25	5	2	1234	151,825	149,890.20	151,533	150,989.67	152,860	152,728.83	**152,951**	**152,770.45**
300	25	5	3	1254	152,233	149,855.35	152,043	151,572.08	153,347	153,183.75	**153,489**	**153,313.50**
300	25	5	4	1262	153,289	149,752.70	155,179	153,822.45	**156,340**	156,235.95	**156,340**	**156,256.20**
300	25	5	5	1227	153,642	150,678.05	153,592	153,022.08	**154,936**	154,724.73	**154,936**	**154,836.83**
300	25	10	1	614	107,929	103,716.35	107,525	107,083.35	**109,400**	109,275.38	**109,400**	**109,319.45**
300	25	10	2	617	100,931	97,292.02	100,699	100,196.70	102,306	102,049.90	**102,383**	**102,078.30**
300	25	10	3	627	102,395	98,351.38	102,338	101,711.60	103,707	**103,510.95**	**103,794**	103,508.18
300	25	10	4	631	103,284	98,986.73	103,177	102,482.77	105,290	105,035.38	**105,294**	**105,049.98**
300	25	10	5	613	103,034	99,035.18	102,649	102,073.38	104,120	103,927.52	**104,218**	**104,019.32**
300	75	3	1	2073	589,453	587,619.35	589,453	589,277.88	589,453	589,139.00	**589,739**	**589,559.28**
300	75	3	2	1892	641,192	640,637.22	641,085	641,082.75	641,085	640,829.32	**641,610**	**641,600.75**
300	75	3	3	2065	598,000	596,776.55	597,965	597,568.53	**598,124**	597,239.70	**598,124**	**598,012.30**
300	75	3	4	2170	**581,227**	580,539.10	**581,227**	581,054.40	**581,227**	580,796.68	**581,227**	**581,121.30**
300	75	3	5	1931	**612,383**	610,555.22	612,369	**612,206.50**	612,373	611,860.40	**612,383**	612,164.12
300	75	5	1	1244	404,902	403,472.90	404,909	404,110.95	405,050	404,716.35	**405,191**	**404,909.03**
300	75	5	2	1135	445,497	444,288.58	445,195	444,656.80	**445,655**	445,306.38	**445,655**	**445,557.30**
300	75	5	3	1239	405,800	405,167.90	405,863	405,123.08	406,556	405,820.47	**406,800**	**406,172.22**
300	75	5	4	1302	395,648	394,453.80	395,422	394,707.42	395,760	395,345.10	**396,021**	**395,514.78**
300	75	5	5	1159	415,057	412,957.67	414,447	414,085.50	415,400	415,032.12	**415,804**	**415,179.58**
300	75	10	1	622	247,429	246,408.15	247,481	246,749.67	248,006	247,650.23	**248,136**	**247,765.67**
300	75	10	2	567	266,922	265,934.88	266,877	266,523.10	267,728	267,399.05	**268,003**	**267,491.38**
300	75	10	3	619	239,088	237,844.67	238,587	238,139.98	239,661	239,358.02	**239,875**	**239,407.88**
300	75	10	4	651	230,355	228,457.08	229,855	229,378.48	231,744	231,043.75	**231,812**	**231,412.27**
300	75	10	5	579	248,488	247,048.98	248,114	247,681.90	249,476	249,113.02	**249,668**	**249,171.75**

Table 5
Wilcoxon test for results of Set II.

Algorithm pair	Best result				Average result			
	R+	*R*−	*p*-value	Diff?	*R*+	*R*−	*p*-value	Diff?
EPR vs TIG	460	0	4.00e−06	Yes	465	0	1.86e−09	Yes
EPR vs SO	435	0	2.70e−06	Yes	464	1	3.73e−09	Yes
EPR vs IRTS	210	0	9.57e−05	Yes	463	2	5.59e−09	Yes

lower bound of each instance to the one obtained by our EPR. For each instance class, we report the number of instances for which the algorithm is able to achieve or improve the best known lower bound (#Bests), the average of the 5 average results (Avg.), the average of the 5 standard deviations (SD.) and the average of the 5 hits (Hit.). From Table 6, we can see that EPR always attains a higher number of best known lower bounds and a better average result than EPR_{GR}. Moreover, EPR usually achieves a higher hit value than EPR_{GR} even if EPR and EPR_{GR} have a comparable standard deviations. A Wilcoxon test with a significance factor of 0.05 is applied to compare both the best results and the average results obtained by these two algorithm variants. The resulting *p-values* of 2.48e−2 and 8.68e−4 clearly show the statistical difference between these two groups of data. The superiority of EPR over EPR_{GR} is validated by the larger positive rank sum compared to the negative ones. This experiment confirms the merit of the adopted repair method with respect to the greedy method for EPR.

5.2. Impact of searching infeasible solutions

EPR allows infeasible solutions to be generated during its path relinking process. To handle this situation, the Repair Method takes an infeasible solution as input and explores infeasible spaces around the feasibility boundary. Indeed, with the evaluation function defined in Formula (6), the search always retains the possibility of re-entering the infeasible region even if it is currently in the feasible region. In the case where only improving solutions are accepted, a transition from a feasible solution (say S) to an infeasible one (say S') can be made when the infeasible solution has a high objective value ($f(S')$) and a low constraint violation ($V(S')$) which yields consequently a higher penalized objective value ($\phi(S')$) compared to the current feasible solution, i.e., $\phi(S') > \phi(S)$. In this section, we discuss how those features contribute to the overall performance of our EPR algorithm. Indeed, it has long been recognized in the constrained optimization literature that it is useful to allow controlled exploration of infeasible

Table 6
Different strategies for the repairing procedure.

	Algo.	#Bests	Avg.	SD.	Hit.
100-25-3	EPR	5	28,288.20	0.00	40.00
	EPR_{GR}	5	28,288.20	0.00	40.00
100-25-5	EPR	5	21,864.26	9.78	27.60
	EPR_{GR}	5	21,862.51	10.84	26.80
100-25-10	EPR	5	15,646.83	21.37	20.00
	EPR_{GR}	5	15,644.08	22.73	19.60
200-25-3	EPR	5	103,290.29	3.27	28.40
	EPR_{GR}	4	103,273.96	6.39	15.60
200-25-5	EPR	5	76,862.77	28.34	26.80
	EPR_{GR}	5	76,860.14	23.18	23.20
200-25-10	EPR	5	53,022.49	69.15	3.60
	EPR_{GR}	2	53,008.23	69.01	4.40
100-75-3	EPR	5	69,606.60	0.00	40.00
	EPR_{GR}	5	69,606.60	0.00	40.00
100-75-5	EPR	5	49,253.62	6.94	27.00
	EPR_{GR}	5	49,253.45	6.00	28.80
100-75-10	EPR	5	30,687.12	39.31	12.80
	EPR_{GR}	5	30,681.94	39.05	12.20
200-75-3	EPR	5	264,927.19	7.29	33.20
	EPR_{GR}	5	264,921.86	22.93	32.20
200-75-5	EPR	5	181,443.21	44.14	12.60
	EPR_{GR}	4	181,391.65	90.60	10.20
200-75-10	EPR	5	109,733.58	143.27	1.00
	EPR_{GR}	1	109,709.86	112.50	1.00

Table 7
Statistics on the relinking method and the repair method. For each representative instance, we show the number of runs over 40 executions in which the relinking method attains the best lower bound of that run (RMHit), and the number of runs in which the repair method achieves the best lower bound of that run (RPHit).

Instance				RMHit	RPMHit
n	d	m	l		
100	25	3	5	0/40	2/40
100	25	5	1	1/40	3/40
100	25	10	2	5/40	1/40
200	25	3	4	2/40	25/40
200	25	5	1	2/40	6/40
200	25	10	2	3/40	0/40
100	75	3	4	1/40	1/40
100	75	5	4	0/40	3/40
100	75	10	2	4/40	1/40
200	75	3	3	0/40	6/40
200	75	5	1	2/40	2/40
200	75	10	3	3/40	2/40

solutions in order to ease solution transitions between structurally different feasible solutions. (The conjectured relevance of this outcome was the basis for the version of the strategic oscillation approach that crosses feasibility boundaries. See [29] for a historical discussion and an illustration of the power of this strategy.) By introducing such a possibility in EPR, we expect that EPR can identify high quality solutions when the search cross back and forth the boundary between feasible and infeasible regions.

To support our expectation, we give some statistical data in Table 7 where we select one representative instance from each instance class and report the number of runs in which the relinking method achieves the best lower bound of that run (RMHit), and the number of runs in which the repair method attains the best lower bound of that run (RPMHit). From Table 7, we can see that these two methods sometimes obtain the best lower bound in a single execution of our EPR algorithm. The relinking method can discover a best lower bound up to 5 out of 40 runs for instance 100-25-10-2, and the repair method is able to find a best lower bound up to 25 out of 40 runs for instance 200-25-3-4. Recall that except for identifying improved solutions, another mission for the relinking method and the repair method is to provide a good starting point for the Local Refinement method (i.e., the responsive threshold search method, see Section 3.6). The excellent performance of EPR presented in Section 4 demonstrates that these methods have fulfilled this mission.

Table 8
Different strategies for the pool updating method.

		Restart	NoRestart
PUS1	#Bests	60	55
	Avg.	0.00	−0.52
PUS2	#Bests	51	51
	Avg.	−1.02	−1.14

5.3. The pool updating strategy

EPR maintains a reference set (*RefSet*) of elite solutions and a pair set (*PairSet*) for path relinking. At each path relinking and Local Refinement, *RefSet* is updated with the offspring solution if the offspring is better than the initiating solution. Then *PairSet* is immediately updated according to the rule presented in Section 3.7. This updating rule for *PairSet* (denoted by PUS1) is different from the conventional strategy (denoted by PUS2) as suggested in [22]. Unlike PUS1 which updates *PairSet* after each path relinking, PUS2 updates *PairSet* only when all solution pairs in the current *PairSet* have been examined, though *RefSet* is always updated by replacing the worst solution with an improved offspring solution.

Apart from the pool updating strategy, we also introduce a restart mechanism within our EPR algorithm. When all solution pairs in *PairSet* have been examined and if the stopping condition is not reached, EPR starts a new round of the evolutionary path relinking process by reinitializing *RefSet* and *PairSet*. The best solution found so far becomes a member of the new *RefSet* and the remaining solutions are generated in the same way as in the first round. To investigate the efficacy of our pool updating strategy and the restart mechanism, we carry out an experiment on the instances of Set I to compare the performance of four algorithm variants EPR_{1R} (PUS1 with restart), EPR_{1NR} (PUS1 without restart), EPR_{2R} (PUS2 with restart), EPR_{2NR} (PUS2 without restart). These four algorithm variants are the same except in the pool updating part. We note that EPR_{1R} corresponds to the EPR algorithm. Table 8 summarizes the computational results which show that the algorithms with the restart mechanism always perform better than those without restarts. When comparing EPR_{1R} (i.e., EPR) and EPR_{2R} which differ only in the pool updating strategy, we observe that EPR_{1R} outperforms EPR_{2R} by attaining more best known lower bounds and better average results. Based on these findings, we conclude that our adopted pool updating strategy and restart mechanism contribute to ensuring the high performance of EPR.

6. Conclusion

The quadratic multiple knapsack problem is a useful model in practical applications that represents an imposing computational challenge. We propose a highly effective algorithm for the QMKP based on the general evolutionary path relinking (EPR) framework. To ensure the efficacy of the proposed EPR algorithm, we address four important issues: the construction method for creating the initial reference set of elite solutions, the path relinking method for generating intermediate solutions from an initial solution to a guiding solution(s), the threshold search for local optimization and the pool updating strategy for maintaining the reference set. Given the highly constrained feature of the QMKP, we also devised a method to explore infeasible regions during the path relinking process which proves to be a critical strategy for the performance of the algorithm.

Comprehensive experimental studies on two sets of 90 benchmarks show that our EPR algorithm competes very favorably with the

state-of-the-art algorithms. On the first set of 60 well-known QMKP benchmark instances, EPR is capable of matching or improving the best known lower bound for all 60 cases. Notably, the proposed algorithm discovers an improved best solution (new lower bound) for 10 challenging benchmarks. On the set of 30 new large-sized instances, EPR achieves the best lower bounds as well as average results over multiple runs that are much better than those of the state-of-the-art reference methods.

There are several elements that account for the high performance of our proposed EPR algorithm. First is its mechanism to explore infeasible solutions. As shown in Sections 5.1 and 5.2, this mechanism allows the search to transition between structurally different feasible solutions and helps the search to locate high quality solutions. Second is manner in which we jointly use of path relinking and local optimization provides the algorithm with a high-level trade-off between diversification and intensification. As revealed by the computational results of Section 4, the path relinking procedure generates diversified (and possibly infeasible) solutions while the local optimization procedure with responsive threshold search ensures an intensified examination of some selected path solutions. Finally, the fitness-based pool updating strategy used in the proposed algorithm maintains the elitism of the reference set while ensuring a healthy diversity, as is shown in Section 5.3.

For future work, we can consider another form of path relinking called Exterior Path Relinking as recently elaborated in [38]. Exterior Path Relinking offers the possibility of including characteristics that are not present in the guiding solution during the relinking process. It would also be interesting to adapt the path relinking approach to other constrained knapsack problems (e.g., the multidimensional knapsack problem [8], the quadratic knapsack problem with multiple constraints [39]) and other quadratic optimization problems (e.g., the quadratic assignment problem [10]).

Acknowledgments

We are grateful to our reviewers for their insightful comments which helped us to improve the paper. We would like to thank Dr. García-Martínez and his co-authors for making the codes of [17,18] available to us. The work is partially supported by the LigeRo project (2009–2013) from the Region of Pays de la Loire (France) and the PGMO (2014-0024H) project from the Jacques Hadamard Mathematical Foundation. Support for Yuning Chen from the China Scholarship Council is also acknowledged.

References

[1] A. Hiley, B. Julstrom, The quadratic multiple knapsack problem and three heuristic approaches to it, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), 2006, pp. 547–552.
[2] H. Kellerer, P. Ulrich, P. David, Knapsack Problems, Springer Science & Business Media, 2004.
[3] C. Kaleli, An entropy-based neighbor selection approach for collaborative filtering, Knowl.-Based Syst. 56 (2014) 273–280.
[4] D. Pisinger, An exact algorithm for large multiple knapsack problems, Eur. J. Oper. Res. 114 (3) (1999) 528–541.
[5] D. Pisinger, The quadratic knapsack problema survey, Discrete Appl. Math. 155 (5) (2007) 623–648.
[6] D. Pisinger, A.B. Rasmussen, R. Sandvik, Solution of large quadratic knapsack problems through aggressive reduction, INFORMS J. Comput. 19 (2) (2007) 280–290.
[7] V. Vasquez, J.K. Hao, A hybrid approach for the multidimensional 0–1 knapsack problem, Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01), Morgan Kaufmann Publishers, Seattle, Washington, USA, 2001, pp. 328–333.
[8] L. Wang, X.-L. Zheng, S.-Y. Wang, A novel binary fruit fly optimization algorithm for solving the multidimensional knapsack problem, Knowl.-Based Syst. 48 (2013) 17–23.
[9] A. Rong, J.R. Figueira, K. Klamroth, Dynamic programming based algorithms for the discounted 01 knapsack problem, Appl. Math. Comput. 218 (2012) 6921–6933.
[10] A. Misevicius, An improved hybrid genetic algorithm: new results for the quadratic assignment problem, Knowl.-Based Syst. 17 (2) (2004) 65–73.
[11] Y. Wang, Z. Lu, F. Glover, J.K. Hao, Path relinking for unconstrained binary quadratic programming, Eur. J. Oper. Res. 223 (3) (2012) 595–604.
[12] P.T. Boggs, J.W. Tolle, Sequential quadratic programming for large-scale nonlinear optimization, J. Comput. Appl. Math. 124 (1–2) (2000) 123–137.
[13] S.H.R. Pasandideh, S.T.A. Niaki, A. Gharaei, Optimization of a multiproduct economic production quantity problem with stochastic constraints using sequential quadratic programming, Knowl.-Based Syst. 84 (2015) 98–107.
[14] T. Saraç, A. Sipahioglu, A genetic algorithm for the quadratic multiple knapsack problem, in: BVAI, in: LNCS, vol. 4729, 2007, pp. 490–498.
[15] A. Singh, A. Baghel, A new grouping genetic algorithm for the quadratic multiple knapsack problem, in: EvoCOP, in: LNCS, vol. 4446, 2007, pp. 97–109.
[16] S.M. Soak, S.W. Lee, A memetic algorithm for the quadratic multiple container packing problem, Appl. Intel. 36 (1) (2012) 119–135.
[17] C. García-Martínez, F.J. Rodríguez-Díaz, M. Lozano, A tabu-enhanced iterated greedy algorithm: a case study in the quadratic multiple knapsack problem, Eur. J. Oper. Res. 232 (3) (2014b) 454–463.
[18] C. García-Martínez, F. Glover, F.J. Rodríguez-Díaz, M.R. Lozano M, Strategic oscillation for the quadratic multiple knapsack problem, Comput. Optim. Appl. 58 (1) (2014a) 161–185.
[19] Y. Chen, J.K. Hao, Iterated responsive threshold search for the quadratic multiple knapsack problem, Ann. Oper. Res. 226 (1) (2015) 101–131.
[20] S. Sundar, A. Singh, A swarm intelligence approach to the quadratic multiple knapsack problem, in: ICONIP, in: LNCS, vol. 6443, 2010, pp. 626–633.
[21] E. Vallada, R. Rubén, Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem, Omega 38 (1) (2010) 57–67.
[22] F. Glover, M. Laguna, R. Martí, Scatter search and path relinking: foundations and advanced designs, in: G.C. Onwubolu, B.V. Babu (Eds.), New Optimization Technologies in Engineering, Studies in Fuzziness and Soft Computing, 141, 2004, pp. 87–100.
[23] Y. Marinakis, M. Magdalene, A particle swarm optimization algorithm with path relinking for the location routing problem, J. Math. Model. Algorithms 7 (1) (2008) 59–78.
[24] A. Rahimi-Vahed, T.G. Crainic, M. Gendreau, W. Rei, A path relinking algorithm for a multi-depot periodic vehicle routing problem, J. Heuristics 19 (3) (2013) 497–524.
[25] M. Resende, R. Martí, M. Gallego, A. Duarte, GRASP and path relinking for the max-min diversity problem, Comput. Oper. Res. 37 (3) (2010) 498–508.
[26] G.Q. Zhang, K.K. Lai, Combining path relinking and genetic algorithms for the multiple-level warehouse layout problem, Eur. J. Oper. Res. 169 (2) (2006) 413–425.
[27] M. Yagiura, I. Toshihide, F. Glover, A path relinking approach with ejection chains for the generalized assignment problem, Eur. J. Oper. Res. 169 (2) (2006) 548–569.
[28] R. Carrasco, P.T. Anthanh, M. Gallego, F. Gortázar, A. Duarte, R. Martí, Tabu search for the max-mean dispersion problem, Knowl.-Based Syst. 85 (2015) 256–264.
[29] F. Glover, J.K. Hao, The case for strategic oscillation, Ann. Oper. Res. 183 (1) (2011) 163–173.
[30] F. Glover, M. Laguna, Tabu Search, Kluwer, 1999.
[31] M. Gendreau, J. Potvin (Eds.), Handbook of Metaheuristics, International Series in Operations Research & Management Science, Springer, 2010.
[32] M. Resende, C. Ribeiro, Greedy randomized adaptive search procedures, Handb. Metaheuristics 57 (2003) 219–249.
[33] F. Glover, Tabu search-part I, ORSA J. Comput. 1 (3) (1989) 190–206.
[34] F. Glover, Tabu search-part II, ORSA J. Comput. 2 (1) (1990) 4–32.
[35] D. Gusfield, Partition-Distance: A problem and class of perfect graphs arising in clustering, Inf. Process. Lett. 82 (3) (2002) 159–164.
[36] H.W. Kuhn, The hungarian method for the assignment problem, Naval Res. Logistics Q. 2 (1955) 83–97.
[37] A. Billionnet, E. Soutif, An exact method for the 0–1 quadratic knapsack problem based on lagrangian decomposition, Eur. J. Oper. Res. 157 (3) (2004) 565–575.
[38] F. Glover, Exterior path relinking for zero-one optimization, Int. J. Appl. Metaheuristic Comput. 5 (3) (2014) 1–8.
[39] H. Wang, G. Kochenberger, F. Glover, A computational study on the quadratic knapsack problem with multiple constraints, Comput. Oper. Res. 39 (1) (2012) 3–11.