Memetic search for the generalized quadratic multiple knapsack problem

Yuning Chen and Jin-Kao Hao

***Abstract*—The generalized quadratic multiple knapsack problem (GQMKP) extends the classical quadratic multiple knapsack problem (QMKP) with setups and knapsack preference of the items. The GQMKP can accommodate a number of real-life applications and is computationally difficult. In this paper, we demonstrate the interest of the memetic search approach for approximating the GQMKP by presenting a highly effective memetic algorithm (denoted by MAGQMK). The algorithm combines a backbone-based crossover operator (to generate offspring solutions) and a multi-neighborhood simulated annealing procedure (to find high quality local optima). To prevent premature convergence of the search, MAGQMK employs a quality-and-distance pool updating strategy. Extensive experiments on two sets of 96 benchmarks show a remarkable performance of the proposed approach. In particular, it discovers improved best solutions in 53 and matches the best known solutions for 39 other cases. A case study on a pseudo real-life problem demonstrates the efficacy of the proposed approach in practical situations. Additional analyses show the important contribution of the novel general-exchange neighborhood, the backbone-based crossover operator as well as the quality-and-distance pool updating rule to the performance of the proposed algorithm.**

***Index Terms*—Constrained quadratic optimization, Knapsack problem, Population based search, Heuristics.**

I. Introduction

THE generalized quadratic multiple knapsack problem (GQMKP), as an extension of the classical quadratic multiple knapsack problem with setups and knapsack preferences of the items, is a difficult combinatorial optimization problem recently introduced in [30].

We are given a set of n objects (also called items) $N = \{1, 2, ..., n\}$ which are classified into r disjoint classes $C = \{C_1, C_2, ..., C_r\}$ where $C_i \cap C_j = \emptyset$ for each i, j, $1 \leq i \neq j \leq r$, and a set of m knapsacks $M = \{1, 2, ..., m\}$. Each knapsack k ($k \in M$) has a capacity B_k. Let $R = \{1, 2, ..., r\}$ be the class index set, $n_i = |C_i|$ be the number of objects of class $C_i \in C$ ($\sum_{i=1}^{r} n_i = n$), σ_i be the set of knapsacks to which the items of C_i can be allocated ($\forall (i, j), 1 \leq i \neq j \leq r$, σ_i and σ_j can be overlapped), β_i ($1 \leq \beta_i \leq m$) be the maximum number of knapsacks for which the items of C_i can be selected. $C_{ij} \in N$ ($i \in R, j \in \{1, 2, ..., n_i\}$) denotes the index of the j^{th} object of the i^{th} class. Each class C_i is associated with a setup resource consumption s_i which is generated when any item of C_i is assigned to knapsack k ($k \in M$, only one s_i is needed when more than one object of C_i are assigned to knapsack k). Each object i ($i \in N$) has a weight w_i, and a knapsack-dependent profit p_{ik} with respect to knapsack k ($k \in M$) which indicates its knapsack preference. Each pair of objects i and j ($1 \leq i \neq j \leq n$) generates a profit q_{ij} which contributes to the optimization objective when these two objects are allocated to the same knapsack. Additionally, let x_{ik} be the decision variable such that $x_{ik} = 1$ if object i is allocated to knapsack k, $x_{ik} = 0$ otherwise; let y_{uk} be the decision variable such that $y_{uk} = 1$ if at least one object of class u is allocated to knapsack k, $y_{uk} = 0$ otherwise. Then the GQMKP can be formulated as a 0-1 quadratic program:

$$\text{Max} \quad \sum_{u=1}^{r} \sum_{i=1}^{n_u} \sum_{k=1}^{m} x_{C_{ui}k} p_{C_{ui}k} + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=1}^{m} x_{ik} x_{jk} q_{ij} \tag{1}$$

subject to:

$$\sum_{u=1}^{r} (\sum_{i=1}^{n_u} x_{C_{ui}k} w_{C_{ui}k} + y_{uk} s_u) \leq B_k, \qquad \forall k \in M \tag{2}$$

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

$$y_{uk} = 0, \qquad \forall u \in R, k \notin \sigma_u \tag{4}$$

$$\sum_{k=1}^{r} y_{uk} \leq \beta_u, \qquad \forall u \in R \tag{5}$$

$$x_{C_{ui}k} \leq y_{uk}, \quad \forall u \in R, i \in \{1, ..., n_u\}, k \in M \tag{6}$$

$$y_{uk} \leq \sum_{i=1}^{n_u} x_{C_{ui}k}, \quad \forall u \in R, k \in M \tag{7}$$

$$x_{ik}, y_{uk} \in \{0, 1\}, \quad \forall i \in N, u \in R, k \in M \tag{8}$$

For each knapsack k, constraint 2 requires that the weight sum of the objects in the knapsack plus the sum of the setup resource consumption must be no larger than its capacity. Constraint 3 ensures that each object can be allocated to at most one knapsack. Constraint 4 indicates that items cannot be assigned to the knapsacks where they are not allowed. Constraint 5 requires that for each class, the number of knapsacks to which its items are allocated must be less than

The work was partially supported by the LigeRo project (2009-2014) 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 (2012-2016) was also acknowledged.

Y. Chen and J.K. Hao (corresponding author) are with the Department of Computer Science, LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers 01, France; J.K. Hao is also affiliated with the Institut Universitaire de France (e-mail: yuning@info.univ-angers.fr; hao@info.univ-angers.fr).

Note: This paper has a supplementary document which is downloadable at: http://www.info.univ-angers.fr/pub/hao/gqmkp.html

or equal to its maximum number. Constraint 6 guarantees that y_{uk} receives the value of 1 when at least one object of class u is allocated to knapsack k. Constraint 7 requires that y_{uk} receives the value of 0 when no object of class u is allocated to knapsack k. Constraint 8 requires that each variable takes the value of 0 or 1.

Notice that the above model is modified from the binary quadratic model described in [30] and has the advantage of being more concise.

The GQMKP is NP-hard in the strong sense, since it generalizes the well-known NP-hard quadratic knapsack problem (QKP) [26]. Indeed, the GQMKP degenerates to the QKP when the number of knapsacks equals 1, the number of classes equals the number of objects (i.e., $r = n$) and the setup of each class equals 0. The GQMKP is also a generalization of the popular quadratic multiple knapsack problem (QMKP) [3], [12]–[14], which is more complicated than the QKP due to the presence of multiple knapsacks. Compared to the classical QMKP, the GQMKP has the following four distinguished features:

- Class condition. Objects are grouped to different classes. Those in the same class share common features.
- Setup requirement. A knapsack may contain objects from different classes. Including more classes in a knapsack requires additional resource consumption, represented by a capacity loss when switching from one class to another class.
- Assignment limitation. Objects from a specific class are allowed to be assigned to a subset of knapsacks, rather than all the knapsacks.
- Knapsack preference. Each object has a knapsack preference which differentiates the return when it is allocated to different knapsacks.

As shown in the case study of [30], the GQMKP can be encountered in companies producing plastic parts which make use of injection machines. In the studied application, jobs are considered as objects; injection machines are regarded as knapsacks and the given planning period is considered as the capacity of the knapsack. Jobs are first classified to different classes according to the mold they require. Switching molds in a machine requires setup time. Molds are fixed to a limited number of machines due to technical reasons. A job is preferred to be assigned to certain machines since its associated mold may not work at the same efficiency at all machines or the decision maker may have preferences to have this job assigned to certain machines. Conducting a job in a machine takes a certain time which is considered as the weight of the job. Apart from the profit returned by performing a single job, additional profits can be attained by assembling similar jobs (i.e., jobs from the same class) to the same machine for the sake of reducing setup times which reflects the quadratic nature of this problem. The purpose of this problem is to determine the jobs to be performed and to decide the assignment of jobs to the machines while satisfying all the underlying constraints. The GQMKP may find other applications in areas like manufacturing systems, computer design and communications where a set of tasks needs to be assigned to a number of machines or processors.

In [30], the authors showed the first study dealing directly with the GQMKP and proposed three solution approaches to this difficult problem. Based on the 0-1 quadratic model proposed in their paper, the first approach is based on the general Gams/Dicopt framework for solving MINP (mixed-integer nonlinear programming) models by integrating a NLP (Nonlinear Programming) or MIP (Mixed-Integer Programming) solver that runs under the Gams system. The second approach is a genetic algorithm (GA). The GA randomly initializes a population and uses a 2-tournament method for selection. It employs a specialized uniform-based operator for crossover and two dedicated operators based on local exchange and greedy construction for mutation. The GA generates solutions satisfying all the constraints except for constraint 5 which is allowed to be violated. To evaluate infeasible solutions, the GA uses a penalized fitness function which is defined as the original objective function (Eq. 1) plus the weighted sum of the excess number of knapsacks of each class. The last approach is a hybrid algorithm combining F-MSG (modified subgradient algorithm operating on feasible values) and the GA. In this approach, constraint 5 is included in the Lagrangian function to construct the sub-problem of the F-MSG algorithm which is then solved by the GA.

This paper introduces a highly effective population-based memetic algorithm MAGQMK for the GQMKP. The main contributions of this work can be summarized as follows.

- From the algorithmic perspective, the proposed approach integrates several original ingredients. First, we devise a backbone-based crossover operator for generating offspring solutions. This operator is specifically designed for the GQMKP and aims to maximally preserve common contributive objects that are shared by parent solutions. Second, to explore efficiently the search space around an offspring solution generated by the crossover operator, we propose a multi-neighbourhood simulated annealing procedure which relies on three specific and complementary neighborhoods. Among these three neighborhoods, the novel general-exchange neighborhood which is first proposed in this work, potentially helps the algorithm to explore the solution space more efficiently with its large-step move size. Finally, we apply a quality-and-distance pool updating strategy to maintain a healthy diversity of the population.
- From the computational perspective, the proposed MAGQMK approach shows a very competitive performance on two sets of 96 benchmark instances. In particular, for the set of 48 small-sized instances, MAGQMK discovers 6 improved best lower bounds and attains the best known result for other 39 instances. More importantly, for the set of 48 large-sized benchmarks, MAGQMK is able to improve all previous best lower bounds with only one exception. Finally, a successful application of the MAGQMK to a pseudo real-life problem demonstrates its effectiveness in handling large-scale practical cases.

The remainder of the paper is organized as follows. Section

II describes in detail the proposed approach. Section III reports the computational results of MAGQMK on the benchmark instances and shows comparisons with respect to the state-of-the-art approaches. Section IV is dedicated to an extensive investigation of the key ingredients of the proposed approach to gain a deep understanding of their impacts on the performance of the algorithm. Conclusions are drawn in Section V.

II. A MEMETIC ALGORITHM FOR THE GQMKP

Memetic search is a powerful framework that promotes the idea of combining evolutionary computing and local optimization [21]. By offering the possibility for the search process to effectively explore the space of local optima, memetic algorithms have proved to be quite effective in solving a number of difficult combinatorial optimization problems [4], [8], [15], [20], [22], [23], [32] and in particular knapsack and quadratic problems [5], [16], [29], [31].

Algorithm 1 shows the working scheme of the proposed memetic algorithm MAGQMK for the GQMKP. The algorithm generates an initial population of solutions which are first ameliorated by the multi-neighborhood simulated annealing procedure described in Section II-C. Then it repeats an evolution process to improve the population until a predefined stop condition (typically a fixed number of generations) is verified. At each generation, the algorithm randomly selects two parent solutions from the population and recombines them to generate an offspring solution using a backbone-based crossover operator (see Section II-D). This newly generated solution is further improved by the multi-neighborhood simulated annealing procedure. Finally a quality-and-distance based rule (see Section II-E) is applied to decide if the improved offspring solution can be inserted into the population. We present the components of the proposed MAGQMK in the following subsections.

Algorithm 1 Pseudo-code of MAGQMK for the GQMKP

Input: P: a $GQMKP$ instance; p: population size; $maxTry$: max number of trials to generate a nonclone solution;
Output: the best solution S^* found;
{ Population initialization, Section II-B }
$POP \leftarrow Pool\_Initlialization(p, maxTry)$;
$p \leftarrow |POP|$; /*p is reset to the number of distinct individuals eventually obtained.*/
$S^* \leftarrow Best(POP)$; /* S^* records the best solution found so far. */
{ Main search procedure }
while stopping condition not reached **do**
  Randomly select two solutions S_i and S_j from POP
  $S_c \leftarrow Crossover(S_i, S_j)$ /* Apply the backbone-based crossover operator to generate an offspring solution S_c, see Section II-D */
  $S_c \leftarrow$ MNSA(S_c) /* Improve S_c with the multi-neighbourhood simulated annealing procedure, see Section II-C */
  if $f(S_c) > f(S^*)$ **then**
    $S^* \leftarrow S_c$
  end if
  $POP \leftarrow Pool\_Updating(S_c, POP)$ /* Update POP with a quality-and-distance based rule, see Section II-E */
end while

A. Search space, solution representation and evaluation function

For a given GQMKP instance, the search space visited by our MAGQMK algorithm is composed of all possible allocations of objects to knapsacks such that the constraints of Eq. 2–8 are satisfied. In other words, MAGQMK visits only feasible solutions.

To encode a feasible solution, 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. Such a solution representation can be also 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.

The quality of any candidate solution S is evaluated directly by the objective function f of the GQMKP. Given a solution $S = \{I_0, I_1, ..., I_m\}$, the objective value $f(S)$ is calculated by the following formula:

$$f(S) = \sum_{k \in M} (\sum_{i \in I_k} p_{ik} + \sum_{i,j \in I_k, i \neq j} q_{ij}) \quad (9)$$

This function defines a total order over the solution space. Given two solution S^1 and S^2, S^2 is better than S^1 if $f(S^2) > f(S^1)$.

B. Population Initialization

The MAGQMK algorithm uses a randomized greedy construction method (RGCM) to create the initial solutions (individuals) of its population. RGCM follows the spirit of the GRASP approach [28] which is able to generate a different solution for each run thanks to its randomized property. RGCM relies on the notions of *contribution* and *object density* whose definitions are given below. Notice that our *object density* extends the classical bang-for-buck ratio of ordinary linear knapsack problems and aims to identify the attractiveness of items in the GQMKP setting.

- **Contribution:** 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:

$$VC(S, i, k) = p_{ik} + \sum_{j \in I_k, j \neq i} q_{ij} \quad (10)$$

- **Density:** Let $ci(i)$ denote the class index of object i. The $density$ of object i $(i \in N)$ with respect to knapsack k $(k \in M)$ is defined as its contribution $VC(S, i, k)$ divided by its weight w_i if another object of its class $C_{ci(i)}$ is already included in knapsack k, or its contribution divided by the sum of its weight and the setup of its class $s_{ci(i)}$ if object i is the first one of its class to be included in knapsack k. Formally, we have:

$$D(S, i, k) = \begin{cases} VC(S, i, k)/w_i, \\ \qquad if\ \exists j \neq i \in C_{ci(i)} \text{ such that } j \in I_k \\ VC(S, i, k)/(w_i + s_{ci(i)}), \qquad \text{otherwise} \end{cases} \quad (11)$$

Starting from an empty solution S and from the first knapsack (i.e., $k = 1$), RGCM iteratively and randomly selects an unallocated object i ($i \in I_0$) from a restricted candidate list $RCL(S, k)$ and assigns it to knapsack k. Let $R(S, k)$ denote the set of unselected objects that can fit into the knapsack k. To build $RCL(S, k)$, we first sort all the objects in $R(S, k)$ in descending order of their density values (calculated by Equation 11), and then we put the first $min\{rcl, |R(S, k)|\}$ (rcl is a parameter) objects into $RCL(S, k)$. Each time $RCL(S, k)$ becomes empty for the current knapsack, the next knapsack is examined. The construction process is repeated until the last knapsack (i.e., $k = m$) is examined.

The solution constructed by RGCM is further improved by the simulated annealing procedure (see Section II-C). The ameliorated solution is either inserted into the population if it is not a clone of any other individual in the population, or discarded if it appears already in the population. The population initialization procedure is iterated until the population is filled with p (population size) distinct individuals or no eligible individual has been generated for $maxTry$ consecutive times. After the initialization procedure, p is reset to the number of distinct individuals eventually obtained.

The time complexity of constructing one solution is bounded by $O(n^2 \log n)$. The whole population initialization procedure takes $O(p * maxTry * n^2 \log n)$ time in the worst case. But for most of the instances we used in this paper, it is easy to obtain p distinct solutions which makes the time complexity of initialization procedure close to $O(p*n^2 \log n)$.

C. The multi-neighborhood simulated annealing procedure

The importance of local search within a memetic algorithm has been recognized for a long time (see e.g., [1], [15], [21], [24], [25]). Our local optimization procedure is based on the popular Simulated Annealing (SA) method [18] (see Algorithm 2). Being the first time applied to the GQMKP problem, our multi-neighborhood SA procedure (denoted by MNSA) is characterized by its combined use of three different and complementary neighborhoods (N_R, N_{SW}, N_{GE}). MNSA basically performs nl cycles of temperature cooling and for each temperature, MNSA runs nsl iterations while at each iteration, it successively explores its three neighborhoods in a sequential way. The acceptance of each sampled solution $S^{'}$ is subject to a probability test $Pr\{S \rightarrow S^{'}\}$ which is given by:

$$Pr\{S \rightarrow S^{'}\} = \begin{cases} 1, & \text{if} \quad f(S^{'}) > f(S) \\ e^{(f(S^{'})-f(S))/T}, & \text{if} \quad f(S^{'}) \leq f(S) \end{cases} \tag{12}$$

where T is a temperature value. Precisely, a neighboring solution $S^{'}$ is accepted to replace the incumbent solution S if a randomly generated value rd ($rd \in [0, 1]$) is less than or equal to $Pr\{S \rightarrow S^{'}\}$ (i.e., $rd \leq Pr\{S \rightarrow S^{'}\}$). In what follows, we introduce the three dedicated neighborhoods.

1) Two traditional neighborhoods: There are two traditional small-sized neighborhoods (denoted by N_R and N_{SW}) which are defined by two basic move operators: REALLOCATE (*REAL* for short) and *SWAP*. These two neighborhoods were first introduced to address the related QMKP [3], [12], [13] and we adapt them to the GQMKP considered in this paper.

For an object i, let $k_i \in \{0, 1, ..., m\}$ be the knapsack to which the object is allocated, let $S \oplus OP$ denote the neighboring solution generated by applying move operator 'OP' to S. Our *REAL* and *SWAP* move operators are described as follows:

- $REAL(i, k)$: This move operator displaces an object i from its current knapsack $k_i \in \{0, 1, ..., m\}$ to another knapsack k ($k \neq k_i, k \neq 0$). All neighboring solutions of a given object i_0 induced by this move operator are given by:

  $N_R(S, i_0) = \{S' : S' = S \oplus REAL(i_0, k), k \in M \setminus \{k_{i_0}\}\}$

- $SWAP(i, j)$: This move operator swaps 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. All neighboring solutions of a given object i_0 induced by the *SWAP* operator are given by:

  $N_{SW}(S, i_0) = \{S' : S' = S \oplus SWAP(i_0, j), j \neq i_0 \in N, k_{i_0} \neq k_j \in M \cup \{0\}\}$

2) New general-exchange neighborhood for the GQMKP: The above two neighborhoods are small-sized in the sense that they include those neighboring solutions that require at most two changes of the incumbent solution. However, we observed

Algorithm 2 Pseudo-code of the MNSA procedure

```
Input: P: a GQMKP instance; S_0: initial solution; T_0: initial temperature; cr: cooling ratio; nl: number of loops; nsl: number of sub-loops
Output: the best solution S*
S ← S_0
S* ← S_0 /* S* records the best solution found so far */
T ← T_0 /* Initialize temperature */
for i = 1 to nl do
    T ← T * cr /* Temperature cooling */
    for j = 1 to nsl do
        for each N ∈ {N_R, N_SW} do
            for k = 1 to n do
                Search the neighborhood N(S,k) for feasible moves and accept moves to replace S according to the probability function (Eq. 12); Update the best solution S*
            end for
        end for
        for k = 1 to m do
            for each class c in knapsack k do
                Search the neighborhood N_GE(S,c,k) for feasible moves and accept moves to replace S according to the probability function (Eq. 12); Update the best solution S*
            end for
        end for
    end for
end for
```

that MNSA with merely these small neighborhoods does not perform well for the highly constrained GQMKP. To reinforce the search ability of MNSA, we introduce a novel large-sized neighborhood N_{GE} based on the General-Exchange operator ($GENEXC$ for short), which is specifically designed for the GQMKP. Unlike $REAL$ and $SWAP$, $GENEXC$ may modify two or more values of the current solution to generate a neighboring solution.

- $GENEXC(k_i, c_i, k_j, c_j)$: This move operator basically exchanges objects of class c_i from knapsack k_i with objects of class c_j ($c_j \neq c_i$) from knapsack k_j ($k_j \neq k_i$). However, simply exchanging those objects may possibly result in infeasible solutions. To always maintain feasibility, our *GENEXC* operator employs a remove-construct method to achieve a feasible "exchange" functionality. Precisely, $GENEXC(k_i, c_i, k_j, c_j)$ removes all the objects of C_{c_i} that are currently allocated to knapsack k_i, and removes all the objects of C_{c_j} ($c_j \neq c_i$) that are currently allocated to knapsack k_j ($k_j \neq k_i$). Then it greedily fills knapsack k_i with unallocated objects of class C_{c_j} which have the highest density values according to the definition of Section II-B; similarly it greedily fills knapsack k_j with unallocated objects of class C_{c_i} having the highest density values. It checks all the constraints before inserting an unallocated item into the target knapsack. If the insertion induces infeasibility, it skips this item and tests the next item with the highest density value. This *GENEXC* operator allows a transition between structurally different feasible solutions which cannot be reached by the *SWAP* and *REAL* operators in certain cases. To illustrate this, consider a simple GQMKP instance shown in Figure 1. The important data of the instance is displayed on the left and two feasible solutions are displayed on the right of the figure. In this example, a transition from *Solution1* to *Solution2* is by no means possible by applying the *SWAP* or *REAL* operator, which however, can be easily achieved by *GENEXC* with a single $GENEXC(k_i = 1, c_i = 3, k_j = 2, c_j = 1)$ move, i.e., exchange the objects of class 3 in knapsack 1 with the objects of class 1 in knapsack 2. For some cases where a transition from one solution to another requires numerous combinations of *SWAP* and *REAL* operators, *GENEXC* can realize the transition much more easily, making the neighborhood search more focused and straightforward. This benefit fundamentally comes from the fact that exchanging two classes of items from two different knapsacks never violates Constraint 5 (a class of items can be assigned only to some knapsacks) which is a hard constraint of the GQMKP.

  Then the neighboring solutions of a given knapsack k_{i_0} and a given class c_{i_0} induced by the *GENEXC* operator are given by:
  $N_{GE}(S, k_{i_0}, c_{i_0}) = \{S' : S' = S \oplus GENEXC(k_{i_0}, c_{i_0}, k_j, c_j), k_j \neq k_{i_0} \in M, c_j \neq c_{i_0} \in R\}$

In Section IV-B, we will investigate the effectiveness of these three neighborhoods and show the particular role of the N_{GE} neighborhood.

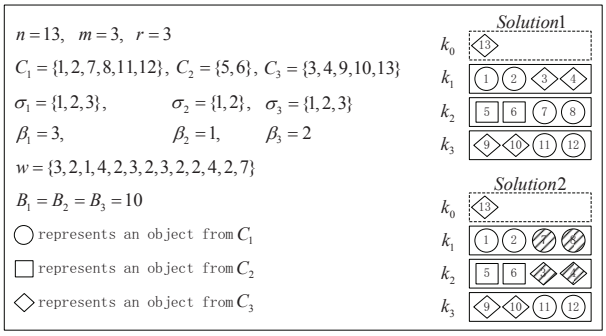


Fig. 1: An example for *GENEXC* neighborhood.

D. Backbone-based crossover operator

Apart from the local optimization procedure, crossover constitutes another key component of our MAGQMK algorithm. A successful crossover operator is usually expected to incorporate domain specific heuristics and should be able to transmit meaningful features (building blocks) from parents to offspring. To devise a specialized crossover operator with strong "semantics" for our problem, we consider the GQMKP as a constrained grouping problem [6], [7]. As described in Section II-A, a solution of the GQMKP can be viewed as a partition of n objects into $m + 1$ groups. For grouping problems, it is more natural and straightforward to manipulate groups of objects rather than individual objects. Such an idea of designing crossover operators has been proved to be very successful in solving a number of grouping problems such as graph coloring [9], [19], [27], bin packing [7] and graph partitioning [2], [11]. In our case, we must additionally take into account the various constraints of our problem.

Preliminary experiments show that high quality local optima share many grouping objects. It is thus expected that objects that are always grouped together are very likely to be part of a global optimum or a high quality solution. Following this observation, the general idea of our proposed crossover operator is to preserve the contributive identical object groupings (backbone) of maximal size from parent solutions to offspring, and to choose the groupings for the rest objects with equal probability from two parents.

Definition 1: Given two parent solutions $S^1 = \{I_0^1, I_1^1, ..., I_m^1\}$ and $S^2 = \{I_0^2, I_1^2, ..., I_m^2\}$, let H denote the set of common objects that has identical object grouping of S^1 and S^2, i.e., $H = \cup_{k=0}^{m}(I_k^1 \cap I_k^2)$. The backbone H^b of S^1 and S^2 is the subset of H such that each object of H^b is a contributive object to both parent solutions, i.e., $H^b = H \backslash (I_0^1 \cap I_0^2)$.

This backbone definition excludes the non-contributive objects (i.e., unallocated objects) of the parent solutions since they are typically small-density and are unlikely to be part of optimal solutions. Based on the notion of backbone, we consider, for each parent solution, m groups of allocated objects taken from all $m+1$ groups for crossover. The proposed backbone-based crossover procedure consists of three main

Algorithm 3 Pseudo-code of the backbone-based crossover procedure

1: **Input**: Two parent solutions $S^1 = \{I_0^1, I_1^1, ..., I_m^1\}$ and $S^2 = \{I_0^2, I_1^2, ..., I_m^2\}$
2: **Output**: An offspring solution $S^0 = \{I_0^0, I_1^0, ..., I_m^0\}$
/* **Step 1:** Group matching */
3: Let $E = \{(g_i^1, g_j^2) | i \in M, j \in M\}$ denote the set of all $m \times m$ group (knapsack) combinations of S^1 and S^2. Compute the number of common objects $w_{g_i^1 g_j^2}$ for each group combination $(g_i^1, g_j^2) \in E$.
4: $J \leftarrow \emptyset$
5: **repeat**
6: Choose the combination (g_i^1, g_j^2) with the largest $w_{g_i^1 g_j^2}$ from E
7: $J \leftarrow J \cup \{(g_i^1, g_j^2)\}$
8: Remove from E all combinations associated with g_i^1 and g_j^2
9: **until** $E = \emptyset$
/* Adjust the group numbering */
10: $rd \leftarrow random\{0, 1\}$
11: **for** each $(g_i^1, g_j^2) \in J$ **do**
12: **if** rd=0 **then**
13: Assign label g_i^1 to group g_j^2 of S^2
14: **else**
15: Assign label g_j^2 to group g_i^1 of S^1
16: **end if**
17: **end for**
/* **Step 2:** Create a partial solution based on backbone */
18: **for** $i := 1$ to m **do**
19: $I_i^0 = \emptyset$
20: $H_i^b \leftarrow I_i^1 \cap I_i^2$; $NH_i^1 \leftarrow I_i^1 \backslash H_i^b$; $NH_i^2 \leftarrow I_i^2 \backslash H_i^b$
21: $I_i^0 \leftarrow I_i^0 \cup H_i^b$
22: $R^1(S^0, i) \subseteq NH_i^1$ denotes a subset of objects of NH_i^1 that can fit into I_i^0, $R^2(S^0, i) \subseteq NH_i^2$ denotes a subset of objects of NH_i^2 that can fit into I_i^0
23: $l \leftarrow 1$
24: **while** $R^1(S^0, i) \neq \emptyset$ or $R^2(S^0, i) \neq \emptyset$ **do**
25: if l *is odd* or $R^2(S^0, i) \neq \emptyset$, then $A \leftarrow 1$, else $A \leftarrow 2$
26: Choose an object o with the highest density from $R^A(S^0, i)$
27: $I_i^0 \leftarrow I_i^0 \cup \{o\}$; $NH_i^A \leftarrow NH_i^A \backslash \{o\}$
28: Update $R^1(S^0, i)$ and $R^2(S^0, i)$
29: $l \leftarrow l + 1$
30: **end while**
31: **end for**
/* **Step 3:** Complete the partial solution */
32: Groups are sorted in random order in SG, $R(S^0, i)$ denotes a subset of unselected objects that can fit into I_i^0;
33: **for** each $i \in SG$ **do**
34: **while** $R(S^0, i) \neq \emptyset$ **do**
35: Choose an object o with the highest density from $R(S^0, i)$
36: $I_i^0 \leftarrow I_i^0 \cup \{o\}$
37: Update $R(S^0, i)$
38: **end while**
39: **end for**

steps, as illustrated in Algorithm 3. The details of these three steps are described as follows.

- **Group matching.** In the context of the GQMKP, two different groups from two parent solutions which share the most common objects typically have different group numbers. For example, group one of the first solution might correspond to group two of the second solution, with possibly most objects in common. Therefore, the first step of the crossover is to properly identify a perfect matching of the groups in order to find out the largest number of common objects of the two parent solutions. This amounts to find a maximum weight matching in a complete bipartite graph $G = (V, E)$ where V consists of m left vertices and m right vertices which correspond respectively to the groups of the first and second solutions; each edge $(g_i^1, g_j^2) \in E$ is associated with a weight $w_{g_i^1 g_j^2}$, which is defined as the number of identical objects in group g_i^1 of solution 1 and group g_j^2 of solution 2. The maximum weight matching problem can be solved by using the classical Hungarian algorithm [17]. However, calling this algorithm for each crossover application would be too computationally expensive ($O(n + m^3)$) in our case. Instead, we apply a fast greedy algorithm to seek a near-optimal weight matching. Our greedy algorithm iteratively chooses an edge $(g_i^1, g_j^2) \in E$ with the largest $w_{g_i^1 g_j^2}$, and then deletes from E all edges incident to vertex g_i^1 and to vertex g_j^2. This procedure is repeated until E becomes empty (lines 3-9 of Algorithm 3).
Steps 1 ends by adjusting the group numbering. We randomly select one solution and adjust its group numbering according to the matched groups of the other solution (lines 10-17 of Algorithm 3).
- **Create a partial solution based on backbone H^b.** Let I_i^1 and I_i^2 denote the set of objects of the i^{th} group in solution 1 and solution 2 respectively; H_i^b denotes the index set of the common objects (backbone) of the i^{th} group, i.e., $H_i^b = I_i^1 \cap I_i^2$; NH_i^1 and NH_i^2 denote the index set of the unshared objects of the i^{th} group of solution 1 and solution 2, i.e., $NH_i^1 = I_i^1 \backslash H_i$, $NH_i^2 = I_i^2 \backslash H_i$. Let I_i^0 denote the set of objects of the i^{th} group in offspring solution ($I_i^0 = \emptyset$ at the beginning). For each pair of matched groups, we first conserve the backbone (i.e., all common objects) to the corresponding group of the offspring solution, i.e., $I_i^0 \leftarrow I_i^0 \cup H_i^b$. Let $R^1(S^0, i) \subseteq NH_i^1$ denote a subset of objects of NH_i^1 that can be allocated to I_i^0 while satisfying all constraints (Eq. 2-8), $R^2(S^0, i) \subseteq NH_i^2$ denotes a subset of objects of NH_i^2 that can be allocated to I_i^0 while satisfying all constraints (Eq. 2-8). We then alternatively choose an object with the highest density from $R^1(S^0, i)$ on odd steps and from $R^2(S^0, i)$ on even steps. Once an object is selected and included into I_i^0, it is removed from the respective set (NH_i^1 or NH_i^2); $R^1(S^0, i)$ and $R^2(S^0, i)$ are updated accordingly. This procedure continues until both $R^1(S^0, i)$ and $R^2(S^0, i)$ become empty. We apply this procedure to all m groups, and we finally obtain a partial offspring solution with some groups (knapsacks) possibly far from being fully filled (lines 18-31 of Algorithm 3).
- **Complete the partial solution.** In order to complete the partial offspring solution, we allocate some of the unassigned objects to the groups (knapsacks) based on a greedy construction strategy. Specifically, we put the m groups (knapsacks) in a set SG in random order. Let $R(S^0, i)$ denote the set of unselected objects that can be

allocated to I_i^0 while satisfying all constraints (Eq. 2-8). For each group $i \in SG$, we continue choosing an object with the highest density from $R(S^0, i)$ until $R(S^0, i)$ becomes empty. This procedure is terminated when all knapsacks are examined (lines 32-38 of Algorithm 3).

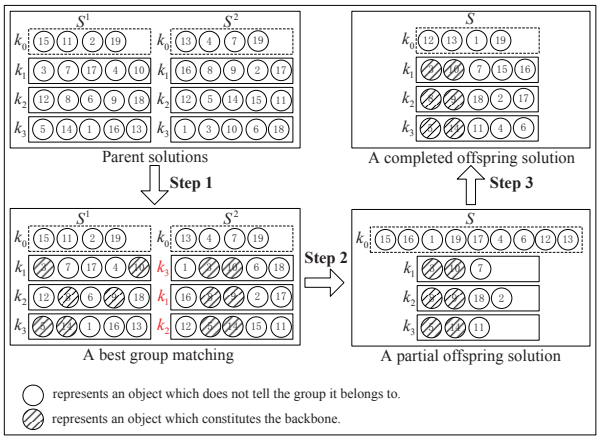


Fig. 2: Illustration of the backbone-based crossover step.

To illustrate the main steps of the backbone-based crossover operator, we use the case of Figure 2 as a working example. The example involves an instance of 19 items and 3 knapsacks, and operates with two parent solutions S^1 and S^2. In the first step, the knapsacks of S^1 and S^2 are matched using the fast greedy algorithm and the results are: (S^1-k_1 matches to S^2-k_3), (S^1-k_2 matches to S^2-k_1) and (S^1-k_3 matches to S^2-k_2). In the second step, a partial offspring solution is constructed by first conserving the backbone objects (striped objects in Figure 2), and then alternatively adding objects with the highest density values from the parent solutions (item 7 from S^1-k_1, 18 from S^1-k_2, 2 from S^2-k_1 and 11 from S^2-k_2 are selected in Figure 2). In the third step, the partial solution is completed by allocating some unassigned items (those in virtual knapsack k_0 of the offspring solution) to other knapsacks (items 15 and 16 are allocated to k_1; items 17 is allocated to k_2; items 4 and 6 are allocated to k_3).

Given that calculating the edge weight and finding the maximum weight matching take respectively $O(n)$ and $O(m^2)$ time, our proposed greedy algorithm for group matching can be realized in time $O(n+m^2)$ in step 1. In step 2 and step 3, finding the highest density object takes $O(n)$ time in the worst case and this can repeat at most n times. So our backbone-based crossover procedure can be realized in $O(m^2+n^2)$ in the worst case.

E. Pool updating strategy

Avoiding premature convergence is another key issue for population-based algorithms. To address this issue, we propose a quality-and-distance pool updating strategy to decide whether S^0, the newly generated offspring, should be inserted into the population or be discarded. This quality-and-distance strategy, as its name implies, considers both the solution quality and the distance between individuals in the population to ensure the population diversity.

Algorithm 4 Quality-and-distance pool updating procedure

1: **Require:** A population $POP = \{S^1, S^2, ..., S^p\}$ and an offspring solution S^0
2: **Ensure:** Updated population POP
3: Insert S^0 into the population: $POP \leftarrow POP \cup \{S^0\}$
4: **for** each $S^i \in POP$ **do**
5: Calculate $AvgDist_{i,POP}$ according to Eq. 14
6: Calculate $QDF(S^i)$ according to Eq. 15
7: **end for**
8: Identify the worst solution S^w: $S^w = argmax\{QDF(S^i)|S^i \in POP)\}$
9: **if** $S^0 \neq S^w$ **then**
10: Remove the worst solution from the population: $POP \leftarrow POP \backslash \{S^w\}$
11: **else**
12: Remove S^0 from the population: $POP \leftarrow POP \backslash \{S^0\}$
13: **end if**

Since the GQMKP can be viewed as a grouping problem, the well-known set-theoretic partition distance [10] seems quite appropriate. Given two solutions S^1 and S^2, the distance between them $Dist(S^1, S^2)$ can be computed as:

$$Dist(S^1, S^2) = n - Sim(S^1, S^2) \tag{13}$$

where the similarity $Sim(S^1, S^2)$ is a complementary measure of the distance representing the maximum number of elements of S^1 that do not need to be displaced to obtain the solution S^2. We apply the group matching algorithm used in Section II-D to identify the similarity $Sim(S^1, S^2)$. Given a population $POP = \{S^1, S^2, ..., S^p\}$ and a distance matrix $Dist$ with $Dist_{ij}$ denoting the distance between individuals S^i and S^j ($i \neq j$ and $i, j \in \{1, ..., p\}$), the average distance between S^i and any other individual in POP is given by:

$$AvgDist_{i,POP} = (\sum_{S^j \in POP, j \neq i} Dist_{ij})/p \tag{14}$$

The general scheme of our quality-and-distance pool updating procedure is described in Algorithm 4. The offspring solution S^0 is first inserted into the population. Then all the solutions in the population are evaluated using the following quality-and-distance fitness (QDF for short) function:

$$QDF(S^i) = \alpha * OR(f(S^i)) + (1-\alpha) * DR(AvgDist_{i,POP}) \tag{15}$$

where $OR(f(S^i))$ and $DR(AvgDist_{i,POP})$ represent respectively the rank of solution S^i with respect to its objective value and the average distance to the population, and α is a parameter empirically set to $\alpha = 0.6$. With this parameter setting, Eq. 15 ensures that the best individual in terms of objective value will never be removed from the population, which formalizes the elitism property of our pool-updating strategy. The worst solution S^w corresponds to the individual with the largest QDF value. If S^0 is different from S^w, we replace S^w with S^0, otherwise we discard S^0.

The distance matrix $Dist$ is a memory structure that we maintain throughout the search. Each time a new solution S^0 is inserted into the population, the distance between S^0 and any other solution in POP needs to be calculated, which takes $O(p * (n + m^2))$ time (given that the greedy group

matching procedure takes $O(n+m^2)$ time, see Section II-D). The other distances are extracted directly from $Dist$. The resting operations of the pool updating procedure, including calculating $AvgDist_{i,POP}$, $QDF(S^i)$ and identifying the worst solution, take $O(p^2+2*p)$ time. $Dist$ is updated if S^0 is accepted and S^w is removed which takes $O(p)$ time. According to the above analysis, our pool updating procedure can be realized in $O(p*(n+m^2+p))$ time.

III. Computational experiments

This section is dedicated to experimental assessment of the proposed MAGQMK algorithm. For this purpose, we show computational results on 96 benchmark instances available in the literature and make comparisons with the current best performing algorithms. To complete this section, we also show a case study on a pseudo real-life problem concerning plastic parts production with injection machines.

A. Experimental settings

Test instances. The 96 GQMKP benchmark instances belong to two different sets and are available at: http://endustri.ogu.edu.tr/Personel/Akademik_personel/Tugba_Sarac_Test_Instances/G-QMKP-instances.rar:

- **Set I:** This set is composed of 48 small-sized instances which are characterized by their number of objects $n = 30$, number of knapsacks $m \in \{1,3\}$, number of classes $r \in \{3,15\}$, density $d \in \{0.25,1.00\}$. Optimal solutions are unknown for these instances.
- **Set II:** The second set includes 48 large-sized instances with the number of objects $n = 300$, number of knapsacks $m \in \{10,30\}$, number of classes $r \in \{30,150\}$, density $d \in \{0.25,1.00\}$. Optimal solutions are unknown for these instances.

The above two sets of instances were first generated in [30] where the authors considered 12 factors that can affect the problem structure. They examined each factor with two levels which, if a full factorial experiment is conducted, leads to 4096 different types of problems. To sample a subset of problems from such a large number, they selected 32 types of test problems where all factors are double leveled. With this design, they generated two sets of 96 benchmarks that we use in this paper.

Parameters. To report computational results of our MAGQMK algorithm, we adopt the parameter values shown in Table I, which are fixed in the following manner. For each of the four parameters (p, nl, nsl, cr), we tested several potential values while fixing the other parameters to our specified default values, and then chose the value yielding the best performance. An analysis in Section IV-A explains why the value of 200 is chosen for the length of the MNSA chain (nl) which is a critical parameter of MAGQMK. For the initial temperature (T_0), we use a formula (see Table I) to determine its value. The formula is identified as follows. Preliminary experiments suggest a quadratic relation between T_0 and the objective value of the initial solution (denoted as f_0), in the form of $T_0 = a*(f_0+b)^2/c+d$. We then selected four representative instances whose initial objective values (f_0) range from small to large and we tuned manually a best T_0 for each instance. With four pairs of (T_0,f_0) values, we obtain the formula by solving four simultaneous equations. In summary, the parameter values of Table I are used in all the experiments reported in this section even if fine-tuning the parameters could lead to better results.

TABLE I: Parameter settings of the MAGQMK algorithm

Parameters	Description	Value	Section
p	population size	10	II-B
$maxTry$	max number of trials to generate a nonclone solution	20	II-B
nl	number of loops	200	II-C
nsl	number of sub-loops	2	II-C
T_0	initial temperature	$\frac{5.0*(f_0+5000)^2}{100000000}+2.0$	II-C
cr	cooling ratio	0.99	II-C

Our MAGQMK algorithm was coded in C++[1] and compiled by GNU gcc 4.1.2 with the '-O3' option. The experiments were conducted on a computer with an AMD Opteron 4184 processor (2.8GHz and 2GB RAM) running Ubuntu 12.04. When solving the DIMACS machine benchmarks[2] without compilation optimization flag, the run time on our machine is 0.40, 2.50 and 9.55 seconds respectively for graphs r300.5, r400.5 and r500.5.

B. Computational results of the MAGQMK algorithm

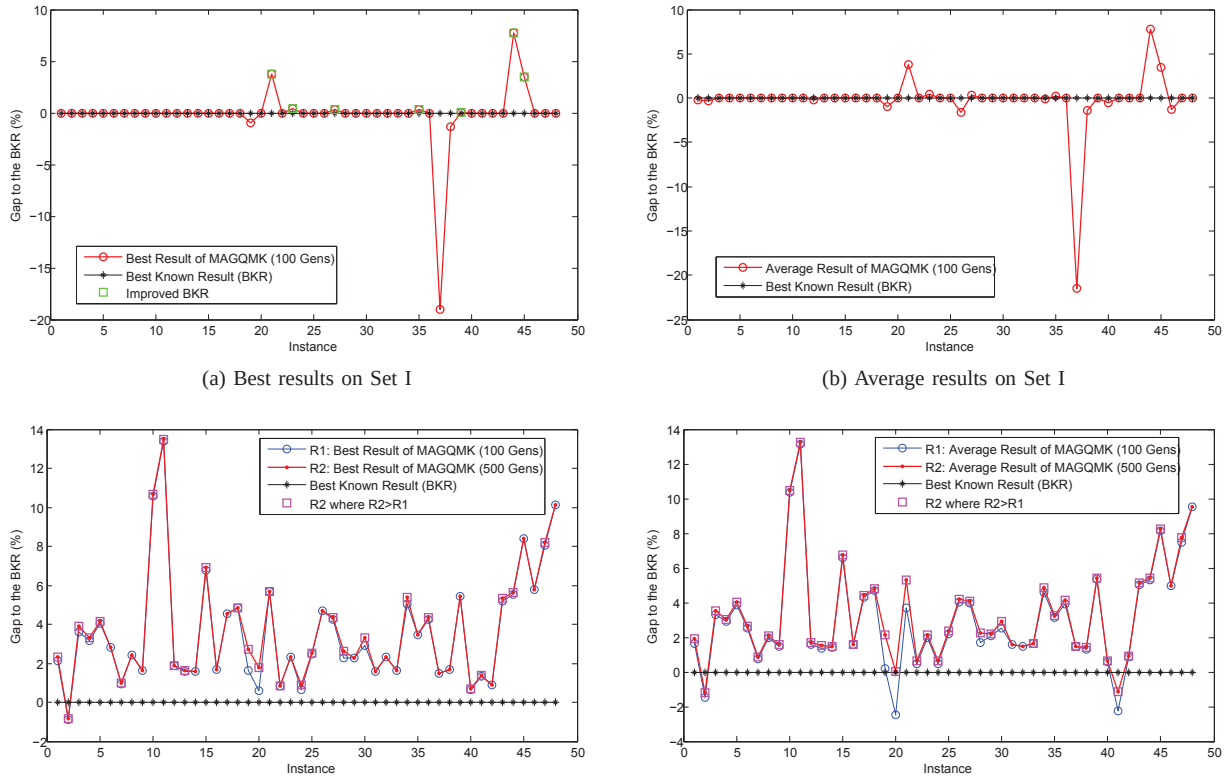
In this section, we show computational results obtained by our MAGQMK algorithm on the two sets of 96 benchmark instances under two different stopping criteria: a short time limit of 100 generations and a long time limit of 500 generations. We use the second stopping criterion to investigate the long-run behavior of our MAGQMK algorithm on the set of 48 large-sized instances. For each stopping criterion and for each instance, our algorithm was executed 30 times independently.

The computational results obtained by our MAGQMK algorithm on the instances of Set I and Set II are displayed respectively in Table A1 and Table A2 in the supplementary document (available at http://www.info.univ-angers.fr/pub/hao/gqmkp.html). In these two tables, column 1 to 4 give the characteristics of each instance, including the instance identity number ($No.$), number of knapsacks (k), number of classes (r) and density (d). Column 5 shows the best known results[3] (f_{bk}) which are compiled from the best results of the three best performing algorithms reported in [30]. Our results are displayed in the remaining columns including the overall best objective value over 30 runs (f_{best}), the average value of the 30 best objective values (f_{avg}), the standard deviation of the 30 best objective values (sd), the earliest CPU time in seconds over the 30 runs when the f_{best} value is first reached (t_{best}) and the average value of the 30 CPU time when the best solution is first encountered in each run (t_{avg}).

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

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

[3]When we examine the solution certificates reported in [30] which are provided at http://endustri.ogu.edu.tr/Personel/Akademik_personel/Tugba_Sarac_Test_Instances/G-QMKP-instances.rar, we find that the reported results for Instance 10-1 (15256.01) and Instance 10-2 (13058.94) correspond to infeasible solutions and consequently they will not be used as reference.



(a) Best results on Set I

(b) Average results on Set I

(c) Best results on Set II

(d) Average results on Set II

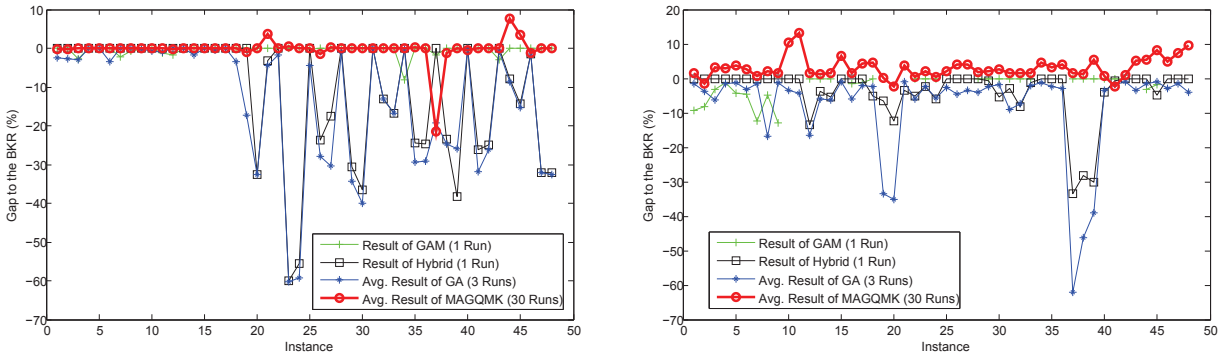
Fig. 3: Computational results of the MAGQMK on two sets of benchmark instances

To give a general picture of MAGQMK's performance with respect to the best known results, we also provide four plots of our best and average results for both sets of instances in Figure 3. The horizontal axis of these plots shows the instance serial number that indicates the order in which the instances appear in Table A1 and Table A2. The vertical axis shows the gap of our results (either best or average) to the best known results in percentage, calculated as $(f - f_{bk})\times 100/f_{bk}$ where f is the best or average solution value. A gap larger than zero means MAGQMK obtains a new best known result for the corresponding instance.

For the 48 small-sized instances of Set I, Table A1 and Figure 3(a) show that our MAGQMK algorithm is able to attain the best known results for 45 out of 48 (93.75%) cases including 7 improved best lower bounds (see values starred in Table A1 and the green square points in Figure 3(a)). For 37 out of 48 instances, even our average results are better than or equal to the previous best known lower bounds (see Figure 3(b)). Moreover, MAGQMK has a perfect standard deviation (sd=0.00) for 38 out of 48 instances, which means that one run is sufficient for MAGQMK to attain its best solution for these instances. Finally, MAGQMK is computationally very efficient and reaches the best solution within a very short computing time (typically less than 1 second). Indeed, across all instances of Set I, the average value of the best solution time (t_{best}) is 0.07 seconds and the average value of the average solution time is 0.35 seconds.

For the 48 large-sized instances of Set II, Table A2 and Figure 3(c) disclose that, within a short time limit (100 generations), our MAGQMK algorithm performs remarkably by improving all the previous best known results except one case (Instance 1-2). Moreover, even its average results reach or improve the previous best lower bounds for 45 out of 48 cases (93.75%, see Figure 3(d)). The average standard deviation is 43.37 which is small with respect to the best objective values that are more than 10 thousands for many cases. Finally, we observe that these results are obtained within a reasonable computing time: the average value of the best solution time is 601.57 seconds and the average value of the average computing time is 693.02 seconds.

When a long time limit is available, our MAGQMK algorithm is able to discover even better results. In particular, MAGQMK (500 Gens) discovers 28 improved best lower bounds (see the magenta square points in Figure 3(c)) with respect to the results of MAGQMK (100 Gens). The average results of MAGQMK (500 Gens) are more interesting which are better than those of MAGQMK (100 Gens) for 44 out of 48 cases (see the magenta square points in Figure 3(d)). Moreover, MAGQMK (500 Gens) decreases the average value of standard deviation (sd) from 43.37 (of MAGQMK (100 Gens)) to 36.02. The average value of the best solution time is 2673.27 seconds and the average value of the average computing time is 3025.82 seconds, which remain reasonable.



(a) Comparative results on Set I

(b) Comparative results on Set II

Fig. 4: Comparative results of the MAGQMK with three state-of-the-art algorithms.

C. Comparisons with state-of-the-art algorithms

To further assess the performance of our proposed MAGQMK, we carry out a comparative study between our algorithm with 3 best performing algorithms proposed in [30] that achieve state-of-the-art results:

- The Gams/Dicopt solver [30]. The Gams/Dicopt solver is a program used for solving MINP (mixed-integer nonlinear programming) problems by integrating a NLP (Nonlinear Programming) or MIP (Mixed-Integer Programming) solver that runs under the Gams system. The reported results of Gams/Dicopt were obtained by executing one run of the algorithm within a time limit of 13000 seconds for each instance.
- A genetic algorithm [30] (GA). For each instance, the GA was run 3 times where each run was imposed a time limit of 13000 seconds. The reported results include the best lower bound, the average value of the 3 best objective values of 3 runs and the average value of the 3 best solution time (in CPU seconds) of 3 runs when the best solution is first encountered.
- A hybrid algorithm combining F-MSG algorithm and GA [30]. The reported results of the hybrid algorithm were obtained by executing one run of the algorithm within a time limit of 13000 seconds for each instance.

The evaluation of the above 3 reference algorithms were performed on a PC with an Intel Core i7 processor (2.8 GHz and 8 GB RAM). This machine is faster than our computer with a factor of 1.16 according to the Standard Performance Evaluation Corporation (www.spec.org).

One notices that the reported results of the three reference algorithms were obtained by executing one run or three runs of the algorithm while our reported results were attained by running MAGQMK 30 times. Notice also that MAGQMK belongs to the family of stochastic algorithms for which executing multiple runs is a common practice in the literature. Given the above remarks, to compare our best results over 30 runs to the reference best results of no larger than 3 runs may be considered unfair. To make the comparison as fair as possible, we focus on the average results (instead of the best results) obtained by our MAGQMK algorithm for this comparative study.

The comparative results of MAGQMK with the 3 reference algorithms on the 48 small-sized instances of Set I and the 48 large-sized instances of Set II are summarized in Table A3 and Table A4 respectively in the supplementary document (available at http://www.info.univ-angers.fr/pub/hao/gqmkp.html). In these two tables, we list the identity number of the instance in the first column (column $INST$). For Gams/Dicopt and the Hybrid algorithm which were executed only once, we indicate the obtained best objective value (f) and the best CPU time (in seconds) when the best solution is first encountered. For GA and MAGQMK which were executed more than one run, we give the average value of the best objective values (f_{avg}) and the average value of the best CPU times (t_{avg}). We also provide the overall best objective value (f_{best}) obtained by GA and MAGQMK for reference purposes. The entry in bold in each row indicates the best objective value for each instance reached by all the algorithms. The underlined value denotes the reference f_{best} value of GA or MAGQMK that matches or improves on the bold value. In the last two rows, we indicate for each algorithm, the number of values highlighted (either in bold or underlined, row $\#Bests$) and the average value of computing time (row $Avg.$). Finally, in reference to the results of the GA algorithm which was run 3 times, the last three columns of Tables A3 and A4 provide the results of 3 runs of our MAGQMK algorithm. These results are displayed for the understanding of MAGQMK's behavior when less runs are applied. From these data, we observe that the average results of MAGQMK vary slightly when the number of runs changes. This finding justifies our practice of using average results for comparative study.

The comparative results are also illustrated using plots in Figure 4, which gives the gaps of our average results to the best known results, in comparison with GA's average results, and the results of the GAM and Hybrid algorithms [30].

From Table A3 and Figure 4(a), we observe that MAGQMK attains 37 best results (values in bold in Table A3) on the 48 small-sized instances of Set I, which is the largest among the results of the 4 algorithms in comparison. Among these 37 best results, 7 correspond to the unique best results which means they were never discovered by the 3 previous reference algorithms. As for the CPU time, MAGQMK requires on

TABLE II: Computational results of the MAGQMK on the pseudo real-life instance

GA				MAGQMK			
n_f/n_i	f_{best}	f_{avg}	t_{avg}	No.Gens	f_{best}	f_{avg}	t_{avg}
10/10	15518.60	15315.40	820.6	20 Gens	16012.30	15879.10	901.02
50/30	15660.70	15578.80	3136.44	60 Gens	16013.30	15897.00	1688.20
1200/150	15884.3	15669.1	9133.48	100 Gens	16018.80	15908.50	2611.21

average 0.35 seconds across Set I which is much shorter than the time required by the Hybrid algorithm (2.48 seconds), and is very competitive to the other two reference algorithms. Moreover, our MAGQMK algorithm displays a very stable behavior that is not observed for GA which was also executed multiple runs. Indeed, MAGQMK achieves a success rate of 100% (i.e., $f_{avg} = f_{best}$) for 39 out of 48 cases (81.25%) while GA has only 8 such cases (16.67%). When we apply the statistical Wilcoxon test with a significance factor of 0.05 for pairwise comparisons, the resulting p-values of 1.97E-4 for MAGQMK v.s. Hybrid, and 1.072E-7 for MAGQMK v.s. GA show a clear dominance of MAGQMK over Hybrid and GA. Though the associated p-value of 0.4613 for MAGQMK v.s. Gams/Dicopt does not disclose a significant difference between these two methods, the superiority of the positive sum rank (205) over the negative sum rank (146) demonstrates that MAGQMK competes very well with Gams/Dicopt.

An even more favorable comparison is observed in Table A4 and Figure 4(b) for the 48 large-sized instance of Set II. Specifically, our MAGQMK algorithm is able to attain a unique best result for 46 out of 48 cases (95.83%) while the Gams/Dicopt solver and the Hybrid algorithm share the remaining two cases. Moreover, MAGQMK consumes on average 693.02 seconds in terms of computing time which is much less than that consumed by the 3 reference algorithms (693.02 v.s. 5184.91 of Gams/Dicopt, 3612.71 of Hybrid and 2598.33 of GA). Our MAGQMK algorithm easily dominates the GA. Indeed, GA obtains 0 best result while MAGQMK attains 46. Additionally, the overall *best* objective value (f_{best}) of GA is worse than our *average* result (f_{avg}) for 46 out of 48 cases. To estimate the validity of our conclusion, we apply the Wilcoxon test with a significance factor of 0.05 to compare MAGQMK with the Gams/Dicopt, Hybrid and GA algorithms respectively. The associated p-values of 7.816E-13, 6.253E-13 and 1.421E-14 confirm a clear dominance of MAGQMK over the three reference algorithms on the instances of Set II.

D. Application of MAGQMK to a pseudo real-life problem

As indicated in Section I, one real-life application of the GQMKP is the plastic parts production with injection machines in a plastic production company [30]. In this section, we study the performance of our MAGQMK algorithm on a pseudo real-life GQMKP instance (denoted by pseudo-RLGQMKP). This pseudo-RLGQMKP instance simulates closely the real-life problem involving plastic parts production used in [30] in terms of both problem size and characteristics[4]. The data file of the pseudo-RLGQMKP instance as well as its detailed characteristics are available at http://www.info.univ-angers.fr/pub/hao/gqmkp.html.

The pseudo real-life problem is very large-sized with 500 jobs (objects) and 40 machines (knapsacks). Its resulting mathematical model has 12840 constraints and 44000 binary decision variables. The pairwise profit parameter q_{ij} alone involves 124750 values. Such a large-sized and non-linear instance poses a real challenge for any existing GQMKP algorithm. As stated in [30], the Gams/Dicopt solver even failed to find a feasible solution for a problem with such difficulty when a time limit of 13000 seconds was given. To test the ability of our MAGQMK algorithm, we ran it 30 times on the pseudo-RLGQMKP instance under the time limits of 100 generations, 60 generations and 20 generations respectively. The first stopping condition (100 generations) is a standard one that was used in previous experiments, while the last two stopping conditions (60 and 20 generations) are used specifically in this experiment to meet the potential efficiency requirement in real situation. The computational results of MAGQMK on the pseudo-RLGQMKP instance are displayed in Table II. For the purpose of comparison, we also report the results obtained by the GA algorithm described in [30][5] for 30 times under three different stopping conditions that were used in [30]. The stopping condition of the GA [30] relies on two criteria: n_f (a fixed number of generations) and n_i (the number of generations without improvement). From Table II, we observe that the best performance of our MAGQMK algorithm in terms of both best result and average result is achieved under the longest time condition (100 generations). However, the decrease of the solution quality is marginal when the time limit is reduced. Indeed, when using the shortest time limit (20 generations), the best and average solution quality decrease by only 0.04% and 0.18% respectively, but with a saving of 28 minutes, compared to the results of using 100 generations. Now if we compare the results of our MAGQMK algorithm with those of the GA algorithm, it is clear that MAGQMK easily dominates the reference algorithm. MAGQMK, even with its shortest time limit (20 generations, 901.02 seconds), outperforms the GA algorithm with its longest time budget ($n_f/n_i = 1200/150$, 9133.48 seconds) in terms of average performance. Finally, one notes that MAGQMK's results remain more stable than GA's results across the three time conditions. This experiment demonstrates that the proposed MAGQMK algorithm is able to handle effectively large real-life cases even under a reduced time condition.

[4]The data file of the real-life problem in [30] is no longer available, as confirmed by the authors. We generated the pseudo real-life instance with the instance generator and the problem features provided by the authors of [30] (Dr. T. Saraç and Dr. A. Sipahioglu). We are grateful to them for this help.

[5]Since the source code of the GA algorithm [30] is not available to us, we have implemented the algorithm strictly following the description presented in the original paper.

IV. Analysis

In this section, we perform additional empirical analyses to gain a deeper understanding of the running behavior of the proposed algorithm and the effectiveness of its underlying mechanisms. Specifically, we explore the running profile of the algorithm, the effectiveness of the three proposed neighborhoods and the efficacy of the pool updating strategy.

A. Running profile

The best (resp. average) running profile is given by the function: $i \mapsto f(i)$, where i is the number of iterations and $f(i)$ is the best (resp. average) objective value known at iteration i. The running profile is a natural way to observe the evolution of the (best or average) objective value during a search process. We mention that the number of loops (nl) of MNSA which represents the search depth of the local optimization procedure is a critical parameter which affects the performance of the proposed MAGQMK algorithm. To understand how this parameter influences the algorithm behavior, we use the running profile to investigate our MAGQMK algorithm with four different values of the parameter nl: nl =50, 100, 200, 400. We consider four representative instances selected from Set II: 3_1, 9_1, 11_2 and 19_2. These instances are of reasonable size and difficulty which are characterized by different levels of k (number of knapsacks), r (number of classes) and d (density). We mention that the observation below on these 4 instances are also valid for other tested instances. For each of the 4 representative instances and for each value of nl, we performed 30 runs of MAGQMK, each run being given 22000 loops (number of temperature cooling). Figure 5 shows the best and average running profiles of MAGQMK with the 4 different values of nl. The figure shows also the running profiles of MNSA alone for comparative purpose. MNSA is also executed 30 times with the parameter values in Table I and initial solutions generated by the RGCM procedure (Section II-B).

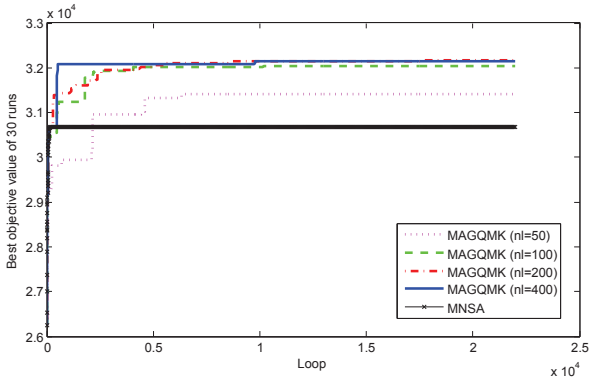
From Figure 5, we observe that MAGQMK with $nl = 200$ always attains the best performance in terms of both best and average objective value. Such a performance cannot be reached by using other values of nl for the four representative instances. With $nl = 50$, MAGQMK typically displays the worst performance. With $nl = 100$ and $nl = 400$, the quality of the best solution found by MAGQMK changes on different instances meaning that none of these two values is definitely better than the other. We observe also that with $nl = 200$ and $nl = 400$, the best objective value increases more quickly at the beginning than with the other two values. Moreover, $nl = 200$ and $nl = 400$ preserve a better population diversity than $nl = 50$ and $nl = 100$, which effectively avoids a premature convergence of the algorithm and makes the search progress steadily. The above observations on the best running profiles are also valid for the average profiles. This analysis confirms that the value of 200 is a reasonable choice for parameter nl.

Concerning the best running profile of MNSA, we observe that even if the best objective value increases sharply at the beginning, the search soon reaches a point from which the best objective value can no longer be improved. The same is also observed on the average running profile of MNSA. Even MAGQMK with the least interesting parameter $nl = 50$ can always obtain a solution that is better than the best solution obtained by MNSA. Comparing the running profiles of MAGQMK and MNSA shows a clear interest of the memetic framework and our proposed crossover operator.

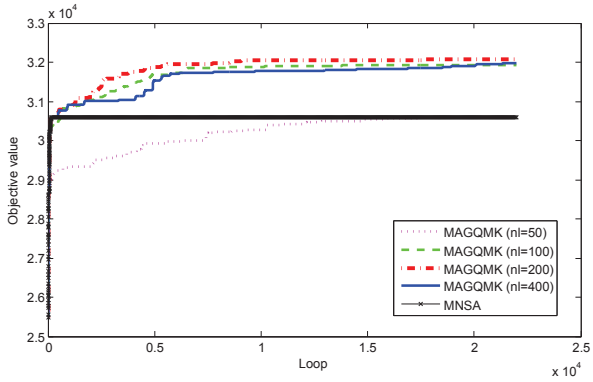
B. Effectiveness of the three neighborhoods

The neighborhood is a critical element that affects the efficacy of a local search procedure. Our proposed MAGQMK algorithm relies on three dedicated neighborhoods: N_R (the neighborhood induced by the *REAL* operator), N_{SW} (the neighborhood induced by the *SWAP* operator) and N_{GE} (the neighborhood induced by the *GENEXC* operator), which are explored in a sequential way in the MNSA procedure. In this section, we investigate the influence of each neighborhood over the performance of the proposed algorithm. For this purpose, we propose six weakened versions of MAGQMK, including three versions with a single neighborhood and three others with double neighborhoods. Except the disabled neighborhood(s), these MAGQMK variants share the same components as the standard MAGQMK algorithm. For the sake of simplicity, we denote these variants as MA_R, MA_S, MA_G, MA_RnS, MA_RnG and MA_SnG, where for example, MA_R indicates an algorithm variant with only N_R neighborhood, and MA_RnS indicates a variant with N_R and N_{SW} neighborhoods. We tested these six variants as well as the standard MAGQMK algorithm on the 48 large-sized instances of Set II. Each algorithm was run 30 times with 100 generations per run. We calculate for each instance the BestGap and the AvgGap, which are computed as $(f - f^*)\times 100/f^*$ where f is the best or average solution value and f^* is the best result found by the standard MAGQMK algorithm. The experimental results are shown in Figure 6 where the left part (a)-(b) and the right part (c)-(d) are dedicated to the one-neighborhood variants and the two-neighborhood variants respectively. The instances are displayed in the same order as they are presented in Table A4. Statistical data are summarized in Table III where for each algorithm variant and for both BestGap and AvgGap, we list the minimum and the average value over 48 gaps (row MIN and AVG). Notice that gaps are in negative values and thus a smaller value means a larger gap.

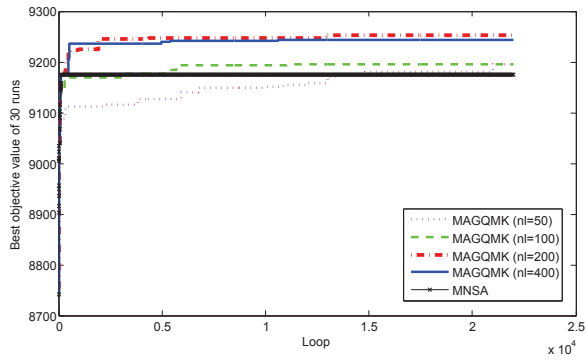
Figure 6 discloses that for each algorithm variant, the gap values (no matter BestGap or AvgGap) are typically below zero, which means all six algorithm variants perform worse than the standard MAGQMK algorithm. Moreover, the performance of the one-neighborhood variants always falls behind the two-neighborhood variants in terms of both the average and the minimum gap values (see Table III). These observations confirm that each of the three neighborhoods makes a significant contribution to the overall performance of MAGQMK. Among the three one-neighborhood algorithms, MA_G which employs the single N_{GE} neighborhood attains the best performance in terms of the average gap value, followed by MA_S and finally MA_R. In the three two-neighborhood variants, again the two algorithms with N_{GE}



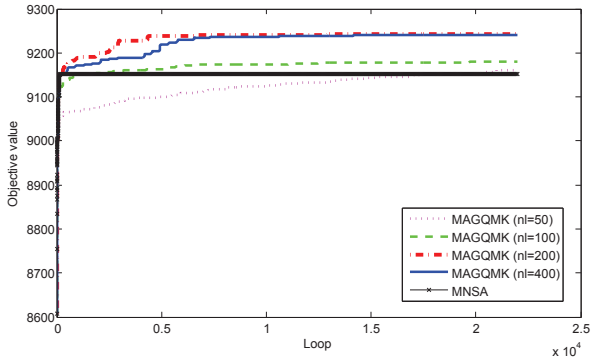
(a) 3_1 Best Profile



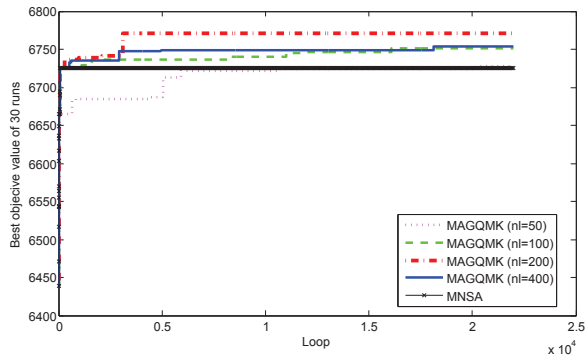
(b) 3_1 Average Profile



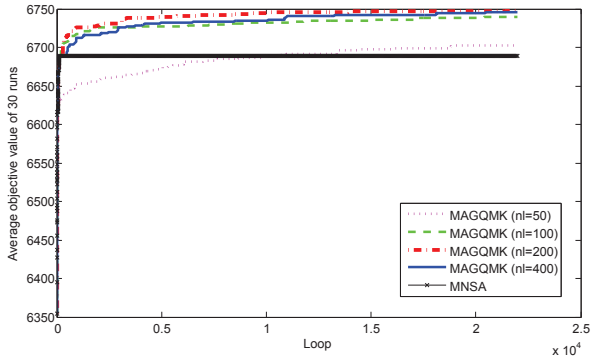
(c) 9_1 Best Profile



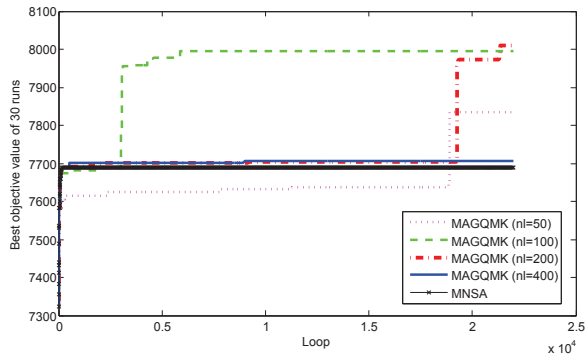
(d) 9_1 Average Profile



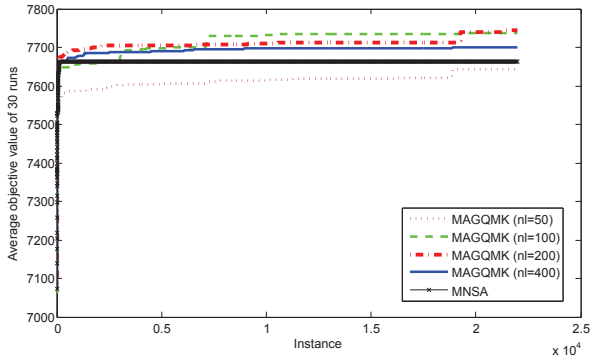
(e) 11_2 Best Profile



(f) 11_2 Average Profile



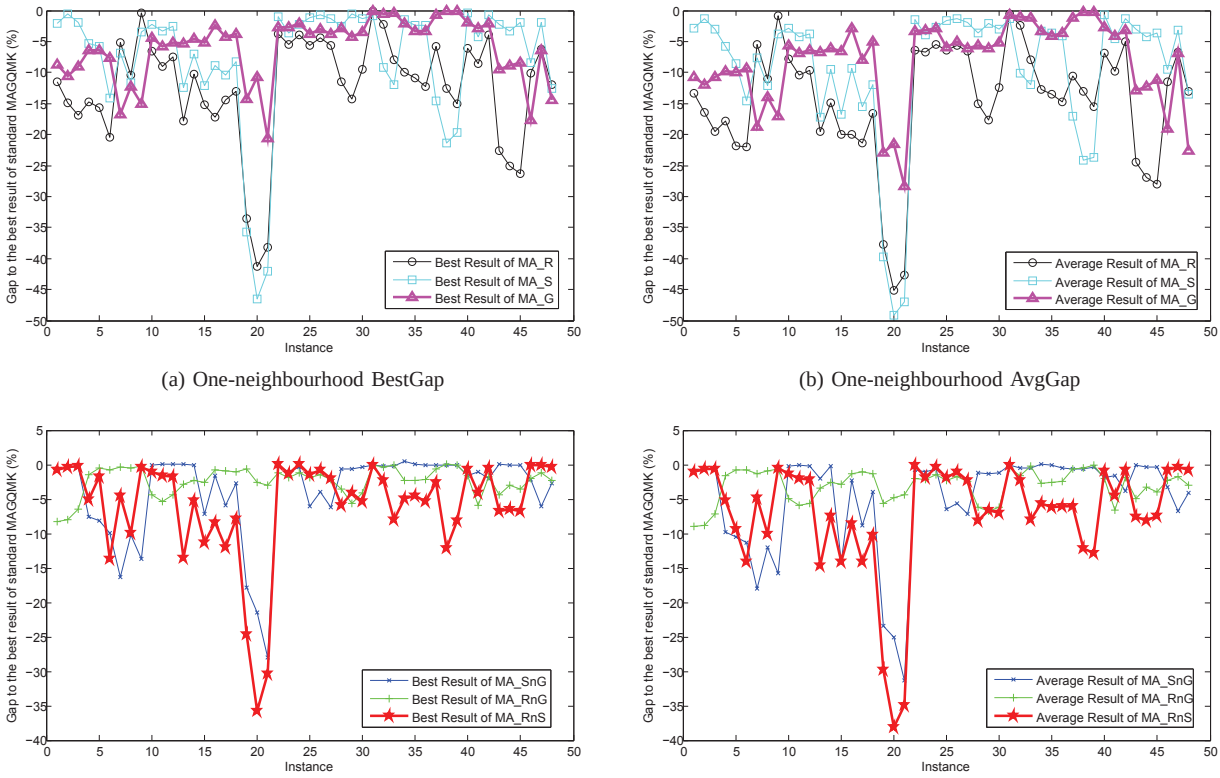
(g) 19_2 Best Profile



(h) 19_2 Average Profile

Fig. 5: Running profiles of the MAGQMK with four different nl values on four representative instances.

as one of its neighborhoods (i.e., MA_RnG and MA_SnG) show an overall better performance than the one without



(a) One-neighbourhood BestGap

(b) One-neighbourhood AvgGap

(c) Two-neighbourhood BestGap

(d) Two-neighbourhood AvgGap

Fig. 6: BestGap and AvgGap of six algorithm variants to the standard MAGQMK.

TABLE III: Statistical data of BestGap and AvgGap.

		MA_R	MA_S	MA_G	MA_RnS	MA_SnG	MA_RnG
BestGap	AVG.	-12.42	-7.67	-6.05	-5.92	-3.89	-2.36
	MIN.	-41.28	-46.49	-20.57	-35.76	-28.05	-8.29
AvgGap	AVG.	-14.57	-9.40	-8.70	-7.15	-5.05	-3.06
	MIN.	-45.19	-49.22	-28.28	-38.15	-31.38	-8.98

it (i.e., MA_RnS). These observations demonstrate that the neighborhood N_{GE}, which is specifically designed for the GQMKP, is the most critical one among the three neighborhoods employed.

C. Impact of the pool updating strategy

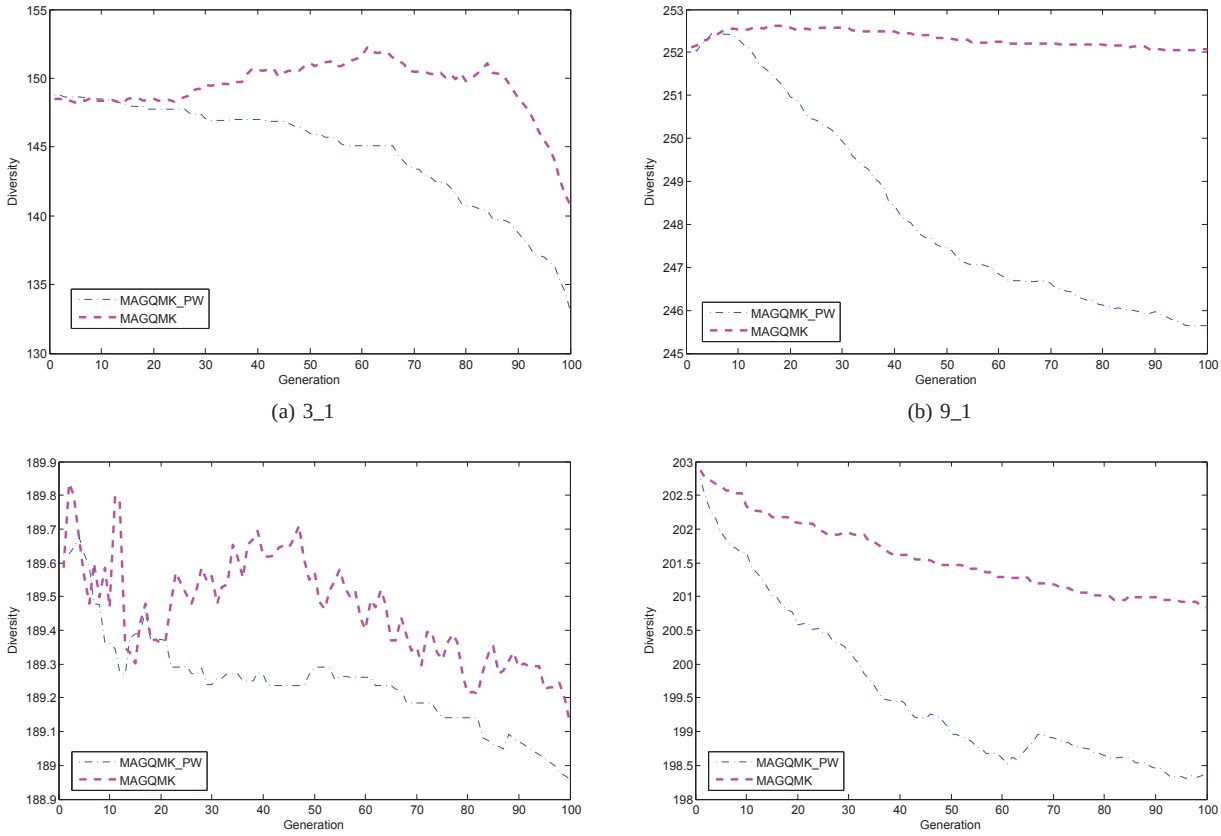
As described in Section II-E, we employ a quality-and-distance (QD) rule to update the solution pool so as to maintain a healthy diversity of the population. To evaluate the merit of this strategy, we compare it to a traditional "Pool Worst" strategy (denoted as PW) which simply replaces the worst solution in terms of objective value in the pool with a new offspring solution. We ran two algorithm variants (standard MAGQMK and MAGQMK$_{PW}$) 30 times on the 48 large-sized instances of Set II. MAGQMK and MAGQMK$_{PW}$ are the same except for the pool updating strategy. Table IV summarizes the statistical data of the computational results. For each algorithm variant, Table IV lists the average value of the best results over 48 instances (column $Avg.f_{best}$), the average value of the average results over 48 instances (column $Avg.f_{avg}$). Table IV also lists, for the best result and the average result respectively, the number of instances where MAGQMK performs better than MAGQMK$_{PW}$ (column $\# > 0$) and where MAGQMK performs equally well as MAGQMK$_{PW}$ (column $\# = 0$).

From Table IV, we observe that compared to MAGQMK$_{PW}$, MAGQMK achieves a better average value for both the best result (21882.41 vs. 21849.57) and the average result (21791.55 vs. 21781.08) obtained over the 48 instances. Moreover, MAGQMK attains a best result which is better than or equal to that of MAGQMK$_{PW}$ for 43 out of 48 cases (36 better, 7 equal), and an average result which is better than or equal to that of MAGQMK$_{PW}$ for 32 out of 48 cases (30 better, 2 equal). The above observation indicates that MAGQMK reaches a better performance than MAGQMK$_{PW}$ which confirms the usefulness of our proposed quality-and-distance pool updating strategy.

To gain some insights on the inner working of the quality-and-distance (QD) pool updating strategy, we provide in Figure 7 the evolution of the population diversity in function of the generations on the four representative instances that were used in Section IV-A. The population diversity is defined as the average distance between all individuals in the population where the distance measure is the set-theoretic partition dis-

TABLE IV: Statistical data of two pool updating strategies on the 48 large-sized instances of Set II.

MAGQMK		MAGQMK$_{PW}$		f_{best}-f_{best}^{PW}		f_{avg}-f_{avg}^{PW}	
$Avg.f_{best}$	$Avg.f_{avg}$	$Avg.f_{best}$	$Avg.f_{avg}$	$\#>0$	$\#=0$	$\#>0$	$\#=0$
21882.41	21791.55	21849.57	21781.08	36	7	30	2



(a) 3_1 (b) 9_1

(c) 11_2 (d) 19_2

Fig. 7: Diversity in function of generations on four representative instances.

tance introduced in Section II-E. We also provide the diversity evolution of the "Pool Worst" (PW) strategy for comparative purpose. From Figure 7, we can make two observations: 1) the diversity decreases regularly for both pool updating strategies; 2) the diversity is better preserved with the QD updating strategy compared to the PW strategy; indeed, the magenta line (QD updating strategy) is generally above the blue line (PW strategy) for all tested instances.

V. Conclusion

In this paper, we have presented an effective memetic algorithm (MAGQMK) for the generalized quadratic multiple knapsack problem (GQMKP). GQMKP is a useful model in practice while representing a real computational challenge. The proposed MAGQMK algorithm combines a dedicated backbone-based crossover operator for solution recombination and a multi-neighborhood simulated annealing procedure for local optimization. A quality-and-distance based pool updating strategy ensures a healthy diversity of the population.

Computational assessments on two sets of 96 benchmark instances reveal that the proposed approach is highly effective compared to the state-of-the-art methods. For the set of 48 small-sized instances, MAGQMK is able to attain 45 best known results where 7 of them correspond to improved best known results. For the set of 48 large-sized benchmarks, our algorithm performs even better by improving all previous best lower bounds except for one case. We have also compared MAGQMK with 3 best performing algorithms published very recently [30] and showed that MAGQMK dominates these reference algorithms both in solution quality and computational efficiency. An application of the MAGQMK approach to a pseudo real-life problem additionally demonstrates the efficacy of the proposed algorithm for practical cases.

Furthermore, we have compared MAGQMK and its underlying simulated annealing procedure to show the interest of the population-based memetic framework and our backbone-based crossover operator. We have illustrated the effectiveness of the proposed dedicated neighborhoods (in particular, the novel general-exchange neighborhood) and the interest of the proposed quality-and-distance pool updating strategy.

Finally, it is expected that the ideas behind the crossover operator and the neighborhoods developed in the work would be useful to other constrained knapsack problems and more

generally constrained grouping problems.

REFERENCES

[1] N. K. Bambha, S. S. Bhattacharyya, J. Teich, E. Zitzler. Systematic integration of parameterized local search into evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 8(2):137–155, 2004.

[2] U. Benlic, J.K. Hao. A multilevel memetic approach for improving graph k-partitions. *IEEE Transactions on Evolutionary Computation*, 15(5):624–472, 2011.

[3] Y. Chen, J.K. Hao. Iterated responsive threshold search for the quadratic multiple knapsack problem. *Annals of Operations Research*, 226(1): 101–131, 2015.

[4] X. Chen, Y.S. Ong, M.H. Lim, and K.C. Tan. A multi-facet survey on memetic computation. *IEEE Transaction on Evolutionary Computation*, 15(5):591–607, 2011.

[5] P.C. Chu, J.E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics* 4(1):63–86, 1998.

[6] E. Falkenauer. New representations and operators for GAs applied to grouping problems. *Evolutionary Computation*, 2: 123–144, 1992.

[7] E. Falkenauer. Genetic algorithms and grouping problems. *New York: Wiley*, 1998.

[8] L. Feng, Y.-S. Ong, M.-H. Lim, I.W.H. Tsang. Memetic search with inter-domain learning: A realization between CVRP and CARP. *IEEE Transactions on Evolutionary Computation*, 19(5): 644–658, 2015.

[9] P. Galinier, J.K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4): 379–397, 1999.

[10] D. Gusfield. Partition-Distance: A Problem and Class of Perfect Graphs Arising in Clustering. *Information Processing Letters*, 82(3): 159–164, 2002.

[11] P. Galinier, Z. Boujbel, M.C. Fernandes. An efficient memetic algorithm for the graph partitioning problem. *Annals of Operations Research*, 191(1): 1–22, 2011.

[12] C. García-Martínez, F. Glover, F.J. Rodriguez, M. Lozano, R. Martí. Strategic oscillation for the quadratic multiple knapsack problem. *Computational Optimization and Applications*, 58(1): 161–185, 2014.

[13] C. García-Martínez, F.J. Rodriguez, M. Lozano. A tabu-enhanced iterated greedy algorithm: A case study in the quadratic multiple knapsack problem. *European Journal of Operational Research*, 232(3): 454–463, 2014.

[14] 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)*, 547–552, 2006.

[15] J.K. Hao. Memetic algorithms in discrete optimization. *In F. Neri, C. Cotta, P. Moscato (Eds.) Handbook of Memetic Algorithms. Studies in Computational Intelligence 379*, Chapter 6, 73–94, 2012.

[16] A. Jaszkiewicz. On the performance of multiple-objective genetic local search on the 0/1 knapsack problem: A comparative experiment. *IEEE Transaction on Evolutionary Computation*, 6(4):402–412, 2002.

[17] H.W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2: 83–97, 1955.

[18] S. Kirkpatrick, Jr. CD. Gelatt, M.P. Vecchi. Optimization by simulated annealing. *Science*, 220: 671–80, 1983.

[19] Z. Lu, J.K. Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1): 241–250, 2010.

[20] P. Merz, B. Freisleben. Fitness landscapes, memetic algorithms, and greedy operators for graph bipartitioning, *Journal of Evolutionary Computation*, 8(1): 61–91, 2000.

[21] P. Moscato, C. Cotta. A gentle introduction to memetic algorithms. *In F. Glover and G. Kochenberger (Eds.), Handbook of Metaheuristics*, Kluwer, Norwell, Massachusetts, USA, 2003.

[22] Y. Mei, X. Li, X. Yao. Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems. *IEEE Transactions on Evolutionary Computation*, 18(3): 435–449, 2014.

[23] F. Neri, C. Cotta, P. Moscato (Eds.) Handbook of Memetic Algorithms. *Studies in Computational Intelligence 379*, Springer, 2012.

[24] Y.S. Ong, A.J. Keane. Meta-lamarckian learning in memetic algorithms. *IEEE Transactions on Evolutionary Computation*, 8(2): 99–110, 2004.

[25] Y.S. Ong, M.H. Lim, X. Chen. Research frontier: memetic computation-past, present & future. *IEEE Computational Intelligence Magazine*, 5(2): 24–36, 2010.

[26] D. Pisinger. The quadratic knapsack problem-a survey. *Discrete Applied Mathematics*, 155: 623–48, 2007.

[27] D.C. Porumbel, J.K. Hao, P. Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers and Operations Research*, 37(10): 1822–1832, 2010.

[28] M. Resende, C. Ribeiro. Greedy randomized adaptive search procedures. *Handbook of Metaheuristics 2003*, 57: 219-249.

[29] T. Saraç A. Sipahioglu. A genetic algorithm for the quadratic multiple knapsack problem. Advances in Brain, Vision, and Artificial Intelligence. Lecture Notes in Computer Science, Volume 4729, pp 490–498, 2007.

[30] T. Saraç A. Sipahioglu. Generalized quadratic multiple knapsack problem and two solution approaches. *Computers & Operations Research*, 43: 78–89, 2014.

[31] A. Singh, A.S. Baghel. A new grouping genetic algorithm for the quadratic multiple knapsack problem. Evolutionary Computation in Combinatorial Optimization. Lecture Notes in Computer Science, 4446, pp. 210–218, 2007.

[32] K. Tang, Y. Mei, X. Yao. Memetic algorithm with extended neighborhood search for capacitated arc routing problems. *IEEE Transactions on Evolutionary Computation*, 13(5): 1151–1166, 2009.

Yuning Chen received the B.Eng. and M.Eng. degrees from the National University of Defense Technology (NUDT), Changsha, China, in 2010 and 2012 respectively. He is currently pursuing the Ph.D. degree in Computer Science at the LERIA laboratory, University of Angers, France. His research interests include evolutionary computation, memetic algorithms, meta-heuristics and multiobjective optimization for quadratic knapsack problems and other combinatorial optimization problems.

Jin-Kao Hao is a Distinguished Professor (Professeur des Universités de classe exceptionnelle) with the Computer Science Department, University of Angers (France) and is Senior Fellow with the Institut Universitaire de France. From 2003-2015, he was the head of the Computer Science Laboratory LERIA. He has authored or co-authored over 200 peer-reviewed publications and co-edited 9 books in Springers LNCS series. His research interests include design of effective algorithms and intelligent computational methods for solving large-scale combinatorial search problems. He has served as an Invited Member of over 180 program committees of international conferences and is on the editorial board of seven International Journals. J.-K. Hao graduated in 1982 from the National University of Defense Technology (School of Computer Science) (China). He received the Master degree (Oct. 1987) from the National Institute of Applied Sciences (INSA Lyon, France), the Ph.D. in Constraint Programming (Feb. 1991) from the University of Franche-Comté, and the Professorship Diploma HDR (Habilitation à Diriger des Recherches) (Jan. 1998) from the University of Science and Technology of Montpellier (France).