A hybrid evolutionary search for the generalized quadratic multiple knapsack problem

Qing Zhou [a], Jin-Kao Hao [a,*] and Qinghua Wu [b,*]

[a] *LERIA, Université d'Angers, 2 bd Lavoisier, 49045 Angers Cedex 01, France*

[b] *School of Management, Huazhong University of Science and Technology, No. 1037, Luoyu Road, Wuhan, China*

Accepted to ***European Journal of Operational Research***, April 2021

Abstract

Knapsack problems are useful models that can formulate many real-life applications. The generalized quadratic multiple knapsack problem (GQMKP) extends the well-known quadratic multiple knapsack problem by taking into account setups and knapsack preference of the items. In this study, an efficient hybrid evolutionary search algorithm (HESA) is proposed to tackle GQMKP, which relies on a knapsack-based crossover operator to generate new offspring solutions, and an adaptive feasible and infeasible tabu search to improve new offspring solutions. Other new features of HESA include a dedicated strategy to ensure a diversified and high-quality initial population, and a streamlining technique to speed up the evaluations of candidate solutions. The experiments on two sets of 96 benchmark instances as well as one large-scale real-life instance show that the proposed algorithm outperforms the state-of-the-art algorithms from the literature. In particular, HESA finds 44 improved best-known solutions (new lower bounds) (for more than 45% cases). The key components of the algorithm are studied to assess their effects on the algorithm's performance.

Keywords: Metaheuristics; generalized knapsack problem; evolutionary search; feasible and infeasible search.

* Corresponding author. Emails: qingzhouqz03@gmail.com; jin-kao.hao@univ-angers.fr and qinghuawu1005@gmail.com

1 Introduction

The knapsack problems (KP) is a classic combinatorial optimization problem, which has numerous real-world applications [28]. Given a knapsack with a predefined capacity, and a set of items such that each item has a weight and a profit, KP is to pack a subset of the items in the knapsack so that the sum of the profits of the selected items is maximized without exceeding the capacity of the knapsack. The multiple knapsack problem (MKP) is a natural generalization of the classic KP by considering m ($m > 1$) knapsacks with capacity $c_1, c_2, ..., c_m$ [9,11,22]. The quadratic multiple knapsack problem (QMKP) deals with multiple knapsacks, and defines a pairwise (quadratic) profit function between items [12,16].

The generalized quadratic multiple knapsack problem (GQMKP) is an extension of the canonical QMKP with setups and knapsack preferences of the items and has shown to be NP-hard [29]. GQMKP has a number of practical applications in manufacturing and production scheduling, where setups and machine preferences need to be considered [2,7] (see [29] for a real-life case study in a plastic injection molding company). The mathematical formulation of the problem is given in Section 2. Below, we review the existing methods for solving GQMKP reported in the literature since the introduction of the problem in 2014.

In [29], Saraç and Sipahioglu presented, along with the definition of the problem, a genetic algorithm (GA) and a hybrid algorithm (HA) combining GA and the F-MSG (modified subgradient algorithm handling on feasible values). GA initializes the population in a random way and uses a 2-tournament strategy for parent selection. GA adopts a dedicated uniform based crossover operator to generate offspring solutions, and two different mutation operators based on local exchange and greedy construction to mutate the offspring. The authors of [29] tested these algorithms on two sets of 96 benchmark instances that they introduced and a large-scale real-life case. Their computational results indicated that the hybrid algorithm was better than GA in terms of solution quality, while GA performed better in terms of solution time.

In 2016, Chen and Hao [7] presented the first memetic algorithm (MA) that combines a backbone-based crossover operator and a multi-neighborhood simulated annealing procedure. They introduced a novel large-sized neighborhood induced by the general-exchange move operator to enhance the search efficiency of the simulated annealing procedure. In addition, a quality-and-distance based population updating rule was used to prevent the search from premature convergence. MA showed a superior performance on the benchmark instances compared with the reference algorithms.

In 2017, Avci and Topaloglu [2] introduced a multi-start iterated local search method (MS-ILS) that integrates a variable neighborhood descent (VND) with an adaptive perturbation mechanism. The adaptive perturbation mechanism helps the algorithm to investigate different search regions while the VND aims to find high-quality local optima in the examined regions. Moreover, MS-ILS employs a tabu list to record recently visited solutions to avoid short-term cycling. The reported results indicated that MS-ILS was capable of producing high-quality solutions for many benchmark instances.

In 2019, Adouani et al. [1] developed a matheuristic variable neighborhood search algorithm (MVNS) that combines variable neighborhood search (VNS) and integer programming (IP). MVNS uses an integer programming based exact algorithm to enhance the performance of VNS. Their results showed that MVNS was very competitive compared with the state-of-the-art algorithms including MA [7] and MS-ILS [2].

According to the reported computational results, MA [7], MS-ILS [2], and MVNS [1] are the best performing methods for GQMKP in the literature. Together these algorithms produced the current best-known results for the two sets of 96 benchmark instances and the large-scale real-life instance.

We observe that compared with other knapsack problems, studies on GQMKP are quite limited. Moreover, the performances of the existing methods vary when they are applied to solve large instances. In this work, we aim to enrich the arsenal of solution methods for GQMKP by proposing an effective hybrid evolutionary search algorithm (HESA). HESA possesses four original characteristics. First, HESA uses a knapsack-based crossover operator to generate new offspring solutions. The design of this operator is motivated by the observation that high quality local optima share common building blocks (grouping items), which need to be preserved from parent solutions to offspring solution. Second, HESA adopts an adaptive feasible and infeasible tabu search, that uses a mixed search strategy guided by an adaptive penalty-based evaluation function, to explore both feasible and infeasible solutions for local optimization. This search strategy is based on the consideration that relaxing the capacity constraints in a controlled manner helps the search to tunnel through feasible and infeasible regions to reach high quality local optima, which are hard to attain otherwise. Third, a fast incremental evaluation technique is designed to rapidly calculate the move gain of each candidate move, which significantly increases the computational efficiency of the proposed algorithm. Finally, HESA starts its search with a high-quality initial population obtained with a greedy randomized construction procedure.

Computational assessments on the two sets of 96 benchmark instances as well as the large-scale real-life instance show that HESA is highly effective compared with the best performing algorithms in the literature. In particular,

HESA updates 44 current best-known lower bounds (for 9 small instances of Set I, 34 large instances of Set II and the real-life instance).

The remainder of the paper is structured as follows. Section 2 presents the problem and its mathematical formulation. Section 3 describes the design of the proposed algorithm. Computational results and comparisons with the state-of-the-art methods are shown in Section 4, followed by an analysis of important components of HESA in Section 5. Finally, Section 6 draws conclusions and shows perspectives.

2 Problem statement and mathematical formulation

Given a set of m knapsacks $K = \{1, 2, ..., m\}$, each with a capacity c_k ($k \in K$) and a set of n items $J = \{1, 2, ..., n\}$, which are categorized into h disjoint classes $R = \{1, 2, ..., h\}$. For each item j ($j \in J$), it is associated with a weight w_j and a profit p_{jk} when j is selected for knapsack k ($k \in K$). Each pair of items i and j ($i, j \in J$) generates a profit q_{ij} if items i and j are allocated to the same knapsack. Each class r ($r \in R$) incurs a setup time s_r when any item that belongs to class r is allocated to a knapsack k ($k \in K$). If several items in a same class are allocated to a knapsack, only one setup time is considered. For each class r ($r \in R$), a maximum number of knapsacks n_r are allowed to which items in class r can be allocated. Let t_{rj} be a parameter indicating whether item j is in class r or not ($j \in J$, $r \in R$): $t_{rj} = 1$ if j is in r, and $t_{rj} = 0$ otherwise. Let U be a parameter representing a positive large number. Let x_{jk} be the binary variable taking the value of 1 if item j is selected for knapsack k, and 0 otherwise. Let y_{rk} be the binary variable such that $y_{rk} = 1$ if any item in class r is allocated to knapsack k, and $y_{rk} = 0$ otherwise. Given the following sets,

- B_k: Set of items that can be allocated to knapsack $k \in K$.
- BB_k: Set of classes that can be activated for knapsack $k \in K$.
- D_j: Set of knapsacks to which item $j \in J$ can be allocated.
- DD_r: Set of knapsacks to which items in class $r \in R$ can be allocated.

GQMKP can then be expressed by the following mathematical formulation [1,2,29].

$$\textbf{Maximize} \quad \sum_{k \in K} \sum_{j \in B_k} p_{jk} x_{jk} + \sum_{k \in K} \sum_{i \in B_k} \sum_{j \in B_k (j>i)} q_{ij} x_{ik} x_{jk} \tag{1}$$

$$\textbf{subject to} \quad \sum_{j \in B_k} w_j x_{jk} + \sum_{r \in BB_k} s_r y_{rk} \leq c_k, \forall k \in K \tag{2}$$

$$\sum_{k \in DD_r} y_{rk} \leq n_r, \forall r \in R \tag{3}$$

$$\sum_{k \in D_j} x_{jk} \leq 1, \forall j \in J \tag{4}$$

$$\sum_{j \in B_k} t_{rj} x_{jk} \leq U y_{rk}, \forall k \in K, \forall r \in BB_k \tag{5}$$

$$x_{jk} \in \{0,1\}, \quad \forall j \in J, \forall k \in K \tag{6}$$

$$y_{rk} \in \{0,1\}, \quad \forall r \in R, \forall k \in K \tag{7}$$

In the above formulation, the objective function (1) is to maximize the total collected profit. Constraint (2) ensures that the weight sum of the selected items of each knapsack plus the sum of setup time consumption does not exceed its capacity. Constraint (3) guarantees that the number of knapsacks including any item in class r cannot be greater than the maximum number n_r of knapsacks of this class. Constraint (4) enforces that each item can be allocated to one knapsack at most. Constraint (5) requires that if an item in class r is selected for knapsack k, then the binary variable y_{rk} must take the value of 1. Binary values for variables x_{jk}, and y_{rk} are imposed in constraints (6) and (7), respectively.

3 Hybrid evolutionary search algorithm for the GQMKP

3.1 Main scheme

Algorithm 1: Hybrid evolutionary search algorithm for generalized quadratic multiple knapsack problem

Input: I: a GQMKP instance, p: population size
Output: Best found solution S^*

```
P = {S^1, S^2, ..., S^p} ← PopulationInitialization() /* P is the population, Section 3.2 */
S* ← Best(P) /* S* records the best solution found so far */
while Stopping condition is not met do
    Select randomly two parents S^a, S^b from P
    S^o ← CrossOver(S^a, S^b) /* Section 3.3, generate an offspring solution */
    S^o ← TabuSearch(S^o) /* Section 3.4, improve the solution */
    if f(S^o) > f(S*) then
        S* ← S^o /* Update the recorded best solution */
    P ← UpdatePopulation(P, S^o) /* Section 3.1, update the population */
```

Memetic algorithm [24,25] is a general hybrid framework that combines population-based evolutionary search and local search. Various memetic algorithms have been successfully employed to tackle a number of difficult combinatorial optimization problems, such as graph partitioning [3,20,21], quadratic assignment [4,30,33], knapsack [17], and critical nodes identification [34].

To obtain an effective MA for GQMKP, the design of our hybrid evolutionary search algorithm (HESA) complies with the design principle of its main search components [15] and pays a particular attention to its crossover operator (Section 3.3) and local optimization procedure (Section 3.4). Algorithm 1 summarizes the general procedure of HESA. HESA starts with an initial population (line 1, Section 3.2) with p (p is a parameter) feasible individuals. Then, HESA performs a number of generations until a stopping condition (typically a given number of generations) is satisfied (lines 3-9). At each generation, two parent solutions in the population are randomly selected (line 4) and recombined by the knapsack-based crossover operator (line 5, Section 3.3) to generate a feasible offspring solution S^o. Then, S^o is improved by the adaptive feasible and infeasible tabu search (line 6, Section 3.4). Finally, S^o is used to update the best recorded solution S^* (lines 7-8) as well as the population (line 9). To update the population, the offspring solution replaces the worst solution of the population if S^o has a better objective value and is not the same as any solution in the population; otherwise the population is kept unchanged.

The main components of HESA are presented in the following subsections.

3.2 Population initialization

The initial population P consists of p (p is the population size) feasible individuals (solutions), and is built by the following population initialization algorithm (PIA, Algorithm 2). The PIA first generates a number of gr (gr is a parameter, $gr > p$) feasible solutions with the greedy randomized construction procedure (GRCP, Algorithm 3) introduced in [2].

Let S denote a candidate solution, which is a partition of n items into $m + 1$ knapsacks $S = \{C_0, C_1, ..., C_m\}$ such that each C_k ($k \in \{1, 2, ..., m\}$) is the set of items allocated to knapsack k, while C_0 is a dummy knapsack containing the unallocated items. GRCP operates as follows. Starting from $S = \{C_0, C_1, ..., C_m\}$, GRCP first initializes the set of available knapsacks KL and the set of unallocated items IL, i.e., $KL = \{1, 2, ..., m\}$, and $IL = \{1, 2, ..., n\} \setminus \{C_1 \cup ... \cup C_m\}$ (lines 1-2, Algorithm 3). Then, GRCP repeats a series of iterations until KL becomes empty (lines 3-17, Algorithm 3). At

Algorithm 2: Population initialization algorithm

Input: I: a GQMKP instance, p: population size, gr: the number of greedy randomized solutions, μ: shake strength
Output: Population P

```
1  P ← ∅
2  S_gr ← ∅ /* S_gr is a set of solutions */
3  for t ← 1 to gr do
4      Initializing C_0 = {1, 2, ..., n}, C_k = ∅ (k ∈ {1, 2, ..., m})
5      S ← {C_0, C_1, ..., C_m}
6      S ← GreedyRandomConstruction(S) /* Generate a feasible solution in
         a greedy randomized manner, Section 3.2 */
7      S_gr ← S_gr ∪ {S}
8  Sort the solutions in S_gr according to their objective values (Eq. (1)) in a
     non-ascending order
9  S_p ← select the first p solutions from S_gr
10 for t ← 1 to p do
11     S ← select the t-th element of S_p
12     S ← TabuSearch(S) /* Improve the solution, Section 3.4 */
13     if S does not exist in P then
14         P ← P ∪ {S} /* Insert S into P */
15     else
16         repeat
17             S ← RandomShake(S, μ)
18         until S is not the same as any solution in P
19         P ← P ∪ {S} /* Insert S into P */
```

each iteration, a knapsack k is selected randomly from KL. Then the set of items FS containing the unallocated items in IL that can fit into knapsack k and satisfy the constraints (2)-(7) is constructed (lines 5-10, Algorithm 3). After that, an item j^* from FS having the largest value density (ties are randomly broken) is allocated to knapsack k (lines 13-14, Algorithm 3). The value density $vd(j,k)$ of an item j ($j \in J$) with respect to a knapsack k ($k \in K$) is defined by [2]:

$$vd(j,k) = (p_{jk} + \sum_{i \in C_k, i \neq j} q_{ij})/w_j \tag{8}$$

Then, j^* is removed from IL and FS is updated accordingly (lines 15-16, Algorithm 3). The operations of selecting an item from FS with the largest value density and updating the sets of C_k, IL as well as FS are repeated until FS becomes empty (lines 11-16, Algorithm 3). In this case, the knapsack k becomes unavailable and is removed from KL (line 17, Algorithm 3). This process is repeated until all knapsacks are examined and an initial solution is then returned. PIA invokes the GRCP procedure gr times to construct the set

Algorithm 3: Greedy randomized construction procedure

Input: a GQMKP instance I, a solution $S = \{C_0, C_1, ..., C_m\}$
Output: A feasible solution denoted by $S = \{C_0, C_1, ..., C_m\}$

```
1  KL ← {1, 2, ..., m} /* KL is the set of available knapsacks */
2  IL ← {1, 2, ..., n} \ {C_1 ∪ ... ∪ C_m} /* IL is the set of unallocated items */
3  while |KL| > 0 do
4      /* |KL| is the size of KL */
5      Select a knapsack k from KL at random
6      FS ← ∅
7      for t ← 1 to |IL| do
8          /* |IL| is the size of IL */
9          if the t-th element j_t of IL can fit into knapsack k then
10             FS ← FS ∪ {j_t} /* FS is the set of unallocated items that can
                fit into knapsack k */
11     while |FS| > 0 do
12         /* |FS| is the size of FS */
13         j* ← argmax{vd(j, k), j ∈ FS} /* j* denotes the item from FS
              having the largest value density with respect to knapsack k */
14         C_k ← C_k ∪ {j*} /* Allocate item j* to knapsack k */
15         IL ← IL \ {j*} /* Remove item j* from IL */
16         Update FS
17     KL ← KL \ {k} /* Remove knapsack k from KL */
18 C_0 = {1, 2, ..., n} \ {C_1 ∪ C_2 ∪ ... ∪ C_m}
19 return S = {C_0, C_1, ..., C_m}
```

of solutions S_{gr} with gr members (lines 3-7, Algorithm 2). Then, PIA sorts the elements in S_{gr} according to their objective values (Eq. (1)) in a non-ascending order, and selects the first p elements as the set S_p (lines 8-9, Algorithm 2). Finally, PIA repeats a series of iterations until the population contains p individuals (lines 10-19, Algorithm 2). At each iteration t, the t-th element S of S_p is chosen and is improved by the tabu search procedure (Section 3.4) (lines 11-12, Algorithm 2). The improved solution S is inserted into the population P if it is not the same as any existing solution in P (lines 13-14, Algorithm 2). Otherwise, a random shake procedure is applied to S to change a small part of S and make it different from the solutions of P, and then S is added to P (lines 15-19, Algorithm 2). The random shake procedure consists of μ (μ is a parameter called the shake strength) operations, each performing a random move from the combined feasible reallocate and swap neighborhoods (Section 3.4.1). By this means, PIA is able to generate an initial population of high quality that contributes to the efficiency of the search process.

We now consider the time complexity of the greedy randomized construction procedure GRCP. At the beginning of GRCP, the set of unallocated items IL is initialized in $O(n)$ time (line 2, Algorithm 3). At each iteration of GRCP,

the set of items in IL that can fit into the selected knapsack k, i.e., FS, is identified in $O(|IL|)$ time, where $|IL|$ is the size of IL (lines 7-10, Algorithm 3). And then an item j^* from FS having the largest value density with respect to k is identified in $O(|FS| \times |C_k|)$ (line 13, Algorithm 3), where $|FS|$ and $|C_k|$ denote the sizes of FS and C_k, respectively, while FS is updated in $O(|IL|)$ (line 16, Algorithm 3). Therefore, the total time complexity of GRCP is bouned by $O(n)+O((|FS|\times|C_k|+|IL|)\times|FS|\times m)$ (denoted by $O(\text{GRCP})$).

3.3 Knapsack-based crossover operator

Algorithm 4: knapsack-based crossover operator

Input: Two randomly selected parents $S^a = \{C_0^a, C_1^a, ..., C_m^a\}$, and $S^b = \{C_0^b, C_1^b, ..., C_m^b\}$
Output: A feasible offspring solution $S^o = \{C_0^o, C_1^o, ..., C_m^o\}$

```
1  Initializing C_0^o = {1, 2, ..., n}, C_k^o = ∅ (k ∈ {1, 2, ..., m})
2  t ← 1 /* Iteration counter */
3  while t <= m do
4      if t % 2 ≠ 0 then
5          Choose a knapsack C_* ∈ S^a (C_* ≠ C_0^a) having the largest profit
6          C_t^o ← C_*
7      else
8          Choose a knapsack C_* ∈ S^b (C_* ≠ C_0^b) having the largest profit
9          C_t^o ← C_*
10     Remove all the items in C_* from S^a and S^b
11     t ← t + 1
12 C_0^o = {1, 2, ..., n} \ {C_1^o ∪ C_2^o ∪ ... ∪ C_m^o}
13 S^o ← {C_0^o, C_1^o, ..., C_m^o}
14 S^o ← GreedyRandomConstruction(S^o) /* Complete the solution in a
       greedy randomized manner, Section 3.2 */
15 return S^o = {C_0^o, C_1^o, ..., C_m^o}
```

The knapsack-based crossover operator (denoted by KCX) is a key search component of the proposed HESA algorithm. At each generation of HESA, KCX is used to generate a new offspring solution by recombining two parent solutions selected randomly from the population. Generally, a successful crossover operator should not only generate diversified solutions but also transfer good properties ("building blocks") from parents to offspring solutions in relation to a given optimization objective [15]. As GQMKP can be viewed as a grouping problem [7], it is more meaningful and natural for a crossover to manipulate groups of items rather than individual items. Crossover operators utilizing such an idea have been successfully used to tackle a number of grouping problems including graph partitioning problem [3] and graph coloring problem [18]. Moreover, a previous study discloses that high quality solutions

of GQMKP share groups of items [7].

The proposed KCX operator follows the general principle presented above and is described in Algorithm 4. Let $S^a = \{C_0^a, C_1^a, ..., C_m^a\}$, and $S^b = \{C_0^b, C_1^b, ..., C_m^b\}$ be the given parent solutions, KCX generates one offspring solution $S^o = \{C_0^o, C_1^o, ..., C_m^o\}$ by alternatively and successively inheriting the knapsacks with the largest profit from the parent solutions. This is reasonable because by inheriting the most profitable knapsacks from both parents, the offspring solution will conserve these good building blocks through the crossover process.

Specifically, KCX operates in two sequential steps. The first step (lines 3-11) consists of m iterations (i.e., one knapsack per iteration), where at each iteration t, a knapsack C_* ($C_* \neq C_0^a$ and $C_* \neq C_0^b$) from a reference parent (determined in an alternative way) is chosen such that C_* has the largest profit. Then, C_* becomes the t-th knapsack of the offspring S^o, followed by the removal of all the items in C_* from the parent solutions S^a and S^b. Note that the first step may result in a solution, where some knapsacks are possibly not fully filled. In other words, some unallocated items could fit into the knapsacks of the solution and satisfy all the constraints. The second step of KCX (lines 12-14) thus completes the solution by the GRCP procedure presented in Section 3.2.

We estimate the time complexity of KCX as follows. KCX first sets the m knapsacks to be empty in $O(1)$ time (line 1, Algorithm 4). At each iteration of KCX, the profit of each knapsack of S^a or S^b is computed in $O(n + n \times n)$ time and the knapsack C_* with the largest profit is identified in $O(m)$ time (line 5 or 8, Algorithm 4). The items in C_* are removed from S^a and S^b in $O(|C_*|)$ time, where $|C_*|$ is the size of C_* (line 10, Algorithm 4). After that, the solution S^o is completed by the GRCP procedure in $O(\text{GRCP})$ time (line 14, Algorithm 4). Therefore, the total time complexity of KCX is bounded by $O((n + n \times n + m + |C_*|) \times m) + O(\text{GRCP})$.

As indicated in Section 1, MA [7], which is one of the best performing algorithms for GQMKP, uses the backbone-based crossover operator (denoted by BCX) for solution recombination. The main difference between BCX and KCX is that BCX aims to transfer common items shared in both parents to the offspring as many as possible, while KCX transfers to the offspring all the items in a knapsack with the largest profit from a reference parent which is determined in an alternative manner.

3.4 Adaptive feasible and infeasible tabu search

The proposed HESA algorithm uses an adaptive feasible and infeasible tabu search (AFITS) to improve initial solutions and offspring solutions. As discussed in [13], for constrained optimization problems, oscillating between infeasible and feasible regions during the search process can help to find high-quality solutions, which are difficult to locate if the search process is confined in feasible regions only (see [26,27,31] for some specific examples). The proposed AFITS algorithm follows this idea and explores an enlarged search space including both feasible and infeasible solutions unlike existing GQMKP algorithms, such as MA [7], MS-ILS [2], and MVNS [1], which only visit feasible regions. AFITS is based on the general framework of the well-known tabu search [14]. In addition, AFITS uses an adaptive parameter β tuned dynamically on the basis of the search context to control the importance given to the infeasible solutions (Section 3.4.2). The neighborhood structures and fast incremental evaluation technique as well as the exploration strategy of AFITS are presented in the following subsections.

3.4.1 Neighborhood structures and fast incremental evaluation technique

AFITS exploits two neighborhoods, i.e., the reallocate neighborhood N_r and the swap neighborhood N_s, which have been successfully employed in previous studies [1,2,7].

Reallocate operator: Given a current solution $S = \{C_0, C_1, ..., C_m\}$, the *Reallocate* operator (denoted by $Real(i, C_u, C_v)$) transfers an item i from its current knapsack $C_u \in \{C_0, C_1, ..., C_m\}$ to another knapsack $C_v \in \{C_1, C_2, ..., C_m\}$ ($C_u \neq C_v$) satisfying constraints (2)-(7). Note that this operator can add an unallocated item collected in C_0 to a knapsack. To efficiently calculate the move gain of a candidate move, AFITS adopts for the first time a fast incremental evaluation technique. The main idea is to maintain a $n \times (m+1)$ matrix γ, where its element $\gamma[j][k]$ records the sum of profits between an item j and all other items in knapsack C_k of the current solution S, i.e., $\gamma[j][k] = \sum_{i \in C_k, i \neq j} q_{ij}$. Given that C_0 of S is a dummy knapsack that has no contribution to the objective value of S, all the values of the 0-th column of γ are initialized to 0, i.e., $\gamma[j][0] = 0, j \in J$. Let P_{jk} be a parameter, where for each item $j \in J$, $P_{jk} = p_{jk}$ when $k \in \{1, 2, ..., m\}$ and $P_{jk} = 0$ when $k = 0$. The move gain of a $Real(i, C_u, C_v)$ operation can then be efficiently calculated by:

$$\Delta_f(Real(i, C_u, C_v)) = \gamma[i][v] - \gamma[i][u] + P_{iv} - P_{iu} \tag{9}$$

After performing a $Real(i, C_u, C_v)$ move, the matrix γ is updated in $O(n)$ time

as follows: $\gamma[j][u] = \gamma[j][u] - q_{ij}$, $\gamma[j][v] = \gamma[j][v] + q_{ij}$, $\forall j \in \{1, 2, ..., n\}, j \neq i, u \neq 0, v \neq 0$.

Swap operator: The *Swap* operator (denoted as $Swap(i, j)$) exchanges two items i and j from two different knapsacks C_u and C_v ($C_u, C_v \in \{C_0, C_1, ..., C_m\}, C_u \neq C_v$), by keeping all constraints satisfied. This operator requires that one of the two selected items is allocated and the other item is unallocated, or both of the two selected items are allocated but they belong to different knapsacks. When a $Swap(i, j)$ move is performed, the associated move gain can be efficiently computed as:

$$\Delta_f(Swap(i,j)) = \begin{cases} (\gamma[i][v] - \gamma[i][u] + \gamma[j][u] - \gamma[j][v] + P_{iv} - P_{iu} \\ \quad +P_{ju} - P_{jv} - q_{ij}, if\ u = 0 \vee v = 0) \\ (\gamma[i][v] - \gamma[i][u] + \gamma[j][u] - \gamma[j][v] + P_{iv} - P_{iu} \\ \quad +P_{ju} - P_{jv} - 2 \times q_{ij}, if\ u \neq 0 \wedge v \neq 0) \end{cases} \tag{10}$$

Given that a *Swap* move is composed of two consecutive *Reallocate* moves, γ is consecutively updated two times as for the *Reallocate* operation. Updating γ after a *Swap* move can also be realized in $O(n)$ time. As such the *Swap* moves benefit from the fast incremental evaluation technique similar to the *Reallocate* moves.

3.4.2 Exploration of feasible and infeasible solutions with tabu search

The main idea behind AFITS is to enable the search to visit intermediary infeasible solutions by relaxing the capacity constraints (Constraint (2)) of the knapsacks in a controlled manner. Complying with the general idea of penalty function for constrained optimization, AFITS uses an extended penalty-based evaluation function F, which combines the objective function with a penalty function to assess the quality of both feasible and infeasible solutions.

Let $S = \{C_0, C_1, ..., C_m\}$ denote a candidate solution in the enlarged search space, the objective function $f(S)$ (Eq. (1)) is equivalent to the sum of total profits of all knapsacks, i.e.,

$$f(S) = \sum_{k=1}^{m} tp(k) \tag{11}$$

where $tp(k)$ is the total profit of the knapsack k, given by $tp(k) = \sum\limits_{j \in C_k} p_{jk} + \sum\limits_{i \in C_k} \sum\limits_{j \in C_k, j > i} q_{ij}$.

Algorithm 5: Adaptive feasible and infeasible tabu search

Input: A feasible solution S, the search depth of tabu search sd
Output: Best feasible solution found S^*

$t \leftarrow 0$
$\beta \leftarrow 1$
$S^* \leftarrow S$ /* S^* records the best feasible solution found so far */
$S^b \leftarrow S$ /* S^b records the best solution found so far in terms of the extended penalty-based evaluation function F */
Initialize the tabu list tl
while $t < sd$ **do**
    Select a best admissible neighboring solution S' from the union of N_r and N_s neighborhoods in terms of F
    $S \leftarrow S'$
    Update the tabu list tl
    if $g(S) = 0 \wedge f(S) > f(S^*)$ **then**
        $S^* \leftarrow S$
    if $F(S) > F(S^b)$ **then**
        $S^b \leftarrow S$
        $t \leftarrow 0$
    else
        $t \leftarrow t + 1$
    if *All previous* λ *solutions are feasible* **then**
        $\beta \leftarrow \beta / \tau$
    else if *They are all infeasible solutions* **then**
        $\beta \leftarrow \beta \times \tau$
    if $\beta < 1$ **then**
        $\beta \leftarrow 1$
return S^*

Let $Z = \{Z_1, Z_2, ..., Z_m\}$ be a m-dimensional vector, where each element Z_k is the set of classes such that for each class $r \in Z_k$, the item selected to the knapsack k has at least one that belongs to r. Let W be another m-dimensional vector $W = \{W_1, W_2, ..., W_m\}$ to record the total weight of each knapsack of solution S, i.e., $W_k = \sum_{j \in C_k} w_j + \sum_{r \in Z_k} s_r$. The penalty function $g(S)$ corresponds to the infeasibility degree of S, given by the overall overloaded parts of all the knapsacks to the capacity limit, i.e., $g(S) = \sum_{k=1}^{m} o_k$, where

$$o_k = \begin{cases} W_k - c_k, & if \quad W_k > c_k \\ 0, & otherwise \end{cases} \tag{12}$$

Then, the extended evaluation function F is a linear combination of the objective function $f(S)$ and the penalty function $g(S)$:

$$F(S) = f(S) - \beta \times g(S) \tag{13}$$

where β is a parameter that controls the degree of infeasibility and is dynamically tuned using a self-adjustment method [31].

AFITS explores the reallocate neighborhood N_r and the swap neighborhood N_s in a union way without considering the capacity constraints of the knapsacks. To quickly calculate the move gain of a candidate move with respect to F, AFITS further uses a fast incremental evaluation technique. Specifically, for a given move denoted as mv $(Reallocate, Swap)$, its move gain is defined by:

$$\Delta_F(mv) = \Delta_f(mv) - \beta \times \Delta_g(mv) \tag{14}$$

With the help of the matrix γ presented above, the move gain $\Delta_f(mv)$ of each move mv relative to the objective function f is efficiently calculated in $O(1)$ time. Similarly, given that each move induced by *Reallocate*, and *Swap* operators involves only two knapsacks, the total weight of the two knapsacks can be directly computed with the assistance of the m-dimensional vector W above. Moreover, the move value $\Delta_g(mv)$ of each move mv related to the change in terms of the penalty function g can be conveniently determined according to Eq. (12) in $O(1)$ time. Therefore, with the data structures γ and W, the move gain that corresponds to the variation in the penalty-based evaluation function F can be efficiently calculated in constant time. As such, the penalty-based evaluation function F of each neighboring solution can be efficiently computed as $F = F + \Delta_F$.

The general scheme of AFITS is presented in Algorithm 5. Starting from a feasible input solution, AFITS iteratively replaces the current solution S with the best admissible neighboring solution S' (ties are randomly broken), in terms of F from the union of N_r and N_s neighborhoods (lines 6-22, Algorithm 5), in $O(n \times (m+1) + n \times n)$ time for each iteration with the help of the fast incremental evaluation technique (Section 3.4.1). To prevent the search from short-term cycling, each time an item j $(j \in J)$ is removed from its original knapsack C_k $(k \in \{0, 1, ..., m\})$, it is forbidden to add j to C_k for the next tt iterations (tt is a parameter called tabu tenure) (line 9, Algorithm 5). A move is however always performed if it leads to the best solution never encountered during the search. During the search process, β is divided (multiplied) by τ if all previous λ solutions are feasible (infeasible) and its initial value is set to be 1 (lines 17-22, Algorithm 5). The values of τ and λ are empirically set to 2 and 5, respectively. Generally, a larger (smaller) value of β imposes a heavier (lighter) penalization to infeasible solutions and results in a search, which grants less importance to infeasible (feasible) solutions. The best feasible solution found

S^* is updated by the current solution S if S is feasible, (i.e., $g(S) = 0$) and has a better quality than S^* in terms of the objective function f (lines 10-11, Algorithm 5). Similarly, the best solution found S^b is renewed with S if S is better than S^b in terms of F (lines 12-14, Algorithm 5). The search stops when the best solution found S^b cannot be updated for sd consecutive iterations, where sd is a parameter called the search depth of AFITS, and then the best feasible solution S^* is returned.

4 Results of computational experiments

To assess the performance of the proposed HESA algorithm, we test the algorithm on the two sets of benchmark instances and a large-scale real-life instance commonly used in the literature, and compare the results with the state-of-the-art methods.

4.1 Benchmark instances and experimental protocol

The two sets of 96 benchmark instances were first introduced in [29] and later used in [1,2,7], while the real-life instance was tested only in [7,29].

- Set I: This set consists of 48 small-sized instances with $n = 30$ items, $m \in \{1, 3\}$ knapsacks, $h \in \{3, 15\}$ classes, and density values $d \in \{0.25, 1.00\}$.
- Set II: It contains 48 large-sized instances, which are characterized by their number of items $n = 300$, number of knapsacks $m \in \{10, 30\}$, number of classes $h \in \{30, 150\}$, and density values $d \in \{0.25, 1.00\}$.
- Real-life: This instance corresponds to a real application concerning plastic parts production with injection machines. The instance was first introduced in [29] and faithfully regenerated by the authors of [7]. This instance has the following features: $n = 500$ items, $m = 40$ knapsacks, $h = 300$ classes, and density $d = 1.00$.

These instances have unknown optimal solutions. All the input data and the certificates of the results obtained by the proposed HESA algorithm are available at `http://www.info.univ-angers.fr/pub/hao/HESA-GQMKP.html`.

Table 1 shows for each instance of Set I and Set II, the following information: instance name (Ins.), number of items (n), number of knapsacks (m), number of classes (h), and density (d).

The HESA algorithm was implemented in C++ and compiled using GNU g++ 8.3.0 with the "-O3" option (We will make our code publicly available at the

Table 1
List of benchmarks.

Ins.	n	m	h	d	Ins.	n	m	h	d
Set I (48 instances)					Set II (48 instances)				
5-1	30	3	15	1.00	1-1	300	10	30	0.25
5-2	30	3	15	1.00	1-2	300	10	30	0.25
5-3	30	3	15	1.00	1-3	300	10	30	0.25
6-1	30	1	3	0.25	2-1	300	30	150	0.25
6-2	30	1	3	0.25	2-2	300	30	150	0.25
6-3	30	1	3	0.25	2-3	300	30	150	0.25
8-1	30	3	15	0.25	3-1	300	10	150	0.25
8-2	30	3	15	0.25	3-2	300	10	150	0.25
8-3	30	3	15	0.25	3-3	300	10	150	0.25
15-1	30	1	3	0.25	4-1	300	10	150	1.00
15-2	30	1	3	0.25	4-2	300	10	150	1.00
15-3	30	1	3	0.25	4-3	300	10	150	1.00
18-1	30	1	3	1.00	7-1	300	10	30	1.00
18-2	30	1	3	1.00	7-2	300	10	30	1.00
18-3	30	1	3	1.00	7-3	300	10	30	1.00
20-1	30	1	15	1.00	9-1	300	30	30	0.25
20-2	30	1	15	1.00	9-2	300	30	30	0.25
20-3	30	1	15	1.00	9-3	300	30	30	0.25
22-1	30	3	3	0.25	10-1	300	30	30	1.00
22-2	30	3	3	0.25	10-2	300	30	30	1.00
22-3	30	3	3	0.25	10-3	300	30	30	1.00
23-1	30	3	3	1.00	11-1	300	10	30	0.25
23-2	30	3	3	1.00	11-2	300	10	30	0.25
23-3	30	3	3	1.00	11-3	300	10	30	0.25
25-1	30	3	15	1.00	12-1	300	30	150	1.00
25-2	30	3	15	1.00	12-2	300	30	150	1.00
25-3	30	3	15	1.00	12-3	300	30	150	1.00
26-1	30	1	15	1.00	13-1	300	30	150	0.25
26-2	30	1	15	1.00	13-2	300	30	150	0.25
26-3	30	1	15	1.00	13-3	300	30	150	0.25
27-1	30	1	15	0.25	14-1	300	10	30	1.00
27-2	30	1	15	0.25	14-2	300	10	30	1.00
27-3	30	1	15	0.25	14-3	300	10	30	1.00
28-1	30	1	15	0.25	16-1	300	30	30	0.25
28-2	30	1	15	0.25	16-2	300	30	30	0.25
28-3	30	1	15	0.25	16-3	300	30	30	0.25
29-1	30	3	3	0.25	17-1	300	30	30	1.00
29-2	30	3	3	0.25	17-2	300	30	30	1.00
29-3	30	3	3	0.25	17-3	300	30	30	1.00
30-1	30	3	3	1.00	19-1	300	10	150	1.00
30-2	30	3	3	1.00	19-2	300	10	150	1.00
30-3	30	3	3	1.00	19-3	300	10	150	1.00
31-1	30	3	15	0.25	21-1	300	10	150	0.25
31-2	30	3	15	0.25	21-2	300	10	150	0.25
31-3	30	3	15	0.25	21-3	300	10	150	0.25
32-1	30	1	3	1.00	24-1	300	30	150	1.00
32-2	30	1	3	1.00	24-2	300	30	150	1.00
32-3	30	1	3	1.00	24-3	300	30	150	1.00

Table 2
Settings of parameters.

Parameter	Section	Description	Considered values	Final value
p	3.2	size of population	{5, 10, 20, 30, 50}	10
gr	3.2	number of greedy randomized solutions	{100, 500, 1000, 2000, 5000}	1000
μ	3.2	shake strength	{0.02, 0.05, 0.08, 0.10, 0.20}	0.10
tt	3.4.2	tabu tenure	{10, 20, 30, 50, 100}	50
sd	3.4.2	search depth of tabu search	{500, 1000, 2000, 5000, 10000}	2000

Table 3
Scaling factors of the processors used in the reference algorithms, based on the processor used in this work.

Algorithm	Reference	Processor type	CPU frequency (GHz)	Scaling factor
HESA	this work	Intel E5-2695 v4	2.10	1.00
MA	[7]	AMD Opteron 4184	2.80	1.33
MS-ILS	[2]	Intel core 2 duo T7500	2.20	1.05
MVNS	[1]	Intel core 2 duo B960	2.40	1.14

link above). All experiments were conducted on a computing platform with an Intel E5-2695 v4 processor (2.10 GHz) and 1 GB RAM, running Debian 8.3.0-6.

4.2 Parameter tuning

HESA requires 5 parameters: the population size p, the number of greedy randomized solutions gr, the shake strength μ, the tabu tenure tt, and the search depth of tabu search sd. To tune these parameters, we used the "irace" package [19], which implements the Iterated F-Race method [5] to determine automatically the most suitable parameter settings from a set of possible parameter configurations. For this tuning experiment, a set of 15 randomly selected instances from Set I and Set II were used, i.e., 1-1, 1-3, 3-1, 3-2, 10-1, 11-3, 13-3, 14-2, 14-3, 16-3, 18-2, 23-1, 24-1, 24-2, and 31-2. The tuning budget (the maximum number of runs of the algorithm) was set to 1000 runs, where each run stops after performing a fixed number of 100 generations. The range of possible parameter values and final values suggested by "irace" are shown in Table 2. The final parameter values were used in all the experiments.

Table 4
Summarized results between HESA and three reference algorithms on the instances of Set I and Set II. The best results are marked in bold.

Instance set	MA	MS-ILS	MVNS	HESA
Set I (48 instances)				
#Best	35/28	39/33	39/38	**48/46**
#Improve/#Match	0/44	0/48	0/48	**9**/39
Average Time(s)	0.35	1.92	1.77	**0.25**
Average $g_{best}/g_{avg}(\%)$	-0.45/-0.58	0.00/-0.09	0.00/0.00	**0.74/0.67**
p-value$_{best}$	3.11e-4	2.70e-3	2.70e-3	
p-value$_{avg}$	1.24e-4	1.62e-3	2.09e-2	
Set II (48 instances)				
#Best	4/0	8/0	14/19	**40/33**
#Improve/#Match	0/8	0/29	0/47	**34**/6
Average Time(s)	3025.75	2896.87	**287.25**	336.61
Average $g_{best}/g_{avg}(\%)$	-0.51/-0.83	-0.10/-1.46	0.00/-0.03	**3.46/3.23**
p-value$_{best}$	1.01e-8	2.96e-7	6.02e-5	
p-value$_{avg}$	4.14e-8	7.76e-9	3.48e-2	

Table 5
Results of the Wilcoxon signed-rank test for HESA and the reference algorithms on the instances of Set I and Set II, with a significance level of 0.05.

Instance set	Comparison	R^+_{best}	R^-_{best}	p-value	R^+_{avg}	R^-_{avg}	p-value
Set I (48 instances)							
	HESA vs. MA	13	0	1.47e-3	20	2	2.01e-4
	HESA vs. MS-ILS	9	0	7.69e-3	15	2	1.18e-3
	HESA vs. MVNS	9	0	7.69e-3	10	2	6.04e-3
Set II (48 instances)							
	HESA vs. MA	41	3	1.03e-7	43	5	5.77e-8
	HESA vs. MS-ILS	39	5	1.92e-6	44	4	1.78e-7
	HESA vs. MVNS	34	8	3.40e-5	29	15	2.82e-2

Table 6
Comparative results between HESA and two state-of-the-art algorithms (GA [29], MA [7]) on the real-life instance. The dominating values are marked in bold.

	GA			MA			HESA		
f_{bk}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
16018.80	15884.30	15669.10	9133.48	16018.80	15908.50	2611.21	**16642.61**	**16599.11**	**1763.87**

4.3 Comparative results with state-of-the-art algorithms

To assess the performance of our HESA algorithm, this section provides an extensive computational comparison between HESA and several state-of-the-art algorithms from the literature on the two sets of 96 benchmark instances

and the large-scale real-life instance. According to the computational results presented in one of the latest studies on GQMKP [1], the following three algorithms can be considered to be the best performing ones for GQMKP: the population-based memetic algorithm (MA) [7], the multi-start iterated local search algorithm (MS-ILS) [2] and the matheuristic variable neighborhood search algorithm (MVNS) [1]. Therefore, we use them as the main reference algorithms for the computational comparisons. On the basis of [1,2,7], HESA was executed independently 30 times per instance by using different random seeds, where for the z-th ($z = \{1, 2, ..., 30\}$) execution, the random seed was set to be z. Considering that the source codes of the reference algorithms are not available, the numerical results reported by the reference algorithms are directly compiled from the related literature.

Note that the stopping conditions of the reference algorithms are not uniform. Specifically, MVNS [1] stops if there is no solution improvement after exploring a fixed number of k_{max} neighborhoods (k_{max} is set to be the number of classes in the paper). MS-ILS [2] is terminated after it performs a number of $maxStart$ (a parameter of MS-ILS) times of the iterated local search (ILS). MA [7] uses two different stopping criteria: a short time limit of 100 generations and a long time limit of 500 generations. The algorithm proposed in [29] utilizes two types of termination conditions together. One of them examines whether the algorithm has performed a given number n_f of generations. The other terminates the algorithm if the solution has not been improved during n_i consecutive generations (n_f and n_i are parameters). In our work, the stopping condition of the HESA algorithm is a fixed number of 100 generations as in [7].

It is a difficult task to provide a fully fair comparison between HESA and the reference algorithms due to the differences in computing platforms, implementations and termination conditions, etc. We thus mainly focus on the quality criterion of the obtained solutions and provide computation times for indicative purposes only. On the basis of [8,23,32], we used the *CPU frequency* to compare the speeds of the processors that were employed to test HESA and the reference algorithms. Using our processor (Intel E5-2695 v4, 2.10 GHz) as a basis, Table 3 shows the scaling factors of the processors used by the reference algorithms, indicating that the processor used in this study is a slightly slower.

Table 4 summarizes the comparative results on the instances of Set I and Set II. Row "#Best" indicates the number of instances for which the corresponding algorithm yields the best result among all the compared algorithms in terms of the best and average objective value. Row "#Improve/#Match" shows the number of cases where each algorithm improves or matches the best-known solution in the literature. The average running time in seconds required by each algorithm to reach its final objective value across all the tested instances is provided in row "Average Time(s)". The average percentage gap from the best

or average objective value to the best-known solution in the literature is given in row "Average $g_{best}/g_{avg}(\%)$". For each instance, the percentage gaps of the best and the average objective value are computed as $(f - f_{bk})/f_{bk} \times 100$, where f is the best or the average objective value obtained by the given algorithm and f_{bk} is the best-known solution from the literature. To verify whether there exists a statistically significant difference between HESA and the compared algorithms in terms of the best (average) objective values, the p-values from the nonparametric Friedman test [10] are provided in row "p-value$_{best}$" ("p-value$_{avg}$"). Detailed results on the benchmark instances of Set I and Set II are presented in Tables 14 and 15 of the Appendix, respectively.

One observes from Table 4 that in terms of the best objective value, HESA produces the best result for 48 (40) cases out of the 48 Set I instances (Set II instances), while MA, MS-ILS and MVNS provide the best result for 35 (4), 39 (8) and 39 (14) instances, respectively. With regard to the average objective value for the two benchmarks, HESA reports 46 (33) best results against 28 (0) for MA, 33 (0) for MS-ILS, and 38 (19) for MVNS. In particular, HESA reports improved best-known solutions for 9 out of the 48 small-sized instances (Set I) and matches the best-known solution for all the remaining instances. More importantly, for the 48 large-sized instances of Set II, HESA improves (matches) 34 (6) best-known solutions, and misses the best-known solution only for the 8 remaining instances. For both sets, HESA further reports the best result on the average percentage gap of the best/average objective value to the best-known solution (0.74%/0.67% and 3.46%/3.23%). In terms of the average running time (in seconds) required to reach the final objective value, for the Set I benchmark, HESA shows a slightly better performance than all the compared algorithms. Meanwhile for the Set II benchmark, HESA performs better than MA and MS-ILS, but slightly worse than MVNS. The p-values from the nonparametric Friedman test (rows p-value$_{best}$ and p-value$_{avg}$) show that there are statistically significant differences between HESA and the reference algorithms, in terms of the best and the average objective value (p-values < 0.05) on both Set I and Set II instances. Table 5 shows the results from the popular Wilcoxon signed-rank test recommended in [6,10] to verify whether there is a statistically significant difference between HESA and each competing algorithm. Column R^+_{best} (R^+_{avg}) denotes the sum of ranks for the instances, where HESA performs better than the compared algorithm, in terms of the best (average) objective value, while R^-_{best} (R^-_{avg}) refers to the sum of ranks for the opposite cases. Table 5 demonstrates that HESA has a significant improvement over the three compared algorithms (MA, MS-ILS and MVNS) with a level of significance $\alpha = 0.05$. These observations highlight the superiority of HESA on both Set I and Set II benchmark instances.

Table 6 presents the comparative results between HESA and the reference algorithms GA [29] and MA [7] on the real-life instance, where the results of the reference algorithms are compiled from [7]. Column "f_{bk}" reports the

Table 7
Summarized results of HESA under the short time limit of 100 generations and the long time limit of 500 generations. The best results are indicated in bold.

	Set I (48 instances)		Set II (48 instances)	
	HESA (100 Gs)	HESA (500 Gs)	HESA (100 Gs)	HESA (500 Gs)
#Best	**48**/44	**48/48**	27/14	**48/48**
Average Time(s)	**0.25**	0.74	**336.61**	584.15
Average $g_{best}/g_{avg}(\%)$	**0.74**/0.67	**0.74/0.71**	3.46/3.23	**3.55/3.32**
p-value$_{best}$	1.00		4.59e-6	
p-value$_{avg}$	2.53e-2		5.51e-9	

Table 8
Comparative results of HESA under the short time limit of 100 generations and the long time limit of 500 generations on the real-life instance.

	HESA (100 Gs)			HESA (500 Gs)		
f_{bk}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
16018.80	16642.61	16599.11	1763.87	**16712.21**	**16667.78**	5339.46

Table 9
Results of the Wilcoxon signed-rank test of HESA with the long time limit of 500 generations on the Set I and Set II benchmarks, with a significance level of 0.05.

Instance set	Comparison	R^+_{best}	R^-_{best}	p-value	R^+_{avg}	R^-_{avg}	p-value
Set I	HESA (100 Gs) vs. HESA (500 Gs)	0	0	1.00	5	0	4.31e-2
Set II	HESA (100 Gs) vs. HESA (500 Gs)	21	0	5.96e-5	34	0	3.65e-7

best-known solution from the literature. Columns "f_{best}", and "f_{avg}" show the best and average objective value across 30 independent runs, respectively. Column "t_{avg}" gives the average running time in seconds required to reach the final objective value. Table 6 indicates that HESA achieves the best performance compared to the reference algorithms, in terms of the best and average objective value, with a shorter computation time. Moreover, HESA updates the best-known solution for this real-life instance.

To sum up, the proposed HESA algorithm competes very favorably with the existing GQMKP algorithms both in terms of solution quality and computational efficiency. With a similar or shorter computation time, the algorithm is able to find equal or better solutions for a majority of the tested instances including the real-life case. The advantage of the algorithm over the other algorithms is more significant on large scale instances than on small instances. As the result, the algorithm is particularly suitable for dealing with large and practical GQMKP applications.

4.4 Comparative results with a long time limit

In this section, we check whether HESA can find still better results if it is given more time. For this, we used the same long time limit of 500 generations as [7]. Tables 7 and 8 summarize the results of HESA under a short time limit of 100 generations (HESA (100 Gs)), and a long time limit of 500 generations (HESA (500 Gs)) on the instances of Set I and Set II as well as the real-life instance, respectively. Detailed results of this experiment on the Set I and Set II benchmarks are given in Tables 16 and 17 of the Appendix.

Tables 7 and 8 demonstrate that HESA is capable of finding even better solutions if a long time limit is allowed. Particularly, in terms of the best objective value, HESA (500 Gs) yields the best result for 48 (48) cases against 48 (27) instances for HESA (100 Gs) on the Set I benchmark (Set II benchmark). In terms of the average objective value, HESA (500 Gs) dominates HESA (100 Gs) by reporting the best result for 48 (48) instances against 44 (14) instances on the Set I (Set II) instances. When considering the average percentage gap of the best/average objective value to the best-known solution, HESA (500 Gs) shows a better result for the two benchmarks (0.74%/0.71% and 3.55%/3.32%). In terms of computational time to reach the final objective value, HESA (100 Gs) requires 0.25/336.61 seconds for the the Set I/II benchmarks against 0.74/584.15 seconds for HESA (500 Gs). The nonparametric Friedman test results (rows p-value$_{best}$ and p-value$_{avg}$) indicate that there exist statistically significant differences between the compared algorithms, in terms of the best and average objective value for the Set II benchmark (p-values < 0.05). For the Set I instances, there exists a statistically significant difference between HESA (100 Gs) and HESA (500 Gs) in terms of the average objective value (p-value < 0.05), however, the difference in terms of the best objective value (p-value > 0.05) is marginal. The statistical results of the Wilcoxon signed-rank test from Table 9 further confirm the advantage of HESA (500 Gs) on the solution quality.

5 Analysis

This section provides an analysis of two key ingredients of the proposed algorithm: the knapsack-based crossover operator and the mixed search strategy of exploring both feasible and infeasible solutions. The experiments were conducted on the 15 randomly selected instances used in Section 4.2.

Table 10
Comparison results of HESA and its two variants HESA_{bb} and HESA_{uni} that employ the backbone-based crossover and the uniform crossover operator, respectively.

	HESA			HESA_{bb}			HESA_{uni}		
Ins.	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
1-1	**5176.49**	**5155.67**	957.37	5133.68	5112.53	1014.88	5167.92	5149.26	769.46
1-3	**5952.06**	**5935.10**	1211.08	5912.10	5891.46	955.53	5944.23	5917.09	1091.53
3-1	**35320.30**	**35238.30**	681.18	35190.50	34860.32	46.10	35255.50	35204.78	655.45
3-2	**43326.30**	**43281.83**	529.87	43236.60	42985.67	43.31	43322.40	43259.52	474.77
10-1	**13208.88**	**13206.60**	234.88	11320.58	10107.15	32.13	12642.19	12127.57	287.29
11-3	**11471.10**	**11404.58**	925.72	11359.70	11322.04	1038.96	11452.00	11401.74	1446.03
13-3	**4722.70**	**4706.42**	527.20	4714.00	4682.50	793.26	4715.00	4706.34	751.90
14-2	**25980.14**	**25980.14**	60.85	25976.66	25916.73	13.87	**25980.14**	**25980.14**	88.08
14-3	**31536.06**	**31536.06**	44.27	**31536.06**	31529.36	11.33	**31536.06**	**31536.06**	130.55
16-3	**14221.90**	**14129.35**	641.44	14029.60	13909.80	726.30	14213.50	14115.18	1484.03
18-2	**8551.08**	8548.01	0.10	**8551.08**	**8551.08**	0.11	**8551.08**	8548.01	0.10
23-1	**503.00**	**503.00**	322.93	**503.00**	**503.00**	199.94	**503.00**	502.87	162.03
24-1	**53814.17**	**53413.35**	1644.66	53059.93	52958.58	955.91	53426.67	53272.92	1309.30
24-2	**60199.73**	**60027.66**	1554.82	59662.11	59334.54	808.09	59908.04	59759.00	1380.31
31-2	**666.00**	**666.00**	3.86	**666.00**	**666.00**	1.23	**666.00**	**666.00**	1.27
#Best	**15**	**14**		4	3		5	3	
p-value$_F$				9.11e-4	2.28e-3		1.57e-3	9.11e-4	
Average	**20976.66**	**20915.47**	622.68	20723.44	20555.38	442.73	20885.58	20809.77	668.81

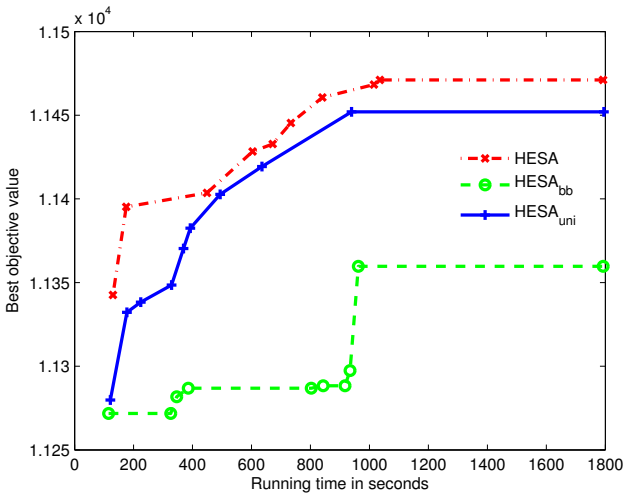
Table 11
Wilcoxon signed-rank test results of HESA and its two variants HESA_{bb} and HESA_{uni} on the 15 randomly selected instances, with a significance level of 0.05.

Comparison	R^+_{best}	R^-_{best}	p-value	R^+_{avg}	R^-_{avg}	p-value
HESA vs. HESA_{bb}	11	0	3.35e-3	12	1	1.87e-3
HESA vs. HESA_{uni}	10	0	5.06e-3	11	0	3.35e-3

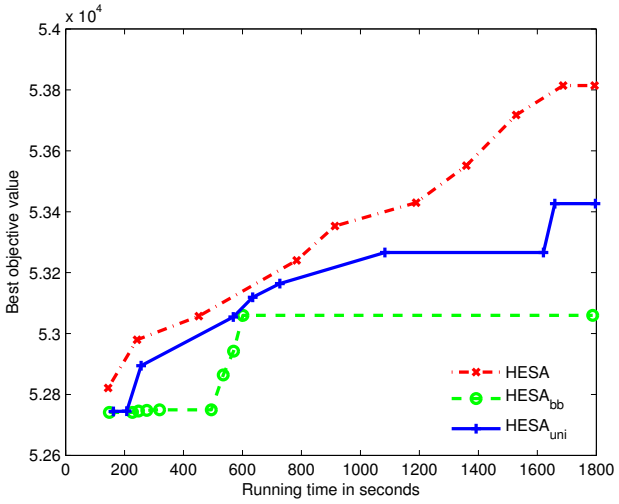
5.1 Effect of the knapsack-based crossover operator

To examine the efficiency of the knapsack-based crossover operator of the proposed algorithm, we compared HESA with two variants HESA_{bb}, and HESA_{uni} that use the backbone-based crossover [7] and the uniform crossover operators [29], respectively. Indeed, the backbone-based crossover was used in the memetic algorithm [7], while the uniform crossover was employed in the genetic algorithm [29]. These two operators are thus reasonable reference crossovers for GQMKP. For this experiment, HESA, HESA_{bb}, and HESA_{uni} were independently ran 30 runs to solve each instance of the 15 selected instances with a time limit of 1800 seconds per run.

Tables 10 and 11 present the comparative results of HESA and its two



(a) 11-3



(b) 24-1

Fig. 1. Convergence graphs of HESA and two variants HESA_{bb} and HESA_{uni} on the instances 11-3 and 24-1.

variants HESA_{bb}, and HESA_{uni}, where the symbols have the same meanings as those in previous tables. Row "p-value$_F$" reports the p-value results from the nonparametric Friedman test between HESA and the compared algorithms, in terms of the best and the average objective value. Row "Average" presents the averaged result across all the tested instances and all the executions. One observes that in terms of the best/average objective value, HESA, HESA_{bb}, and HESA_{uni} report the best result in 15/14, 4/3, and 5/3 out of 15 instances, respectively. The Wilcoxon signed-rank test further confirms the superiority of HESA with a significance level of 0.05. To illustrate the evolution of the best objective value during the search process, Fig. 1 provides the running profile (convergence graph) of HESA and the variants HESA_{bb}, and HESA_{uni}

Table 12
Comparative results of AFITS and its variant FLS that visits only feasible solutions on 15 randomly selected instances.

	AFITS			FLS		
Ins.	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
1-1	**5147.01**	**5117.39**	425.82	5128.22	5076.95	522.36
1-3	**5932.39**	**5898.53**	581.12	5915.25	5863.14	462.43
3-1	35164.80	34884.45	627.33	**35285.00**	**35070.56**	776.79
3-2	**43326.10**	**43130.08**	701.24	43277.40	43005.13	580.28
10-1	**11427.61**	**9988.28**	171.02	9696.74	8657.48	9.31
11-3	**11409.20**	**11316.60**	435.44	11344.90	11208.23	513.73
13-3	**4725.80**	**4705.03**	582.86	4714.20	4681.48	380.70
14-2	**25980.14**	**25980.14**	74.21	25951.34	25941.25	231.10
14-3	**31536.06**	**31536.06**	175.53	**31536.06**	**31536.06**	20.02
16-3	**14155.00**	**14047.20**	618.48	14121.40	13980.42	624.20
18-2	8535.72	8535.72	0.04	**8551.08**	**8551.08**	0.03
23-1	**503.00**	**503.00**	586.29	501.00	494.08	240.87
24-1	**53534.18**	**53183.14**	551.60	52966.73	52351.63	377.37
24-2	**59858.88**	**59482.29**	487.73	59516.93	58874.00	192.56
31-2	**666.00**	**666.00**	8.86	**666.00**	**666.00**	89.02
#Best	**13**	**13**		4	4	
p-value$_F$				1.26e-2	1.26e-2	
Average	**20793.46**	**20598.26**	401.84	20611.48	20397.17	334.72

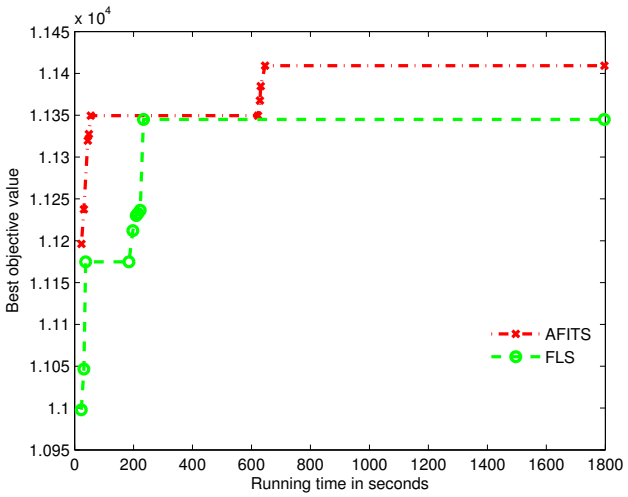
Table 13
Wilcoxon signed-rank test results of AFITS and its variant FLS on the 15 randomly selected instances, with a significance level of 0.05.

Comparison	R^+_{best}	R^-_{best}	p-value	R^+_{avg}	R^-_{avg}	p-value
AFITS vs. FLS	11	2	2.31e-2	11	2	1.92e-2

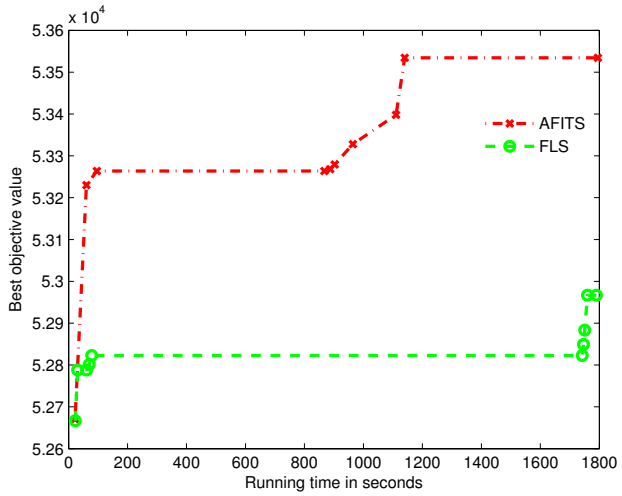
on instances 11-3, and 24-1, where X-axis and Y-axis show the running time in seconds and the best objective value, respectively. Fig. 1 shows that HESA requires less times to reach even better final solutions compared with HESA_{bb} and HESA_{uni}. Similar results are also observed for other instances. This experiment highlights the usefulness of the knapsack-based crossover operator.

5.2 Advantage of searching both feasible and infeasible solutions

To evaluate the effect of the strategy that oscillates between feasible and infeasible search areas of AFITS, an experiment was conduced to compare AFITS and a variant FLS that explores only feasible solutions during the search process. For a fair comparison, AFITS and FLS were ran 30 independent times per instance, with a time limit of 1800 seconds for each run, under the



(a) 11-3



(b) 24-1

Fig. 2. Convergence graphs of AFITS and its variant FLS on instances 11-3 and 24-1.

same experimental protocol as shown in Section 4.1.

Table 12 summarizes the comparative results. One observes that AFITS produces 13 best results against 4 best results for FLS in terms of both the best and average objective value. The nonparametric Friedman test indicates that there exists a significant difference between AFITS and FLS for both the best and average objective values (p-values < 0.05). The Wilcoxon signed-rank test results from Table 13 further indicate that AFITS performs better than FLS with a significance level of 0.05. The convergence graphs of AFITS and FLS on instances 11-3, and 24-1 provided in Fig. 2 show that AFITS requires less running times to reach even better solutions compared with FLS. This experiment clearly demonstrates that exploring both feasible and infeasible

solutions contributes to the performance of the algorithm.

6 Conclusions

The generalized quadratic multiple knapsack problem (GQMKP) is a significant member of the large class of knapsack problems. In this work, we introduced an effective hybrid evolutionary search algorithm to solve GQMKP with the following original features: (i) a knapsack-based crossover for new offspring generation; (ii) an adaptive feasible and infeasible tabu search for effective local optimization; (iii) a fast incremental technique to streamline the evaluations of candidate solutions; (iv) a dedicated strategy to ensure a diversified and high-quality initial population.

The performance of the proposed HESA algorithm was verified on the two sets of 96 commonly used benchmark instances as well as one large-scale real-life instance. The computational results showed that HESA competes very favorably with the current state-of-the-art methods in the literature. In terms of solution quality, HESA found 44 improved best-known solutions (new lower bounds) (for 9 small instances of Set I, 34 large instances of Set II and the real-life instance). HESA is also computationally effective by reaching equal or better solutions for most tested instances with a shorter time compared with the best reference algorithms. To shed lights on the impacts of the mixed feasible and infeasible search strategy and the knapsack-based crossover operator, we presented additional experiments to investigate these two key components.

In addition to the reported computational results that can be useful for future studies on GQMKP, the code of our HESA algorithm that we make publicly available can be used by researchers and practitioners to solve various problems that can be formulated as GQMKP. Moreover, the design ideas adopted in the proposed algorithm are of general nature, they can be adapted to design effective algorithms for other related problems.

Given that the proposed algorithm is a heuristic approach, the gap between the reported solutions and the optimal solutions cannot be determined. Therefore, additional research is needed to study exact and approximate methods with quality guarantee. In addition, to further improve the search efficiency of the algorithm, other neighborhoods could be considered to complement the reallocate and the swap neighborhoods used in GQMKP studies. Finally, the proposed algorithm requires several parameters whose tuning may be fastidious. It would be interesting to study self-adaptive approaches that can dynamically adapt the parameters to the search landscape under examination.

Acknowledgments

We are grateful to the reviewers for their valuable comments and suggestions which helped us to improve the paper. We also thank Dr. Yuning Chen [7] and Dr. Tugba Saraç [29] for sharing the problem instances.

References

[1] Adouani, Y., Jarboui, B., & Masmoudi, M. (2019). A matheuristic for the 0–1 generalized quadratic multiple knapsack problem. Optimization Letters, in press, https://doi.org/10.1007/s11590-019-01503-z

[2] Avci, M., & Topaloglu, S. (2017). A multi-start iterated local search algorithm for the generalized quadratic multiple knapsack problem. Computers & Operations Research, 83, 54-65.

[3] Benlic, U., & Hao, J. K. (2011). A multilevel memetic approach for improving graph K-partitions. IEEE Transactions on Evolutionary Computation, 15(5), 624-642.

[4] Benlic, U., & Hao, J. K. (2015). Memetic search for the quadratic assignment problem. Expert Systems with Applications, 42(1), 584-595.

[5] Birattari, M., Yuan, Z., Balaprakash, P., & Stützle, T. (2010). F-race and iterated F-race: an overview. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., & Preuss, M. (Eds.), Experimental Methods for the Analysis of Optimization Algorithms (pp. 311-336). Berlin: Springer.

[6] Carrasco, J., García, S., Rueda, M. M., Das, S., & Herrera, F. (2020). Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review. Swarm and Evolutionary Computation, 54, 100665.

[7] Chen, Y. N., & Hao, J. K. (2016). Memetic search for the generalized quadratic multiple knapsack problem. IEEE Transactions on Evolutionary Computation, 20(6), 908-923.

[8] Chen, Y. N., & Hao, J. K. (2017). An iterated "hyperplane exploration" approach for the quadratic knapsack problem. Computers & Operations Research, 77, 226-239.

[9] Dell'Amico, M., Delorme, M., Iori M., & Martello, S. (2018). Mathematical models and decomposition methods for the multiple knapsack problem. European Journal of Operational Research, 274(3), 886-899.

[10] Derrac, J., García, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation, 1(1), 3-18.

[11] Detti, P. (2021). A new upper bound for the multiple knapsack problem. Computers & Operations Research, 129, 105210.

[12] Galli, L., Martello, S., Rey, C., & Toth, P. (2020). Polynomial-size formulations and relaxations for the quadratic multiple knapsack problem. European Journal of Operational Research, 291(3), 871-882.

[13] Glover, F., & Hao, J. K. (2011). The case for strategic oscillation. Annals of Operations Research, 183(1), 163-173.

[14] Glover, F., & Laguna, M. (1998). Tabu search. In: DU, DZ., & Pardalos, P. M. (Eds.), Handbook of Combinatorial Optimization (pp. 2093-2229). Boston: Springer.

[15] Hao, J. K. (2012). Memetic algorithms in discrete optimization. In: Neri, F., Cotta, C., & Moscato, P. (Eds.), Handbook of Memetic Algorithms, Studies in Computational Intelligence, Vol 379 (pp. 73-94). Berlin: Springer.

[16] Hiley, A., & Julstrom, B. A. (2006). The quadratic multiple knapsack problem and three heuristic approaches to it. In Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (pp. 547-552). Washington: ACM.

[17] Ishibuchi, H., Hitotsuyanagi, Y., Tsukamoto, N., & Nojima, Y. (2009). Implementation of multiobjective memetic algorithms for combinatorial optimization problems: a knapsack problem case study. In: Goh, CK., Ong, YS., & Tan, K. C. (Eds.), Multi-Objective Memetic Algorithms, Studies in Computational Intelligence, vol 171 (pp. 27-49). Berlin: Springer.

[18] Jin, Y., & Hao, J. K. (2019). Solving the Latin square completion problem by memetic graph coloring. IEEE Transactions on Evolutionary Computation, 23(6), 1015-1028.

[19] López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: iterated racing for automatic algorithm configuration. Operations Research Perspectives, 3, 43-58.

[20] Lu, Z., Hao, J. K., & Wu, Q. H. (2020). A hybrid evolutionary algorithm for finding low conductance of large graphs. Future Generation Computer Systems, 106, 105-120.

[21] Lu, Z., Zhou, Y., & Hao, J. K. (2021). A hybrid evolutionary algorithm for the clique partitioning problem. IEEE Transactions on Cybernetics, in press, https://doi.org/10.1109/TCYB.2021.3051243

[22] Martello, S., & Toth, P. (1980). Solution of the zero-one multiple knapsack problem. European Journal of Operational Research, 4(4), 276-283.

[23] Martinelli, R., Poggi, M., & Subramanian, A. (2013). Improved bounds for large scale capacitated arc routing problem. Computers & Operations Research, 40(8), 2145-2160.

[24] Moscato, P. (1999). Memetic algorithms: A short introduction. In New ideas in optimization (pp. 219-234). McGraw-Hill Ltd., UK.

[25] Neri, F., Cotta, C., & Moscato, P. (2012). Handbook of Memetic Algorithms, Studies in Computational Intelligence (Vol 379). Berlin: Springer.

[26] Paraskevopoulos, D. C., Laporte, G., Repoussis, P. P., & Tarantilis, C. D. (2017). Resource constrained routing and scheduling: review and research prospects. European Journal of Operational Research, 263(3), 737-754.

[27] Qin, J., Xu, X. H., Wu, Q. H., & Cheng, T. C. E. (2016). Hybridization of tabu search with feasible and infeasible local searches for the quadratic multiple knapsack problem. Computers & Operations Research, 66, 199-214.

[28] Salkin, H. M., & Kluyver, C. D. (1975). The knapsack problem: a survey. Naval Research Logistics Quarterly, 22(1), 127-144.

[29] Saraç, T., & Sipahioglu, A. (2014). Generalized quadratic multiple knapsack problem and two solution approaches. Computers & Operations Research, 43, 78-89.

[30] Silva, A., Coelho, L. C., & Darvish, M. (2021). Quadratic assignment problem variants: A survey and an effective parallel memetic iterated tabu search. European Journal of Operational Research, 292(3), 1066-1084.

[31] Sun, W., Hao, J. K., Lai, X. J., & Wu, Q. H. (2018). Adaptive feasible and infeasible tabu search for weighted vertex coloring. Information Sciences, 466, 203-219.

[32] Usberti, F. L., França, P. M., & França, A. L. M. (2013). GRASP with evolutionary path-relinking for the capacitated arc routing problem. Computers & Operations Research, 40(12), 3206-3217.

[33] Zhang, H. Z., Liu, F., Zhou, Y. Y., & Zhang, Z. Y. (2020). A hybrid method integrating an elite genetic algorithm with tabu search for the quadratic assignment problem. Information Sciences, 539, 347-374.

[34] Zhou, Y., Hao, J. K., & Glover, F. (2019). Memetic search for identifying critical nodes in sparse graphs. IEEE Transactions on Cybernetics 49(10), 3699-3712.

Appendix

This appendix provides 1) detailed comparative results between HESA and the state-of-the-art methods in the literature on the 48 small-sized instances of Set I and 48 large-sized instances of Set II under the condition indicated in Section 4.3 (Tables 14 and 15), and 2) comparative results of HESA under a short time limit of 100 generations and a long time limit of 500 generations on the instances of Set I and Set II (Tables 16 and 17). In each table,

column "Ins." indicates the instance name, and column "f_{bk}" reports the best-known solution from the literature. Columns "f_{best}", "f_{avg}", and "t_{avg}" show for each algorithm the best objective value, the average objective value and the average running time in seconds to reach the final objective value across 30 independent runs, respectively. Row "#Best" indicates the number of instances for which the corresponding algorithm reports the best result among all the compared algorithms, in terms of the best and average objective value. Row "#Improve" ("#Match") denotes the number of cases that an algorithm improves (matches) the best-known solution in the literature. The best results are marked in bold.

Table 14. Comparative results between HESA and three state-of-the-art algorithms (MA [7], MS-ILS [2] and MVNS [1]) on the 48 small-sized instances of Set I (part 1).

		MA			MS-ILS			MVNS			HESA		
Ins.	f_{bk}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
5-1	**2835.30**	**2835.30**	2828.22	1.23	**2835.30**	**2835.30**	3.24	**2835.30**	**2835.30**	2.23	**2835.30**	**2835.30**	0.03
5-2	**3304.80**	3293.48	3293.90	0.83	**3304.80**	3293.48	1.65	**3304.80**	**3304.80**	2.04	**3304.80**	**3304.80**	0.06
5-3	1678.00	1678.00	1678.00	0.01	1678.00	1678.00	3.17	1678.00	1678.00	1.86	**1756.30**	**1756.30**	0.03
6-1	**346.40**	**346.40**	**346.40**	0.01	**346.40**	**346.40**	2.48	**346.40**	**346.40**	2.25	**346.40**	**346.40**	0.03
6-2	**554.00**	**554.00**	**554.00**	0.01	**554.00**	**554.00**	1.41	**554.00**	**554.00**	2.09	**554.00**	**554.00**	0.02
6-3	**428.70**	**428.70**	**428.70**	0.01	**428.70**	**428.70**	1.75	**428.70**	**428.70**	1.16	**428.70**	**428.70**	0.03
8-1	309.21	309.21	309.21	0.91	309.21	309.21	2.25	309.21	309.21	2.02	**318.01**	**317.38**	0.74
8-2	353.85	353.85	353.69	0.11	353.85	353.85	2.97	353.85	353.85	2.09	**374.30**	**374.30**	0.13
8-3	541.57	541.57	541.57	0.03	541.57	541.57	2.85	541.57	541.57	2.25	**556.57**	**556.57**	0.50
15-1	**91.54**	**91.54**	**91.54**	0.32	**91.54**	**91.54**	1.60	**91.54**	**91.54**	2.22	**91.54**	**91.54**	0.03
15-2	**306.38**	**306.38**	**306.38**	0.02	**306.38**	**306.38**	2.85	**306.38**	**306.38**	2.13	**306.38**	**306.38**	0.03
15-3	**75.62**	**75.62**	75.45	0.37	**75.62**	**75.62**	2.77	**75.62**	**75.62**	1.38	**75.62**	**75.62**	0.03
18-1	**5387.70**	**5387.70**	**5387.70**	0.01	**5387.70**	**5387.70**	2.11	**5387.70**	**5387.70**	2.03	**5387.70**	**5387.70**	0.04
18-2	**8551.08**	**8551.08**	**8551.08**	0.00	**8551.08**	**8551.08**	3.03	**8551.08**	**8551.08**	1.29	**8551.08**	8548.01	0.11
18-3	**7760.51**	**7760.51**	**7760.51**	0.00	**7760.51**	**7760.51**	1.43	**7760.51**	**7760.51**	1.05	**7760.51**	**7760.51**	0.05
20-1	**1599.85**	**1599.85**	**1599.85**	0.01	**1599.85**	**1599.85**	1.99	**1599.85**	**1599.85**	1.02	**1599.85**	**1599.85**	0.04
20-2	**925.59**	**925.59**	**925.59**	0.01	**925.59**	**925.59**	2.81	**925.59**	**925.59**	1.55	**925.59**	**925.59**	0.04
20-3	**931.33**	**931.33**	**931.33**	0.01	**931.33**	**931.33**	2.83	**931.33**	**931.33**	1.42	**931.33**	**931.33**	0.06
22-1	**1923.61**	1904.86	1904.86	0.02	**1923.61**	1911.11	3.30	**1923.61**	**1923.61**	2.03	**1923.61**	**1923.61**	0.03
22-2	**1314.09**	**1314.09**	**1314.09**	0.01	**1314.09**	**1314.09**	1.33	**1314.09**	**1314.09**	2.15	**1314.09**	**1314.09**	0.04
22-3	**1799.09**	**1799.09**	**1799.09**	0.02	**1799.09**	**1799.09**	2.04	**1799.09**	**1799.09**	1.13	**1799.09**	**1799.09**	0.03
23-1	471.00	471.00	471.00	0.02	471.00	471.00	2.53	471.00	471.00	2.05	**503.00**	**496.08**	1.47
23-2	959.70	959.70	959.70	0.06	959.70	959.70	1.02	959.70	959.70	2.99	**989.70**	**983.43**	2.41
23-3	**1241.00**	**1241.00**	**1241.00**	0.32	**1241.00**	**1241.00**	1.20	**1241.00**	**1241.00**	2.81	**1241.00**	1231.66	3.22
25-1	**2118.33**	**2118.33**	**2118.33**	1.52	**2118.33**	**2118.33**	1.34	**2118.33**	**2118.33**	1.05	**2118.33**	**2118.33**	0.09
25-2	**4262.64**	**4262.64**	4195.05	1.66	**4262.64**	4193.01	1.12	**4262.64**	**4262.64**	0.98	**4262.64**	**4262.64**	0.07
25-3	**2962.06**	**2962.06**	**2962.06**	1.03	**2962.06**	**2962.06**	1.02	**2962.06**	**2962.06**	1.25	**2962.06**	**2962.06**	0.04

Table 14. Comparative results between HESA and three state-of-the-art algorithms (MA [7], MS-ILS [2] and MVNS [1]) on the 48 small-sized instances of Set I (part 2).

		MA			MS-ILS			MVNS			HESA		
Ins.	f_{bk}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
26-1	**1747.60**	**1747.60**	**1747.60**	0.01	**1747.60**	**1747.60**	2.66	**1747.60**	**1747.60**	0.93	**1747.60**	**1747.60**	0.04
26-2	**2433.60**	**2433.60**	**2433.60**	0.01	**2433.60**	**2433.60**	1.20	**2433.60**	**2433.60**	1.17	**2433.60**	**2433.60**	0.06
26-3	**2293.20**	**2293.20**	**2293.20**	0.01	**2293.20**	**2293.20**	1.32	**2293.20**	**2293.20**	2.07	**2293.20**	**2293.20**	0.04
27-1	**2247.95**	**2247.95**	**2247.95**	0.01	**2247.95**	**2247.95**	2.76	**2247.95**	**2247.95**	1.99	**2247.95**	**2247.95**	0.02
27-2	**1966.52**	**1966.52**	**1966.52**	0.01	**1966.52**	**1966.52**	1.05	**1966.52**	**1966.52**	2.14	**1966.52**	**1966.52**	0.03
27-3	**1383.49**	**1383.49**	**1383.49**	0.01	**1383.49**	**1383.49**	1.09	**1383.49**	**1383.49**	1.86	**1383.49**	**1383.49**	0.04
28-1	**978.80**	**978.80**	978.07	0.12	**978.80**	**978.80**	2.67	**978.80**	**978.80**	1.23	**978.80**	**978.80**	0.04
28-2	**4036.00**	**4036.00**	4035.62	0.04	**4036.00**	**4036.00**	1.12	**4036.00**	**4036.00**	1.11	**4036.00**	**4036.00**	0.03
28-3	**2634.00**	**2634.00**	**2634.00**	0.01	**2634.00**	**2634.00**	1.14	**2634.00**	**2634.00**	1.19	**2634.00**	**2634.00**	0.03
29-1	**1935.80**	1567.60	1520.33	0.19	**1935.80**	**1935.80**	2.46	**1935.80**	**1935.80**	2.01	**1935.80**	**1935.80**	0.03
29-2	**2820.00**	2782.00	2782.00	0.10	**2820.00**	**2820.00**	1.08	**2820.00**	**2820.00**	1.64	**2820.00**	**2820.00**	0.05
29-3	**3285.60**	**3285.60**	**3285.60**	0.05	**3285.60**	**3285.60**	1.03	**3285.60**	**3285.60**	1.45	**3285.60**	**3285.60**	0.04
30-1	**721.39**	**721.39**	717.27	0.40	**721.39**	719.58	2.47	**721.39**	**721.39**	2.01	**721.39**	**721.39**	0.05
30-2	**612.59**	**612.59**	**612.59**	0.03	**612.59**	**612.59**	1.02	**612.59**	**612.59**	1.89	**612.59**	**612.59**	0.06
30-3	**1032.35**	**1032.35**	**1032.35**	0.04	**1032.35**	1031.94	1.88	**1032.35**	**1032.35**	1.55	**1032.35**	**1032.35**	0.10
31-1	491.90	491.90	491.90	1.52	491.90	491.90	1.98	491.90	491.90	3.11	**507.90**	**507.90**	0.57
31-2	640.00	640.00	640.00	0.49	640.00	640.00	1.21	640.00	640.00	2.05	**666.00**	**665.20**	0.97
31-3	526.10	526.10	526.10	5.37	526.10	526.10	1.16	526.10	526.10	2.04	**538.40**	**538.40**	0.41
32-1	**11425.20**	**11425.20**	11271.90	0.02	**11425.20**	11283.21	2.61	**11425.20**	11393.75	2.53	**11425.20**	**11425.20**	0.05
32-2	**15914.20**	**15914.20**	**15914.20**	0.00	**15914.20**	**15914.20**	1.03	**15914.20**	**15914.20**	1.13	**15914.20**	**15914.20**	0.04
32-3	**19273.50**	**19273.50**	**19273.50**	0.00	**19273.50**	**19273.50**	1.09	**19273.50**	**19273.50**	1.25	**19273.50**	**19273.50**	0.04
#Best		35	28		39	33		39	38		**48**	**46**	
#Improve		0			0			0			9		
#Match		44			48			48			39		

Table 15. Comparative results between HESA and three state-of-the-art algorithms (MA [7], MS-ILS [2] and MVNS [1]) on the 48 large-sized instances of Set II (part 1).

		MA			MS-ILS			MVNS			HESA		
Ins.	f_{bk}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
1-1	5107.80	5093.06	5074.50	7419.16	5100.54	5016.82	3793.42	5107.80	5102.17	225.23	**5169.59**	**5142.55**	656.64
1-2	4889.58	4848.58	4830.20	8101.91	4858.84	4784.07	4493.57	4889.58	**4889.58**	295.09	**4909.03**	4884.49	605.44
1-3	5902.86	5896.01	5876.05	6823.41	5902.86	5823.73	4710.79	5902.86	5898.23	259.01	**5940.14**	**5920.00**	646.45
2-1	2608.12	2607.84	2601.31	3530.13	2608.12	2557.22	5175.50	2608.12	2601.20	443.04	**2608.61**	**2601.42**	217.92
2-2	**2285.32**	**2285.32**	2281.63	3570.48	2257.88	2249.62	3925.30	**2285.32**	**2285.32**	525.11	2282.16	2276.14	216.35
2-3	**2580.62**	2578.14	2573.40	2946.75	**2580.62**	2574.96	3464.23	**2580.62**	**2577.08**	390.03	2579.45	2573.31	223.07
3-1	32216.20	32189.10	32147.30	2693.57	32210.80	32163.74	1734.37	32216.20	32210.32	307.98	**35320.30**	**35222.61**	395.27
3-2	40354.90	40302.40	40169.70	1437.15	40354.90	40239.63	2399.47	40354.90	40354.90	523.02	**43306.60**	**43276.31**	340.27
3-3	32772.40	32766.70	32749.40	3414.05	32768.20	32704.85	4220.01	32772.40	32772.40	309.15	**35990.90**	**35900.85**	379.35
4-1	9048.40	9045.80	9027.86	4323.70	9048.40	9029.01	2720.37	9048.40	9048.40	223.06	**9074.70**	**9066.59**	520.75
4-2	**8468.50**	8465.00	8448.00	4871.10	**8468.50**	8425.58	2207.61	**8468.50**	8459.36	255.03	8467.60	**8465.77**	440.30
4-3	8497.20	8491.30	8475.10	4467.05	8494.20	8450.93	2058.68	8497.20	8490.23	189.45	**8504.20**	**8492.15**	495.77
7-1	**68165.50**	68129.00	68029.40	3314.59	**68165.50**	68060.77	1669.41	**68165.50**	68160.81	301.09	**68165.50**	**68161.32**	163.46
7-2	**65643.50**	65616.80	65546.20	2542.84	**65643.50**	65559.26	1943.83	**65643.50**	**65643.50**	208.01	**65643.50**	**65643.50**	178.78
7-3	69440.90	69397.60	69279.30	3104.20	69440.90	69295.39	2304.84	69440.90	**69440.90**	205.21	**69482.10**	69423.92	179.43
9-1	**9256.47**	9252.47	9242.60	1485.96	**9256.47**	9245.74	3262.62	**9256.47**	**9256.47**	558.82	**9256.47**	9255.00	115.15
9-2	13013.20	13007.30	12988.90	3120.53	13009.08	12943.85	3215.71	13013.20	13013.20	398.86	**13053.61**	**13018.26**	305.48
9-3	16385.97	16372.00	16359.20	2822.24	16385.97	16364.29	3547.43	16385.97	16385.97	480.20	**16398.19**	**16397.05**	332.00
10-1	**13214.66**	13196.30	13125.80	3761.90	**13214.66**	11147.24	5078.34	**13214.66**	**13214.66**	508.11	13208.88	13205.06	145.96
10-2	13015.08	13003.30	12779.20	3796.61	13015.08	11209.88	5482.66	13015.08	13015.08	514.20	**13066.20**	**13066.20**	171.40
10-3	13068.47	13057.00	13008.60	4114.58	13068.47	11672.09	4046.23	13068.47	13068.47	397.03	**13079.47**	**13079.25**	163.02
11-1	7121.90	7116.50	7103.55	711.63	7121.90	7108.17	2536.74	7121.90	7121.90	306.05	**9189.30**	**9148.06**	517.42
11-2	6774.70	6771.50	6758.19	537.44	6774.70	6760.15	2546.20	6774.70	6774.70	414.89	**11496.70**	**11441.58**	578.77
11-3	7747.10	7745.10	7726.96	911.53	7747.10	7705.36	2952.58	7747.10	7747.10	262.40	**11446.40**	**11384.61**	591.70
12-1	59592.00	59234.10	59137.70	6140.09	59592.00	59381.42	4665.91	59592.00	**59592.00**	301.17	**59631.00**	59522.59	460.96
12-2	61737.40	61489.70	61181.70	4800.17	61725.20	61449.70	4163.09	61737.40	**61730.08**	360.23	**62154.10**	61695.47	437.63
12-3	61165.70	60899.30	60749.70	5156.66	61165.70	60988.89	4541.59	61165.70	61165.70	489.20	**61379.70**	**61250.17**	366.86

Table 15. Comparative results between HESA and three state-of-the-art algorithms (MA [7], MS-ILS [2] and MVNS [1]) on the 48 large-sized instances of Set II (part 2).

		MA			MS-ILS			MVNS			HESA		
Ins.	f_{bk}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
13-1	4210.10	4210.10	4194.59	1901.71	4196.20	4132.31	1176.32	4207.20	4198.17	190.13	**4212.10**	**4212.03**	241.43
13-2	4139.90	4139.90	4136.01	1318.51	4119.60	4086.26	1181.03	4139.90	**4139.90**	201.25	**4145.60**	4126.51	265.25
13-3	**4745.10**	4734.90	4717.27	2123.22	4722.10	4666.32	1160.96	**4745.10**	**4742.85**	186.17	4722.70	4700.99	177.00
14-1	26868.60	26868.60	26868.60	4.61	26868.60	26868.60	1416.34	26868.60	26868.60	213.07	**26920.15**	**26920.15**	9.53
14-2	25976.20	25929.60	25720.00	304.66	25885.67	25743.61	1290.07	25976.20	25972.27	287.06	**25980.14**	**25979.72**	58.91
14-3	31448.20	31448.20	31448.20	301.21	31448.20	31444.55	1353.25	31448.20	31448.20	266.52	**31536.06**	**31535.83**	40.42
16-1	14166.80	14129.10	14060.90	1932.42	14166.80	14086.34	1011.32	14166.80	**14166.80**	198.03	**14207.30**	14065.21	429.43
16-2	16612.40	16611.90	16577.60	1634.65	16612.40	16582.07	811.82	16612.40	16612.40	183.06	**16643.90**	**16626.78**	337.77
16-3	**14251.00**	14240.80	14210.10	2391.58	**14251.00**	14230.64	940.71	**14251.00**	**14251.00**	192.71	14187.80	14108.88	360.80
17-1	**4157.20**	**4157.20**	4147.09	1221.69	**4157.20**	4149.12	3242.94	**4157.20**	**4157.20**	198.65	**4157.20**	**4157.20**	68.15
17-2	**3911.00**	3901.30	3891.48	1500.73	3892.00	3881.10	3009.30	**3911.00**	**3911.00**	103.33	3901.30	3899.37	92.01
17-3	**3767.70**	**3767.70**	3767.67	1444.65	3756.80	3744.68	3460.16	**3767.70**	**3767.70**	196.13	**3767.70**	**3767.70**	94.32
19-1	6873.07	6869.80	6866.33	843.08	6873.07	6853.02	1641.79	6873.07	6865.86	153.69	**6877.74**	**6875.31**	173.06
19-2	8042.79	8028.54	7831.85	1847.38	8042.79	7888.13	2137.21	8042.79	**8042.79**	201.06	**8043.47**	8033.44	157.89
19-3	**8155.05**	**8155.05**	8154.87	1410.02	8142.84	8131.62	1724.09	**8155.05**	**8155.05**	113.20	**8155.05**	**8155.05**	194.56
21-1	22230.23	22221.90	22187.00	7570.70	22210.23	22121.54	3100.56	22230.23	22225.07	167.23	**22240.40**	**22225.67**	333.72
21-2	25266.50	25254.50	25199.70	5544.17	25266.50	25165.91	3867.18	25266.50	25260.06	200.03	**25364.93**	**25309.08**	288.56
21-3	**28593.40**	24574.10	24541.20	8984.81	28565.63	27847.15	3914.01	**28593.40**	**28589.17**	188.14	24612.36	24589.88	303.04
24-1	53320.50	52652.70	52253.30	91.59	53318.76	53173.81	3103.33	53320.50	53312.35	203.48	**53470.20**	**53350.10**	903.00
24-2	59723.20	57771.60	57513.40	2868.01	59712.09	59169.73	3190.82	59723.20	59719.32	203.60	**60010.86**	**59812.06**	877.93
24-3	53073.72	52642.70	52361.60	77.34	53073.72	52929.65	3456.26	53073.72	53073.72	290.73	**53668.12**	**53433.89**	904.91
#Best		4	0		8	0		14	19		**40**	**33**	
#Improve		0			0			0			34		
#Match		8			29			47			6		

Table 16
Comparison results of HESA under a short time limit of 100 generations and a long time limit of 500 generations on the 48 small-sized instances of Set I.

		HESA (100 Gs)			HESA (500 Gs)		
Ins.	f_{bk}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
5-1	**2835.30**	**2835.30**	**2835.30**	0.03	**2835.30**	**2835.30**	0.03
5-2	**3304.80**	**3304.80**	**3304.80**	0.06	**3304.80**	**3304.80**	0.06
5-3	1678.00	**1756.30**	**1756.30**	0.03	**1756.30**	**1756.30**	0.03
6-1	**346.40**	**346.40**	**346.40**	0.03	**346.40**	**346.40**	0.03
6-2	**554.00**	**554.00**	**554.00**	0.02	**554.00**	**554.00**	0.02
6-3	**428.70**	**428.70**	**428.70**	0.03	**428.70**	**428.70**	0.02
8-1	309.21	**318.01**	317.38	0.74	**318.01**	**317.53**	3.86
8-2	353.85	**374.30**	**374.30**	0.13	**374.30**	**374.30**	0.14
8-3	541.57	**556.57**	**556.57**	0.50	**556.57**	**556.57**	0.47
15-1	**91.54**	**91.54**	**91.54**	0.03	**91.54**	**91.54**	0.03
15-2	**306.38**	**306.38**	**306.38**	0.03	**306.38**	**306.38**	0.03
15-3	**75.62**	**75.62**	**75.62**	0.03	**75.62**	**75.62**	0.03
18-1	**5387.70**	**5387.70**	**5387.70**	0.04	**5387.70**	**5387.70**	0.04
18-2	**8551.08**	**8551.08**	**8548.01**	0.11	**8551.08**	**8548.01**	0.11
18-3	**7760.51**	**7760.51**	**7760.51**	0.05	**7760.51**	**7760.51**	0.06
20-1	**1599.85**	**1599.85**	**1599.85**	0.04	**1599.85**	**1599.85**	0.04
20-2	**925.59**	**925.59**	**925.59**	0.04	**925.59**	**925.59**	0.04
20-3	**931.33**	**931.33**	**931.33**	0.06	**931.33**	**931.33**	0.06
22-1	**1923.61**	**1923.61**	**1923.61**	0.03	**1923.61**	**1923.61**	0.03
22-2	**1314.09**	**1314.09**	**1314.09**	0.04	**1314.09**	**1314.09**	0.04
22-3	**1799.09**	**1799.09**	**1799.09**	0.03	**1799.09**	**1799.09**	0.03
23-1	471.00	**503.00**	496.08	1.47	**503.00**	**496.68**	3.69
23-2	959.70	**989.70**	983.43	2.41	**989.70**	**989.30**	11.62
23-3	**1241.00**	**1241.00**	1231.66	3.22	**1241.00**	**1240.33**	10.13
25-1	**2118.33**	**2118.33**	**2118.33**	0.09	**2118.33**	**2118.33**	0.08
25-2	**4262.64**	**4262.64**	**4262.64**	0.07	**4262.64**	**4262.64**	0.06
25-3	**2962.06**	**2962.06**	**2962.06**	0.04	**2962.06**	**2962.06**	0.04
26-1	**1747.60**	**1747.60**	**1747.60**	0.04	**1747.60**	**1747.60**	0.04
26-2	**2433.60**	**2433.60**	**2433.60**	0.06	**2433.60**	**2433.60**	0.06
26-3	**2293.20**	**2293.20**	**2293.20**	0.04	**2293.20**	**2293.20**	0.05
27-1	**2247.95**	**2247.95**	**2247.95**	0.02	**2247.95**	**2247.95**	0.03
27-2	**1966.52**	**1966.52**	**1966.52**	0.03	**1966.52**	**1966.52**	0.04
27-3	**1383.49**	**1383.49**	**1383.49**	0.04	**1383.49**	**1383.49**	0.04
28-1	**978.80**	**978.80**	**978.80**	0.04	**978.80**	**978.80**	0.04
28-2	**4036.00**	**4036.00**	**4036.00**	0.03	**4036.00**	**4036.00**	0.03
28-3	**2634.00**	**2634.00**	**2634.00**	0.03	**2634.00**	**2634.00**	0.03
29-1	**1935.80**	**1935.80**	**1935.80**	0.03	**1935.80**	**1935.80**	0.03
29-2	**2820.00**	**2820.00**	**2820.00**	0.05	**2820.00**	**2820.00**	0.04
29-3	**3285.60**	**3285.60**	**3285.60**	0.04	**3285.60**	**3285.60**	0.05
30-1	**721.39**	**721.39**	**721.39**	0.05	**721.39**	**721.39**	0.04
30-2	**612.59**	**612.59**	**612.59**	0.06	**612.59**	**612.59**	0.06
30-3	**1032.35**	**1032.35**	**1032.35**	0.10	**1032.35**	**1032.35**	0.09
31-1	491.90	**507.90**	**507.90**	0.57	**507.90**	**507.90**	0.55
31-2	640.00	**666.00**	665.20	0.97	**666.00**	**665.87**	2.80
31-3	526.10	**538.40**	**538.40**	0.41	**538.40**	**538.40**	0.39
32-1	**11425.20**	**11425.20**	**11425.20**	0.05	**11425.20**	**11425.20**	0.06
32-2	**15914.20**	**15914.20**	**15914.20**	0.04	**15914.20**	**15914.20**	0.04
32-3	**19273.50**	**19273.50**	**19273.50**	0.04	**19273.50**	**19273.50**	0.04
#Best		48	44		48	48	
#Improve		9			9		
#Match		39			39		

Table 17
Comparison results of HESA under a short time limit of 100 generations and a long time limit of 500 generations on the 48 large-sized instances of Set II.

		HESA (100 Gs)			HESA (500 Gs)		
Ins.	f_{bk}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
1-1	5107.80	5169.59	5142.55	656.64	**5176.49**	**5156.50**	1086.81
1-2	4889.58	4909.03	4884.49	605.44	**4926.24**	**4900.18**	1336.93
1-3	5902.86	5940.14	5920.00	646.45	**5952.06**	**5935.80**	1281.99
2-1	2608.12	2608.61	2601.42	217.92	**2611.19**	**2602.09**	538.31
2-2	2285.32	2282.16	2276.14	216.35	**2293.89**	**2281.91**	437.27
2-3	2580.62	2579.45	2573.31	223.07	**2583.64**	**2574.91**	339.97
3-1	32216.20	**35320.30**	35222.61	395.27	**35320.30**	**35238.30**	703.23
3-2	40354.90	43306.60	43276.31	340.27	**43326.30**	**43282.06**	579.85
3-3	32772.40	35990.90	35900.85	379.35	**35996.80**	**35915.76**	622.82
4-1	9048.40	**9074.70**	9066.59	520.75	**9074.70**	**9068.58**	613.76
4-2	**8468.50**	8467.60	8465.77	440.30	8467.60	**8466.46**	535.99
4-3	8497.20	**8504.20**	**8492.15**	495.77	**8504.20**	**8492.15**	535.83
7-1	**68165.50**	**68165.50**	68161.32	163.46	**68165.50**	**68164.72**	180.12
7-2	**65643.50**	**65643.50**	**65643.50**	178.78	**65643.50**	**65643.50**	212.81
7-3	69440.90	**69482.10**	**69423.92**	179.43	**69482.10**	**69423.92**	158.23
9-1	**9256.47**	**9256.47**	**9255.00**	115.15	**9256.47**	**9255.00**	107.28
9-2	13013.20	**13053.61**	13018.26	305.48	**13053.61**	**13022.58**	313.65
9-3	16385.97	**16398.19**	**16397.05**	332.00	**16398.19**	**16397.05**	322.61
10-1	**13214.66**	13208.88	**13205.06**	145.96	13208.88	**13205.06**	150.72
10-2	13015.08	**13066.20**	**13066.20**	171.40	**13066.20**	**13066.20**	156.45
10-3	13068.47	**13079.47**	**13079.25**	163.02	**13079.47**	**13079.25**	159.77
11-1	7121.90	9189.30	9148.06	517.42	**9194.60**	**9162.40**	899.51
11-2	6774.70	11496.70	11441.58	578.77	**11501.40**	**11460.98**	938.16
11-3	7747.10	11446.40	11384.61	591.70	**11471.10**	**11404.83**	1003.03
12-1	59592.00	59631.00	59522.59	460.96	**59643.00**	**59547.94**	870.19
12-2	61737.40	62154.10	61695.47	437.63	**62190.20**	**61804.70**	767.83
12-3	61165.70	61379.70	61250.17	366.86	**61478.90**	**61262.68**	752.29
13-1	4210.10	**4212.10**	**4212.03**	241.43	**4212.10**	**4212.03**	272.16
13-2	4139.90	4145.60	4126.51	265.25	**4149.60**	**4129.82**	562.04
13-3	**4745.10**	4722.70	4700.99	177.00	4722.70	**4706.24**	492.17
14-1	26868.60	**26920.15**	**26920.15**	9.53	**26920.15**	**26920.15**	9.41
14-2	25976.20	**25980.14**	25979.72	58.91	**25980.14**	**25980.14**	63.13
14-3	31448.20	**31536.06**	31535.83	40.42	**31536.06**	**31536.06**	47.03
16-1	14166.80	**14207.30**	14065.21	429.43	**14207.30**	**14071.73**	607.83
16-2	16612.40	**16643.90**	16626.78	337.77	**16643.90**	**16629.38**	715.08
16-3	14251.00	14187.80	14108.88	360.80	**14272.90**	**14131.05**	700.10
17-1	**4157.20**	**4157.20**	**4157.20**	68.15	**4157.20**	**4157.20**	65.98
17-2	**3911.00**	3901.30	3899.37	92.01	3901.30	**3899.93**	186.95
17-3	**3767.70**	**3767.70**	**3767.70**	94.32	**3767.70**	**3767.70**	102.76
19-1	6873.07	**6877.74**	6875.31	173.06	**6877.74**	**6875.54**	347.36
19-2	8042.79	**8043.47**	8033.44	157.89	**8043.47**	**8034.03**	202.88
19-3	**8155.05**	**8155.05**	**8155.05**	194.56	**8155.05**	**8155.05**	231.39
21-1	22230.23	22240.40	22225.67	333.72	**22242.40**	**22225.80**	471.63
21-2	25266.50	25364.93	25309.08	288.56	**25391.93**	**25309.98**	490.32
21-3	**28593.40**	24612.36	**24589.88**	303.04	24612.36	**24589.88**	370.47
24-1	53320.50	53470.20	53350.10	903.00	**53832.25**	**53561.95**	2575.78
24-2	59723.20	60010.86	59812.06	877.93	**60234.98**	**60087.76**	1923.42
24-3	53073.72	53668.12	53433.89	904.91	**53797.22**	**53668.75**	1995.95
#Best		27	14		48	48	
#Improve		34			37		
#Match		6			6		