An iterated "hyperplane exploration" approach for the Quadratic Knapsack Problem

Yuning Chen [a] and Jin-Kao Hao [a,b,*]

[a] *LERIA, Université d'Angers, 2 Bd Lavoisier, 49045 Angers, France*

[b] *Institut Universitaire de France, Paris, France*

Computers & Operations Research 77: 226-239, 2017

Abstract

The quadratic knapsack problem (QKP) is a well-known combinatorial optimization problem with numerous applications. Given its NP-hard nature, finding optimal solutions or even high quality suboptimal solutions to QKP in the general case is a highly challenging task. In this paper, we propose an iterated "hyperplane exploration" approach (IHEA) to solve QKP approximately. Instead of considering the whole solution space, the proposed approach adopts the idea of searching over a set of hyperplanes defined by a cardinality constraint to delimit the search to promising areas of the solution space. To explore these hyperplanes efficiently, IHEA employs a variable fixing strategy to reduce each hyperplane-constrained sub-problem and then applies a dedicated tabu search procedure to locate high quality solutions within the reduced solution space. Extensive experimental studies over three sets of 220 QKP instances indicate that IHEA competes very favorably with the state-of-the-art algorithms both in terms of solution quality and computing efficiency. We provide additional information to gain insight into the key components of the proposed approach.

Keywords: Quadratic Knapsack problem; Hyperplane exploration; Tabu search; Variable fixing; Heuristics.

* Corresponding author.
Email addresses: `yuning@info.univ-angers.fr` (Yuning Chen), `hao@info.univ-angers.fr` (Jin-Kao Hao).

1 Introduction

The quadratic knapsack problem (QKP) [14] can be informally described as follows. We are given a capacity-constrained knapsack and a set of candidate objects (or items). Each object has a positive weight, and if selected, generates an object profit and a pairwise profit with any other selected object. The purpose of QKP is to select a subset of objects to fill the knapsack so as to maximize the overall profit while the total weight of the selected objects does not exceed the knapsack capacity.

Formally, let c be the knapsack capacity and $N = \{1, 2, ..., n\}$ the set of objects. Let p_{ii} be the profit of object i ($i \in N$), w_i be its weight. For each pair of objects i and j ($1 \leq i \neq j \leq n$), p_{ij} denotes the pairwise profit which is added to the total profit only when both objects are selected. Let x_i ($1 \leq i \leq n$) be the decision variables such that $x_i = 1$ if object i is selected, $x_i = 0$ otherwise. Then QKP can be formulated as follows:

$$\text{Maximize} \quad f(x) = \sum_{i=1}^{n} \sum_{j=i}^{n} p_{ij} x_i x_j \tag{1}$$

subject to:

$$\sum_{j=1}^{n} w_j x_j \leq c \tag{2}$$

$$x \in \{0, 1\}^n \tag{3}$$

QKP can be reduced to the classical knapsack problem (KP) by restricting the pairwise profit p_{ij} to 0 for all $1 \leq i \neq j \leq n$ and thus generalizes KP. From a graph-theoretic view of point, QKP is also a generalization of the Clique problem [6, 28]. From the perspective of computational complexity, QKP is NP-hard in the strong sense [6]. It is also tightly related to other challenging problems (e.g., edge-weighted maximum clique [9] and weighted maximum b-clique [27]), and appears as a column generation sub-problem when solving the graph partitioning problem [22]. In addition to its theoretical importance, QKP is capable of formulating a number of practical applications [28] like satellite site selection in telecommunications, locations of airports and railway stations in logistics and compiler design.

QKP is well studied in the literature. Over the last decades, much effort has been devoted to developing exact algorithms. Most of these algorithms are based on the general branch-and-bound (B&B) framework. In this context, many useful techniques were introduced including upper planes [14], Lagrangian relaxation [2, 6], linearization [1, 22], semidefinite programming [21]. The progress made on exact methods continually enlarged the class of QKP instances that can be solved optimally. Among state-of-the-art exact approaches, the B&B algorithm introduced in [29] is probably one of the most successful methods which uses aggressive problem reduction techniques. Today's state-of-the-art exact methods are able to solve instances with up to 1500 variables.

On the other hand, to handle problems whose optimal solutions cannot be reached by an exact algorithm, heuristics constitute a useful and complementary approach which aims to find sub-optimal solutions as good as possible to large problems within a reasonable time. Existing QKP heuristic methods can be classified into two categories, namely the randomized or stochastic heuristics (which use random choices in their search components) and deterministic heuristics (which, given a particular input, always produce the same output). Representative randomized heuristic approaches for QKP include three greedy, genetic and greedy genetic algorithms [23], a Mini-Swarm algorithm [35], and a GRASP-tabu search algorithm [37]. Typical deterministic heuristic approaches include an upper plane based heuristic [14], a greedy constructive heuristic [5], a hybrid method [1] combining the greedy heuristic of [5] and the "fill-up and exchange" procedure of [14], a linearization and exchange heuristic [20] and a dynamic programming heuristic [12]. Different from the methods dealing directly with QKP, the approach of [18] reformulates it as an unconstrained binary quadratic problem (UBQP) and applies a tabu search algorithm designed for UBQP to solve the reformulated model. Among the aforementioned heuristics, the Mini-Swarm algorithm [35], the GRASP-tabu search algorithm of [37] and the very recent dynamic programming heuristic of [12] are the state-of-the-art approaches which are used as our references for performance assessment and comparisons. Finally, several approximation algorithms can be found in [24,36]. For a comprehensive survey of different solution methods prior to 2007, the reader is referred to [28].

One observes that, compared to the research effort on exact algorithms which has taken place for a long time, studies on heuristics for QKP are more recent and less abundant. In this work, we are interested in solving large scale QKP instances approximately and present an effective heuristic approach which explores the idea of searching over promising hyperplanes. The proposed approach basically introduces an additional cardinality constraint to the original model to prune regions of the search space where no optimal solution exists. Such an idea proved to be very useful for designing effective heuristics for the multidimensional knapsack problem [4,11,32]. Similar ideas were also explored as a type of "generalized branching" [30] or "constraint branching" [13] within the general B&B framework. In this work, we adapt for the first time the idea of hyperplane exploration in the context of QKP. For this purpose, we address two relevant issues which are critical to make our approach successful. First, we need to identify the promising hyperplanes which are likely to contain high quality solutions. Second, we want to search efficiently inside the identified hyperplanes since each hyperplane, though already much reduced relative to the original model, may still contain a very large number of candidate solutions.

Based on the above considerations, our proposed iterated "hyperplane exploration" algorithm (IHEA) for QKP makes two original contributions.

- From the algorithmic perspectives, we present for the first time a decomposition approach for solving QKP. By introducing the cardinality constraint, IHEA divides the initial problem into several disjoint hyperplane-constrained sub-problems and focuses its exploration within the most promising hyperplanes while discarding unpromising sub-problems. To further reduce the search space of each hyperplane-constrained problem, IHEA uses specific rules to fix a large number of variables. To seek high quality solutions within each reduced sub-problem, IHEA employs a dedicated tabu search procedure which is able to tunnel through infeasible regions to facilitate transition between structurally different feasible solutions. An informed perturbation strategy is also applied to establish a global form of diversification and thereby to allow examining additional unexplored hyperplanes.
- From the perspective of computational results, the proposed IHEA approach displays very competitive performances on three groups of 220 benchmark instances. Particularly, it is able to attain easily all the optimal solutions with a 100% success rate for the first group of 100 instances with up to 300 objects. For the second group of 80 larger instances with 1000 to 2000 objects, IHEA provides improved best results for 6 instances (new lower bounds) and attains the best known results for the remaining cases. We also report results for a third group of very large instances with up to 6000 variables for which our approach achieves an average gap of less than 1.359% between the best lower bound and the well-known upper bound $\hat{U}^2_{CPT}$ [6].

The rest of the paper is organized as follows. Section 2 describes the proposed approach. Section 3 presents an extensive computational assessment in comparison with the state-of-the-art approaches and reports new results for a set of very large-sized instances. Section 4 studies some key ingredients of the proposed approach. Conclusions are drawn in Section 5.

2 Iterated "hyperplane exploration" algorithm for QKP

In this section, we present our iterated "hyperplane exploration" algorithm for QKP. We begin with some useful notations and definitions, and then present the main components of the proposed approach.

2.1 Basic notations and definitions

For a precise presentation of the IHEA algorithm, the following notations and definitions are first introduced.

- Given a solution $x \in \{0,1\}^n$ of a QKP instance P, $I_1(x)$ and $I_0(x)$ denote respectively the index set of variables receiving the value of 1 and 0 in x;
- Given a solution $x \in \{0,1\}^n$ of a QKP instance P, $\sigma(x)$ denotes the sum of the values of all variables in x (i.e., $\sigma(x) = |I_1(x)|$).
- Given two solutions $x \in \{0,1\}^n$ and $x^{'} \in \{0,1\}^n$, $|x, x^{'}| = \sum_{j=1}^{n} |x_j - x_j^{'}|$ denotes the Hamming distance between x and $x^{'}$.
- Function $f_r(x)$ calculates a raw objective value of solution $x \in \{0,1\}^n$ which could be either feasible or infeasible with respect to the capacity constraint.

Definition 1 Given a solution $x \in \{0,1\}^n$, the *contribution* of object i ($i \in N$) to the objective value with respect to x is given by:

$$C(i,x) = p_{ii} + \sum_{j \in I_1(x), j \neq i} p_{ij} \tag{4}$$

Definition 2 Given a solution $x \in \{0,1\}^n$, the *density* of object i ($i \in N$) with respect to x is given by:

$$D(i,x) = C(i,x)/w_i \tag{5}$$

Definition 3 A constrained QKP with a k dimensional hyperplane constraint is defined as:

$$CQKP[k] = \begin{cases} \max \sum_{i=1}^{n} \sum_{j=i}^{n} p_{ij} x_i x_j \\ s.t. \\ \sum_{j=1}^{n} w_j x_j \leq c \\ \sum_{j=1}^{n} x_j = k \\ x \in \{0,1\}^n \end{cases} \tag{6}$$

where $\sum_{j=1}^{n} x_j = k$ is a hyperplane constraint which restricts the solution space of QKP in the k dimensional hyperplane. Each feasible solution of problem $CQKP[k]$ has exactly k selected objects.

Note that the notion of contribution $C(i,x)$ (Definition 1) is related to the upper planes of [14] with two notable differences. First, $C(i,x)$ involves both a problem instance and a solution x while an upper plane only depends on the problem instance (and is independent of any solution). Second, an upper plane can be used to produce an upper bound for QKP while $C(i,x)$ does not have such a utility. In our work, $C(i,x)$ is used as an indicator to evaluate the contribution of a particular item i to the total profit with respect to the given solution x. This indicator allows us to dynamically identify the most profitable item (according to the search state) to be included into the current solution in order to maximize the total profit.

2.2 General idea of the "hyperplane exploration" algorithm

Let $CQKP[k]$ $(1 \leq k \leq n)$ be a constrained QKP (see Definition 3), it is obvious that its solution space is a subspace of the original QKP. Therefore, any feasible solution of the $CQKP[k]$ $(1 \leq k \leq n)$ is also a feasible solution of the original QKP. Let Ω^F be the feasible solution space of a QKP instance such that:

$$\Omega^F = \{x \in \{0,1\}^n : \sum_{j=1}^{n} w_j x_j \leq c\},$$

Then the feasible solution space of the constrained QKP with a k dimensional hyperplane constraint ($CQKP[k]$) is given by:

$$\Omega_{[k]} = \{x \in \Omega^F : \sigma(x) = k\}.$$

QKP can be decomposed into n independent sub-problems (constrained QKPs): $CQKP[1], CQKP[2], ..., CQKP[n]$. These n sub-problems represent n disjoint subspaces and the feasible solution space of QKP is the union of the solution space of all its sub-problems, i.e., $\Omega^F = \cup_{k=1}^{n} \Omega_{[k]}$.

For QKP, if items are sorted in non-decreasing order according to their weight w_j $(j \in N)$, there must exist only one positive integer k_{UB} simultaneously verifying the following two constraints: 1) $\sum_{j=1}^{k_{UB}} w_j \leq c$, 2) $\sum_{j=1}^{k_{UB}+1} w_j > c$; similarly, if items are sorted in non-increasing order according to their weight w_j $(j \in N)$, there must be only one positive integer k_{LB} simultaneously verifying the following two constraints: 1) $\sum_{j=1}^{k_{LB}} w_j \leq c$, 2) $\sum_{j=1}^{k_{LB}+1} w_j > c$. Following the literature like [1, 2, 6, 14, 28, 29], we assume non-negative entries in the profit matrix. Notice that the existing QKP benchmark instances follow this assumption. This leads to the following proposition.

Proposition 1. There must exist optimal solutions of QKP in hyperplanes whose dimensions satisfy: $k_{LB} \leq k \leq k_{UB}$.

Proof. Given a feasible solution $x^1 \in \Omega_{[k_1]}$ $(0 < k_1 < k_{LB})$, and a feasible solution $x^2 \in \Omega_{[k_2]}$ $(k_{LB} \leq k_2 \leq k_{UB},\ k_1 < k_2)$ which is obtained by including another $(k_2 - k_1)$ unselected items to x^1. Then x^2 and x^1 have k_1 selected items in common. Since $(k_2 - k_1) \geq 1$, and the contribution of any selected item in x^2 is greater than or equal to 0, we have: $f(x^2) \geq f(x^1)$. Consequently, there must exist solutions in $\cup_{k=k_{LB}}^{k_{UB}} \Omega_{[k]}$ whose objective value is greater than or equal to the objective value of the best solution in $\cup_{k=1}^{k_{LB}-1} \Omega_{[k]}$. On the other hand, k_{UB} is the maximum number of items that can be contained in the knapsack which means any solution in hyperplanes with a dimension greater than k_{UB} is infeasible. Therefore, no feasible solution in $\cup_{k=k_{UB}+1}^{n} \Omega_{[k]}$ has an objective value better than the objective value of the best solution in $\cup_{k=k_{LB}}^{k_{UB}} \Omega_{[k]}$. $\square$

To make an effective search, we can focus on hyperplanes with a dimension k within the interval $[k_{LB}, k_{UB}]$ and definitely discard other subspaces [1] . This leads to a first (and large) reduction of the solution space from $\cup_{k=1}^{n} \Omega_{[k]}$ to $\cup_{k=k_{LB}}^{k_{UB}} \Omega_{[k]}$.

However, the remaining search space $\cup_{k=k_{LB}}^{k_{UB}} \Omega_{[k]}$ could still be too large to be explored efficiently, especially when the gap between k_{LB} and k_{UB} is large which is typically the general case (see Section 4.1). In order to further prune some less-promising solution spaces, we explore the hypothesis that high quality solutions are located in a set of "promising" hyperplanes whose dimensions are close to k_{UB}. This hypothesis is confirmed by the experimental results presented in Section 3 and Section 4.1.

Based on the above hypothesis, the general idea of our IHEA approach is to progressively and intensively explore a small number of "interesting" hyperplanes whose dimensions are close to k_{UB} so that a large subspace is effectively pruned and the search effort is more focused. We carry out the hyperplane exploration in increasing order of their dimensions aiming to identify solutions of increasing quality, and restart this process with a perturbation when no improvement can be found. When the search seems to stagnate in one hyperplane, we seek better solutions in a higher dimensional hyperplane. To explore a given hyperplane, additional variable fixing techniques are applied to fix a number of variables so as to further shrink the subspace to be examined.

[1] In case an optimal solution exists in a hyperplane with a dimension $k < k_{LB}$, an optimal solution x^* identified in $k \in [k_{LB}, k_{UB}]$ must include items with zero contribution. In a practical situation where the (positive) weight of an item represents a cost, an additional step is required to remove from x^* the zero contribution items.

2.3 General procedure of the "hyperplane exploration" algorithm

Our iterated "hyperplane exploration" approach is composed of three steps:

(1) Construct an initial high quality solution x^0 and use $\sigma(x^0)$ to determine the starting dimension of the "promising" hyperplanes; ($\sigma(x^0) \in [k_{LB}, k_{UB}]$ and $\sigma(x^0)$ is close to k_{UB})

(2) For each hyperplane dimension $k = \sigma(x^0), \sigma(x^0)+1, \sigma(x^0)+2...$, execute the following three steps:

   a. Construct a constrained problem $CQKP[k]$;

   b. Identify some variables that are very likely to be part of the optimal solution, fix these variables to 1 and remove them from $CQKP[k]$, leading to a reduced constrained problem $CQKP'[k]$;

   c. Run a hyperplane exploration algorithm to solve $CQKP'[k]$. In our case, we employ a search procedure based on tabu search [16] to find a high quality solution within the current search space $\Omega_{[k]}$. If this solution improves on the best feasible solution found in previous hyperplanes, we move to the next hyperplane; otherwise we skip to step (3).

(3) Apply a perturbation to restart the search from a new starting point. This perturbation (removing some specifically selected items) typically displaces the search from the current hyperplane to a lower dimensional hyperplane which introduces a possibility to visit more not-explored-yet search spaces (e.g., hyperplanes with a dimension lower than $\sigma(x^0)$) where high quality solutions might exist (though with a small probability).

Algorithm 1 shows the pseudo-code of the IHEA algorithm for QKP, whose components are detailed in the following sections. At the very beginning, an initial solution is generated by a greedy randomized construction procedure (Section 2.4.1) and is further improved by a descent procedure (Section 2.4.2). Then, the initial dimension of the hyperplane as well as the first constrained problem are determined from the initial solution. From this point, the algorithm enters the "hyperplane exploration" phase which examines a series of hyperplane constrained QKP problems. At each iteration of the 'while' loop, IHEA first applies specific variable fixing rules to construct a reduced constrained problem $RCP(V_{fixed}, CQKP[k], x')$ where V_{fixed} contains a set of fixed variables (Section 2.5). $RCP(V_{fixed}, CQKP[k], x')$ is then solved by the tabu search procedure (Section 2.6). Each time a better solution $x^*_{[k]}$ is discovered (i.e., $f(x^*_{[k])} > f(x^b))$ by tabu search, IHEA updates the best solution found x^b and moves on to solve another constrained problem in a higher dimensional hyperplane. To do this, the algorithm increments k by one and reinitializes x' by randomly adding one unselected item to x^b. The 'while' loop terminates when no improving solution can be found. The global best solution is updated at the end of the 'while' loop. IHEA then makes some dedicated changes (perturbations) to x^b , improves the perturbed solution (with the descent procedure) to a local optimum, and use the local optimum to start a new round of the "hyperplane exploration" procedure. The above whole procedure continues until a maximum number of allowed iterations is reached.

Algorithm 1 Pseudo-code of the IHEA algorithm for QKP.

```
1: **Input**: $P$: an instance of the $QKP$; $L$: size of running list; $rcl$: max allowed size of
   restricted candidate list; $MaxIter$: max number of iterations;
2: **Output**: the best solution $x^*$ found so far
3: $x^0 \leftarrow Greedy\_Randomized\_Construction(rcl)$
4: $x^0 \leftarrow Descent(x^0)$
5: $x^{'} \leftarrow x^0$ /* $x^{'}$ represents the current solution */
6: $Iter \leftarrow 0$ /* Iteration counter */
7: $x^b \leftarrow x^{'}$ /* $x^b$ records the best solution found in current iteration */
8: $x^* \leftarrow x^b$ /* $x^*$ records the global best solution */
9: **repeat**
10:    $SolutionImproved \leftarrow true$
11:    $k \leftarrow \sigma(x^{'})$
12:    Construct constrained problem $CQKP[k]$
13:    /* "hyperplane exploration" phase */
14:    **while** $SolutionImproved$ **do**
15:       $V_{fixed} \leftarrow Determine\_Fixed\_Variables(k, x^{'})$
16:       Construct reduced constrained problem $RCP(V_{fixed}, CQKP[k], x^{'})$
17:       Run $TabuSearch\_Engine(L, x^{'}, x^b)$ to solve $RCP(V_{fixed}, CQKP[k], x^{'})$ and
          keep a best solution $x^*_{[k]}$
18:       **if** $f(x^*_{[k]}) > f(x^b)$ **then**
19:          $x^b \leftarrow x^*_{[k]}$
20:          $k \leftarrow k + 1$
21:          $x^{'} \leftarrow$ Randomly add one item to $x^b$
22:          Construct constrained problem $CQKP[k]$
23:       **else**
24:          $SolutionImproved \leftarrow false$
25:       **end if**
26:    **end while**
27:    **if** $f(x^b) > f(x^*)$ **then**
28:       $x^* \leftarrow x^b$
29:    **end if**
30:    /* Perturbation phase */
31:    $x^{'} \leftarrow Perturbation(x^b, Iter)$
32:    $x^{'} \leftarrow Descent(x^{'})$
33:    $x^b \leftarrow x^{'}$
34:    $Iter \leftarrow Iter + 1$
35: **until** $Iter \geq MaxIter$
```

2.4 Initial solution

IHEA constructs an initial solution according to a greedy randomized constructive heuristic. In order to place the initial solution in a "good" hyperplane, we additionally improve the constructed solution with a descent procedure. In this section, we explain these two procedures.

2.4.1 Greedy randomized construction procedure

Our greedy randomized construction procedure follows the spirit of the GRASP approach [10] which has been investigated in [37] to solve QKP. Different from [37] where the construction procedure is used as the main search algorithm, our IHEA algorithm uses this construction procedure to obtain an initial solution.

Starting from a partial solution x where all items are set unselected initially, the construction procedure iteratively and adaptively selects some items to be included in x (i.e., the corresponding variables receive the value of 1) while maintaining the solution feasibility. At each iteration, we randomly select one unselected item from a restricted candidate list RCL and add the item to the partial solution.

Let $R(x) = \{i \in I_0(x) : w_i + \sum_{j \in I_1(x)} w_j \leq c\}$ be the set of unselected items that can fit into the knapsack with respect to x. Let rcl be the maximum size of the restricted candidate list. Then RCL contains $min\{rcl, |R(x)|\}$ unselected items which have the largest *density* values (Section 2.1) and do not violate the capacity constraint when any of them is included to the current partial solution x. Formally, $\forall i \in RCL$, the following two conditions hold: 1) $i \in R(x)$, 2) $D(i, x) \geq D(j, x)$ $(\forall j \in I_0(x) \backslash RCL)$.

The items in RCL are ranked according to their *density* values and the r^{th} $(1 \leq r \leq |RCL|)$ ranked item is associated with a bias $b_r = 1/e^r$. Thus, the r^{th} item is selected with a probability $p(r)$ which is calculated as: $p(r) = b_r / \sum_{j=1}^{|RCL|} b_j$. Once an item is selected and added to the partial solution x, RCL is updated as well as the objective value of the partial solution. This procedure repeats until RCL becomes empty.

One notices that the initial solution constructed by this greedy randomized procedure must be in a hyperplane whose dimension is not smaller than k_{LB} since it is a tight packing plan that cannot include any more unselected item, and not greater than k_{UB} since it is a feasible solution. Also, this dimension is experimentally proved to be close to k_{UB} (see Section 4.1). The underlying reason is that items with a small weight are biased towards being selected to join the knapsack.

2.4.2 Descent procedure

Starting from a solution generated through the greedy randomized construction procedure, IHEA uses a descent procedure to reach a local optimum (The descent procedure is also applied after a perturbation, see Section 2.7). This descent procedure helps: 1) to generate a promising hyperplane which may contain high quality solutions; 2) to improve the quality of the initial solution which allows the hyperplane exploration to start from a high platform.

Our descent procedure jointly employs two different neighborhoods defined by two basic move operators: ADD and $SWAP$.

- $ADD(i)$: This move operator adds an unselected item i ($i \in I_0(x)$) to a given solution x. It can be considered as a special case of $FLIP$ used in [37] by restricting the flipped variables to those having the value of 0 in the given solution. N_A denotes the neighborhood induced by the ADD operator, and N_A^F is a subset of N_A that contains only feasible neighbor solutions.
- $SWAP(i,j)$: Given a solution x, $SWAP(i,j)$ exchanges an unselected item i ($i \in I_0(x)$) with a selected item j of x ($j \in I_1(x)$). This operator is commonly used in the existing QKP approaches [1,37]. A $SWAP$ operation can be realized as two consecutive $FLIP$ operations where one is to flip a variable from 0 to 1 and the other is to flip another variable from 1 to 0. N_S denotes the neighborhood induced by the $SWAP$ operator, and N_S^F is a subset of N_S that contains only feasible neighbor solutions.

The aim of the descent procedure is to attain a local optimum (as good as possible) in both neighborhoods N_A^F and N_S^F starting from a solution x (either obtained by the construction procedure or the perturbation procedure). To this end, the algorithm iteratively explores N_A^F and N_S^F in a token-ring way $N_A^F \to N_S^F \to N_A^F \to N_S^F$.... For each iteration, a feasible neighbor solution x' is picked at random from the neighborhood under consideration and replaces the incumbent solution x if x' is better than x (i.e., $f(x') > f(x)$).

It should be noted that our descent procedure is different from the "fill-up and exchange" procedure of [14] which also relies on ADD and $SWAP$. Our descent procedure examines neighbor solutions in random order and accepts the first encountered improving neighbor while the "fill-up and exchange" procedure checks neighbor solutions in a deterministic way. Moreover, our descent procedure explores N_A^F and N_S^F in a token-ring way while the "fill-up and exchange" process explores these neighborhoods in a sequential way.

2.5 Variable fixing and problem reduction

By adding a hyperplane constraint $\sigma(x) = k$ to QKP, the induced constrained problem $CQKP[k]$ removes a large part of the solution space (of order $O(2^n - C_n^k)$) from the original QKP. However, the solution space of the constrained problem $CQKP[k]$ might still be too large to be efficiently searched by an algorithm. Indeed, as we employ a search algorithm which explores both feasible and infeasible solutions (see Section 2.6), $CQKP[k]$ has a search space of 2^k where k, though smaller than n, could still be large. To further reduce the search space to explore, we fix some variables in $CQKP[k]$.

Recall that in Algorithm 1, the starting solution $x^{'}$ of $TabuSearch\_Engine$ for solving each constrained problem $CQKP[k]$ is modified from the best solution x^b such that $x^{'}$ and x^b are either the same in the first hyperplane (see Line 7 of Algorithm 1) or different only in one variable (see Line 21 of Algorithm 1). For each $CQKP[k]$, based on $x^{'}$, our variable fixing step tries to identify a set of variables V_{fixed} that are highly likely to be part of the optimal solution and fixes them to the value of 1. We then remove these variables (these variables are said fixed) from $CQKP[k]$, leading to a reduced constrained problem $RCP(V_{fixed}, CQKP[k], x^{'})$.

In order to limit the risk of fixing wrong variables, we follow the general idea of identifying a set of "strongly determined" variables [15]. For this purpose, we make use of information from the *density* value associated with each item. According to the definition presented in Section 2.1, for those selected items in a given solution, the *density* of an item represents its contribution (profit) per weight unit. Thus, the *density* value is a good indicator of the importance of a selected item. Given a QKP solution $x^{'} \in \{0, 1\}^n$, our variable fixing rules can be summarized as a three-step method:

(1) For each variable $x^{'}_i$ such that $i \in I_1(x^{'})$, calculate its *density* value $D(i, x^{'})$;
(2) Sort all variables in $I_1(x^{'})$ in non-increasing order according to their *density* values $D(i, x^{'})$ $(i \in I_1(x^{'}))$, leading to a sorted index set $SI_1(x^{'})$;
(3) Extract the first n_f variables in $SI_1(x^{'})$ to form the fixed variable set V_{fixed} (n_f is the number of fixed variables, i.e., $n_f = |V_{fixed}|$). Fix the variables in V_{fixed} with the value of 1, leading to a reduced constrained problem $RCP(V_{fixed}, CQKP[k], x^{'})$ whose variable set is $(I_1(x^{'}) \backslash V_{fixed}) \cup I_0(x^{'})$.

In the last step, n_f is determined using the following empirical formula:

$$n_f = k_{LB} + max\{(|I_1(x^{'})| - k_{LB}) * (1 - 1/(0.008 * n)), 0\} \tag{7}$$

where k_{LB} is the minimum number of items that can be contained in the knapsack. Typically, $|I_1(x^{'})|$ is larger than k_{LB}. It is easy to understand that a solution with only k_{LB} selected items is unlikely to be a good solution since the packing plan is not tight enough. The number of items that can be fixed (n_f) is specified by formula (7).

Formula (7) was identified in the following manner. We first obtained optimal solutions for a set of instances of different sizes (ranging from $n = 100$ to 500) with the exact solver of [6]. We also obtained a set of near-optimal solutions provided by our solution initialization procedure of Section 2.4. We sorted the selected items of the near-optimal solutions according to their density values, and then compared them to the optimal solutions. Finally, we arrived at the following observations: 1) the number of items that can be fixed (n_f) is somewhere between k_{LB} and $|I_1(x^{'})|$; 2) the number of items that can be fixed enlarges as the size of the instance increases. The design of formula (7) basically integrates these observations.

Note that different strategies for temporary or definitive variable fixing were explored in [2] for QKP and studied in other contexts like 0-1 mixed integer programming, integer linear programming and binary quadratic programming [7, 33, 34]. Moreover, the notion of item density was previously used in other construction procedures [5, 37]. For instance, from the set of given items, the greedy construction procedure in [5] drops iteratively the "lightest" items one by one until the remaining items form a feasible solution (i.e., the knapsack constraint becomes satisfied). Furthermore, unlike our procedure where the density of each item is calculated only once, the procedure of [5] updates, after each iteration, the density value of each remaining item.

2.6 Hyperplane exploration with tabu search

The tabu search procedure ($TabuSearch\_Engine$) described in this section is designed to solve the reduced constrained problem $RCP(V_{fixed}, CQKP[k], x^{'})$, i.e., identify feasible solutions that are better than x^b which is the current best solution found in the current course of the "hyperplane exploration" procedure. The key ingredients of $TabuSearch\_Engine$ are described as follows.

- **Neighborhood:** It is known that allowing a controlled exploration of infeasible solutions may enhance the performance of a heuristic search algorithm, by facilitating transitions between structurally different feasible solutions [17]. Following this idea, we employ the N_S neighborhood (defined in Section 2.4.2) which contains both feasible and infeasible neighbor solutions. In order to explore effectively the search space of a given hyperplane, we restrict our tabu search procedure to visit solutions that are considered better than the best feasible solution found so far, leading to a restricted $SWAP$ neighborhood N_S^R. Precisely, given the objective value of the current best feasible solution with objective value f_{min}, the neighborhood $N_S^R(x)$ of a solution x (either feasible or infeasible) is defined as: $N_S^R(x) = \{x^{'} \in N_S(x) : f_r(x^{'}) > f_{min}\}$ where f_r is the raw objective value.
- **Tabu list management:** We use the reverse elimination method (REM) introduced in [16] for our tabu list management. REM defines an exact tabu mechanism which prevents any visited solution from being revisited. REM uses a *running list* to store the attributes of all implemented moves. One can trace back the *running list* to identify the tabu status of a move by making use of another list called residual cancellation sequence (RCS) where an

attribute is either added if it is not yet in the RCS or removed from RCS otherwise. Interested readers are referred to [8,16,32] for more details on this method. For the $SWAP$ operator used by $TabuSearch\_Engine$, when there are only two attributes left in RCS (i.e., $|RCS| = 2$), the move composed of these two attributes is declared tabu in the next iteration. The procedure of updating the tabu status is described in Algorithm 2.

- **Evaluation function:** The evaluation function used by $TabuSearch\_Engine$ considers two factors to assess a solution x: 1) raw objective value $f_r(x)$, 2) capacity violation $v_c(x) = c - \sum_{j=1}^{n} w_j x_j$. A transition is made from the current solution x to a neighbor solution $x' \in N_S^R(x)$ if $\forall x'' \in (N_S^R(x) \backslash \{x'\})$, x' verifies one of the following two conditions: 1) $v_c(x') < v_c(x'')$, 2) $v_c(x') = v_c(x'')$ and $f_r(x') \geq f_r(x'')$.

Algorithm 2 presents the pseudo-code of $TabuSearch\_Engine$ which takes three elements as its input: 1) the max size of the running list which serves as the termination condition; 2) an initial solution (feasible or unfeasible) which serves as its starting point; 3) a reference feasible solution x^{ref} which restricts the search to visit solutions whose objective value is better than that of x^{ref}. At each iteration, the algorithm identifies the best non-tabu $SWAP$ move relative to the above evaluation function, and applies the move to obtain a new solution. Each time a feasible solution is discovered (i.e., $v_{min} = 0$), the running list is reset and f_{min} is updated. $TabuSearch\_Engine$ terminates if one of the following two conditions is verified: 1) all moves are tabu, i.e., $v_{min} = \infty$; 2) the running list is full, i.e., $erl \geq L$.

Note that tabu search was also used in [37] to enhance a GRASP procedure. However, these two studies are quite different. First, our general IHEA approach focuses on decomposing the initial solution space into a set of hyperplane-constrained and reduced solution spaces while GRASP+tabu of [37] explores the original solution space. Second, our tabu search procedure is dedicated to effectively explore the solution space of a hyperplane-constrained sub-problem. As such, its design (in terms of neighborhood, tabu list management and evaluation function) is different from GRASP+tabu. Third, our tabu search explores both feasible and infeasible solutions while GRASP+tabu only visits feasible solutions. Compared to the approach of [18], their tabu search algorithm was dedicated to the unconstrained binary quadratic problem and operates with the “one-flip” move operator (called “shift” in [37]) which adds or removes an item ($SWAP$ was not used).

Algorithm 2 Pseudo-code of $TabuSearch\_Engine$

```
1: Input: $L$: max size of running list; $x^{in}$: an initial solution; $x^{ref}$: a reference feasible solution
2: Output: the best feasible solution found so far $x^*$
3: $|RL| \leftarrow L$ /* Initialize the size of running list to $L$ */
4: $f_{min} \leftarrow f(x^{ref})$ /* $f_{min}$ records the objective value of the current best feasible solution */
5: $x^* \leftarrow x^{ref}$ /* $x^*$ records the best feasible solution found so far */
6: $erl \leftarrow 0$
7: $x \leftarrow x^{in}$
8: while $v_{min} \neq \infty \vee erl < L$ do
9:    $(v_{min}, f_{max}) \leftarrow (\infty, -\infty)$
10:   for each $i \in I_0(x)$ do
11:      for each $j \in I_1(x)$ do
12:         if $tabu[i][j] \neq iter$ then
13:            $(x_i, x_j) \leftarrow (1, 0)$
14:            if $(f_r(x) > f_{min}) \wedge ((v_c(x) < v_{min}) \vee ((v_c(x) = v_{min}) \wedge (f_r(x) \geq f_{max})))$ then
15:               $(i^*, j^*) \leftarrow (i, j)$; $(v_{min}, f_{max}) \leftarrow (v_c(x), f_r(x))$
16:            end if
17:            $(x_i, x_j) \leftarrow (0, 1)$
18:         end if
19:      end for
20:   end for
21:   if $v_{min} \neq \infty$ then
22:      $(x_{i^*}, x_{j^*}) \leftarrow (1, 0)$
23:      if $v_{min} = 0$ then
24:         $erl \leftarrow 0$; $f_{min} \leftarrow f_r(x)$; $x^* \leftarrow x$
25:      else
26:         $iter \leftarrow iter + 1$; $RL \leftarrow RL \cup \{i^*\} \cup \{j^*\}$; $erl \leftarrow erl + 2$
            /* UPDATE TABU STATUS */
27:         $i \leftarrow (erl - 1)$
28:         while $i \geq 0$ do
29:            $j \leftarrow RL[i]$
30:            if $j \in RCS$ then
31:               $RCS \leftarrow RCS \backslash \{j\}$
32:            else
33:               $RCS \leftarrow RCS \cup \{j\}$
34:            end if
35:            if $|RCS| = 2$ then
36:               $tabu[RCS[0]][RCS[1]] \leftarrow iter$; $tabu[RCS[1]][RCS[0]] \leftarrow iter$
37:            end if
38:            $i \leftarrow i - 1$
39:         end while
40:      end if
41:   end if
42: end while
```

2.7 Perturbation

To establish a global form of diversification and reinforce the capacity of the algorithm to visit unexplored "promising" hyperplanes, we employ a perturbation strategy to restart the search from a new starting point (usually in a lower dimensional hyperplane *w.r.t.* the current hyperplane). A perturbation is applied when the hyperplane-based search stagnates, i.e., the local optimum $x^*_{[k]}$

of the current constrained problem $CQKP[k]$ does not improve the best solution found in the current hyperplane exploration phase x^b. This best solution x^b could be either the local optimum obtained by the initializing construction procedure (or perturbation procedure) followed by the descent procedure, or the local optimum of the last constrained problem $CQKP[k-1]$ ($x^*_{[k-1]}$).

Given a feasible solution x, the general idea of our perturbation strategy is to remove a small number (say s, $s > 0$) of items which are chosen from t ($s \leq t \leq (I_1(x) - n_f)$) selected items with the lowest densities and then replace the removed items with some other unselected items. To do this, we first sort all the selected items in x according to the ascending order of their densities $D(x, i)$ ($i \in I_1(x)$). We then remove s items which are selected from the first t sorted items and re-construct the solution using the greedy randomized construction procedure of Section 2.4.1, leading to a perturbed solution. This perturbed solution is further improved by using the descent procedure of Section 2.4.2. The calibration of the parameters t and s is discussed in Section 3.2.

To improve the diversification effect of the perturbation, we employ a short-term memory to prevent recently removed items from being added back to the solution in subsequent iterations. Each time an item is removed from the solution, it is not allowed to be inserted to the solution in next $rand(1, s)$ iterations where $rand(1, s)$ takes a random value between 1 and s.

2.8 Implementation improvement

A fast incremental evaluation technique was introduced in [14] to determine rapidly the effect of the *ADD* and *SWAP* moves for their "fill-up and exchange" procedure which explores only feasible regions. We slightly extend this technique to make it applicable to evaluate the raw objective values of the new solutions encountered in either feasible or infeasible solution spaces. In addition, we also introduce a fast feasibility checking technique.

Given a solution x, which may be either feasible or infeasible, flipping a variable x_i produces a new solution $x^{'}$ whose raw objective value can be conveniently calculated in *O(1)* time as: $f_r(x^{'}) = f_r(x) + (1 - 2 * x_i) * C(i, x)$, where $C(i, x)$ is the contribution of variable x_i. Therefore, any solution generated by a move operator which can be realized with constant times of flip operations can be evaluated in *O(1)* time as well. This is the case for the *ADD* and *SWAP* operators which can be realized with one and two flip operations respectively. Using this fast evaluation technique leads to a significant acceleration compared to a complete evaluation which requires O(n^2) time.

To achieve this saving, we maintain a memory structure Δ to store the current contribution value of each variable Δ_i (corresponding to $C(i, x)$ for a given solution x) which is updated each time a flip operation is performed. Given an empty solution where all variables are assigned the value of 0, the contribution of flipping any variable is initialized to its profit value, i.e., $\Delta_i \leftarrow p_i, i \in N$. Thereafter, once a move is performed, the contribution value of each variable after flipping a variable x_i can be efficiently updated as follows:

$$\Delta_j = \begin{cases} \Delta_j, & if \quad j = i \\ \Delta_j + q_{ij}, & if \quad x_i = 1 \quad and \quad j \in N\backslash\{i\} \\ \Delta_j - q_{ij}, & if \quad x_i = 0 \quad and \quad j \in N\backslash\{i\} \end{cases} \tag{8}$$

The total time of updating the structure Δ is bounded by *O(n)*. Using this memory structure, a new solution $x^{'}$ transitioned from the current solution x by adding an unselected item x_i can be evaluated using the following equation:

$$f_r(x^{'}) = f_r(x) + \Delta_i \tag{9}$$

Similarly, a new solution $x^{'}$ produced by swapping items x_i ($x_i = 1$) and x_j ($x_j = 0$) of the current solution x can be evaluated by the following equation:

$$f_r(x^{'}) = f_r(x) - \Delta_i + \Delta_j - p_{ij} \tag{10}$$

In addition to Δ, we maintain another memory structure which stores the sum of weights of all the selected items in the current solution, which is updated accordingly after an operation is performed. This memory structure allows the capacity constraint checking to be achieved with a time complexity of *O(1)*.

3 Computational Experiments

This section is dedicated to a computational assessment of the proposed algorithm and comparisons with the state-of-the-art QKP approaches.

3.1 Experimental protocol

To evaluate the efficiency of the proposed algorithm, we carry out extensive experiments on a set of 220 instances ranging from small to very large sizes. These instances can be divided into three groups:

- **Group I**. This group is composed of 100 small and medium sized benchmark instances generated by Billionnet and Soutif [2]. These instances, generated in the same way as in [5, 6, 14, 20], are very popular and used to test many QKP algorithms. These instances are characterized by their number of objects $n \in \{100, 200, 300\}$, density $d \in \{25\%, 50\%, 75\%, 100\%\}$ (i.e., number of non-zero coefficients of the objective function divided by $n(n+1)/2$). Each (n, d) combination involves 10 different instances distinguished by their labels except for (300,75%) and (300,100%) where instances are missing. Optimal solutions are known for these instances. The instance data files can be downloaded at `http://cedric.cnam.fr/~soutif/QKP/QKP.html`.
- **Group II**. The second group includes 80 large-sized benchmark instances which are recently generated by Yang et al. [37]. These instances have a number of objects from 1000 to 2000, a value of density from 25% to 100%. Due to their large size, optimal solutions are still unknown for these instances. The instance data files are available at `http://www.info.univ-angers.fr/pub/hao/QKP.html`
- **Group III**. This group is composed of 40 new instances of very large sizes that we propose for this study. They are characterized by their number of objects $n \in \{5000, 6000\}$ and density $d \in \{25\%, 50\%, 75\%, 100\%\}$. For each (n, d) combination, 5 instances were proposed. These instances are available from the authors of this work (they are too large to be put on our web).

The above three groups of instances were all generated using the same generator that was very popular and commonly used in QKP literature [1,6,14,20]. The parameter settings for the generator are the same as well: the coefficients p_{ij} of the objective function are integers that are uniformly distributed in the interval [0,100]; each weight w_j is uniformly distributed in [1,50]; the capacity c is randomly selected from $[50,\sum_{j=1}^{n} w_j]$. However, to ensure the hardness of the new instances of Group III, we have performed the following selection process. For each instance, we attained the best objective value of ten feasible solutions, each of which was built by randomly filling the knapsack. A gap between the objective value of the random solution and the best objective value from our IHEA algorithm was then calculated. We denote this gap as *randGap*. For each (n, d) combination, we generated 10 instances, 5 of which having the largest *randGap*s were selected to join Group III. The average *randGap* (in percentage) of all instances in Group III is 69.36% which indicates that a randomly generated solution is rather poor. From this perspective, finding a high quality solution for these instances is a non-trivial task.

Our IHEA algorithm was coded in C++ [2] and compiled by GNU gcc 4.1.2 with the '-O3' option. The experiments were conducted on a computer with an AMD Opteron 4184 processor (2.8GHz and 2GB RAM) running Ubuntu

[2] Our best solution certificates are available at: `http://www.info.univ-angers.fr/pub/hao/QKPResults.zip`.

Table 1
Parameter settings of the IHEA algorithm

Para.	*Description*	*Value*	*Section*
rcl	max allowed size of restricted candidate list	20	2.4.1
L	size of the running list	300	2.6
t	size of the least-density items for perturbation	$min\{10, (I_1(x) - n_f)\}$	2.7
s	number of items to be perturbed	$min\{3, t\}$	2.7
$MaxIter$	max number of iterations	$\sqrt{n}$+65	2.3

12.04. When running the DIMACS machine benchmark program dfmax.c on graphs r300.5, r400.5 and r500.5 (available at `ftp://dimacs.rutgers.edu/pub/dsj/clique/`) (compliled without optimization flag), the run time on our machine is 0.40, 2.50 and 9.55 seconds respectively for these graphs.

3.2 Parameter calibration

Our IHEA algorithm relies on five parameters (see Table 1). To calibrate these parameters, we employed an automatic configuration method called Iterated F-race (IFR) [3] which was implemented in the irace package [25]. For each parameter to be tuned, IFR requires a range of values as input. Based on preliminary experiments, we used the range of values as follows: $rcl \in [10, 30]$, $L \in [100, 400]$, $p_1 \in [5, 20]$, and $p_2 \in [1, 7]$. p_1 and p_2 are two parameters associated with t and s, i.e., $t = min\{p_1, (I_1(x) - n_f)\}$ and $s = min\{p_2, t\}$. We restricted the training set to 26 representative instances: one instance from each (n, d) combination. To run IFR, we used the tuning budget of 3000 IHEA runs, each run being given 50 iterations. Once the previous four parameters are determined, the termination condition parameter $MaxIter$ can be easily tuned by taking into account the balance between quality and efficiency of the IHEA algorithm. The parameter values shown in Table 1 were used in all experiments in the following sections unless otherwise mentioned.

3.3 Comparative results on small and medium instances of Group I

Our first experiment was performed on the benchmark instances of Group I. These instances were first solved to optimality by the exact algorithm of [2], with hundreds or thousands of CPU seconds on a 300 MHz Pentium II Processor. Several recent heuristic approaches are able to attain these optimal solutions with much less computing efforts (typically a few seconds) [35, 37]. To evaluate the performance of our IHEA algorithm, three leading heuristic methods were considered for our comparison:

- A Mini-Swarm approach [35]. The experiments reported in [35] were per-

Table 2
Scaling factors for the computers used in the reference algorithms. Our computer (AMD Opteron 4184) serves as the basis.

Algorithm	Reference	Processor type	Frequency (GHz)	Factor
IHEA	-	AMD Opteron 4184	2.8	1.0
Min-Swarm	[35]	Pentium IV	3.06	1.09
GRASP+tabu	[37]	Pentium	1.73	0.62

formed on a computer with a 3.06 GHz P4 processor.

- A recent Dynamic Programming heuristic approach [12]. Among all algorithm variants studied in [12], the best performance was achieved by the version using $\tilde{\pi}_i^3/w_i$ as the item ordering rule coupled with the "fill-up and exchange" procedure proposed in [14]. This algorithm version, denoted as DP+FE, was included for our comparative study. The results of DP+FE in this study were obtained by running the source code (which was kindly given to us by the authors of [12]) on our machine. Since DP+FE is a deterministic heuristic algorithm, it was executed for a single run.
- A GRASP-tabu approach [37]. This approach includes two algorithm variants (GRASP and GRASP+tabu). We used the results of GRASP+tabu for our comparative study since it dominates GRASP alone. The reported results were obtained on a Pentium 1.73 GHz processor with 2 GB RAM.

It is not a straightforward task to make a fully fair comparative analysis with the reference approaches due to the differences in computing hardware, termination criterion, etc. This is particularly true for the computing times. For this reason, we focus our study on the quality criterion of the solutions found. Nevertheless, we include information on computing time for indicative purposes. Following [26, 31], we used the CPU frequency of our computer (AMD Opteron 4184 2.8 GHz) as a basis to scale the times of the reference algorithms reported in [35, 37] (see Table 2 for the scaling factors).

Like the two reference randomized algorithms (Min-Swarm and GRASP+tabu), we ran our IHEA algorithm 100 times to solve each problem instance. To show a general picture and simplify the presentation, we divide the whole instance set into 8 classes according to the (n, d) combination. Table 3 displays for each instance class and for each randomized algorithm, the average value of three indicators: 1) success rate (SR), i.e., the number of the trials over 100 runs hitting the known optimal solution; 2) relative percentage deviation (RPD), the average gap between the best lower bound (f_{LB}) and the best solution value (f_{best}) in percentage over 100 trials where the gap is calculated by $((f_{LB}$ - $f_{best})/f_{LB} \times 100)$; 3) the average CPU time in seconds for one trial ($t(s)$). Note that for the single-run deterministic algorithm (DP+FE), the average values over multiple runs are not needed. For our IHEA algorithm, we also report the average time when the algorithm first encounters the best solution ($t_b(s)$) over 100 trials. Since our proposed algorithm as well as the reference algorithms can easily attain the optimal results for all the benchmark in-

Table 3
Comparative results of IHEA with 3 state-of-the-art algorithms on the 100 benchmark instances of Group I [2]. The values in bold indicate the improved results of IHEA.

INST.	Min-Swarm [35]			DP+FE [12]			GRASP+tabu [37]			IHEA			
	SR(%)	RPD	t(s)	SR(%)	RPD	t(s)	SR(%)	RPD	t(s)	SR(%)	RPD	t(s)	$t_b(s)$
100_25	93.900	0.012	0.482	40.000	0.319	0.697	100.000	0.000	0.060	100.000	0.000	0.325	0.004
100_50	94.200	0.004	0.442	80.000	0.018	0.708	100.000	0.000	0.057	100.000	0.000	0.253	0.002
100_75	97.500	0.001	0.396	80.000	0.008	0.704	100.000	0.000	0.052	100.000	0.000	0.334	0.003
100_100[1]	100.000	0.000	0.224	77.777	0.005	0.657	100.000	0.000	0.048	100.000	0.000	0.248	0.002
200_25	90.300	0.009	1.559	60.000	0.056	7.341	100.000	0.000	0.286	100.000	0.000	0.714	0.029
200_50	92.400	0.001	1.967	50.000	0.009	8.239	99.900	9.4E-6	0.301	**100.000**	**0.000**	0.827	0.035
200_75	90.900	0.003	2.361	60.000	0.010	7.055	100.000	0.000	0.318	100.000	0.000	0.946	0.010
200_100	100.000	0.000	1.305	60.000	0.004	6.683	100.000	0.000	0.251	100.000	0.000	0.722	0.005
300_25[2]	-	-	-	33.333	0.061	28.341	99.667	0.001	0.735	**100.000**	**0.000**	1.122	0.018
300_50	-	-	-	50.000	0.003	31.324	100.000	0.000	0.763	100.000	0.000	1.156	0.015

[1] Instance 100_100_4 is not available and is not considered when we calculate the average value for the instance class 100_100.
[2] Instance 300_25_3 is not available and is not considered when we calculate the average value for the instance class 300_25.

stances under consideration, these optimal results are not listed in the table (see [2,35,37] for these optimal results). The results of the reference algorithms are extracted from the corresponding papers [35,37] (the code of the reference algorithms are not available).

From Table 3, we observe that our IHEA algorithm attains the known optimal values with a successful rate of 100% for *all* these instances with an average computing time of no more than 1.156 seconds. The average best solution time $t_b(s)$ of IHEA is even more interesting since the maximum time is only 0.035 seconds (for 200_50). IHEA outperforms the Min-Swarm approach in the success rate and the relative percentage deviation by consuming typically less CPU seconds. Meanwhile, IHEA dominates the deterministic DP+FE algorithm in terms of both solution quality and computational efficiency for all instance classes. Compared to GRASP+tabu which is one of the current best performing heuristic algorithms, IHEA remains very competitive since it solves all these instances to optimality with a 100% success rate while there are 2 instances for which GRASP+tabu is not able to achieve a success rate of 100%. Another interesting feature of IHEA is that its average computing time is approximately linear relative to the size of the instance which was not observed for the reference algorithms.

3.4 Comparative results on large instances of Group II

In this section, we investigate the behavior of our algorithm on the second group of 80 large instances (n = 1000 or 2000) with unknown optima. Like GRASP+tabu of [37], our IHEA algorithm was executed 100 times for each instance. As reference algorithms, we again used GRASP+tabu (which reported the current best known lower bounds for these instances) and DP+FE. The results of DP+FE were obtained by executing a single run of its source code on our machine while the results of GRASP+tabu were extracted from [37]. Table 4 (for instances with 1000 objects) and Table 5 (for instances with 2000 objects) summarize the comparative results. The listed indicators include the best lower bound discovered (column Best), the success rate (column SR) to attain the best lower bound, the relative percentage deviation (column RPD) and the average computing time for one trial ($t(s)$). We also report the average best time over 100 trials for our IHEA algorithm (column $t_b(s)$). The best known results are indicated in italic and the new improved results are highlighted in bold. In the last two rows, *#Bests* indicates the number of italic and bold values, and *Avg.* denotes the average value for each column.

Table 4 and 5 disclose that our IHEA algorithm outperforms both GRASP+tabu and DP+FE in terms of all indicators. Firstly, IHEA attains the best known lower bounds or improves these lower bounds for all 80 instances. Specifically, it discovers improved best lower bounds for 6 instances and attains the previous best known lower bounds for the remaining 74 instances. Secondly, the success rate and relative percentage deviation achieved by IHEA are consistently better than or equal to those achieved by the randomized GRASP+tabu. In particular, IHEA performs better in these two indicators for 24 out of 80 cases and is equal for the remaining 56 cases. GRASP+tabu has 24 cases for which the success rate is under 100% while IHEA has only 3 such cases. Moreover, IHEA is better both in its lowest success rate (95% vs. 6%) and in its average success rate (99.9% vs. 91.63%) compared to GRASP+tabu. Notice that the success rate of IHEA can be further improved by simply extending the max iteration parameter (*MaxIter*). For example, when we set the *MaxIter* to ($\sqrt{n}$+130), IHEA achieves a 100% success rate for *all* these instances but at the expense of more computing time. Thirdly, IHEA always needs much less computing effort to achieve a similar or better performance compared to GRASP+tabu and DP+FE.

To solve instances with 1000 variables and 2000 variables respectively, IHEA consumes an average computing time of 6.0 seconds and 22.73 seconds while these values are 27.96 seconds and 329.65 seconds for GRASP+tabu, 2917.70 seconds and 51695.75 seconds for DP+FE. This implies that IHEA scales very well with a weak increasing ratio of its average computing time (i.e., 22.73/6.0 $\approx$ 3.79) when the problem size grows from 1000 to 2000 objects while this ratio is much higher for GRASP+tabu (329.65/27.96 $\approx$ 11.79) and DP+FE (51695.75/2917.70 $\approx$ 17.72). Notice also that compared to the average time which is proportional to the *MaxIter* parameter, the average best solution time (column $t_b(s)$) is even much shorter (i.e., 0.47 seconds) across the whole instance set. Given this fact, it is easy to see that the large gap between the average $t_b(s)$ and $t(s)$ is consumed by the algorithm only to complete its run, but useless for improving the best solution. Finally, comparing GRASP+tabu with DP+FE, the former outperforms the latter by achieving significantly more best known lower bounds (74 vs. 11) with much less average computing

Table 4
Comparative results of IHEA with two state-of-the-art algorithms on the 40 instances with 1000 objects of Group II. The best known results are in italic and the new best known results are in boldface.

INST.	DP+FE [12]		GRASP+tabu [37]				IHEA				
	Best	t(s)	Best	SR(%)	RPD	t(s)	Best	SR(%)	RPD	t(s)	*tb(s)*
1000_25_1	*6172407*	1682.280	*6172407*	*100.000*	*0.000*	18.234	*6172407*	*100.000*	*0.000*	2.765	0.094
1000_25_2	229833	2103.290	*229941*	66.000	0.008	20.448	*229941*	**100.000**	**0.000**	5.390	0.091
1000_25_3	*172418*	1919.350	*172418*	*100.000*	*0.000*	13.429	*172418*	*100.000*	*0.000*	5.892	0.050
1000_25_4	367365	2537.720	*367426*	*100.000*	*0.000*	16.188	*367426*	*100.000*	*0.000*	7.293	0.068
1000_25_5	4885569	2626.970	*4885611*	*100.000*	*0.000*	23.368	*4885611*	*100.000*	*0.000*	5.543	0.144
1000_25_6	15528	608.550	*15689*	*100.000*	*0.000*	5.072	*15689*	*100.000*	*0.000*	1.635	0.020
1000_25_7	4945741	2725.220	*4945810*	*100.000*	*0.000*	22.636	*4945810*	*100.000*	*0.000*	4.804	0.306
1000_25_8	1709954	3762.890	*1710198*	*100.000*	*0.000*	44.150	*1710198*	*100.000*	*0.000*	7.104	0.416
1000_25_9	*496315*	2839.990	*496315*	*100.000*	*0.000*	18.619	*496315*	*100.000*	*0.000*	6.891	0.060
1000_25_10	1173686	3607.270	*1173792*	*100.000*	*0.000*	36.537	*1173792*	*100.000*	*0.000*	7.573	0.148
1000_50_1	5663517	3722.470	*5663590*	*100.000*	*0.000*	31.459	*5663590*	*100.000*	*0.000*	6.870	0.108
1000_50_2	*180831*	1450.870	*180831*	*100.000*	*0.000*	0.893	*180831*	*100.000*	*0.000*	3.692	0.045
1000_50_3	11384139	2071.250	*11384283*	*100.000*	*0.000*	19.753	*11384283*	*100.000*	*0.000*	3.338	0.088
1000_50_4	322184	1868.860	*322226*	*100.000*	*0.000*	13.677	*322226*	*100.000*	*0.000*	5.433	0.059
1000_50_5	9983477	2570.760	*9984247*	86.000	1.5E-4	25.315	*9984247*	**98.000**	**6.0E-6**	3.662	0.541
1000_50_6	4106186	3801.720	*4106261*	*100.000*	*0.000*	36.010	*4106261*	*100.000*	*0.000*	7.691	0.113
1000_50_7	10498135	2322.160	*10498370*	84.000	1.3E-4	20.727	*10498370*	**100.000**	**0.000**	3.584	0.271
1000_50_8	4981017	3826.980	*4981146*	20.000	0.012	72.100	*4981146*	**99.000**	**1.0E-6**	9.155	1.648
1000_50_9	1727727	3382.020	*1727861*	*100.000*	*0.000*	32.717	*1727861*	*100.000*	*0.000*	9.381	0.847
1000_50_10	2340590	3605.070	*2340724*	94.000	2.3E-4	59.074	*2340724*	**100.000**	**0.000**	7.416	0.163
1000_75_1	11569498	3334.210	*11570056*	65.000	7.6E-5	39.680	*11570056*	**100.000**	**0.000**	4.892	0.514
1000_75_2	1901119	3094.560	*1901389*	*100.000*	*0.000*	20.131	*1901389*	*100.000*	*0.000*	6.492	0.129
1000_75_3	2096415	3208.980	*2096485*	*100.000*	*0.000*	24.713	*2096485*	*100.000*	*0.000*	8.742	0.107
1000_75_4	7305195	3821.020	*7305321*	*100.000*	*0.000*	34.156	*7305321*	*100.000*	*0.000*	6.846	0.119
1000_75_5	13969705	2887.190	13970240	93.000	4.0E-4	23.182	**13970842**	**100.000**	**0.000**	6.022	0.130
1000_75_6	12288299	3178.950	*12288738*	*100.000*	*0.000*	20.733	*12288738*	*100.000*	*0.000*	4.463	0.161
1000_75_7	*1095837*	2580.270	*1095837*	*100.000*	*0.000*	14.359	*1095837*	*100.000*	*0.000*	7.119	0.099
1000_75_8	5575592	3804.420	*5575813*	*100.000*	*0.000*	42.451	*5575813*	*100.000*	*0.000*	7.833	0.142
1000_75_9	695595	2171.330	*695774*	*100.000*	*0.000*	14.062	*695774*	*100.000*	*0.000*	4.624	0.126
1000_75_10	2507627	3349.440	*2507677*	*100.000*	*0.000*	29.338	*2507677*	*100.000*	*0.000*	6.863	0.074
1000_100_1	6243330	3849.500	*6243494*	*100.000*	*0.000*	44.646	*6243494*	*100.000*	*0.000*	7.018	0.116
1000_100_2	4853927	3627.050	*4854086*	61.000	0.001	52.601	*4854086*	**100.000**	**0.000**	7.092	0.193
1000_100_3	3171955	3320.520	*3172022*	*100.000*	*0.000*	29.177	*3172022*	*100.000*	*0.000*	6.391	0.096
1000_100_4	754542	1990.800	*754727*	*100.000*	*0.000*	14.651	*754727*	*100.000*	*0.000*	5.207	0.075
1000_100_5	18646607	2829.350	*18646620*	99.000	6.9E-7	24.273	*18646620*	**100.000**	**0.000**	4.070	0.289
1000_100_6	16019697	3247.810	16018298	96.000	4.2E-6	25.780	**16020232**	**100.000**	**0.000**	5.204	0.117
1000_100_7	*12936205*	3587.160	*12936205*	*100.000*	*0.000*	27.590	*12936205*	*100.000*	*0.000*	5.533	0.129
1000_100_8	6927342	3850.890	*6927738*	*100.000*	0.000	59.551	*6927738*	*100.000*	**0.000**	7.298	0.113
1000_100_9	*3874959*	3463.920	*3874959*	*100.000*	*0.000*	32.414	*3874959*	*100.000*	*0.000*	7.085	0.067
1000_100_10	1334389	2474.890	*1334494*	*100.000*	*0.000*	14.651	*1334494*	*100.000*	*0.000*	6.270	0.094
#Bests	7	-	38	30	30	-	40	40	40	-	-
Avg.	5128111	2917.699	5128228	94.100	6E-4	27.964	5128291	99.925	0.000	6.005	0.204

time (178.87 seconds vs. 27306.73 seconds) across 80 instances of Group II. This observation confirms a safe ranking of the three compared algorithms in decreasing order of their performance: IHEA, GRASP+tabu and DP+FE.

Table 5
Comparative results of IHEA with two state-of-the-art algorithms on the 40 instances with 2000 objects of Group II. The best known results are in italic and the new best known results are in boldface.

INST.	DP+FE [12]		GRASP+tabu [37]				IHEA				
	Best	t(s)	Best	SR(%)	RPD	t(s)	Best	SR(%)	RPD	t(s)	*tb(s)*
2000_25_1	5268004	57726.920	*5268188*	*100.000*	*0.000*	320.273	*5268188*	*100.000*	*0.000*	22.264	0.479
2000_25_2	13293940	51050.130	*13294030*	*100.000*	*0.000*	205.053	*13294030*	*100.000*	*0.000*	24.917	2.191
2000_25_3	5500323	57419.270	*5500433*	56.000	4.5E-4	496.081	*5500433*	**100.000**	**0.000**	28.933	0.780
2000_25_4	14624769	46620.160	*14625118*	*100.000*	*0.000*	215.072	*14625118*	*100.000*	*0.000*	17.050	0.766
2000_25_5	5975645	57416.960	*5975751*	*100.000*	*0.000*	457.765	*5975751*	*100.000*	*0.000*	28.102	0.502
2000_25_6	4491533	56155.800	*4491691*	*100.000*	*0.000*	294.252	*4491691*	*100.000*	*0.000*	23.442	0.767
2000_25_7	6388475	57116.940	*6388756*	*100.000*	*0.000*	346.090	*6388756*	*100.000*	*0.000*	25.178	0.671
2000_25_8	11769395	52832.060	*11769873*	*100.000*	*0.000*	277.109	*11769873*	*100.000*	*0.000*	22.584	0.718
2000_25_9	10959388	54258.650	*10960328*	*100.000*	*0.000*	278.882	*10960328*	*100.000*	*0.000*	22.420	0.564
2000_25_10	139233	14686.960	*139236*	*100.000*	*0.000*	68.070	*139236*	*100.000*	*0.000*	7.551	0.087
2000_50_1	*7070736*	52860.690	*7070736*	39.000	0.027	294.078	*7070736*	**100.000**	**0.000**	28.016	0.441
2000_50_2	12586693	57518.440	*12587545*	*100.000*	*0.000*	331.619	*12587545*	*100.000*	*0.000*	23.943	1.349
2000_50_3	27266846	48397.300	*27268336*	*100.000*	*0.000*	191.506	*27268336*	*100.000*	*0.000*	22.691	0.359
2000_50_4	17754391	57376.090	*17754434*	*100.000*	*0.000*	485.249	*17754434*	*100.000*	*0.000*	24.506	0.447
2000_50_5	16804699	57563.580	16805490	90.000	9.3E-4	923.936	**16806059**	**100.000**	**0.000**	32.057	0.538
2000_50_6	23075693	52613.210	*23076155*	50.000	5.1E-4	285.256	*23076155*	**100.000**	**0.000**	21.579	0.631
2000_50_7	28757657	46437.960	*28759759*	6.000	0.008	442.792	*28759759*	**100.000**	**0.000**	25.365	0.525
2000_50_8	*1580242*	32416.870	*1580242*	*100.000*	*0.000*	102.412	*1580242*	*100.000*	*0.000*	13.937	0.169
2000_50_9	26523637	48529.930	*26523791*	*100.000*	*0.000*	212.114	*26523791*	*100.000*	*0.000*	19.695	0.438
2000_50_10	24746249	50565.420	*24747047*	*100.000*	*0.000*	253.202	*24747047*	*100.000*	*0.000*	20.613	0.377
2000_75_1	25121327	57579.990	*25121998*	*100.000*	*0.000*	500.371	*25121998*	*100.000*	*0.000*	22.721	0.529
2000_75_2	12663927	54629.120	*12664670*	89.000	4.7E-4	316.231	*12664670*	**100.000**	**0.000**	21.584	0.401
2000_75_3	43943294	45151.420	*43943994*	*100.000*	*0.000*	171.362	*43943994*	*100.000*	*0.000*	18.723	0.763
2000_75_4	37496414	50255.520	*37496613*	*100.000*	*0.000*	219.561	*37496613*	*100.000*	*0.000*	19.901	0.434
2000_75_5	24835254	56840.030	24834948	73.000	2.1E-4	424.285	**24835349**	**100.000**	**0.000**	27.439	0.533
2000_75_6	45137702	44437.730	*45137758*	*100.000*	*0.000*	190.011	*45137758*	*100.000*	*0.000*	20.862	0.345
2000_75_7	25502503	57480.680	*25502608*	*100.000*	*0.000*	303.887	*25502608*	*100.000*	*0.000*	21.848	0.402
2000_75_8	10067752	52566.820	*10067892*	*100.000*	*0.000*	213.795	*10067892*	*100.000*	*0.000*	21.560	0.333
2000_75_9	**14177079**	55684.210	14171994	97.000	1.6E-5	329.877	**14177079**	**100.000**	**0.000**	32.008	0.482
2000_75_10	7815419	48717.480	*7815755*	78.000	8.7E-5	201.636	*7815755*	**100.000**	**0.000**	20.537	1.730
2000_100_1	37929562	57195.970	*37929909*	*100.000*	*0.000*	270.140	*37929909*	*100.000*	*0.000*	21.622	0.418
2000_100_2	**33665281**	57844.250	33647322	95.000	8.2E-5	490.736	**33665281**	**100.000**	**0.000**	34.322	0.548
2000_100_3	29951509	57198.420	*29952019*	34.000	0.003	923.360	*29952019*	**100.000**	**0.000**	23.249	0.436
2000_100_4	26948234	57484.560	*26949268*	*100.000*	*0.000*	440.690	*26949268*	*100.000*	*0.000*	23.800	0.542
2000_100_5	22040523	58316.780	*22041715*	70.000	3.0E-4	466.252	*22041715*	**95.000**	**1.1E-5**	23.346	6.286
2000_100_6	18868630	56282.860	*18868887*	*100.000*	*0.000*	339.878	*18868887*	*100.000*	*0.000*	22.315	0.387
2000_100_7	15850198	54333.570	*15850597*	*100.000*	*0.000*	358.472	*15850597*	*100.000*	*0.000*	22.555	0.399
2000_100_8	13628210	52206.350	*13628967*	*100.000*	*0.000*	231.923	*13628967*	*100.000*	*0.000*	22.250	0.356
2000_100_9	8394440	45817.310	*8394562*	97.000	2.2E-4	188.672	*8394562*	**100.000**	**0.000**	18.686	0.361
2000_100_10	4923413	38243.750	*4923559*	92.000	1.4E-4	124.031	*4923559*	**100.000**	**0.000**	15.041	0.913
#Bests	4	-	36	26	26	-	40	40	40	-	-
Avg.	18088455	51695.754	18088299	89.150	0.001	329.650	18088900	99.875	0.000	22.730	0.735

3.5 Computational results on very large instances of Group III

We now present the last experiment on the 40 very large instances of Group III with 5000 to 6000 variables with unknown optima. In addition to their size, the hardness of these instances is also ensured through the selection process when they were created (see Section 3.1). To measure the solution quality obtained by our IHEA algorithm, we used the method proposed in [6] to calculate upper bounds $\hat{U}^2_{CPT}$ (the current tightest bounds), which was previously used in [37] to evaluate their algorithms. The results of DP+PE [12] on these instances are not reported since DP+PE requires intolerable amount of computing time (indeed, we did not obtain any result with a time limit of one week).

Our IHEA algorithm was executed 100 times for each instance. Table 6 summarizes the results: the best lower bound (column Best), the gap between the upper bound $\hat{U}^2_{CPT}$ and the best lower bound in percentage (column GAP, calculated by ($\hat{U}^2_{CPT}$ - Best)/Best $\times$ 100), success rate (column SR), relative percentage deviation (column RPD), the average computational time for one trial (column t(s)) and the average best solution time aver 100 trials (column $t_b(s)$). The last row (*Avg.*) indicates the average value of each column. From Table 6, we observe that IHEA is able to attain high quality lower bounds for all these large and difficult instances. These lower bounds are typically very close to the corresponding upper bounds. Indeed, the average gap between the best lower bound and the upper bound $\hat{U}^2_{CPT}$ is 1.359% for the whole instance set. Moreover, IHEA achieves a success rate of 100% for 30 out of 40 instances (75%). The average success rate across all instances is 87.675%. When we examine the computing time, the results are quite acceptable. Specifically, the average run time for one trial is 174.075 seconds. The best solution time is much shorter with an average value of 14.456 seconds.

4 Discussion

The computational outcomes and comparisons with state-of-the-art algorithms presented in Section 3 demonstrated the effectiveness of the proposed IHEA approach. In this section, we provide additional information to gain more insights into the "hyperplane exploration" component (Section 4.1), and further investigate two other important ingredients of the IHEA algorithm: the variable fixing strategy (Section 4.2) and the perturbation strategy (Section 4.3). To simplify the presentation of Sections 4.1 and 4.2, we used a subset of 8 representative instances (see Table 7-8) from the 80 benchmarks of Group II [37]. These instances cover all sizes and all densities of Group II. We denote this subset of instances as Group ii.

4.1 Insight into the "hyperplane exploration" phase

To show the influence of the "hyperplane exploration" component on the efficiency of IHEA, we provide additional information on the 8 representative instances of Group ii in Table 7 (complementary to Tables 4 and 5) includ-

Table 6
Computational results of IHEA on the 40 very large instances of Group III

INST.	Best	Gap(%)	SR(%)	RPD	t(s)	$t_b(s)$
5000_25_1	23667450	2.729	100.000	0.000	130.664	1.973
5000_25_2	37914560	3.323	100.000	0.000	143.679	2.843
5000_25_3	68295820	1.718	100.000	0.000	126.904	2.668
5000_25_4	33866053	2.292	100.000	0.000	139.453	3.374
5000_25_5	9533115	5.409	100.000	0.000	111.366	3.227
5000_50_1	45194685	2.432	100.000	0.000	144.125	2.503
5000_50_2	88355678	0.830	100.000	0.000	143.188	2.488
5000_50_3	152447303	0.376	100.000	0.000	143.813	2.960
5000_50_4	171000228	0.915	100.000	0.000	148.015	3.392
5000_50_5	1187339	6.570	100.000	0.000	61.106	0.326
5000_75_1	28170819	1.230	100.000	0.000	105.745	1.674
5000_75_2	195434758	0.406	100.000	0.000	149.977	2.834
5000_75_3	64324704	0.970	76.000	3.9E-5	141.571	36.113
5000_75_4	247348595	0.796	100.000	0.000	144.213	2.942
5000_75_5	46462750	0.478	4.000	1.0E-4	136.119	45.054
5000_100_1	214425886	0.083	100.000	0.000	150.076	2.974
5000_100_2	18783132	0.367	100.000	0.000	76.661	1.164
5000_100_3	10784650	0.203	100.000	0.000	61.450	0.648
5000_100_4	160539947	0.065	100.000	0.000	153.082	2.780
5000_100_5	33166524	0.552	67.000	9.0E-5	105.708	38.976
6000_25_1	69832542	1.788	100.000	0.000	204.230	5.621
6000_25_2	3697236	5.232	100.000	0.000	123.770	2.797
6000_25_3	79300092	1.964	100.000	0.000	246.285	3.490
6000_25_4	191531304	0.276	87.000	1.2E-5	238.917	9.093
6000_25_5	36121510	1.423	100.000	0.000	208.762	3.447
6000_50_1	194344567	1.622	100.000	0.000	214.187	4.060
6000_50_2	323753804	0.376	66.000	9.6E-5	272.235	4.864
6000_50_3	31913824	2.502	100.000	0.000	220.343	3.015
6000_50_4	225556641	0.999	100.000	0.000	198.893	5.776
6000_50_5	40931924	1.885	100.000	0.000	186.351	6.849
6000_75_1	204512250	0.999	12.000	5.2E-5	267.433	86.722
6000_75_2	42422207	1.187	100.000	0.000	182.990	2.754
6000_75_3	524508156	0.477	60.000	3.4E-3	177.873	24.377
6000_75_4	197004931	1.083	100.000	0.000	220.513	4.100
6000_75_5	74350712	0.344	100.000	0.000	282.668	3.822
6000_100_1	292257056	0.069	100.000	0.000	219.599	4.054
6000_100_2	219791358	0.149	1.000	9.7E-4	257.679	122.758
6000_100_3	376967122	0.087	96.000	1.0E-6	266.202	46.105
6000_100_4	355609720	0.058	100.000	0.000	245.857	3.710
6000_100_5	686364195	0.100	38.000	4.2E-5	211.295	69.904
Avg.	-	1.359	87.675	1.2E-4	174.075	14.456

ing the best hyperplane where IHEA locates the best solution (column k^*), the average dimension of the initial hyperplane ($Avg.k_0$), the hyperplane dimension lower bound (k_{LB}), the hyperplane dimension upper bound (k_{UB}) and the average number of hyperplanes explored ($Avg.k$). Table 7 shows that the average number of hyperplanes that IHEA explores is always less than 3 for these instances. There is one case where $Avg.k$ is exactly 2 which means $TabuSearch\_Engine$ finds improved solutions in the first hyperplane but not in the second. For the other 7 cases, $TabuSearch\_Engine$ sometimes discovers improved solutions in the second hyperplane, which explains why their

Table 7
Additional information of IHEA on the 8 representative instances of Group ii

INST.	k^*	Avg.k_0	k_{LB}	k_{UB}	Avg.k
1000_25_5	880	879.010	520	881	2.904
1000_50_8	627	626.000	242	627	2.928
1000_75_5	858	857.000	490	858	2.924
1000_100_6	796	796.000	397	796	2.000
2000_25_3	929	928.010	236	931	2.896
2000_50_5	1153	1152.000	378	1153	2.969
2000_75_9	864	863.000	207	864	2.967
2000_100_2	1154	1153.000	387	1154	2.970
Avg.	907.63	906.75	357.13	908.00	2.82

Avg.k value is more than 2 but less than 3. Based on this observation, we conclude that each iteration of the "hyperplane exploration" phase of IHEA typically explores a very limited number of hyperplanes which contributes to its performance. To understand why such a small number of hyperplanes is explored, we examine the initial hyperplane ($Avg.k_0$). It can be seen that the initial hyperplane is always equal or very close to the best hyperplane (see column k^*). Indeed, there is 1 out of 8 cases where the initial hyperplane is exactly the best hyperplane where the best solution is found, and 7 cases where the initial hyperplane is only one dimension away from the best hyperplane. Moreover, Table 7 indicates that the dimension of the initial hyperplane (column $Avg.k_0$), which is always within the interval $[k_{LB}, k_{UB}]$, is very close to k_{UB}. The difference between the average values of $Avg.k_0$ and k_{UB} is only 908.00-906.75.00=1.25. Similar observations can be made for other instances of Group II.

4.2 Impact of the variable fixing strategy

As shown in Section 2.5, for a given hyperplane, IHEA has a "high quality" solution as its input. Before exploring the hyperplane with tabu search, we use the density criterion to fix a number of selected items (to the value of 1) and thus generate a reduced problem for tabu search. This variable fixing strategy is motivated by the idea that selected items with high density in a high quality solution are strongly determined and should not be destroyed during tabu search. To verify the usefulness of this variable fixing strategy, we conducted an experiment to compare IHEA with a variant IHEA$_{NoVF}$ where the variable fixing strategy is disabled (i.e., removing lines 15-17 from Algorithm 1). As such $TabuSearch\_Engine$ explores directly the search space of $CQKP[k]$ instead of $RCP(V_{fixed}, CQKP[k], x')$ (see Section 2.3).

We ran IHEA and IHEA$_{NoVF}$ 100 times to solve all instances of Group II under the same condition as before. We divide the whole instance set into 8 classes according to the (n, d) combination. Table 8 summarizes the results. In addition of the average best solution value (Avg.Best), average success rate (Avg.SR)

Table 8
Comparative results of IHEA and IHEA$_{NoVF}$ on the benchmarks of Group II

INST.	IHEA					IHEA$_{NoVF}$		
	Avg.Best	Avg.SR	t(s)	Avg.%$_{FV}$	Avg.%$_{WFV}$	Avg.Best	Avg.SR	t(s)
1000_25	2016960.700	**100.000**	5.489	0.928	0.000	2016960.700	99.400	101.332
1000_50	5118953.900	**99.700**	6.022	0.938	0.000	5118953.900	90.900	114.268
1000_75	5900793.200	**100.000**	6.390	0.932	0.000	5900793.200	83.100	120.730
1000_100	7476457.700	**100.000**	6.117	0.932	0.000	7476457.700	99.400	102.621
2000_25	7841340.400	**100.000**	22.244	0.964	0.000	7841340.400	99.400	591.328
2000_50	18617410.400	**100.000**	23.240	0.967	0.000	18617410.400	99.900	617.926
2000_75	**24676371.600**	**100.000**	22.718	0.967	0.000	24676368.000	88.600	546.975
2000_100	**21220476.400**	**99.500**	22.719	0.963	0.000	21220453.500	80.000	466.538
Avg.	**11608595.538**	**99.900**	**14.367**	0.949	0	11608592.225	92.587	332.715

and average total computing time (Avg.t(s)), we also list for IHEA the average percentage of fixed variable (Avg.%$_{FV}$) and average percentage of wrongly fixed variable (Avg.%$_{WFV}$), where the percentage of the fixed variables is calculated by: #fixed_variable/k × 100 and the percentage of wrongly fixed variable is calculated by: #wrongly_fixed_variable/#fixed_variable × 100. The wrongly fixed variables are identified by comparing the fixed variables to the best known solution reported in Tables 4 and 5. The last row (*Avg.*) indicates the average value of some columns.

Table 8 shows that IHEA performs better than IHEA$_{NoVF}$ in terms of both average best solution value (Avg.Best) and average success rate (Avg.SR) for 2 out of 8 instance classes. For the remaining 6 instance classes where both algorithms achieve the same best results, IHEA always attains a higher average success rate than IHEA$_{NoVF}$. A Wilcoxon signed rank test with a significance factor of 0.05 was applied to these two groups of success rates and the resulting p-value of 0.001602 clearly shows that IHEA is significantly better than IHEA$_{NoVF}$. Moreover, when we examine the average computing time, we find that IHEA is 23 times faster than IHEA$_{NoVF}$ (14.376 vs 332.715 seconds). Such a drastic speed-up is achieved thanks to the fact that a large number of variables are fixed and few of them are wrongly fixed. Indeed, the average value of the average number of fixed variables in percentage (see column Avg.%$_{FV}$) is 94.9% which means that 94.9% of the search space is eliminated on average, and the average number of wrongly fixed variables (see column Avg.%$_{WFV}$) is always 0 which means no variable is wrongly fixed for these instances.

4.3 Impact of the perturbation strategy

IHEA uses a density based perturbation strategy to introduce a form of global diversification for a better exploration of the solution space (Section 2.7). To assess the impact of the adopted perturbation strategy, we conducted an experiment to compare the performance of IHEA with two variants IHEA$_{RDPT}$

Table 9
Comparative results of IHEA with IHEA$_{RDPT}$ and IHEA$_{NOPT}$ on the 80 large instances of Group II [37]. The values in bold denote the best results of row *Sum*.

INST.	IHEA			IHEA$_{RDPT}$			IHEA$_{NOPT}$		
	Avg.Best	Avg.SR	Avg.t(s)	Avg.Best	Avg.SR	Avg.t(s)	Avg.Best	Avg.SR	Avg.t(s)
1000_25	2016960.700	100.000	5.489	2016960.700	96.200	7.649	2016958.100	78.300	0.134
1000_50	5118953.900	99.700	6.022	5118953.900	78.200	8.327	5118885.300	61.100	0.144
1000_75	5900793.200	100.000	6.390	5900793.200	98.400	9.218	5900733.000	75.700	0.149
1000_100	7476457.700	100.000	6.117	7476457.700	93.600	9.010	7476457.700	79.900	0.146
2000_25	7841340.400	100.000	22.244	7841340.400	89.900	31.684	7841340.400	57.900	0.569
2000_50	18617410.400	100.000	23.240	18617410.400	97.600	32.179	18617161.000	88.300	0.576
2000_75	24676371.600	100.000	22.718	24676371.600	97.500	31.554	24675823.000	81.100	0.563
2000_100	21220476.400	99.500	22.719	21220476.400	100.000	31.750	21218753.400	77.800	0.533
Avg.	**11608595.538**	**99.900**	14.367	**11608595.538**	93.925	20.171	11608263.988	75.012	0.352

and IHEA$_{NOPT}$. IHEA$_{RDPT}$ randomly removes s selected items without considering their densities while IHEA$_{NOPT}$ eliminates the perturbation phase from IHEA. This experiment was also performed on the 80 instances of Group II. As usual, each algorithm was executed 100 times on each instance.

Table 9 summarizes the results based on three indicators: 1) average best solution value (Avg.Best); 2) average success rate (Avg.SR); 3) average computing time for one trial (Avg.t(s)). The last row of the table (*Avg.*) indicates the average of the listed values of each column. From Table 9, we observe that eliminating the perturbation phase from the IHEA algorithm causes a great deterioration of its performance in terms of both best solution value and success rate. Indeed, compared to IHEA and IHEA$_{RDPT}$, IHEA$_{NOPT}$ achieved a smaller average best solution (Avg.Best) value for 6 out of 8 instance classes and even a smaller average success rate (Avg.SR) for all 8 classes. When comparing IHEA$_{RDPT}$ with IHEA, one observes that, although IHEA$_{RDPT}$ does not deteriorate the best solution value, it is less stable than IHEA by achieving a smaller average success rate for 7 out of 8 instance classes. A Wilcoxon signed rank test with a significance factor of 0.05 was applied to compare the success rates of IHEA$_{RDPT}$ and IHEA, and the resulting p-value of 0.001602 discloses that IHEA$_{RDPT}$ is significantly worse than IHEA. Moreover, IHEA$_{RDPT}$ required on average more computing time than IHEA (20.171 seconds v.s. 14.367 seconds). This experiment confirms that the perturbation phase of IHEA is useful and the adopted density based strategy is effective.

5 Conclusions

This paper deals with the NP-hard Quadratic Knapsack Problem which is a highly useful model in practice. To approximate this hard combinatorial problem, we developed an iterated “hyperplane exploration” approach mixing problem reduction techniques and local optimization with tabu search. The proposed approach introduces a hyperplane constraint to the original QKP model to generate a series of "interesting" hyperplane-constrained problems whose solution space represents a small subset of the original solution space of QKP. To further reduce the hyperplane-constrained problem, we employ specific variable fixing rules based on item density information to fix "strongly determined" variables. The dedicated tabu search procedure is then used to explore the reduced hyperplane-constrained problem. Finally a perturbation strategy is applied to help the search to escape from deep local optima.

We assessed the performance of the proposed approach and presented comparisons with three best performing methods on two sets of 180 well-known benchmark instances (up to 2000 items) and a set of 40 new instances (with 5000 and 6000 items). The computational experiments showed that the proposed approach competes very favorably with the state-of-the-art algorithms. Specifically, IHEA consistently attained the known optimal solution with a 100% success rate for all 100 small-sized benchmarks (with 100 to 300 objects). For the set of 80 large benchmarks with 1000 and 2000 objects, IHEA discovered 6 improved results (new best lower bounds) and attained the remaining 74 best known results. Encouraging results on 40 new very large instances (with 5000 and 6000 objects) additionally confirmed the effectiveness of our approach where the average gap between the best lower bound and the well known upper bound $\hat{U}^2_{CPT}$ is bounded by 1.359%. The experiments also showed that the proposed approach is more computationally effective than the existing heuristics. Furthermore, we performed additional experiments to gain insight into the "hyperplane exploration" component of the proposed approach, and investigate the beneficial role of two key strategies of the proposed approach: the variable fixing strategy and the perturbation strategy.

We comment that even if IHEA follows the common assumption of non-negative profits (see Section 2.2, Proposition 1), this is not a necessary condition to apply it. Indeed, in the case of negative profits, Proposition 1 does not hold necessarily and as such the hyperplanes containing the optimal solutions can no more be bounded correctly. Nevertheless, IHEA can still be applied to locate high quality solutions within a set of promising hyperplanes which can be identified by any specific means. In this sense, the proposed IHEA approach is general and applicable to any QKP instance even though its performance may decrease for instances with negative profits.

For future work, there are several interesting directions that could be investigated. First, given the current IHEA algorithm, we can improve the hyperplane exploration (tabu search) by introducing adaptive memory techniques based on recency and/or frequency information projection [19]. Some advanced search frameworks like path relinking and scatter search [16] could also be integrated to reinforce the tabu search engine. Generally, under the IHEA approach, the task of hyperplane exploration can be performed by any effective search algorithm which can be either a heuristic or an exact solver. Second, it would be useful to investigate additional methods able to identify the most "promising" hyperplanes and thus reduce the number of hyperplanes to be explored. Finally, given that the idea of hyperplane decomposition is quite general, it would be interesting to investigate its merit for solving other knapsack and related problems.

Acknowledgment

We are grateful to the reviewers for their insightful comments which helped us to improve the paper. We would also like to thank Dr. Letchford and Dr. Fomeni for providing us with their source code of [12]. The work was partially supported by the LigeRo project (2009-2013, No. 012, Region of Pays de la Loire, France) and the PGMO project (2014-0024H, Jacques Hadamard Mathematical Foundation Paris). Support for Yuning Chen from the China Scholarship Council is also acknowledged.

References

[1] Billionnet A, Calmels F. Linear programming for the 0-1 quadratic knapsack problem. European Journal of Operational Research 1996; 92: 310-325.

[2] Billionnet A, Soutif E. An exact method based on Lagrangian decomposition for the 0-1 quadratic knapsack problem. European Journal of Operational Research 2004; 157: 565-575.

[3] Birattari M., Yuan Z., Balaprakash P., Sttzle T. F-Race and iterated F-Race: An overview. Experimental methods for the analysis of optimization algorithms 2010; Berlin, Germany: Springer, pp. 311-336.

[4] Boussier S., Vasquez M., Vimont Y., Hanafi S., Michelon P. A multi-level search strategy for the 0-1 multidimensional knapsack problem. Discrete Applied Mathematics 2010; 158(2): 97-109.

[5] Chaillou P., Hansen P., Mahieu Y. Best network flow bound for the quadratic knapsack problem. Lecture Notes in Mathematics 1983; 1403:226-235.

[6] Caprara A, Pisinger D, Toth P. Exact solution of the quadratic knapsack problem. INFORMS Journal on Computing 1999; 11: 125-137.

[7] Chen Y., Hao J.K. A "reduce and solve" approach for the multiple-choice multidimensional knapsack problem. European Journal of Operational Research 2014; 239(2): 313-322.

[8] Dammeyer F., Voβ S. Dynamic tabu list management using the reverse elimination method. Annals of Operations Research 1993; 41: 31-46.

[9] Dijkhuizen G, Faigle U. A cutting-plane approach to the edge-weighted maximal clique problem. European Journal of Operational Research 1993; 69: 121-30.

[10] Feo T, Resende M.G.C. Greedy randomized adaptive search procedures. Journal of Global Optimization 1995; 2: 1-27.

[11] Fleszar K., Hindi K.S. Fast, effective heuristics for the 0-1 multi-dimensional knapsack problem. Computers & Operations Research 2009; 36(5): 1602-1607.

[12] Fomeni F.D., Letchford A.N. A dynamic programming heuristic for the quadratic knapsack problem. INFORMS Journal on Computing 2014; 26(1): 173-182.

[13] Foster B.A., Ryan D.M. An integer programming approach to scheduling. Computer Scheduling of Public Transport (A. Wren, ed.). North-Holland Publishing Company 1981; 269-280.

[14] Gallo G., Hammer P., Simeone B. Quadratic knapsack problems, Mathematical Programming Studies 1980; 12: 132-149.

[15] Glover F. Heuristics for integer programming using surrogate constraints. Decision Sciences 1977; 8(1): 156-166.

[16] Glover F., Laguna M. Tabu Search. Kluwer Academic Publishers, Boston, 1997.

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

[18] Glover F., Kochenberger G., Alidaee B., Amini M. Solving quadratic knapsack problems by reformulation and tabu search. single constraint case. Series on Applied Mathematics 2002; (14): 111-122, World Scientific Publishing Company.

[19] Glover F., Lü Z.P., Hao J.K. Diversification-driven tabu search for unconstrained binary quadratic problems. 4OR - A Quarterly Journal of Operations Research 2010; 8(3): 239-253.

[20] Hammer P.L., Rader D.J. Efficient methods for solving quadratic 0-1 knapsack problem. INFOR 1997; 35: 170-182.

[21] Helmberg C, Rendl F, Weismantel R. A semidefinite programming approach to the quadratic knapsack problem. Journal of Combinatorial Optimization 2000; 4: 197-215.

[22] Johnson E, Mehrotra A, Nemhauser G. Min-cut clustering. Mathematical Programming 1993; 62: 133-151.

[23] Julstorm B.A. Greedy, genetic, and greedy genetic algorithms for the quadratic knapsack problem. Genetic and Evolutionary Computation Conference, Washington DC, USA; 2005. pp. 607-614.

[24] Kellerer H, Strusevich V.A. Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. Algorithmica 2010; 57(4): 769-795.

[25] López-Ibáñez M., Dubois-Lacoste J., Stützle T., Birattari M. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Universit Libre de Bruxelles, Belgium.

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

[27] Park K., Lee K., Park S. An extended formulation approach to the edge-weighted maximal clique problem. European Journal of Operational Research 1996; 95: 671-82.

[28] Pisinger D. The quadratic knapsack problem-a survey. Discrete Applied Mathematics 2007; 155: 623-648.

[29] Pisinger D, Rasmussen A.B., Sandvik R. Solution of large-sized quadratic knapsack problems through aggressive reduction. INFORMS Journal on Computing 2007; 19(2): 280-290.

[30] Sil M. Column generation techniques for pickup and delivery problems. Ph.D. thesis, Technische Universiteit Eindhoven 1994.

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

[32] Vasquez M., Hao J.K. An hybrid approach for the 0-1 multidimensional knapsack problem. Proc. of the 17th International Joint Conference of Artificial Intelligence (IJCAI-2001), Morgan Kaufmann, San Francisco, pp. 328-333.

[33] Wang Y., Lü Z.P., Glover F., Hao J.K. Backbone guided tabu search for solving the UBQP problem. Journal of Heuristics 2013; 19(4): 679-695.

[34] Wilbaut C., Hanafi S. New convergent heuristics for 0-1 mixed integer programming. European Journal of Operational Research 2009; 195(1): 62-74.

[35] Xie X., Liu J. A Mini-Swarm for the quadratic knapsack problem. IEEE Swarm Intelligence Symposium, Honolulu, HI, USA; 2007. pp. 190-197.

[36] Xu Z. A strongly polynomial FPTAS for the symmetric quadratic knapsack problem. European Journal of Operational Research 2012; 218(2): 377-381.

[37] Yang Z., Wang G., Chu F. An effective GRASP and tabu search for the 0-1 quadratic knapsack problem. Computers & Operations Research 2013; 40: 1176-1185.