Diversity-preserving quantum particle swarm optimization for the multidimensional knapsack problem

Xiangjing Lai[a], Jin-Kao Hao[b,c,*], Zhang-Hua Fu[d], Dong Yue[a]

[a] *Institute of Advanced Technology, Nanjing University of Posts and Telecommunications, Nanjing 210023, China*
[b] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*
[c] *Institut Universitaire de France, 1 Rue Descartes, 75231 Paris, France*
[d] *Robotics Laboratory for Logistics Service, Institute of Robotics and Intelligent Manufacturing, The Chinese University of Hong Kong, Shenzhen, 518172, China.*

Expert Systems with Applications, https://doi.org/10.1016/j.eswa.2020.113310

Abstract

Quantum particle swarm optimization is a population-based metaheuristic that becomes popular in recent years in the field of binary optimization. In this paper, we investigate a novel quantum particle swarm optimization algorithm, which integrates a distanced-based diversity-preserving strategy for population management and a local optimization method based on variable neighborhood descent for solution improvement. We evaluate the proposed method on the classic NP-hard 0–1 multidimensional knapsack problem. We present extensive computational results on the 270 benchmark instances commonly used in the literature to show the competitiveness of the proposed algorithm compared to several population based algorithms. The ideas of using the diversity-preserving strategy and the probabilistic application of a local optimization procedure are of general interest and can be used to reinforce other quantum particle swarm algorithms.

Keywords: Binary optimization; Multidimensional knapsack problem; population-based metaheuristics; Quantum particle swarm optimization; Diversity-preserving population updating strategy.

1. Introduction

Given a knapsack with a m-dimensional capacity vector $\mathbf{c}$ and a set V of n items, let $p_j > 0$ $(j = 1, 2, \ldots, n)$ be the profit of item j, and let $\mathbf{a}$ be a $m \times n$ matrix composed of positive values where the jth column a_{*j} represents the

*Corresponding author.
Email addresses: laixiangjing@gmail.com (Xiangjing Lai), jin-kao.hao@univ-angers.fr (Jin-Kao Hao), fuzhanghua@cuhk.edu.cn (Zhang-Hua Fu), medongy@vip.163.com (Dong Yue)

m-dimensional weights of item j. The classic 0–1 multidimensional knapsack problem (MKP) involves packing a subset of items of V to the knapsack so that the sum of the profits of the items in the knapsack is maximized while the sum of weights in each dimension i ($i = 1, 2, \ldots, m$) does not exceed the capacity c_i. Formally, the MKP can be stated as follows:

$$\text{Maximize} \quad f(s) = \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} \mathbf{a}_{ij} x_j \leq c_i, \forall i \in \{1, 2, \ldots, m\} \tag{2}$$

$$x_j \in \{0, 1\}, \forall j \in \{1, 2, \ldots, n\} \tag{3}$$

where x_j ($j = 1, 2, \ldots, n$) are binary decision variables such that $x_j = 1$ if item j is packed in the knapsack, $x_j = 0$ otherwise. The objective in Eq. (1) aims to maximize the total profit of the selected items, while the constraints in Eq. (2) ensure that the selected items satisfy the m capacity constraints of the knapsack.

The MKP has numerous applications, including cutting stock (Gilmore and Gomory, 1966), loading (Shih, 1979), resource allocation (Gavish and Pirkul, 1982) and so on. However, the problem is known to be NP-hard (Garey, 1979) and thus computationally challenging. As one of the most studied combinatorial optimization problems, a large number of solution approaches have been proposed for the MKP. A comprehensive review of representative studies up to 2004 can be found in (Fréville, 2004) and more recent studies are discussed in (Lai et al., 2018). Notice that the MKP has some interesting variants such as the multiple multidimensional knapsack problem (Mancini et al., 2019), the multiple-choice multidimensional knapsack problem (Chen and Hao, 2014), the robust multiple-choice multidimensional knapsack problem (Caserta and Voβ, 2019), and the multidemand multidimensional knapsack problem (Lai et al., 2019). Below, we discuss some recent and most representative studies on the MKP.

Existing algorithms for the MKP can be classified into exact and heuristic algorithms. Representative exact algorithms are mainly based on the branch & bound method (Shih, 1979; Vimont et al., 2008) and hybrid approaches combining branch & bound and other strategies (Boussier et al., 2010; Mansini and Speranza, 2012). The best performing exact algorithms like those presented in (Boussier et al., 2010; Mansini and Speranza, 2012; Vimont et al., 2008) are quite successful to yield optimal solutions in an acceptable computation time for benchmark instances of limited sizes (e.g., $n = 250$ or 500 and $m \in \{5, 10\}$). However, for larger instances with $n \geq 250$ and $m \geq 30$, heuristic algorithms become more suitable methods to find sub-optimal (or non provable optimal) solutions.

Heuristic algorithms for the MKP belong to two large categories, namely trajectory-based local search algorithms and population-based evolutionary algorithms. Representative trajectory-based algorithms include tabu search (Glover

and Kochenberger, 1966; Hanafi and Fréville, 1998; Khemakhem et al., 2012; Vasquez and Hao, 2001; Vasquez and Vimont, 2005), simulated annealing (Drexl, 1988), and kernel search (Angelelli et al., 2010), while representative population-based algorithms include binary particle swarm optimization (Chih, 2015; Haddar et al., 2016; Ktari and Chabchoub, 2013; Lin et al., 2016), genetic and memetic algorithms (Chu and Beasley, 1998; Drake et al., 2016; Lai et al., 2018; Puchinger et al., 2009), steady-state evolutionary algorithm (Raidl and Gottlieb, 2005), ant colony optimization (Al-Shihabi and Ólafsson, 2010; Ke et al., 2010; Kong et al., 2008), and hybrid estimation of distribution algorithm (Wang et al., 2012), among other.

Our goal of this work is twofold. First, according to our literature review, most existing MKP algorithms in the literature fail to achieve simultaneously a high performance in terms of both solution quality and computation speed. For example, tabu search based algorithms like those in (Khemakhem et al., 2012; Lai et al., 2018; Vasquez and Hao, 2001) are among the best MKP methods to obtain high quality solutions especially for instances with a large number of constraints. However, these methods are generally quite time consuming. On the other hand, bio-inspired evolutionary algorithms like (Chih, 2018; Chu and Beasley, 1998) are often more time effective, but yield less competitive solutions than tabu search based algorithms. Second, in several interesting studies (Haddar et al., 2016; Yang et al., 2004), quantum particle swarm optimization (QPSO) has shown promising performances on the MKP. In this work, in addition to developing an effective algorithm for the MKP, we aim also to further enhance the general QPSO approach by introducing a diversity-preserving strategy.

We summarize our work as follows. First, we propose a diversity-preserving quantum particle swarm optimization (DQPSO*) approach, which enhances the conventional QPSO method. The diversity-preserving strategy is used to control the population diversity of a QPSO algorithm and helps to avoid premature convergence of the algorithm. The proposed algorithm integrates an effective local optimization procedure which is applied in a probabilistic way to reinforce its exploitation capacity. We show extensive computational results and comparisons with representative (mainly population-based) algorithms based on well-known benchmark instances. It is worth noting that the ideas of diversity-preserving strategy and local optimization are of general interest. As a result, they could be advantageously adopted in other QPSO algorithms to control the balance of exploitation and exploration of the search process, such that they can help to effectively solve other binary optimization problems such as the MKP variants mentioned above.

The remainder of the paper is organized as follows. In Section 2, we provide a brief introduction on the quantum particle swarm optimization. In Section 3, we present the proposed DQPSO* algorithm. In Section 4, we evaluate the proposed algorithm by providing experimental results and making a comparison with several state-of-the-art MKP algorithms. In Section 5, we analyze two essential components of the algorithm to show their influences on the performance of the algorithm, followed by concluding comments and discussions on future

research.

2. A Review of Quantum Particle Swarm Optimization

In this section, we provide a brief introduction of particle swarm optimization (PSO) for continuous problems and quantum particle swarm optimization which is an adaptation of PSO to binary optimization problems.

2.1. Basic Particle Swarm Optimization

Particle swarm optimization was originally developed for optimization of continuous nonlinear functions (Kennedy and Eberhart, 1995). For a given problem in a n-dimensional continuous space where n represents the number of variables, PSO searches for the global optimum through mimicking the behavior of a swarm or population of particles (e.g., birds), where each particle i represents a candidate solution characterized by a n-dimensional position vector $\vec{X}_i^t = (x_{i1}^t, x_{i2}^t, \ldots, x_{in}^t)$ and a velocity vector $\vec{V}_i^t = (v_{i1}^t, v_{i2}^t, \ldots, v_{in}^t)$ where t is the t-th iteration of the algorithm.

To reach the global optimal solution, the particles in the swarm move iteratively in the search space, and the position vector $\vec{X}_i^t$ and velocity vector $\vec{V}_i^t$ of particle i at t-th iteration are updated by the following formulas (Liu et al., 2010; Kennedy and Eberhart, 1995; Shi and Eberhart, 1998; Zhan et al., 2011).

$$v_{ij}^{t+1} = \omega v_{ij}^t + c_1 r_1 (pBest_{ij}^t - x_{ij}^t) + c_2 r_2 (nBest_j^t - x_{ij}^t) \tag{4}$$

$$x_{ij}^{t+1} = x_{ij}^t + v_{ij}^{t+1} \tag{5}$$

where $j \in \{1, 2, \ldots, n\}$, $\omega \in [0, 1]$ is the inertia factor, c_1 and c_2 are two positive constants, r_1 and r_2 are two random numbers in $[0, 1]$, $pBest_i^t$ is the personal historical best position vector for particle i, and $nBest^t$ is the neighborhood's historical best position for particle i. It is worth noting that the neighborhood relation between particles is defined by some topological structure, such as a ring topology where only the particles $i-1$ and $i+1$ are the neighbors of particle i and a clique topology where the particles are pair-wisely connected.

According to the topological structure between particles, a PSO algorithm can be roughly divided into two categories (Zhan et al., 2011), i.e., global version PSO (GPSO) and local version PSO (LPSO). In GPSO, the clique topology is adopted, i.e., any two particles in the swarm are neighbors, and thus the neighborhood's historical best position $nBest^t$ is also the historical best position $gBest^t$ of the entire swarm. For LPSO, the neighborhood's historical best position $nBest^t$ of particles depends on the used topological structure.

2.2. Quantum Particle Swarm Optimization

Due to the fact that the basic PSO method is not applicable to binary optimization problems, a number of PSO variants have been proposed in the past 20 years to deal with binary optimization (Beheshti et al., 2015; Chen et al., 2010; Lin and Guan, 2018; Kennedy and Eberhart, 1997; Yang et al., 2004)

among which quantum particle swarm optimization (QPSO) is a representative example (Yang et al., 2004).

In a QPSO algorithm, a swarm $Q = \{Q(1), Q(2), \ldots, Q(np)\}$ of np quantum particles is maintained and evolves, where each quantum particle $Q(i)$ is a n-dimensional real-valued vector $(q_1^i, q_2^i, \ldots, q_n^i)$ with $q_j^i \in [0, 1]$. For each component q_j^i $(1 \leq j \leq n)$ of quantum particle $Q(i)$, its value represents the probability that the associated binary decision variable x_j takes the value of 0.

As described in Algorithm 1, a QPSO algorithm typically performs a number of evolution iterations until a maximum number of iterations is reached. Starting with a randomly initialized Q_t in which the notation t denotes the current number of iterations, the algorithm first transforms each quantum particle $Q(i) = (q_1^i, q_2^i, \ldots, q_n^i)$ of Q_t into a n-dimensional binary vector (called discrete particle) $D(i) = (d_1^i, d_2^i, \ldots, d_n^i)$ by applying a random observation:

$$d_j^i = \begin{cases} 1, & \text{if } q_j^i < rand(0,1); \quad (6) \\ 0, & \text{otherwise;} \quad (7) \end{cases}$$

where $rand(0, 1)$ denotes a random real number in $[0, 1]$. Then, at each iteration t, the evolution of the quantum particle swarm is described by the following evolution formulas:

$$Q_{t+1}^*(i) = \alpha \times D_t^*(i) + (1 - \alpha) \times (\vec{e} - D_t^*(i)) \quad (8)$$

$$Q_{t+1}^{lb}(i) = \alpha \times D_t^{lb}(i) + (1 - \alpha) \times (\vec{e} - D_t^{lb}(i)) \quad (9)$$

$$Q_{t+1}(i) = c_1 \times Q_t(i) + c_2 \times Q_{t+1}^{lb}(i) + (1 - c_1 - c_2) \times Q_{t+1}^*(i) \quad (10)$$

where α, c_1 and c_2 are three parameters satisfying $\alpha \in [0, 1]$, $c_1 \in [0, 1]$, $c_2 \in [0, 1]$, and $0 < c_1 + c_2 < 1$. In addition, $\vec{e} = (1, 1, \ldots, 1)$ is a n-dimensional vector in which each entry takes 1, $D_t^{lb}(i)$ and $D_t^*(i)$ $(1 \leq i \leq np)$ denote respectively the personal and neighborhood's historical best positions for the discrete particle $D(i)$ at iteration t, and $Q_{t+1}^{lb}(i)$ and $Q_{t+1}^*(i)$ $(1 \leq i \leq np)$ represent respectively the personal and neighborhood's historical best positions for the quantum particle $Q(i)$ at iteration $t + 1$. After a new quantum particle $Q_{t+1}(i)$ is generated by Eqs. (8)–(10), $Q_{t+1}(i)$ is used to replace $Q_t(i)$ and is at the same time transformed into a discrete particle d, which is then used to update $D_{t+1}^*(i)$ and $D_{t+1}^{lb}(i)$ accordingly.

3. Diversity-Preserving Quantum Particle Swarm Optimization for the MKP

The DQPSO* algorithm for the MKP proposed in this work shares ideas from the studies (Haddar et al., 2016; Yang et al., 2004) and distinguishes itself with two new features. First, DQPSO* introduces a diversity-preserving mechanism to guarantee a healthy diversity of the particle swarm, thus avoiding a premature convergence of the algorithm. Second, DQPSO* applies in a

Algorithm 1: General procedure of the QPSO algorithm for a binary optimization problem with a form of maximization

Function *QPSO*
Input: Instance I, size of particle swarm (np), maximum number of iterations ($IterMax$), α, c_1, and c_2.
Output: The best discrete solution d^* found
/* $Q_t = \{Q_t(i) : 1 \leq i \leq np\}$ denotes the quantum particle swarm at the iteration t, $D_t^{lb} = \{D_t^{lb}(i) : 1 \leq i \leq np\}$ denotes the set of personal historical best positions for discrete particles */
$t \leftarrow 0$ /* t denotes the current number of iterations */
/* Initialization of quantum particle swarm */
$Q_t \leftarrow InitialQuantumSwarm(np)$
for $i \leftarrow 1$ ***to*** np **do**
  $D_t^{lb}(i) \leftarrow Transform(Q_t(i))$ /* Eqs.(6)-(7) */
end
$d^* \leftarrow argmax\{f(d) : d \in D_t^{lb}\}$ /* d^* denotes the best discrete particle found so far */
/* Evolution of particle swarm */
while $t < IterMax$ **do**
  for $i \leftarrow 0$ ***to*** np **do**
    $D_t^*(i) \leftarrow argmax\{f(d^{'}) : d^{'} \in D_t^{lb}\}$ /* $D_t^*(i)$ denotes the best discrete particle in D_t^{lb} */
    $Q_{t+1}^*(i) \leftarrow \alpha \times D_t^*(i) + (1-\alpha) \times (\vec{e} - D_t^*(i)))$
    $Q_{t+1}^{lb}(i) \leftarrow \alpha \times D_t^{lb}(i) + (1-\alpha) \times (\vec{e} - D_t^{lb}(i)))$
    $Q_{t+1}(i) \leftarrow c_1 \times Q_t(i) + c_2 \times Q_{t+1}^{lb}(i) + (1-c_1-c_2) \times Q_{t+1}^*(i)$
    $d \leftarrow Transform(Q_{t+1}(i))$ /* d is a discrete solution */
    if $f(d) > f(D_t^{lb}(i))$ **then**
      $D_t^{lb}(i) \leftarrow d$ /* $D_t^{lb}(i) \leftarrow LocalSearch(d)$ is used for some variants of QPSO like QPSO* in (Haddar et al., 2016) */
    end
    if $f(d) > f(d^*)$ **then**
      $d^* \leftarrow d$ /* $d^* \leftarrow LocalSearch(d)$ is used for some variants of QPSO like QPSO* in (Haddar et al., 2016) */
    end
  end
  $t \leftarrow t + 1$
end

probabilistic way a powerful local optimization procedure to enhance the intensification search ability of the algorithm. The proposed algorithm and its components are described in the following subsections.

3.1. Solution Representation and Search Space

Given a MKP instance with n items, a candidate solution can be represented by a n-dimensional 0–1 vector $s = (x_1, x_2, \ldots, x_n)$ where $x_i = 1$ if item i is selected, $x_i = 0$ otherwise. As a result, the search space Ω explored by the DQPSO* algorithm is composed of all possible n-dimensional 0–1 vectors (also called discrete solutions or discrete particles in this paper), including the feasible and infeasible solutions, i.e.,

$$\Omega = \{(x_1, x_2, \ldots, x_n) : x_i \in \{0, 1\}, 1 \leq i \leq n\} \tag{11}$$

In addition, DQPSO* uses a n-dimensional real-valued vector $q = (q_1, q_2, \ldots, q_n)$ (called quantum solution or quantum particle), where q_i $(1 \leq i \leq n)$ is a real number in $[0, 1]$ and represents the probability that the binary variable x_i takes 0. This vector indicates approximately a discrete solution in the search space.

3.2. Main Framework of the Algorithm

Algorithm 2: Pseudo-code of generating the initial quantum particle swarm

```
1 Function InitialQuantumSwarm
  Input: Size of particle swarm np, number of items n
  Output: A quantum particle swarm Q = {Q(i) : 1 ≤ i ≤ np}
2 for i ← 1 to np do
3 |   for j ← 1 to n do
4 |   |   Q(i)_j ← rand(0, 1)
5 |   end
6 |   Q(i) ← (Q(i)_1, Q(i)_2, ..., Q(i)_n)
7 end
```

The proposed DQPSO* algorithm consists of six components, including the initialization of the quantum particle swarm, the repair operator to ensure the feasibility of generated solutions, the updating strategy of the personal historical best positions (D_t^{lb}) of the discrete particles, the rule of transforming a quantum particle to a discrete particle, the local optimization method to improve the solutions generated by the repair operator, and the evolution formulas of the quantum particle swarm. The DQPSO* method is described in Algorithm 4, where $Q_t = \{Q_t(i) : 1 \leq i \leq np\}$ denotes the swarm of np quantum particles at iteration t, $D_t^{lb} = \{D_t^{lb}(i) : 1 \leq i \leq np\}$ is the set of personal historical best positions for discrete particles at iteration t, and d^* records the best discrete particle found so far. As mentioned in Section 3.1, each quantum particle $Q_t(i) \in Q_t$

Algorithm 3: Pseudo-code of transforming a quantum solution into a discrete solution

```
1 Function Transform
  Input: A quantum particle q = (q_1, q_2, ..., q_n), where q_j ∈ [0,1]
         (1 ≤ j ≤ n)
  Output: A discrete particle d = (d_1, d_2, ..., d_n), where d_j ∈ {0,1}
          (1 ≤ j ≤ n)
2 for j ← 1 to n do
3 |   if rand(0,1) > q_j then
4 |   |   d_j ← 1
5 |   end
6 |   else
7 |   |   d_j ← 0
8 |   end
9 end
```

is a n-dimensional real-valued vector $(q_1^i, q_2^i, \ldots, q_n^i)$, and each discrete particle $D_t^{lb}(i) \in D_t^{lb}$ is a n-dimensional 0–1 vector $(d_1^i, d_2^i, \ldots, d_n^i)$ representing a candidate solution in the search space.

The DQPSO* algorithm starts with an initial Q_t $(t = 0)$ which is randomly generated by the initialization method presented in Algorithm 2. Then, each quantum particle in Q_t is transformed into a discrete particle by the transforming procedure given in Algorithm 3 and the infeasibility of the resulting discrete particle is subsequently repaired by the repair operator of Section 3.3 (lines 4–7). At the same time, D_t^{lb} is accordingly initialized and the best discrete particle found in this process is recorded as d^* (lines 6 and 8).

After the initialization of Q_t and D_t^{lb}, DQPSO* performs $IterMax$ iterations (lines 9–27) to search for a best discrete solution of the MKP instance. Specifically, at each iteration t, the particles $Q_t(i)$ $(1 \leq i \leq np)$ are processed by applying the following steps: (1) K (which is a parameter) discrete solutions $S = \{d^1, d^2, \ldots, d^K\}$ are randomly selected from D_t^{lb}, the corresponding quantum particles are tentatively recorded as the neighbors of the particle $Q_t(i)$, and the best individual in S is tentatively recorded as the neighborhood's best position $D_t^*(i)$ for the corresponding discrete particle of $Q_t(i)$ (lines 12–13). One observes that the evolution of the particle swarm in DQPSO* is based on a random and dynamic neighborhood topology. (2) A new quantum particle $Q_{t+1}(i)$ is generated by the evolution formulas in Eqs. (8)-(10), where $Q_{t+1}^{lb}(i)$ and $Q_{t+1}^*(i)$ represent in some sense the personal and neighborhood's historical best positions for the quantum particle $Q_{t+1}(i)$ (lines 13–15). (3) The newly generated quantum solution $Q_{t+1}(i)$ is transformed into a discrete solution d and its infeasibility is subsequently repaired by the repair operator (lines 16–17). (4) The local optimization method (denoted by VND) is applied with a probability of p to further improve the solution (line 18–20). (5) The resulting solution is then used to update D_t^{lb} by means of a diversity-preserving updating strategy (line 21).

DQPSO* stops once a maximum allowed number of iterations is reached and

the best discrete solution found (d^*) is returned as the result of the algorithm.

Algorithm 4: Main frame of the DQPSO* algorithm for the MKP

Function *DQPSO**
Input: Instance I, size of particle swarm (np), maximum number of iterations ($IterMax$), parameters K, p, α, c_1, and c_2.
Output: The best discrete solution d^* found
/* $Q_t = \{Q_t(i) : 1 \leq i \leq np\}$ denotes the quantum particle swarm at the iteration t, $D_t^{lb} = \{D_t^{lb}(i) : 1 \leq i \leq np\}$ denotes the set of personal historical best positions for discrete particles */
$t \leftarrow 0$ /* t denotes the current number of iterations */
/* Initialization of quantum particle swarm */
$Q_t \leftarrow InitialQuantumSwarm(np)$ /* Algorithm 2 */
for $i \leftarrow 1$ ***to*** np **do**
  $d \leftarrow Transform(Q_t(i))$ /* Algorithm 3 */
  $D_t^{lb}(i) \leftarrow RepairOperator(d)$ /* Section 3.3 */
end
$d^* \leftarrow argmax\{f(d) : d \in D_t^{lb}\}$
/* Evolution of particle swarm */
while $t < IterMax$ **do**
  for $i \leftarrow 0$ ***to*** np **do**
    Randomly select a subset $S = \{d^1, d^2, \ldots, d^K\}$ of size K from D_t^{lb}, and the corresponding quantum particles are regarded as the neighbors of particle $Q_t(i)$
    $D_t^*(i) \leftarrow argmax\{f(d^{'}) : d^{'} \in S\}$ /* $D_t^*(i)$ is recorded as the neighborhood's historical best position for particle $D_t(i)$ */
    $Q_{t+1}^*(i) \leftarrow \alpha \times D_t^*(i) + (1-\alpha) \times (\vec{e} - D_t^*(i)))$
    $Q_{t+1}^{lb}(i) \leftarrow \alpha \times D_t^{lb}(i) + (1-\alpha) \times (\vec{e} - D_t^{lb}(i)))$
    $Q_{t+1}(i) \leftarrow c_1 \times Q_t(i) + c_2 \times Q_{t+1}^{lb}(i) + (1 - c_1 - c_2) \times Q_{t+1}^*(i)$
    $d \leftarrow Transform(Q_{t+1}(i))$ /* d is a discrete solution */
    $d \leftarrow RepairOperator(d)$
    if $rand(0,1) < p$ **then**
      $d \leftarrow VND(d)$ /* Algorithm 7 */
    end
    $SwarmUpdating(d, D_t^{lb})$ /* Section 3.5 */
    if $f(d) > f(d^*)$ **then**
      $d^* \leftarrow d$
    end
  end
  $t \leftarrow t + 1$
end

3.3. Repair Operator

Like previous studies (Chih, 2018; Chu and Beasley, 1998; Haddar et al., 2016; Lai et al., 2018), the DQPSO* algorithm uses a popular repair operator (denoted by *RepairOperator*()) to restore the feasibility of an infeasible solution. In addition to converting an infeasible solution into a feasible one, the repair operator serves also as a local optimization method.

To implement efficiently the repair operator, we apply a preprocessing procedure to first process the given MKP instance, so that the items are renumbered in an ascending order according to their scaled pseudo-utility ratios σ_j (Puchinger et al., 2009) defined as:

$$\sigma_j = \frac{p_j}{\sum_{i=1}^{m} \frac{a_{ij}}{c_i}}, \forall j \in \{1, 2, \ldots, n\} \tag{12}$$

After that, the vectors $(p_1, p_2, \ldots, p_n)$ and a_{ij} $(i = 1, 2, \ldots, m, j = 1, 2, \ldots, n)$ are adjusted accordingly.

Based on the resulting order of items, the repair operator is performed in two phases. Given an input infeasible solution, the first phase drops the least profitable items one by one according to the scaled pseudo-utility ratios until the solution becomes feasible. Then the second phase adds one by one the most profitable missing items according to their scaled pseudo-utility ratios, while keeping each intermediate solution feasible. Given its greedy nature, the repair operator is very fast with a time complexity bounded by $O(n \times m)$.

3.4. Local Optimization by Variable Neighborhood Descent

Algorithm 5: Neighborhood search with N_1

```
Function LSN1
  Input: A discrete solution d = (d_1, d_2, ..., d_n)
  Output: The improved solution d
Flag ← true
while Flag do
    Flag ← false
    for j ← n to 1 do
        if (d_j = 0) ∧ (d ⊕ Add(j) ∈ N_1(d)) then
            d ← d ⊕ Add(j)
            Flag ← true
        end
    end
end
```

To reinforce further its intensification ability, the DQPSO* algorithm employs, in a probabilistic way, a dedicated variable neighborhood descent (VND) procedure for local optimization. This VND procedure follows the standard VND framework (Mladenović and Hansen, 1997) and relies on two basic neighborhoods, i.e., the restricted 'Add' neighborhood N_1 and the restricted 'Swap'

Algorithm 6: Neighborhood search with N_2

```
1  Function LSN2
   Input: A discrete solution d = (d_1, d_2, ..., d_n)
   Output: The improved solution d
2  Flag ← false
3  for i ← 1 to n do
4  |  for j ← i + 1 to n do
5  |  |  if (d_i ≠ d_j) ∧ (d ⊕ Swap(i, j) ∈ N_2(d)) then
6  |  |  |  d ← d ⊕ Swap(i, j)
7  |  |  |  Flag ← true
8  |  |  |  return {d, Flag}
9  |  |  end
10 |  end
11 end
12 return {d, Flag}
```

Algorithm 7: The variable neighborhood descent (VND) method

```
1  Function VND
   Input: A discrete solution d = (d_1, d_2, ..., d_n)
   Output: The improved solution d
2  Flag ← true
3  while Flag do
4  |  d ← LSN1(d)                          /* Algorithm 5 */
5  |  (Flag,d) ← LSN2(d)                   /* Algorithm 6 */
6  end
```

neighborhood N_2. Given a discrete solution $s = (x_1, x_2, \ldots, x_n)$, the N_1 neighborhood is composed of all possible feasible solutions that can be obtained by changing the value of one variable x_i $(1 \leq i \leq n)$ from 0 to 1, and the N_2 neighborhood is composed of all possible feasible solutions that can be obtained by swapping the values of two variables x_v and x_u taking distinct values. Formally, the N_1 and N_2 neighborhoods can be described as follows:

$$N_1(s) = \{s \oplus Add(l) : \sum_{j=1}^{n} a_{ij}x_j + a_{il} \leq c_i, x_l = 0, 1 \leq l \leq n, 1 \leq i \leq m\} \quad (13)$$

$$N_2(s) = \{s \oplus Swap(v, u) : x_v \neq x_u = 0; \sum_{j=1}^{n} a_{ij}x_j + a_{iu} - a_{iv} \leq c_i, i \leq m\} \quad (14)$$

where $s \oplus Op$ $(Op \in \{Add, Swap\})$ designates the neighbor solution obtained by applying the 'Add' or 'Swap' operator to transform the incumbent solution s.

The size of $N_1(s)$ is bounded by $|I^0|$ $(\leq n)$, where I^0 denotes the set of variables taking the value of 0 in s, i.e., $I^0 = \{x_i : x_i = 0 \text{ in } s\}$. Thus,

the computational complexity of examining the whole $N_1(s)$ is bounded by $O(|I^0| \times m)$, where m is the number of capacity constraints. The size of $N_2(s)$ is bounded by $|I^1| \times |I^0|$, where I^1 is the set of variables taking the value of 1 in s, i.e., $I^1 = \{x_i : x_i = 1 \text{ in } s\}$. The computational complexity of examining the whole $N_2(s)$ is bounded by $O(|I^1| \times |I^0| \times m)$.

Based on these two neighborhoods, the VND procedure improves the input solution as follows. First, it starts with N_1 and makes a complete exploitation of the neighborhood by means of the first improvement descent strategy. Then, it switches to N_2 to search for an improving solution when a local optimal solution with respect to N_1 is reached. Moreover, VND switches immediately to N_1 once an improving solution is found with N_2. Finally, the search process stops when N_2 does not contain any improving solution and the best solution found is returned as the result of the VND procedure.

Algorithms 5 and 6 show how the neighborhoods N_1 and N_2 are examined, while Algorithm 7 summarizes the main framework of the VND procedure.

3.5. Population Updating Strategy for the Historical Discrete Best Positions of Particle Swarm

Algorithm 8: Pseudo-code of population updating method for D^{lb}

1 **Function** *SwarmUpdating*
  Input: A set of personal historical best positions (D^{lb}) for the discrete particles, a discrete solution (d), and parameter θ
  Output: Updated D^{lb}
2 $d_w \leftarrow argmin\{f(d^{'}) : d^{'} \in D^{lb})\}$
  /* d_w denotes the worst solution in D^{lb} */
3 $d_c \leftarrow argmin\{||d - d^{'}||_H : d^{'} \in D^{lb}\}$
  /* $||d - d^{'}||_H$ denotes the Hamming distance between d and $d^{'}$ */
4 $dist \leftarrow ||d - d_c||_H$
5 **if** $(f(d) > f(d_c)) \wedge (dist \leq \theta)$ **then**
6 | $D^{lb} \leftarrow D^{lb} \cup \{d\} \setminus \{d_c\}$ /* replace d_c by d */
7 **end**
8 **else if** $f(d) > f(d_w)) \wedge (dist > \theta)$ **then**
9 | $D^{lb} \leftarrow D^{lb} \cup \{d\} \setminus \{d_w\}$ /* replace d_w by d */
10 **end**

Like any population algorithm, it is crucial for the DQPSO* algorithm to maintain a healthy swarm in terms of diversity. For this purpose, DQPSO* uses a diversity-preserving strategy to update the set of personal historical best positions of the discrete particles D^{lb}.

Given a discrete solution d generated by the repair operator or the VND procedure and D^{lb}, the diversity-preserving updating strategy is performed as follows. First, the Hamming distance ($dist$) between d and its closest solution d_c in D^{lb} is calculated. Then, D^{lb} is updated according to one of the following

two situations, which is inspired by the work in (Lai and Hao, 2015) where a diversity-preserving pool updating strategy is employed as a key component of an evolutionary path relinking algorithm designed for the fixed spectrum frequency assignment problem. 1) If $f(d) > f(d_c)$ and $dist \leq \theta$, then d_c in D^{lb} is replaced by d, where θ is a parameter used to control the diversity of D^{lb} and $f(d)$ denotes the objective value of solution d. 2) If $f(d) > f(d_w)$ and $dist > \theta$, then the worst solution in D^{lb} (denoted by d_w) is replaced by d. In other cases, the offspring solution d is discarded, while keeping D^{lb} unchanged. The pseudo-code of the this population updating strategy is provided in Algorithm 8.

3.6. Discussions

As we show above, the proposed DQPSO* algorithm integrates especially two original strategies that distinguish itself from the existing binary PSO algorithms for the MKP in the literature such as (Chih, 2018; Haddar et al., 2016). First, DQPSO* employs the diversity-preserving updating strategy (see Section 3.5) to enhance the diversity of discrete particle swarm D^{lb}, where the distances among discrete particles are directly controlled by a parameter θ. To the best of our knowledge, such a strategy was never used in previous binary PSO algorithms. The analysis in Section 5.1 shows that this updating strategy helps to preserve population diversity and improves significantly the search ability of the algorithm. Second, the proposed algorithm integrates for the first time a VND method as the local optimization procedure, which is applied in a probabilistic way each time an offspring solution is generated during the search process. Once again, this technique was not available in existing binary PSO algorithms. As the computational experiments in Section 5.2 show, the probability-controlled VND method ensures the key intensification role and contributes to the performance of the algorithm. Finally, it is worth mentioning that these two strategies are of general interest and can be applied within binary PSO algorithms designed for other binary optimization problems.

4. Computational Experiments

To assess the performance of the DQPSO* algorithm, we carried out extensive experiments by testing the algorithm on benchmark instances commonly used in the literature and making a comparison with a number of state-of-the-art MKP algorithms.

4.1. Benchmark Instances

To carry out our computational experiments, we use 270 popular benchmark instances, which are described in (Chu and Beasley, 1998) and available at OR-Library[1]. These instances can be divided into three sets and their characteristics can be summarized as follows.

[1] `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html`

- Set I: This set consists of 90 small instances with $n = 100$ which can be divided into three subsets, where each subset contains 30 instances with $m = 5$, 10 or 30, respectively. The coefficients a_{ij} $(1 \leq i \leq m,\ 1 \leq j \leq n)$ are integers randomly generated in $[0, 1000]$ and c_i is set to $\beta \times \sum_{j=1}^{n} a_{ij}$ $(1 \leq i \leq m)$ where β is a parameter called the tightness ratio and is set to 0.25, 0.5, and 0.75. Optimal solutions for these instances are provided in (Mansini and Speranza, 2012).

- Set II: This set contains 90 medium-sized instances with $n = 250$ and $m \in \{5, 10, 30\}$, and the coefficients a_{ij} $(1 \leq i \leq m,\ 1 \leq j \leq n)$ and c_i were generated in the same way as for the instances of Set I. Optimal solutions for the instances with $m = 5$ and 10 are provided in (Boussier et al., 2010; Mansini and Speranza, 2012; Vimont et al., 2008).

- Set III: This set contains 90 large instances with $n = 500$ and $m \in \{5, 10, 30\}$, and the coefficients a_{ij} $(1 \leq i \leq m,\ 1 \leq j \leq n)$ and c_i were generated in the same way as for the instances of Sets I and II. Optimal solutions for the instances with $m = 5$ and 10 are provided in (Boussier et al., 2010; Mansini and Speranza, 2012; Vimont et al., 2008).

One notices that the optimal solutions for 217 out of 270 instances reported in (Boussier et al., 2010; Mansini and Speranza, 2012; Vimont et al., 2008) have been obtained with large computation times up to 150 hours for some instances with $n = 500$ and $m = 10$.

4.2. Parameter Settings and Experimental Protocol

Table 1: Settings of parameters

Parameters	Section	Description	Values
np	3.2	Number of particles	{2n,4n,5n}
$IterMax$	3.2	Maximum number of iterations	{200m,500m,10^4}
K	3.2	Number of neighbors of particles	10
α	3.2	parameter used in Eq.(8)	0.0
c_1	3.2	parameter used in Eq.(10)	0.2
c_2	3.2	parameter used in Eq.(10)	0.4
p	3.2	probability of applying the VND	0.01
θ	3.5	parameter for the population updating	2

The DQPSO* algorithm employs eight parameters whose descriptions and settings are given in Table 1, where the values of np and $IterMax$ were set according to the values of n and m of instances to guarantee that the computational effort of our algorithm is the same as that of a recent binary PSO algorithm (3R-BPSO) (Chih, 2018): $np = 5n$ and $IterMax = 200m$ for the instances with $n = 100$ and $m \in \{5, 10, 30\}$, $np = 4n$ and $IterMax = 500m$ for the instances with $n = 250$ and $m \in \{5, 10, 30\}$, and $np = 2n$ and $IterMax = 500m$ for the instances with $n = 500$ and $m \in \{5, 10\}$. For the instances with $n = 500$ and $m = 30$ which are shown to be very hard to solve for most existing heuristic algorithms, np and $IterMax$ were respectively set to $2n$ and 10^4. For the parameters p and α, their values were set according to the experiments shown in

Section 5.2 and Section 5.3, respectively. For the other parameters, the default values were empirically set according to a preliminary experiment.

The DQPSO* algorithm was implemented in C++ language and compiled by the g++ compiler with the -O3 flag [2]. All the experiments were carried out on a computer with an Intel E5-2670 processor (2.5 GHz and 2G RAM), running the Linux operating system. Due to its stochastic nature, DQPSO* was independently run 100 times to solve each instance. To run the algorithm, we use consistently the parameter setting of Table 1. We mention that using an extended number of iterations (i.e., larger $IterMax$ values) does not significantly change the final results.

To assess the performance of DQPSO*, we use six representative MKP algorithms in the literature as our reference algorithms, including a genetic algorithm (GA) (Chu and Beasley, 1998) (as a baseline reference), a hybrid quantum particle swarm optimization algorithm (QPSO*) (Haddar et al., 2016), a binary PSO algorithm (3R-BPSO) that employs three repair operators to repair infeasible solutions (Chih, 2018), a filter-and-fan heuristic (F&F) (Khemakhem et al., 2012), a two-phase tabu search (TP+TS) (Vasquez and Hao, 2001), and a very recent two-phase tabu-evolutionary algorithm (TPTEA) published in 2018 (Lai et al., 2018). The results of the reference algorithms are compiled from the corresponding papers. If the results of an algorithm for a set of benchmark instances are not available, the algorithm will be ignored in the comparative study (e.g., this is the case of 3R-BPSO for some instances of set I and the instances of set II). Moreover, given that the compared algorithms are written in different programming languages and run on various computing platforms under different stopping conditions, it is impossible to perform a fair comparison of computation times. As a result, we mainly focus on solution quality for our computational study (this is also a common practice in the literature). Only for indicative purposes, we provide the timing information for DQPSO* and TPTEA (whose codes are available and were run on the same computing platform).

4.3. Computational Results and Comparison

Our first experiment aims to assess the proposed DQPSO* algorithm on the small instances of Set I with $n = 100$, and the experimental results are summarized in Tables 2–4, along with the results of five references algorithms whose results are available. In Table 2, columns 1 and 2 give the names and the known optimum results (Opt.) of the instances with $m = 5$, columns 3–8 provide the best objective values (f_{best}) obtained for the reference algorithms as well as our DQPSO* algorithm, columns 9–12 indicate the average objective values (f_{avg}) obtained for three reference algorithms as well as our DQPSO* algorithm, and the last two columns report the average computational time (t_{avg}) in seconds

[2] The source code of the DQPSO* algorithm will be made available at `http://www.info.univ-angers.fr/pub/hao/DQPSO.html`

needed to reach the final objective value for TPTEA and DQPSO*. In addition, the row "Avg." shows the average result for each column, and the rows "#better", "#equal", and "#worse" show the number of instances for which the associated reference algorithm obtained a better, equal, or worse result in terms of f_{best}, f_{avg}, and t_{avg} in comparison with the proposed DQPSO* algorithm. To verify the statistical difference between the proposed DQPSO* algorithm and the reference algorithms in terms of f_{best} and f_{avg}, the *p-values* from the Wilcoxon signed-rank tests are provided in the last row of the tables, where a *p-value* less than 0.05 means that there exists a significant difference between the compared results. Tables 3 and 4 present the results on the instances with $m = 10$ and $m = 30$ with the same information as in Table 2.

Table 2 shows that for all the 30 small instances with $n = 100$ and a small number ($m = 5$) of constraints, our DQPSO* algorithm performs very well and is able to obtain the known optimum solution with a success rate of 100% within a short computing time ($t_{avg} = 0.5$). Moreover, compared to the reference algorithms, DQPSO* achieves a similar or better performance in terms of f_{best}, f_{avg}, and t_{avg}. In terms of f_{best}, DQPSO* obtains the same result compared to GA, F&F, QPSO* and TPTEA, and reports a better results for 2 instances compared to 3R-BPSO. In terms of f_{avg}, DQPSO* obtains the same result compared to QPSO* and TPTEA, and reports a better result for 25 out of 30 instances compared to 3R-BPSO.

Table 3 shows that for the instances with $n = 100$ and a medium-sized number ($m = 10$) of constraints our algorithm also performs well. For all the 30 instances, our algorithm obtains the known optimum result reported in the literature, and the corresponding success rate of our algorithm is 100% for 26 out of 30 instances. Compared to the first 4 reference algorithms, i.e., GA, F&F, 3R-BPSO, QPSO*, the DQPSO* algorithm is very competitive and obtains a better result in terms of f_{best} for one instance and an equal result for the 29 remaining instances. Compared to the TPTEA algorithm, in terms of f_{best}, the DQPSO* algorithm obtains the same result for all the 30 instances. In terms of f_{avg}, the DQPSO* algorithm outperforms significantly 3R-BPSO by obtaining better results on all the 30 instances, reaches comparable results relative to QPSO*, and obtains slightly worse results than the latest TPTEA algorithm.

Table 4 shows that for the instances with $n = 100$ and a large number ($m = 30$) of constraints, the DQPSO* algorithm has a similar performance compared to GA, F&F, and QPSO*, but performs slightly worse than the tabu-based TPTEA algorithm. Specifically, in terms of f_{best}, DQPSO* obtains a better result on 3 instances than the GA algorithm, and a better and worse result on 2 instances compared to the F&F, and QPSO* algorithms, respectively. In terms of f_{avg}, the DQPSO* algorithm obtains comparable results compared to QPSO*. Compared to TPTEA, DQPSO* performs worse in terms of both f_{best} and f_{avg}. TPTEA attains the known optimum solution with a success rate of 100% for all the instances while this is the case of DQPSO* only for 16 instances.

On the other hand, Tables 2–4 show that for these small instances the differences between DQPSO* and the reference algorithms in terms of f_{best} are

Table 2: Comparative results of DQPSO* with 5 reference algorithms from the literature on the small instances with $n = 100$ and $m = 5$.

Instance	Opt.	f_{best}						f_{avg}				t_{avg}(s)	
		GA	F&F	3R-BPSO	QPSO*	TPTEA	DQPSO*	3R-BPSO	QPSO*	TPTEA	DQPSO*	TPTEA	DQPSO*
5.100.0	24381	24381	24381	24381	24381	24381	24381	24381.0	24381.0	24381.00	24381.0	0.55	0.11
5.100.1	24274	24274	24274	24274	24274	24274	24274	24273.5	24274.0	24274.00	24274.0	0.39	0.12
5.100.2	23551	23551	23551	23538	23551	23551	23551	23537.8	23551.0	23551.00	23551.0	0.39	1.02
5.100.3	23534	23534	23534	23534	23534	23534	23534	23508.9	23534.0	23534.00	23534.0	1.49	0.82
5.100.4	23991	23991	23991	23991	23991	23991	23991	23961.4	23991.0	23991.00	23991.0	0.85	1.57
5.100.5	24613	24613	24613	24613	24613	24613	24613	24608.5	24613.0	24613.00	24613.0	0.37	0.17
5.100.6	25591	25591	25591	25591	25591	25591	25591	25554.6	25591.0	25591.00	25591.0	0.42	0.09
5.100.7	23410	23410	23410	23410	23410	23410	23410	23375.4	23410.0	23410.00	23410.0	0.32	0.12
5.100.8	24216	24216	24216	24216	24216	24216	24216	24213.6	24216.0	24216.00	24216.0	1.71	2.05
5.100.9	24411	24411	24411	24411	24411	24411	24411	24404.3	24411.0	24411.00	24411.0	0.53	0.11
5.100.10	42757	42757	42757	42757	42757	42757	42757	42704.4	42757.0	42757.00	42757.0	1.08	0.24
5.100.11	42545	42545	42545	42545	42545	42545	42545	42471.2	42545.0	42545.00	42545.0	9.35	0.82
5.100.12	41968	41968	41968	41967	41968	41968	41968	41958.3	41968.0	41968.00	41968.0	0.91	1.25
5.100.13	45090	45090	45090	45090	45090	45090	45090	45083.2	45090.0	45090.00	45090.0	10.80	0.72
5.100.14	42218	42218	42218	42218	42218	42218	42218	42218.0	42218.0	42218.00	42218.0	0.51	0.32
5.100.15	42927	42927	42927	42927	42927	42927	42927	42917.8	42927.0	42927.00	42927.0	0.47	0.06
5.100.16	42009	42009	42009	42009	42009	42009	42009	42009.0	42009.0	42009.00	42009.0	0.40	0.05
5.100.17	45020	45020	45020	45020	45020	45020	45020	45019.5	45020.0	45020.00	45020.0	0.52	1.40
5.100.18	43441	43441	43441	43441	43441	43441	43441	43425.6	43441.0	43441.00	43441.0	1.72	0.12
5.100.19	44554	44554	44554	44554	44554	44554	44554	44515.5	44554.0	44554.00	44554.0	2.71	0.81
5.100.20	59822	59822	59822	59822	59822	59822	59822	59822.0	59822.0	59822.00	59822.0	0.32	0.02
5.100.21	62081	62081	62081	62081	62081	62081	62081	62060.8	62081.0	62081.00	62081.0	0.57	0.11
5.100.22	59802	59802	59802	59802	59802	59802	59802	59761.9	59802.0	59802.00	59802.0	0.30	0.28
5.100.23	60479	60479	60479	60479	60479	60479	60479	60478.2	60479.0	60479.00	60479.0	0.33	0.15
5.100.24	61091	61091	61091	61091	61091	61091	61091	61089.9	61091.0	61091.00	61091.0	0.37	0.13
5.100.25	58959	58959	58959	58959	58959	58959	58959	58954.4	58959.0	58959.00	58959.0	0.40	0.05
5.100.26	61538	61538	61538	61538	61538	61538	61538	61523.5	61538.0	61538.00	61538.0	0.35	0.16
5.100.27	61520	61520	61520	61520	61520	61520	61520	61505.1	61520.0	61520.00	61520.0	0.32	0.08
5.100.28	59453	59453	59453	59453	59453	59453	59453	59453.0	59453.0	59453.00	59453.0	0.34	0.04
5.100.29	59965	59965	59965	59965	59965	59965	59965	59960.4	59965.0	59965.00	59965.0	0.65	1.96
Avg.	42640.37	**42640.37**	**42640.37**	42639.90	**42640.37**	**42640.37**	**42640.37**	42625.02	**42640.37**	**42640.37**	**42640.37**	1.31	**0.50**
#better		0	0	0	0	0		0	0	0		6	
#equal		30	30	28	30	30		5	30	30		0	
#worse		0	0	2	0	0		25	0	0		24	
p-value		1.0	1.0	1.80E-1	1.0	1.0		1.23E-5	1.0	1.0			

Table 3: Comparative results of DQPSO* with 5 reference algorithms from the literature on the small instances with $n = 100$ and $m = 10$.

Instance	Opt.	f_{best}						f_{avg}				t_{avg}(s)	
		GA	F&F	3R-BPSO	QPSO*	TPTEA	DQPSO*	3R-BPSO	QPSO*	TPTEA	DQPSO*	TPTEA	DQPSO*
10.100.0	23064	23064	23064	23064	23064	23064	23064	23050.4	23064.00	23064.00	23064.00	1.51	1.90
10.100.1	22801	22801	22801	22801	22801	22801	22801	22752.2	22801.00	22801.00	22801.00	1.42	1.53
10.100.2	22131	22131	22131	22131	22131	22131	22131	22119.8	22131.00	22131.00	22131.00	1.14	0.66
10.100.3	22772	22772	22772	22772	22772	22772	22772	22744.8	22772.00	22772.00	22772.00	19.73	1.65
10.100.4	22751	22751	22751	22751	22751	22751	22751	22651.1	22751.00	22751.00	22737.29	0.36	7.47
10.100.5	22777	22777	22739	22777	22777	22777	22777	22692.1	22758.00	22777.00	22777.00	5.92	4.54
10.100.6	21875	21875	21875	21875	21875	21875	21875	21832.5	21875.00	21875.00	21875.00	0.47	0.42
10.100.7	22635	22635	22635	22635	22635	22635	22635	22547.8	22635.00	22635.00	22635.00	4.97	2.95
10.100.8	22511	22511	22511	22511	22511	22511	22511	22429.1	22511.00	22511.00	22511.00	0.45	1.64
10.100.9	22702	22702	22702	22702	22702	22702	22702	22699.5	22702.00	22702.00	22702.00	0.53	0.17
10.100.10	41395	41395	41395	41395	41395	41395	41395	41356.4	41395.00	41395.00	41395.00	16.63	4.86
10.100.11	42344	42344	42344	42344	42344	42344	42344	42344.0	42344.00	42344.00	42344.00	0.86	1.19
10.100.12	42401	42401	42401	42401	42401	42401	42401	42339.3	42401.00	42401.00	42401.00	13.08	3.62
10.100.13	45624	45624	45624	45624	45624	45624	45624	45571.0	45611.00	45624.00	45624.00	21.52	7.24
10.100.14	41884	41884	41884	41884	41884	41884	41884	41812.5	41884.00	41884.00	41876.95	6.53	5.34
10.100.15	42995	42995	42995	42995	42995	42995	42995	42993.9	42995.00	42995.00	42995.00	0.57	0.32
10.100.16	43574	43559	43574	43559	43553	43574	43574	43494.4	43553.00	43574.00	43570.34	16.23	4.75
10.100.17	42970	42970	42970	42970	42970	42970	42970	42949.6	42970.00	42970.00	42970.00	15.30	0.54
10.100.18	42212	42212	42212	42212	42212	42212	42212	42207.6	42212.00	42212.00	42212.00	0.51	0.49
10.100.19	41207	41207	41207	41207	41207	41207	41207	41128.6	41207.00	41207.00	41207.00	18.76	3.60
10.100.20	57375	57375	57375	57375	57375	57375	57375	57375.0	57375.00	57375.00	57375.00	0.43	0.23
10.100.21	58978	58978	58978	58978	58978	58978	58978	58978.0	58978.00	58978.00	58978.00	0.96	0.76
10.100.22	58391	58391	58391	58391	58391	58391	58391	58349.8	58391.00	58391.00	58391.00	0.53	2.94
10.100.23	61966	61966	61966	61966	61966	61966	61966	61966.0	61966.00	61966.00	61966.00	1.74	2.25
10.100.24	60803	60803	60803	60803	60803	60803	60803	60802.9	60803.00	60803.00	60803.00	0.52	0.25
10.100.25	61437	61437	61437	61437	61437	61437	61437	61382.7	61437.00	61437.00	61402.50	5.65	2.74
10.100.26	56377	56377	56377	56377	56377	56377	56377	56361.7	56377.00	56377.00	56377.00	7.90	1.24
10.100.27	59391	59391	59391	59391	59391	59391	59391	59372.8	59391.00	59391.00	59391.00	0.43	0.56
10.100.28	60205	60205	60205	60205	60205	60205	60205	60197.7	60205.00	60205.00	60205.00	0.58	0.22
10.100.29	60633	60633	60633	60633	60633	60633	60633	60631.2	60633.00	60633.00	60633.00	0.50	0.22
Avg.	41606.03	41605.53	41604.77	41605.53	41605.33	**41606.03**	**41606.03**	41571.15	41604.27	**41606.03**	41604.07	5.52	**2.21**
#better		0	0	0	0	0		0	3	4		7	
#equal		29	29	29	29	30		0	25	26		0	
#worse		1	1	1	1	0		30	2	0		23	
p-value		3.17E-1	3.17E-1	3.17E-1	3.17E-1	1.0		8.30E-6	9.16E-1	6.79E-2			

Table 4: Comparative results of DQPSO* with 4 reference algorithms from the literature on the small instances with $n = 100$ and $m = 30$.

		f_{best}					f_{avg}			t_{avg}(s)	
Instance	Opt.	GA	F&F	QPSO*	TPTEA	DQPSO*	QPSO*	TPTEA	DQPSO*	TPTEA	DQPSO*
30.100.0	21946	21946	21946	21946	21946	21946	21946.00	21946.00	21946.00	7.5	1.1
30.100.1	21716	21716	21716	21716	21716	21716	21716.00	21716.00	21716.00	17.7	0.9
30.100.2	20754	20754	20754	20754	20754	20754	20754.00	20754.00	20754.00	10.4	1.1
30.100.3	21464	21464	21464	21464	21464	21464	21448.00	21464.00	21464.00	12.9	5.7
30.100.4	21844	21814	21844	21844	21844	21844	21828.50	21844.00	21833.09	15.6	26.6
30.100.5	22176	22176	22176	22176	22176	22176	22176.00	22176.00	22176.00	0.7	0.9
30.100.6	21799	21799	21799	21772	21799	21799	21772.00	21799.00	21793.60	19.5	12.8
30.100.7	21397	21397	21397	21397	21397	21397	21361.50	21397.00	21396.04	15.8	14.5
30.100.8	22525	22493	22493	22525	22525	22493	22503.50	22525.00	22493.00	15.0	10.6
30.100.9	20983	20983	20983	20983	20983	20983	20983.00	20983.00	20983.00	0.8	2.5
30.100.10	40767	40767	40767	40767	40767	40767	40728.50	40767.00	40764.18	22.5	27.3
30.100.11	41308	41304	41304	41308	41308	41308	41306.00	41308.00	41305.88	19.2	14.9
30.100.12	41630	41560	41630	41630	41630	41612	41606.00	41630.00	41585.22	28.5	29.8
30.100.13	41041	41041	41041	41041	41041	41041	41041.00	41041.00	41041.00	22.1	8.1
30.100.14	40889	40872	40889	40872	40889	40872	40872.00	40889.00	40872.00	21.6	0.9
30.100.15	41058	41058	41058	41058	41058	41058	41058.00	41058.00	41058.00	0.9	2.7
30.100.16	41062	41062	41062	41062	41062	41062	41062.00	41062.00	41062.00	11.3	14.9
30.100.17	42719	42719	42719	42719	42719	42719	42719.00	42719.00	42718.73	19.4	0.6
30.100.18	42230	42230	42230	42230	42230	42230	42230.00	42230.00	42230.00	1.2	2.1
30.100.19	41700	41700	41700	41700	41700	41700	41700.00	41700.00	41700.00	14.3	3.9
30.100.20	57494	57494	57494	57494	57494	57494	57494.00	57494.00	57494.00	0.6	0.2
30.100.21	60027	60027	60027	60027	60027	60027	60027.00	60027.00	60021.94	1.4	29.8
30.100.22	58052	58025	58052	58052	58052	58052	58052.00	58052.00	58027.85	18.2	28.4
30.100.23	60776	60776	60776	60776	60776	60776	60776.00	60776.00	60775.78	4.5	6.3
30.100.24	58884	58884	58884	58884	58884	58884	58884.00	58884.00	58865.52	4.9	2.4
30.100.25	60011	60011	60011	60011	60011	60011	60011.00	60011.00	60005.30	2.3	2.0
30.100.26	58132	58132	58104	58132	58132	58132	58118.00	58132.00	58132.00	0.7	1.7
30.100.27	59064	59064	59064	59064	59064	59064	59064.00	59064.00	59064.00	0.7	1.2
30.100.28	58975	58975	58975	58975	58975	58975	58975.00	58975.00	58975.00	19.6	11.3
30.100.29	60603	60603	60603	60593	60603	60603	60593.00	60603.00	60603.00	2.3	3.2
Avg.	40767.53	40761.53	40765.40	40765.73	**40767.53**	40765.30	40760.17	**40767.53**	40761.87	11.1	**8.9**
#better		0	2	2	3		9	14		14	
#equal		27	26	26	27		14	16		0	
#worse		3	2	2	0		7	0		16	
p-value		6.79E-2	1.0	5.15E-1	1.09E-1		7.56E-1	9.81E-4			

marginal, which is confirmed by the large *p-values* (≥ 0.05). However, in terms of f_{avg}, the DQPSO* algorithm outperforms significantly the 3R-BPSO algorithm.

The second experiment aims to assess and compare the DQPSO* algorithm on the medium-sized instances with $n = 250$, and the experimental results are summarized in Tables 5 to 7, where the BKR denotes the best known results reported in the literature and other symbols are same as in the previous tables.

The results on the instances with a small number ($m = 5$) of constraints are provided in Table 5. One observes that for these instances, the proposed DQPSO* algorithm outperforms GA, F&F, QPSO* in terms of f_{best}, and obtains comparable results with the TPTEA algorithm. Specifically, DQPSO* yields respectively a better result for 11, 7, and 5 instances compared to GA, F&F, QPSO*, and matches the result of the TPTEA algorithm for 29 out of 30 instances, while yielding a worse result for one instance. As for the f_{avg}, DQPSO* performs better than QPSO* by reporting a better result on 15 instances and the same result on 12 instances, but performs marginally worse than TPTEA (the average value of f_{avg} over all the 30 instances is 107087.05 for DQPSO* against 107088.59 for TPTEA). Moreover, TPTEA and DQPSO* obtain the optimum solution with a success rate of 100% for 27 and 18 out of 30 instances, respectively, while DQPSO* is more computationally efficient.

Tables 6 and 7 report respectively the results for the instances with $m = 10$ and $m = 30$. These two tables show that for the medium-sized instances with a large number of constraints, DQPSO* outperforms significantly GA, F&F, and

Table 5: Comparative results of DQPSO* with 4 reference algorithms from the literature on the medium-sized instances with $n = 250$ and $m = 5$.

		f_{best}					f_{avg}			t_{avg}(s)	
Instance	Opt.	GA	F&F	QPSO*	TPTEA	DQPSO*	QPSO*	TPTEA	DQPSO*	TPTEA	DQPSO*
5.250.0	59312	59312	59312	59312	59312	59312	59312.00	59312.00	59312.00	67.0	7.1
5.250.1	61472	61472	61468	61472	61472	61472	61470.00	61472.00	61471.72	251.5	24.9
5.250.2	62130	62130	62130	62130	62130	62130	62130.00	62130.00	62130.00	82.9	3.5
5.250.3	59463	59446	59436	59427	59463	59463	59427.00	59462.33	59441.93	1017.0	28.3
5.250.4	58951	58951	58951	58951	58951	58951	58951.00	58951.00	58951.00	99.3	3.4
5.250.5	60077	60056	60062	60077	60077	60077	60056.00	60069.50	60070.34	741.8	27.4
5.250.6	60414	60414	60414	60414	60414	60414	60414.00	60414.00	60414.00	96.7	5.8
5.250.7	61472	61472	61454	61472	61472	61472	61460.50	61472.00	61472.00	153.2	18.3
5.250.8	61885	61885	61885	61885	61885	61885	61885.00	61885.00	61885.00	85.1	11.7
5.250.9	58959	58959	58959	58959	58959	58959	58925.50	58959.00	58959.00	79.2	3.1
5.250.10	109109	109109	109109	109066	109109	109109	109058.50	109109.00	109107.46	288.7	18.4
5.250.11	109841	109841	109841	109841	109841	109841	109841.00	109841.00	109841.00	118.1	5.9
5.250.12	108508	108489	108508	108508	108508	108508	108508.00	108508.00	108508.00	107.2	18.0
5.250.13	109383	109383	109383	109356	109383	109383	109347.50	109383.00	109383.00	144.8	9.7
5.250.14	110720	110720	110720	110720	110720	110720	110710.00	110720.00	110718.10	611.2	12.0
5.250.15	110256	110256	110256	110256	110256	110256	110256.00	110256.00	110250.71	257.9	46.0
5.250.16	109040	109016	109040	109040	109040	109040	109022.50	109040.00	109040.00	114.7	24.9
5.250.17	109042	109037	109016	109042	109042	109042	109018.50	109042.00	109041.32	102.8	22.8
5.250.18	109971	109957	109957	109971	109971	109971	109955.00	109971.00	109971.00	212.0	8.4
5.250.19	107058	107038	107058	107058	107058	107058	107048.00	107058.00	107057.09	210.9	14.3
5.250.20	149665	149659	149659	149665	149665	149665	149650.50	149665.00	149661.10	250.4	23.3
5.250.21	155944	155940	155944	155944	155944	155940	155942.00	155943.87	155940.00	60.5	6.0
5.250.22	149334	149316	149334	149334	149334	149334	149334.00	149334.00	149332.46	119.1	20.9
5.250.23	152130	152130	152130	152130	152130	152130	152130.00	152130.00	152130.00	55.9	3.6
5.250.24	150353	150353	150353	150353	150353	150353	150353.00	150353.00	150353.00	58.0	16.3
5.250.25	150045	150045	150045	150045	150045	150045	150045.00	150045.00	150045.00	41.9	3.3
5.250.26	148607	148607	148607	148607	148607	148607	148607.00	148607.00	148607.00	36.4	7.9
5.250.27	149782	149772	149782	149772	149782	149782	149762.50	149782.00	149775.40	51.8	18.1
5.250.28	155075	155075	155075	155057	155075	155075	155045.00	155075.00	155075.00	41.8	26.1
5.250.29	154668	154662	154668	154668	154668	154668	154668.00	154668.00	154668.00	118.0	16.9
Avg.	107088.87	107083.40	107085.20	107084.40	**107088.87**	107088.73	107077.77	**107088.59**	107087.05	189.20	**15.21**
#better		0	1	1	1		3	11		0	
#equal		19	22	24	29		12	18		0	
#worse		11	7	5	0		15	1		30	
p-value		3.33E-3	2.07E-2	4.64E-2	3.17E-1		7.38E-4	4.74E-3			

Table 6: Comparative results of DQPSO* with 4 reference algorithms from the literature on the medium-sized instances with $n = 250$ and $m = 10$.

		f_{best}					f_{avg}			t_{avg}(s)	
Instance	Opt.	GA	F&F	QPSO*	TPTEA	DQPSO*	QPSO*	TPTEA	DQPSO*	TPTEA	DQPSO*
10.250.0	59187	59187	59164	59182	59187	59187	59173.00	59187.00	59187.00	194.8	20.0
10.250.1	58781	58662	58693	58781	58781	58705	58733.00	58743.13	58686.12	715.2	51.7
10.250.2	58097	58094	58094	58097	58097	58097	58095.50	58097.00	58086.86	189.9	39.4
10.250.3	61000	61000	60972	61000	61000	61000	60986.00	60998.57	60989.07	839.2	61.4
10.250.4	58092	58092	58092	58092	58092	58092	58092.00	58090.57	58088.38	821.7	67.4
10.250.5	58824	58803	58824	58824	58824	58824	58824.00	58822.60	58803.42	462.1	24.4
10.250.6	58704	58607	58632	58606	58704	58704	58596.50	58704.00	58692.39	385.2	59.1
10.250.7	58936	58917	58917	58902	58936	58930	58889.50	58932.10	58921.47	732.7	54.7
10.250.8	59387	59384	59381	59372	59387	59387	59357.50	59387.00	59383.41	102.3	74.0
10.250.9	59208	59193	59208	59208	59208	59208	59208.00	59208.00	59208.00	327.1	23.3
10.250.10	110913	110863	110889	110857	110913	110913	110843.00	110913.00	110913.00	370.7	22.3
10.250.11	108717	108659	108702	108687	108717	108717	108687.00	108717.00	108702.55	529.3	9.6
10.250.12	108932	108932	108922	108891	108932	108932	108889.00	108932.00	108930.63	77.4	40.3
10.250.13	110086	110037	110059	110086	110086	110086	110060.50	110086.00	110061.33	1070.9	81.6
10.250.14	108485	108423	108485	108485	108485	108485	108459.50	108485.00	108485.00	129.1	32.4
10.250.15	110845	110841	110841	110845	110845	110845	110843.00	110843.67	110840.22	1064.0	46.3
10.250.16	106077	106075	106075	106047	106077	106077	106036.00	106075.73	106076.41	239.2	56.3
10.250.17	106686	106686	106685	106686	106686	106686	106681.50	106686.00	106686.00	563.2	24.8
10.250.18	109829	109825	109822	109788	109829	109825	109755.00	109827.40	109823.00	845.3	46.8
10.250.19	106723	106723	106723	106723	106723	106723	106723.00	106723.00	106723.00	80.5	34.1
10.250.20	151809	151790	151790	151779	151809	151809	151769.00	151809.00	151806.92	177.4	49.4
10.250.21	148772	148772	148772	148772	148772	148772	148772.00	148772.00	148772.00	24.6	3.1
10.250.22	151909	151900	151909	151909	151909	151909	151909.00	151909.00	151909.00	85.6	41.6
10.250.23	151324	151275	151281	151281	151324	151324	151281.00	151324.00	151276.39	629.1	48.4
10.250.24	151966	151948	151966	151966	151966	151966	151938.00	151961.80	151953.94	413.8	33.3
10.250.25	152109	152109	152109	152109	152109	152109	152109.00	152109.00	152109.00	51.2	6.9
10.250.26	153131	153131	153131	153131	153131	153131	153131.00	153131.00	153131.00	36.3	9.0
10.250.27	153578	153520	153533	153529	153578	153578	153529.00	153578.00	153560.40	95.8	70.2
10.250.28	149160	149155	149160	149160	149160	149160	149145.00	149160.00	149156.53	59.1	92.4
10.250.29	149704	149704	149688	149646	149704	149704	149637.00	149704.00	149704.00	56.2	10.3
Avg.	106365.70	106343.57	106350.63	106348.03	**106365.70**	106362.83	106338.42	**106363.89**	106355.55	379.0	**41.1**
#better		0	0	1	3		6	18		0	
#equal		11	11	16	27		6	11		0	
#worse		19	19	13	0		18	1		30	
p-value		1.31E-4	1.32E-4	1.31E-2	1.09E-1		4.68E-3	1.55E-4			

Table 7: Comparative results of DQPSO* with 4 reference algorithms from the literature on the medium-sized instances with $n = 250$ and $m = 30$.

		f_{best}					f_{avg}			t_{avg}(s)	
Instance	BKR	GA	F&F	QPSO*	TPTEA	DQPSO*	QPSO*	TPTEA	DQPSO*	TPTEA	DQPSO*
30.250.0	56842	56693	56796	56796	56824	56796	56745.50	56824.00	56745.30	130.5	191.2
30.250.1	58520	58318	58333	58302	58520	58351	58302.00	58520.00	58319.88	216.3	81.4
30.250.2	56614	56553	56553	56614	56614	56614	56570.50	56614.00	56556.16	216.4	274.2
30.250.3	56930	56863	56930	56930	56930	56930	56892.00	56930.00	56929.35	90.7	81.0
30.250.4	56629	56629	56629	56629	56629	56629	56629.00	56629.00	56629.00	74.2	28.4
30.250.5	57205	57119	57149	57146	57205	57189	57115.50	57205.00	57147.28	374.4	178.8
30.250.6	56348	56292	56263	56303	56357	56303	56246.50	56333.40	56223.06	1155.3	432.6
30.250.7	56457	56403	56457	56392	56457	56457	56374.50	56457.00	56456.91	103.3	171.2
30.250.8	57474	57442	57373	57447	57474	57474	57407.50	57458.90	57419.36	971.1	279.7
30.250.9	56447	56447	56447	56447	56447	56447	56447.00	56447.00	56447.00	99.5	12.6
30.250.10	107770	107689	107735	107703	107770	107732	107696.00	107763.10	107719.89	1034.2	299.5
30.250.11	108392	108338	108338	108338	108392	108379	108336.50	108387.23	108377.71	437.6	81.6
30.250.12	106442	106385	106415	106442	106442	106442	106413.50	106439.60	106427.69	587.2	136.1
30.250.13	106876	106796	106832	106851	106876	106876	106828.00	106876.00	106821.63	204.5	213.5
30.250.14	107414	107396	107414	107382	107414	107396	107382.00	107414.00	107396.00	230.4	196.0
30.250.15	107271	107246	107271	107271	107271	107271	107236.50	107271.00	107244.81	293.9	210.4
30.250.16	106372	106308	106277	106248	106372	106365	106242.00	106371.77	106319.30	682.5	259.9
30.250.17	104032	103993	104003	103988	104032	104014	103988.00	104019.00	104000.59	497.2	285.7
30.250.18	106856	106835	106835	106856	106856	106835	106845.50	106852.50	106807.00	322.2	164.2
30.250.19	105780	105751	105742	105751	105780	105751	105740.00	105779.17	105751.00	440.6	138.7
30.250.20	150163	150083	150138	150096	150163	150138	150052.00	150163.00	150111.33	456.9	335.6
30.250.21	149958	149907	149958	149958	149958	149907	149932.50	149958.00	149907.00	100.7	52.3
30.250.22	153007	152993	153007	153007	153007	153007	153007.00	153007.00	152993.42	130.9	86.3
30.250.23	153234	153169	153182	153234	153234	153234	153200.00	153234.00	153188.81	83.8	279.3
30.250.24	150287	150287	150287	150287	150287	150287	150287.00	150287.00	150287.00	51.2	7.8
30.250.25	148574	148544	148549	148544	148574	148574	148528.50	148574.00	148560.74	77.0	139.9
30.250.26	147477	147471	147455	147471	147477	147477	147463.00	147477.00	147477.00	78.5	25.3
30.250.27	152912	152841	152841	152835	152912	152912	152835.00	152912.00	152894.37	70.6	213.2
30.250.28	149570	149568	149570	149570	149570	149570	149541.00	149570.00	149569.86	61.3	377.3
30.250.29	149668	149572	149587	149668	149668	149601	149620.00	149668.00	149601.00	741.8	34.0
Avg.	104717.37	104664.37	104678.87	104683.53	**104717.07**	104698.60	104663.47	**104714.72**	104677.65	333.8	**175.6**
#better		0	3	4	14		9	26		8	
#equal		7	11	13	16		3	4		0	
#worse		23	16	13	0		18	0		22	
p-value		2.70E-5	2.71E-3	3.13E-2	9.79E-4		2.55E-2	8.30E-6			

QPSO* in terms of f_{best}, which is confirmed by the small *p-values* (≤ 0.05), but performs worse than the tabu-based TPTEA algorithm. In terms of f_{avg}, DQPSO* performs better than QPSO*, but worse than TPTEA.

The third experiment aims to assess the DQPSO* algorithm on the largest instances with $n = 500$, and the experimental results are respectively summarized in Tables 8–10 according to the value of m ($m = 5, 10, 30$), along with the results of the reference algorithms.

We observe from Table 8 that for the large instances with a small number ($m = 5$) of constraints, DQPSO* performs very well compared to six reference algorithms. In terms of f_{best}, DQPSO* obtains a better result respectively for 23, 10, 15, 13 and 15 out of 30 instances compared to five reference algorithms (GA, F&F, 3R-BPSO, TP+TS and QPSO*), while matching their best results for 7, 19, 15, 16, 12 instances, respectively. Such an outcome indicates that DQPSO* outperforms significantly these five reference algorithms, which is confirmed by the small *p-values* (≤ 0.05). In addition, compared to the latest TPTEA algorithm, DQPSO* obtains a better, equal, and worse result for 1, 25, and 4 instances in terms of f_{best}, which means that DQPSO* performs slightly worse than TPTEA. Nevertheless, the large *p-value* (≥ 0.05) means that there does not exist a significant difference between DQPSO* and TPTEA in terms of f_{best}. On the other hand, DQPSO* obtains a better result for 30, 30, and 18 instances in terms of f_{avg} compared to 3 reference algorithms (3R-BPSO, QPSO* and TPTEA). Moreover, DQPSO* and TPTEA have a very similar performance in terms of both f_{best} and f_{avg} with an advantage for DQPSO* in

Table 8: Comparative results of DQPSO* with 6 several reference algorithms from the literature on the large instances with $n = 500$ and $m = 5$.

Instance	Opt. Vimont et al. (2008)	f_{best}							f_{avg}				t_{avg}(s)	
		GA	F&F	3R-BPSO	TP+TS	QPSO*	TPTEA	DQPSO*	3R-BPSO	QPSO*	TPTEA	DQPSO*	TPTEA	DQPSO*
5.500.0	120148	120130	120134	120141	120134	120130	120148	120148	120101.50	120105.70	120126.90	120137.80	3753.2	110.6
5.500.1	117879	117837	117864	117864	117864	117844	117879	117864	117825.50	117834.30	117850.83	117852.40	3876.2	102.3
5.500.2	121131	121109	121131	121129	121112	121131	121131	121131	121103.50	121092.00	121112.23	121125.87	3148.0	99.6
5.500.3	120804	120798	120794	120804	120804	120752	120804	120804	120772.30	120740.30	120786.40	120795.59	2917.9	156.6
5.500.4	122319	122319	122319	122319	122319	122319	122319	122319	122310.30	122300.70	122319.00	122316.00	1936.3	96.9
5.500.5	122024	122007	122024	122024	122024	122024	122024	122024	121981.10	121981.70	122008.83	122014.48	4421.1	108.9
5.500.6	119127	119113	119109	119127	119127	119094	119127	119127	119090.50	119075.00	119120.50	119122.50	3419.2	109.7
5.500.7	120568	120568	120568	120545	120568	120536	120568	120568	120534.70	120513.30	120548.10	120564.88	2738.8	119.5
5.500.8	121586	121575	121575	121575	121575	121586	121575	121575	121537.10	121527.30	121559.17	121560.67	2719.1	136.8
5.500.9	120717	120699	120707	120717	120707	120685	120717	120717	120674.80	120662.30	120695.00	120707.86	3353.3	134.4
5.500.10	218428	218422	218428	218415	218428	218428	218428	218426	218397.10	218394.70	218411.27	218414.45	4100.0	103.4
5.500.11	221202	221191	221202	221191	221191	221202	221191	221202	221167.40	221152.30	221184.90	221177.44	3560.1	99.7
5.500.12	217542	217534	217534	217534	217534	217528	217542	217536	217518.30	217513.00	217525.90	217533.89	3836.3	119.0
5.500.13	223560	223558	223558	223560	223558	223560	223560	223560	223536.40	223537.70	223558.87	223559.98	2139.3	39.8
5.500.14	218966	218962	218966	218966	218966	218965	218966	218966	218933.60	218964.30	218966.00	218965.68	171.5	73.3
5.500.15	220530	220514	220530	220530	220530	220527	220530	220530	220493.20	220498.70	220528.07	220526.80	3183.0	68.2
5.500.16	219989	219987	219989	219987	219989	219943	219989	219989	219973.20	219931.30	219985.90	219988.26	2798.7	115.0
5.500.17	218215	218194	218215	218194	218194	218215	218215	218215	218168.50	218185.00	218200.37	218198.35	3700.4	86.5
5.500.18	216976	216976	216976	216976	216976	216976	216976	216976	216942.40	216955.30	216976.00	216976.00	607.9	67.5
5.500.19	219719	219693	219719	219709	219704	219719	219719	219719	219691.80	219698.00	219715.60	219717.02	2632.9	22.7
5.500.20	295828	295828	295828	295805	295828	295828	295828	295828	295786.70	295797.70	295828.00	295828.00	552.3	33.0
5.500.21	308086	308077	308079	308081	308083	308086	308086	308086	308069.90	308064.00	308081.87	308079.50	3225.2	99.4
5.500.22	299796	299796	299796	299796	299796	299788	299796	299796	299761.60	299778.00	299796.00	299796.00	637.5	20.7
5.500.23	306480	306476	306476	306478	306478	306480	306480	306480	306466.40	306466.30	306478.47	306478.56	2626.0	62.3
5.500.24	300342	300342	300342	300342	300342	300342	300342	300342	300322.20	300310.00	300340.67	300342.00	3454.1	23.2
5.500.25	302571	302560	302571	302560	302561	302560	302571	302571	302546.30	302547.00	302565.40	302562.25	2156.6	65.9
5.500.26	301339	301322	301329	301327	301329	301322	301339	301339	301310.40	301317.30	301330.67	301329.21	589.5	71.2
5.500.27	306454	306430	306430	306438	306454	306422	306454	306454	306422.20	306409.00	306454.00	306448.36	1707.3	97.3
5.500.28	302828	302814	302814	302828	302822	302828	302828	302828	302812.00	302808.70	302820.70	302823.70	2852.2	58.8
5.500.29	299910	299904	299904	299904	299904	299910	299910	299904	299888.50	299885.30	299901.80	299902.19	3491.8	128.7
Avg	214168.80	214157.83	214163.70	214162.20	214163.37	214157.67	**214168.07**	214167.47	214137.98	214134.87	214159.25	**214161.52**	2676.86	**87.70**
#better		0	1	0	1	3	4		0	0	9		0	
#equal		7	19	15	16	12	25		0	0	3		0	
#worse		23	10	15	13	15	1		30	30	18		30	
p-value		3.95E-5	5.58E-3	6.31E-4	1.63E-3	2.84E-3	3.43E-1		1.73E-6	1.73E-6	6.97E-2			

Table 9: Comparative results of DQPSO* with 6 reference algorithms from the literature on the large instances with $n = 500$ and $m = 10$.

Instance	Opt.	f_{best}							f_{avg}				t_{avg}(s)	
		GA	F&F	3R-BPSO	TP+TS	QPSO*	TPTEA	DQPSO*	3R-BPSO	QPSO*	TPTEA	DQPSO*	TPTEA	DQPSO*
10.500.0	117821	117726	117734	117790	117779	117744	117801	117779	117699.30	117733.50	117736.17	117754.82	3664.7	105.3
10.500.1	119249	119139	119181	119155	119190	119177	119200	119206	119125.00	119148.50	119137.47	119179.74	3354.9	152.7
10.500.2	119215	119159	119194	119211	119194	119215	119159	119215	119094.90	119146.50	119108.27	119162.61	4469.7	127.8
10.500.3	118829	118802	118784	118813	118813	118775	118829	118813	118754.20	118747.50	118793.93	118777.36	3150.1	39.8
10.500.4	116530	116434	116471	116470	116462	116502	116456	116509	116402.80	116449.50	116405.17	116460.83	3582.2	146.2
10.500.5	119504	119454	119442	119461	119504	119402	119483	119470	119426.60	119391.50	119441.80	119435.57	3646.9	157.4
10.500.6	119827	119749	119764	119764	119782	119827	119775	119827	119720.50	119784.00	119739.70	119782.76	3789.5	134.1
10.500.7	118344	118288	118309	118288	118307	118309	118323	118320	118243.30	118282.50	118258.27	118265.30	4228.5	146.7
10.500.8	117815	117779	117781	117752	117781	117721	117801	117781	117698.30	117710.00	117705.97	117763.24	3168.7	122.3
10.500.9	119251	119125	119183	119186	119186	119251	119196	119212	119127.30	119200.50	119161.90	119166.54	3338.8	124.7
10.500.10	217377	217318	217318	217345	217343	217308	217351	217365	217283.00	217289.50	217313.67	217326.03	3984.4	76.7
10.500.11	219077	219022	219036	219053	219036	219077	219059	219063	219002.10	219049.50	219022.70	219027.54	3598.2	136.0
10.500.12	217847	217772	217797	217755	217797	217797	217847	217847	217743.80	217772.00	217786.73	217760.63	4010.4	173.1
10.500.13	216868	216802	216843	216832	216836	216868	216868	216843	216770.90	216826.00	216836.33	216824.05	3403.7	125.3
10.500.14	213873	213809	213811	213843	213859	213795	213814	213843	213777.90	213783.00	213780.27	213817.08	4095.0	134.4
10.500.15	215086	215013	215021	215058	215034	215086	215086	215062	214992.90	215053.50	215049.57	215034.68	3622.3	166.4
10.500.16	217940	217896	217880	217896	217903	217868	217926	217931	217858.30	217853.00	217884.80	217884.36	4281.3	181.3
10.500.17	219990	219949	219969	219949	219965	219949	219984	219984	219906.60	219919.50	219947.37	219965.07	3908.2	102.0
10.500.18	214382	214332	214346	214351	214341	214382	214363	214382	214286.00	214364.00	214327.43	214337.14	3999.5	165.9
10.500.19	220899	220833	220849	220840	220865	220827	220887	220865	220806.30	220814.50	220864.43	220847.10	3211.7	152.5
10.500.20	304387	304344	304344	304344	304351	304344	304387	304387	304311.80	304329.50	304364.47	304354.01	3025.1	67.4
10.500.21	302379	302332	302345	302379	302333	302341	302379	302358	302315.60	302341.00	302364.47	302346.04	2883.6	98.6
10.500.22	302417	302354	302408	302396	302408	302417	302416	302408	302348.80	302386.50	302398.13	302399.10	3497.9	77.4
10.500.23	300784	300743	300743	300743	300757	300784	300784	300784	300712.20	300763.50	300758.80	300745.46	967.0	58.6
10.500.24	304374	304344	304357	304374	304344	304340	304374	304374	304341.00	304328.50	304361.13	304353.75	3509.5	112.6
10.500.25	301836	301730	301742	301796	301754	301836	301796	301766	301712.70	301787.50	301740.43	301752.48	3808.8	106.4
10.500.26	304952	304949	304911	304949	304949	304952	304952	304949	304944.00	304924.50	304952.00	304949.00	3132.6	16.2
10.500.27	296478	296437	296447	296438	296441	296437	296478	296459	296416.70	296432.00	296455.53	296444.16	3138.7	153.6
10.500.28	301359	301313	301331	301353	301331	301293	301359	301357	301290.40	301284.00	301349.27	301332.24	3187.4	163.1
10.500.29	307089	307014	307078	307072	307078	307002	307089	307089	307004.90	306963.50	307088.23	307068.83	2351.9	83.1
Avg.	212859.30	212798.70	212813.97	212821.87	212824.10	212820.87	212840.7	**212841.60**	212770.60	212795.30	212804.48	**212810.58**	3467.04	**120.26**
#better		0	0	2	2	7	14		0	10	15		0	
#equal		1	3	3	6	4	6		0	0	0		0	
#worse		29	27	25	22	19	10		30	20	15		30	
p-value		2.55E-6	5.59E-6	3.25E-4	3.35E-4	5.85E-3	6.89E-1		1.73E-6	1.48E-2	5.72E-1			

Table 10: Comparative results of DQPSO* with 5 reference algorithms from the literature on the large instances with $n = 500$ and $m = 30$.

		f_{best}						f_{avg}			t_{avg}(s)	
Instance	BKR	GA	F&F	TP+TS	QPSO*	TPTEA	DQPSO*	QPSO*	TPTEA	DQPSO*	TPTEA	DQPSO*
30.500.0	116056	115868	115903	115950	115991	115968	115952	115906.00	115897.20	115805.36	3969.2	346.1
30.500.1	114810	114667	114718	114810	114684	114769	114734	114661.00	114733.00	114633.84	3548.9	525.6
30.500.2	116741	116661	116583	116683	116712	116708	116741	116642.50	116619.10	116581.00	4413.0	334.7
30.500.3	115354	115237	115198	115301	115354	115313	115236	115062.50	115251.60	115206.66	3499.0	460.4
30.500.4	116525	116353	116474	116435	116435	116455	116372	116378.50	116364.80	116296.31	3436.2	484.0
30.500.5	115741	115604	115734	115694	115594	115734	115673	115583.50	115674.00	115631.11	3847.9	347.6
30.500.6	114181	113952	113996	114003	113987	114085	113961	113936.50	114037.10	113929.22	4784.7	732.4
30.500.7	114348	114199	114266	114213	114184	114278	114189	114135.50	114164.40	114043.17	4109.9	445.9
30.500.8	115419	115247	115419	115288	115419	115288	115319	115271.00	115221.43	115170.67	4043.3	524.3
30.500.9	117116	116947	117011	117055	116909	117112	117003	116909.00	116984.37	116920.85	3778.6	564.0
30.500.10	218104	217995	218068	218068	218068	218104	218043	218068.00	218069.60	218008.51	3163.2	328.6
30.500.11	214648	214534	214626	214562	214626	214645	214551	214546.50	214544.93	214473.44	3796.3	321.6
30.500.12	215978	215854	215836	215903	215839	215946	215883	215839.00	215898.80	215841.16	4007.6	376.1
30.500.13	217910	217836	217862	217910	217816	217910	217807	217816.00	217831.33	217774.97	3259.2	423.1
30.500.14	215689	215596	215592	215596	215544	215689	215601	215544.00	215602.07	215514.71	3945.5	437.2
30.500.15	215919	215762	215784	215842	215753	215840	215774	215753.00	215766.23	215698.08	3558.5	520.9
30.500.16	215907	215772	215824	215838	215789	215907	215871	215784.50	215857.23	215773.37	3176.9	316.2
30.500.17	216542	216336	216418	216419	216387	216542	216452	216387.00	216459.73	216348.59	3642.6	459.7
30.500.18	217340	217290	217225	217305	217217	217340	217290	217211.00	217304.30	217244.15	3460.7	457.6
30.500.19	214739	214624	214663	214671	214739	214739	214681	214686.50	214671.30	214626.78	3417.9	438.7
30.500.20	301675	301627	301643	301643	301643	301675	301643	301635.00	301641.63	301628.38	2849.2	580.0
30.500.21	300055	299985	299982	300055	299965	300055	300013	299963.50	300035.73	299944.24	3862.8	401.2
30.500.22	305087	304995	305062	305028	305038	305087	305055	305038.00	305080.47	304993.46	3784.6	482.8
30.500.23	302032	301935	301982	302004	301982	302015	302004	301982.00	301983.60	301928.64	3200.1	310.3
30.500.24	304462	304404	304413	304411	304346	304462	304404	304346.00	304427.53	304393.26	3102.0	296.8
30.500.25	297012	296894	296918	296961	296892	296999	296962	296892.00	296964.97	296864.15	3743.2	459.0
30.500.26	303364	303233	303320	303328	303287	303364	303360	303287.00	303335.60	303253.82	2828.3	402.5
30.500.27	307007	306944	306908	306999	306915	306999	306999	306915.00	306972.50	306930.64	3922.4	535.4
30.500.28	303199	303057	303109	303080	303169	303199	303162	303169.00	303168.53	303099.06	3283.0	486.2
30.500.29	300596	300460	300471	300532	300449	300596	300536	300449.00	300530.23	300499.76	3936.9	383.5
Avg.	211451.9	211328.9	211366.9	211386.2	211357.8	**211427.4**	211375.7	211326.6	**211369.8**	211301.9	3645.7	**439.4**
#better		3	12	15	10	27		22	30		0	
#equal		2	1	3	1	1		0	0		0	
#worse		25	17	12	19	2		8	0		30	
p-value		1.96E-6	3.87E-1	2.20E-1	1.0E-2	7.98E-6		8.22E-3	1.73E-6			

terms of computational efficiency.

Similarly, Table 9 also shows that DQPSO* performs very well on the instances with a medium-sized number ($m = 10$) of constraints in comparison with six reference algorithms. For f_{best}, DQPSO* yields respectively a better result for 29, 27, 25, 22, 19 instances compared to GA, F&F, 3R-BPSO, TP+TS and QPSO*. Moreover, compared to the TPTEA algorithm, DQPSO* obtains a better, equal, and worse result for 10, 6, and 14 instances, respectively. For the average value *Avg.* of f_{best}, the result of the DQPSO* algorithm is 212841.60 that is slightly superior to 212840.7 of the TPTEA algorithm. In terms of f_{avg}, DQPSO* is superior to three reference algorithms (3R-BPSO, QPSO*, and TPTEA) with a *Avg.* value of 212810.58 which is better than those of the reference algorithms. On the other hand, from the Wilcoxon tests, we observe that the differences between the DQPSO* algorithm and the first five reference algorithms are statistically significant both in terms of f_{best} and f_{best}, while there does not exist a significant difference between DQPSO* and TPTEA.

Table 10 reports the computational results for the instances with a large number ($m = 30$) of constraints, which are known to be the hardest instances among the tested instances. We observe from the table that for these instances, the DQPSO* algorithm has a comparable performance compared with the popular MKP algorithms. In terms of f_{best}, DQPSO* outperforms GA and QPSO* by obtaining a better result for 25 and 19 out of 30 instances, respectively. DQPSO* yields comparable results with respect to two tabu-based algorithms (F&F and TP+TS), which is confirmed by the large *p-values*. However, when comparing with the latest TPTEA algorithm, DQPSO* performs significantly worse in terms of f_{best}. Moreover, the average results of DQPSO* are much worse for most instances than QPSO* and TPTEA, even if it attains its solutions within a short computation time compared to TPTEA.

In summary, the above computational results and comparisons indicate that the proposed DQPSO* algorithm performs very well for the instances with $m \leq 10$ knapsack constraints in terms of both solution quality and computation efficiency in comparison with the compared algorithms from the literature. However, for the instances with a large number ($m = 30$) of constraints, the performance of DQPSO* decreases and fails to compete with the best performing algorithms. Moreover, DQPSO* has a fast convergence, but its results on a number of instances (especially the largest and the most constrained instances) are unstable across multiple runs, indicating that its robustness could be further improved.

5. Analysis and Discussions

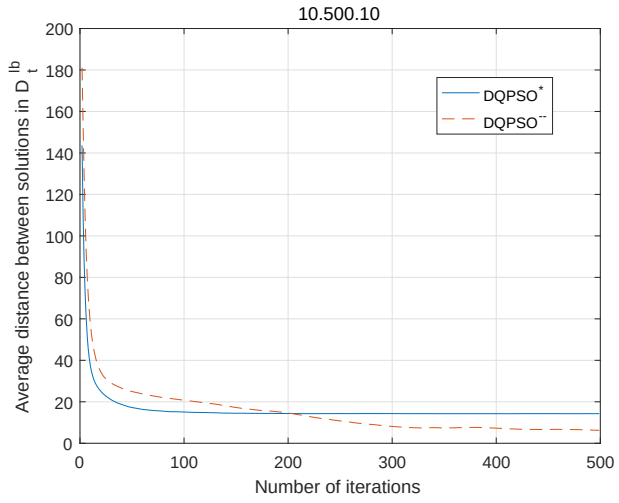
We now turn our attention to several essential components of the proposed algorithm to analyze their impacts on the performance of the algorithm, i.e., the diversity-preserving population updating mechanism, the variable neighborhood descent method, and the setting of parameter α.

Table 11: Comparison of the quantum particle swarm optimization algorithms with and without the diversity-preserving strategy. For each performance indicator, the better results between the compared algorithms are indicated in bold.

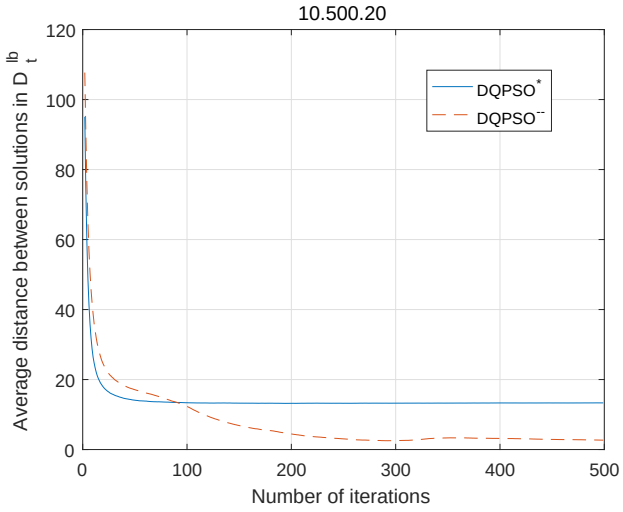
	DQPSO$^-$				DQPSO*			
Instance	f_{best}	f_{avg}	std	$t_{avg}(s)$	f_{best}	f_{avg}	std	$t_{avg}(s)$
10.500.0	117740	117684.52	30.94	9.59	**117779**	**117754.82**	**15.35**	105.33
10.500.1	119150	119099.22	21.54	7.60	**119206**	**119179.74**	**7.43**	152.69
10.500.2	119163	119080.89	38.38	11.94	**119215**	**119162.61**	**12.88**	127.80
10.500.3	118775	118727.86	38.22	6.27	**118813**	**118777.36**	**6.26**	39.85
10.500.4	116453	116366.48	31.19	15.32	**116509**	**116460.83**	**28.82**	146.17
10.500.5	119414	119361.04	25.56	12.64	**119470**	**119435.57**	**18.18**	157.44
10.500.6	119777	119727.00	19.22	2.28	**119827**	**119782.76**	**12.78**	134.07
10.500.7	118248	118177.46	44.65	11.67	**118320**	**118265.30**	**14.24**	146.69
10.500.8	117751	117678.45	38.86	10.08	**117781**	**117763.24**	**18.14**	122.31
10.500.9	119203	119129.00	**17.69**	8.52	**119212**	**119166.54**	18.03	124.73
10.500.10	217318	217260.40	30.77	11.56	**217365**	**217326.03**	**12.69**	76.68
10.500.11	219009	218945.35	26.79	8.89	**219063**	**219027.54**	**7.66**	136.02
10.500.12	217737	217678.75	27.76	20.05	**217847**	**217760.63**	**20.03**	173.14
10.500.13	216827	216753.71	32.00	10.38	**216843**	**216824.05**	**9.90**	125.30
10.500.14	213828	213761.85	34.59	27.07	**213843**	**213817.08**	**18.51**	134.41
10.500.15	215040	214968.59	32.35	9.06	**215062**	**215034.68**	**9.44**	166.43
10.500.16	217876	217799.39	32.60	13.21	**217931**	**217884.36**	**10.94**	181.26
10.500.17	219949	219885.72	32.90	9.87	**219984**	**219965.07**	**16.97**	101.98
10.500.18	214352	214275.96	37.26	24.41	**214382**	**214337.14**	**13.77**	165.88
10.500.19	**220865**	220782.14	16.30	3.75	**220865**	**220847.10**	**14.44**	152.54
10.500.20	304344	304295.74	27.32	6.62	**304387**	**304354.01**	**10.58**	67.44
10.500.21	**302371**	302307.26	28.17	5.41	302358	**302346.04**	**11.60**	98.60
10.500.22	302396	302329.70	21.11	6.26	**302408**	**302399.10**	**5.17**	77.43
10.500.23	300743	300687.89	18.39	3.45	**300784**	**300745.46**	**8.12**	58.63
10.500.24	304344	304335.83	**6.31**	2.15	**304374**	**304353.75**	7.66	112.58
10.500.25	**301766**	301687.16	22.15	14.47	**301766**	**301752.48**	**6.42**	106.35
10.500.26	**304949**	304887.98	25.37	7.13	**304949**	**304949.00**	**0.00**	16.24
10.500.27	296440	296404.64	23.12	14.06	**296459**	**296444.16**	**5.88**	153.58
10.500.28	301322	301280.43	12.35	3.99	**301357**	**301332.24**	**14.30**	163.07
10.500.29	307072	306988.09	32.10	13.67	**307089**	**307068.83**	**10.74**	83.13
Avg.	212807.40	212744.95	27.53	10.38	**212841.60**	**212810.58**	**12.23**	120.26
#best	4	0	2	**30**	**29**	**30**	**28**	0

5.1. Importance of the diversity-preserving Population Updating Strategy

The diversity-preserving population updating strategy of DQPSO* aims to maintain a healthy diversity for the population composed of the personal historical best positions of particles (D^{lb}). To investigate its influence on the performance of DQPSO*, we created a variant DQPSO$^-$ by replacing the population updating strategy of the algorithm (Algorithm 8) by a popular replacement strategy that is described in the lines 15-17 of Algorithm 1 and uses the current offspring solution d to replace $D^{lb}(i)$ which represents the historical discrete best position of current particle i, while keeping other ingredients of algorithm unchanged. We carried out an experiment on the set of 30 instances with $n = 500$ and $m = 10$ by running the DQPSO$^-$ and DQPSO* 100 times, according to the experimental protocol in Section 4.2. The experimental results are summarized in Table 11 with the same information as in the previous tables, where the row "#best" shows the number of instances for which the associated algorithm obtained the best result between the two compared algorithms for the considered performance indicator, and the best results of the compared results are indicated in bold. In addition, in order to investigate the influence of the diversity-preserving updating strategy on the diversity of the population D^{lb}, we ran the DQPSO$^-$ and DQPSO* algorithms 10 times on two representative instances (i.e., 10.500.10 and 10.500.20) and recorded the evolution of the average distance (denoted by $dist_{avg}$) between the solutions in D^{lb} respectively. The average results over 10 runs with a maximum number 500 of iterations are



(a)



(b)

Figure 1: Evolution of the average distance between solutions in the discrete particle swarm D^{lb} as a function of the number of iterations for the diversity-preserving population updating strategy (DQPSO*) and the popular population updating strategy in the literature (DQPSO$^-$).

summarized in Fig. 1, where Y-axis indicates the value of $dist_{avg}$ and X-axis indicates the number of iterations.

Table 11 shows that DQPSO* dominates DQPSO$^-$ in terms of f_{best}, f_{avg}, and std. First, in terms of f_{best}, both algorithms obtained the best result

respectively for 4 and 29 instances. Second, in terms of f_{avg}, DQPSO* obtained a better result for all the 30 instances. Third, the standard deviation *std* of the objective values from the DQPSO* algorithm is smaller than that of the DQPSO$^-$ algorithm, which implies DQPSO* is more robust than DQPSO$^-$. On the other hand, the computation time to reach the final objective value is much shorter for DQPSO$^-$ than for DQPSO*, which implies a premature convergence of DQPSO$^-$ compared to DQPSO*.

In addition, one observes from Fig. 1 that the average distance $dist_{avg}$ between the solutions in the population D^{lb}, which measures the diversity of population D^{lb}, decreases quickly at the beginning of search for both of the DQPSO$^-$ and DQPSO* algorithms, and then the $dist_{avg}$ value of the DQPSO* algorithm outperforms gradually that of the DQPSO$^-$ algorithm as the search progresses, which means the diversity-preserving updating strategy of DQPSO* is able to provide a better diversity for the population than the popular population updating strategy that is used in most existing binary PSO algorithms.

The above two experiments show thus that the diversity-preserving strategy helps DQPSO* to avoid a premature convergence and plays a crucial role for enhancing the performance of the algorithm.

5.2. Effect of the Variable Neighborhood Descent Method

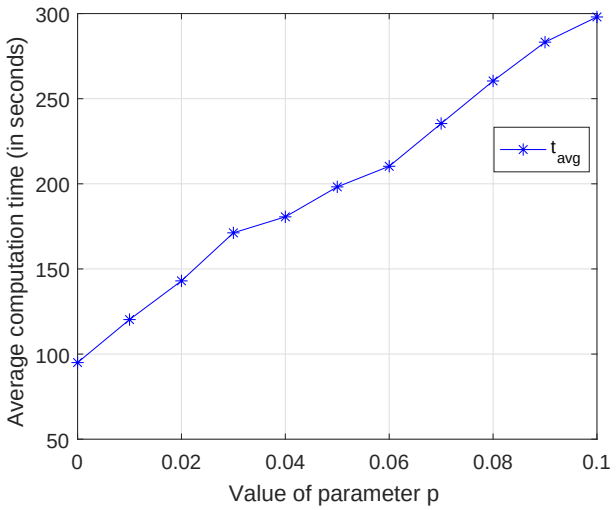


Figure 2: Evolution of the average computation time (i.e., t_{avg}) of the DQPSO* algorithm needed to reach the final objective value as a function of the value of p.

The VND procedure in Section 3.4 is another essential ingredient of the proposed algorithm and it is applied with a probability of p after each repair operator to reinforce search intensification. To investigate the effect of this local search method on the performance of the algorithm, we carried out another

Table 12: Computational results of DQPSO* with different p values on the large instances with $n = 500$ and $m = 10$.

p	0.0		0.01		0.02		0.03		0.04		0.1	
Instance	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
10.500.0	117799	117740.06	117779	117754.82	117779	117755.55	117779	117757.08	117790	117757.89	117790	117759.99
10.500.0	117799	117740.06	117779	117754.82	117779	117755.55	117779	117757.08	117790	117757.89	117790	117759.99
10.500.1	119211	119178.13	119206	119179.74	119206	119180.40	119206	119180.84	119200	119179.84	119206	119181.86
10.500.2	119215	119157.00	119215	119162.61	119181	119162.42	119181	119164.00	119181	119162.62	119187	119153.68
10.500.3	118813	118781.95	118813	118777.36	118813	118777.32	118813	118779.55	118813	118777.96	118813	118779.16
10.500.4	116509	116462.60	116509	116460.83	116509	116448.96	116509	116449.85	116509	116444.39	116481	116440.96
10.500.5	119475	119424.87	119470	119435.57	119470	119431.17	119481	119429.94	119470	119433.91	119466	119434.49
10.500.6	119813	119782.36	119827	119782.76	119813	119780.83	119813	119780.01	119813	119779.77	119813	119784.81
10.500.7	118296	118259.80	118320	118265.30	118333	118272.52	118320	118274.77	118333	118275.97	118320	118276.97
10.500.8	117781	117753.21	117781	117763.24	117781	117748.40	117781	117743.64	117781	117740.13	117781	117737.47
10.500.9	119207	119162.97	119212	119166.54	119203	119163.18	119211	119159.75	119218	119160.82	119251	119158.61
10.500.10	217365	217320.57	217365	217326.03	217377	217327.51	217377	217328.24	217377	217330.65	217377	217329.45
10.500.11	219042	219011.39	219063	219027.54	219063	219025.57	219050	219023.73	219063	219025.79	219063	219023.12
10.500.12	217792	217756.47	217847	217760.63	217792	217749.91	217847	217745.58	217847	217743.90	217847	217742.87
10.500.13	216840	216810.21	216843	216824.05	216840	216825.40	216868	216826.59	216843	216825.10	216843	216824.82
10.500.14	213842	213791.15	213843	213817.08	213850	213826.18	213843	213828.49	213846	213831.50	213843	213830.31
10.500.15	215041	215028.76	215062	215034.68	215045	215030.79	215062	215033.61	215050	215032.63	215050	215032.56
10.500.16	217895	217872.77	217931	217884.36	217931	217885.00	217931	217884.31	217931	217881.28	217931	217879.70
10.500.17	219977	219934.21	219984	219965.07	219984	219964.25	219984	219960.87	219984	219958.66	219984	219955.61
10.500.18	214346	214310.17	214382	214337.14	214352	214333.58	214352	214329.13	214352	214323.34	214346	214320.78
10.500.19	220865	220860.02	220865	220847.10	220865	220845.13	220872	220843.95	220865	220842.69	220865	220842.65
10.500.20	304387	304347.02	304387	304354.01	304363	304355.69	304387	304355.45	304370	304355.73	304363	304357.68
10.500.21	302358	302331.95	302358	302346.04	302371	302349.02	302371	302351.32	302371	302351.43	302371	302355.08
10.500.22	302408	302387.44	302408	302399.10	302408	302400.35	302408	302401.58	302411	302402.27	302411	302404.44
10.500.23	300784	300744.19	300784	300745.46	300784	300745.52	300784	300748.14	300784	300746.57	300784	300751.47
10.500.24	304357	304345.77	304374	304353.75	304374	304355.01	304366	304354.65	304374	304354.67	304366	304354.76
10.500.25	301774	301733.94	301766	301752.48	301766	301753.29	301768	301753.44	301766	301754.17	301767	301755.07
10.500.26	304949	304949.00	304949	304949.00	304950	304949.01	304950	304949.01	304950	304949.02	304950	304949.14
10.500.27	296456	296435.24	296459	296444.16	296459	296447.08	296457	296448.24	296466	296449.24	296466	296454.10
10.500.28	301357	301316.16	301357	301332.24	301357	301329.79	301357	301329.79	301357	301330.47	301357	301331.80
10.500.29	307072	307037.93	307089	307068.83	307089	307072.34	307072	307072.00	307089	307072.17	307089	307072.34
Avg.	212834.20	212800.91	**212841.60**	**212810.58**	212836.93	212809.71	212840.00	212809.59	212840.13	212809.15	212839.37	212809.19

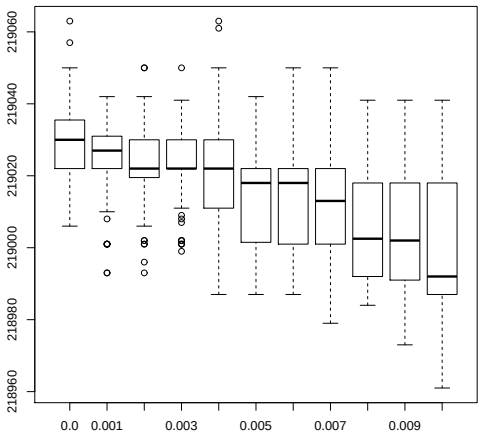
experiment based on the set of 30 instances with $n = 500$ and $m = 10$, where for each p value in the set $\{0.0, 0.01, 0.02, \ldots, 0.09, 0.10\}$, the DQPSO* algorithm was independently performed 100 times according to the experimental protocol in Section 4.2. It is worth noting that a larger value of p implies a higher computation effort and a stronger local optimization ability for the proposed algorithm, and vice versa. Specially, the setting of $p = 0.0$ means that the VND method is disabled in the algorithm.

The experimental results of this experiment are summarized in Table 12 and Fig. 2, where we show the results with p values in $\{0.0, 0.01, 0.02, 0.03, 0.04, 0.10\}$. The first column and the first row of the table give the name of instances and the settings of p, and the best objective value (f_{best}) over 100 runs and the average objective value (f_{avg}) for the tested p values are reported in columns 2–13. The last row "Avg." of the table gives the average value for each column, and the best result among the compared p values are indicated in bold in terms of f_{best} and f_{avg}. The average computation time (t_{avg}) needed to reach the final objective value is plotted in Fig. 2 as a function of p.

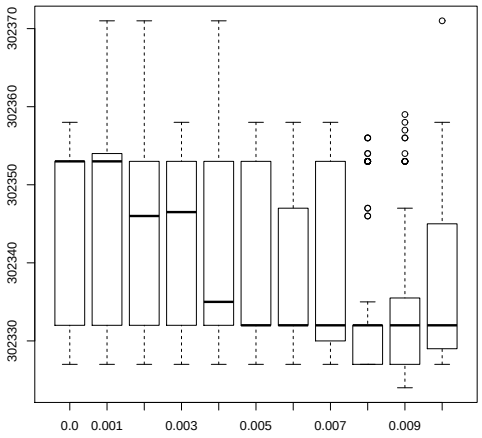
Table 12 shows that $p = 0$ that is equivalent to disabling the VND procedure leads to much worse results than the other values ($p > 0$) in terms of both f_{best} and f_{avg}, which means that the VND method plays a crucial role for the performance of the algorithm. Moreover, one observes that in terms of $Avg.$ the results with $p = 0.01$ are the best ones among the compared results, indicating that running the local search method more often with a probability $p > 0.01$ does not improve the final results in terms of f_{best} and f_{avg}. Meanwhile, Fig. 2 shows the computation time t_{avg} increases almost linearly with the increase of p, which confirms that the VND procedure is very time-consuming relative to other components of the algorithm. This experiment thus shows that 1) the VND procedure reinforces the performance of the algorithm; 2) applying VND too often (with a probability $p > 0.01$) only increases the computation time, without improving the search performance of the algorithm, and 3) a small p value ($p = 0.01$) is appropriate for the proposed algorithm to reach simultaneously a high performance in terms of computation time and solution quality.

5.3. Sensitivity Analysis of Parameter α

According to Eqs. (8)–(10) in Section 2.2, the DQPSO* algorithm uses the parameter α to control the appearance probability of items of D_t^* and D_t^{lb} in the discrete particle $D(i)$ generated by the evolution process and the *transform* function. A smaller α value means a larger appearance probability, and vice versa. To investigate the sensitivity of this parameter on the performance of the algorithm, we carried out an experiment based on two representative instances (10.500.11 and 10.500.21). We ran the DQPSO* algorithm 100 times for each instance and each α value from 0.0 to 0.01 with an increment of 0.001, and the experimental results are summarized in Fig. 3 with the popular box and whisker plots, where the X-axis represents the values of parameter α and the Y-axis represents the objective values obtained over 100 runs.



(a) Instance 10.500.11



(b) Instance 10.500.21

Figure 3: Sensitivity analysis of parameter α on two representative instances, where the X-axis represents the values of parameter α and the Y-axis represents the objective values.

One observes from Fig. 3 that a small α value leads generally to a better result than a large α value. For the instance 10.500.11, the result of the algorithm deteriorates as the value of α increases, and the setting of $\alpha = 0.0$ leads to the best result among all the tested α values. For the instance 10.500.21, the algorithm exhibits a similar behavior in general, and the setting of $\alpha = 0.001$ leads to the best result among all the tested α values. Thus, based on the outcomes of this experiment, the default value of α was set to 0.0 for the DQPSO* algorithm.

6. Conclusions and Future Work

We have presented a diversity-preserving quantum particle swarm optimization algorithm for solving the classic 0–1 multidimensional knapsack problem. In comparison with the popular QPSO algorithm, the proposed algorithm contains two new original features, namely a diversity-preserving population updating strategy to maintain a healthy diversity of particle swarm and a variable neighborhood descent procedure applied in a probabilistic way to reinforce search intensification.

The experimental results on 270 instances commonly used in the literature showed that the proposed algorithm is particularly efficient in terms of both the solution quality and the computational efficiency on the instances with a small or medium-sized number ($m \leq 10$) of constraints in comparison with several state-of-the-art MKP algorithms in the literature. As such, the algorithm can be advantageously applied to effectively find high-quality solutions for MKP instances with a limited number of constraints. However, the performance of the proposed algorithm decreases considerably on the tested instances with a large number ($m = 30$) of constraints, even if the algorithm remains very fast in terms of computation time. We also presented additional experiments to get insights on the interest of the diversity-preserving updating strategy, the local search procedure, as well as key parameters.

There are several potential directions for future research. First, the performance of the algorithm may vary across multiple runs on instances with many constraints. It is thus useful to investigate additional strategies to improve the robustness of the algorithm. Second, to enhance the effectiveness of the repair operator, different pseudo-utility ratios can be used in a combined way. Third, the ideas of the diversity-preserving updating strategy and the probabilistic application of local optimization are general and independent of the problem studied in this work. Consequently, it would be interesting to check their effectiveness and efficiency within other QPSO algorithms for MKP variants such as those mentioned in the introduction as well as other binary optimization problems (e.g., the set covering problem (Gao et al., 2015) and the maximum diversity problem (Wu and Hao , 2013)).

Acknowledgments

We are grateful to the reviewers for their valuable comments and suggestions which helped us to improve the paper. This work was partially supported by the Natural Science Foundation of Jiangsu Province of China (Grant No. BK20170904), the National Natural Science Foundation of China (Grant No. 61703213), six talent peaks project in Jiangsu Province (Grant No. RJFW-011), and NUPTSF (Grant Nos. NY217154 and RK043YZZ18004).

References

Al-Shihabi, S. & Ólafsson, S. (2010). A hybrid of nested partition, binary ant system, and linear programming for the multidimensional knapsack problem. *Computers & Operations Research*, 37(2), 247–255.

Angelelli, E., Mansini2012, R. & Speranza, M.G. (2010). Kernel search: A general heuristic for the multidimensional knapsack problem. *Computers & Operations Research* 37, 2017–2026.

Beheshti, Z., Shamsuddin, S. & Hasan, S. (2015). Memetic binary particle swarm optimization for discrete optimization problems. *Information Sciences* 299, 58–84.

Boussier, S., Vasquez, M., Vimont, Y., Hanafi, S. & Michelon, P. (2010). A multi-level search strategy for the 0–1 multidimensional knapsack problem. *Discrete Applied Mathematics* 158, 97–109.

Caserta M., & Voβ, S. (2019). The robust multiple-choice multidimensional knapsack problem. *Omega* 86, 16–27.

Chen, Y.N. & Hao, J.K. (2014). A "reduce and solve" approach for the multiple-choice multidimensional knapsack problem. *European Journal of Operational Research* 239(2), 313–322.

Chen, W.N., Zhang, J., Chung, H.S.H., Zhong, W.L., Wu, W.G. & Shi, Y.H. (2010). A novel set-based particle swarm optimization method for discrete optimization problems. *IEEE Transactions on Evolutionary Computation* 14(2), 278–299.

Chih, M. (2018). Three pseudo-utility ratio-inspired particle swarm optimization with local search for multidimensional knapsack problem. *Swarm and Evolutionary Computation* 39, 279–296.

Chih, M. (2015). Self-adaptive check and repair operator-based particle swarm optimization for the multidimensional knapsack problem. *Applied Soft Computing* 26, 378–389.

Chu, P.C. & Beasley, J.E. (1998) A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics* 4, 63–86.

Drake, J.H., Özcan, E. & Burke, E.K. (2016). A case study of controlling crossover in a selection hyper-heuristic framework with MKP. *Evolutionary Computation* 24(1), 113–141.

Drexl, A. (1988). A simulated annealing approach to the multiconstraint zero-one knapsack problem. *Computing* 40, 1–8.

Fréville, A. (2004). The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research* 155, 1–21.

Gao, C., Yao, X., Weise, T. & Li, J. (2015). An efficient local search heuristic with row weighting for the unicost set covering problem. *European Journal of Operational Research* 246, 750–761.

Garey, M.R. & Johnson, D.S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. A Series of Books in the Mathematical Sciences. San Francisco, Calif.: W. H. Freeman and Co.

Gavish, B. & Pirkul, H. (1982). Allocation of databases and processors in a distributed data processing, in: J. Akola (Ed.), Management of Distributed Data Processing, North-Holland, Amsterdam, pp. 215–231.

Gilmore, P.C. & Gomory, R.E. (1966). The theory and computation of knapsack functions. *Operations Research* 14(6), 1045–1075.

Glover, F. & Kochenberger, G.A. (1996). Critical event tabu search for multidimensional knapsack problems. In: Meta-hueristics, Springer, pp.407–427.

Haddar, B., Khemakhem, M., Hanafi, S. & Wilbaut, C. (2016). A hybrid quantum particle swarm optimization for the multidimensional knapsack problem. *Engineering Applications of Artificial Intelligence* 55, 1–13.

Hanafi, S.A. & Fréville, A. (1998). An efficient tabu search approach for the 0–1 multidimensional knapsack problem. *European Journal of Operational Research* 106, 659–675.

Ke, L., Feng, Zuren, Ren, Z. & Wei, X. (2010). An ant colony optimization approach for the multidimensional knapsack problem. *Journal of Heuristics* 16, 65–83.

Kennedy, J. & Eberhart, R.C. (1995). Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks IV* vol. 4, 1942–1948.

Kennedy, J. & Eberhart, R.C. (1997). A discrete binary version of the particle swarm algorithm. *IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation* 4104–4108.

Khemakhem, M., Haddar, B., Chebil, K. & Hanafi S. (2012). A filter-and-fan metaheuristic for the 0–1 multidimensional knapsack problem. *International Journal of Applied Metaheuristic Computing* 3(4), 43–63.

Ktari, R. & Chabchoub, H. (2013). Essential particle swarm optimization queen with tabu search for MKP resolution. *Computing* 95, 897–921.

Kong, M., Tian, Peng & Kao, Y. (2008). A new ant colony optimization algorithm for the multidimensional knapsack problem. *Computers & Operations Research* 35, 2672–2683.

Lai, X.J. & Hao, J.K. (2015). Path relinking for the fixed spectrum frequency assignment problem. *Expert Systems with Applications* 42, 4755–4767.

Lai, X.J., Hao, J.K., Glover, F. & Lü, Z.P. (2018). A two-phase tabu-evolutionary algorithm for the 0–1 multidimensional knapsack problem. *Information Sciences* 436, 282–301.

Lai, X.J., Hao, J.K., & Yue, D. (2019). Two-stage solution-based tabu search for the multidemand multidimensional knapsack problem. *European Journal of Operational Research* 274, 35–48.

Lin, G. & Guan, J. (2018). Solving maximum set k-covering problem by an adaptive binary particle swarm optimization method. *Knowledge-Based Systems* 142, 95–107.

Lin, C.J., Chern, M.S. & Chih, M. (2016). A binary particle swarm optimization based on the surrogate information with proportional acceleration coefficients for the 0-1 multidimensional knapsack problem. *Journal of Industrial and Production Engineering* 33, 77–102.

Liu, H., Cai, Z. & Wang, Y. (2010). Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. *Applied Soft Computing* 10, 629–640.

Mansini, R. & Speranza, M.G. (2012). CORAL: an exact algorithm for the multidimensional knapsack problem. *INFORMS Journal on Computing* 24(3), 399–415.

Mancini, S., Ciavotta, M., & Meloni, C. (2019). The multiple multidimensional knapsack with family-split penalties. *European Journal of Operational Research*, https://doi.org/10.1016/j.ejor.2019.07.052.

Mladenović, N. & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research* 24(1), 1097–1100.

Puchinger, J., Raidl, G.R. & Pferschy, U. (2009). The multidimensional knapsack problem: structure and algorithms. *INFORMS Journal on Computing* 22(2), 250–265.

Raidl, G.R. & Gottlieb, J. (2005). Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation* 13(4), 441–475.

Shih, W., 1979, A branch & bound method for the multiconstraint zero-one knapsack problem. *Journal of the Operational Research Society* 30, 369–378.

Shi, Y.H. & Eberhart, R.C. (1998). A modified particle swarm optimizer. *IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence* 69–73.

Vasquez, M. & Hao, J.K. (2001). A hybrid approach for the 0–1 multidimensional knapsack problem. Proc. of the 17th Intl. Joint Conference on Artificial Intelligence (IJCAI-01), pages 328-333, Seattle, Washington, USA, August 2001. Morgan Kaufmann Publishers.

Vasquez, M. & Vimont, Y. (2005). Improved results on the 0–1 multidimensional knapsack problem. *European Journal of Operational Research* 165, 70–81.

Vimont, Y., Boussier, S. & Vasquez, M. (2008). Reduced costs propagation in an efficient implicit enumeration for the 0–1 multidimensional knapsack problem. *Journal of Combinatorial Optimization* 15, 165–178.

Wang, L., Wang, S.Y. & Xu, Y. (2012). An effective hybrid EDA-based algorithm for solving multidimensional knapsack problem. *Expert Systems with Applications* 39(5), 5593–5599.

Wu, Q. & Hao, J.K. (2013). A hybrid metaheuristic method for the maximum diversity problem. *European Journal of Operational Research* 231(2), 452–464.

Yang, S., Wang, M. & Jiao, L. (2004). A quantum particle swarm optimization. In: Proceedings of the 2004 Congress IEEE Conference on Evolutionary Computation, Vol.1, pp.320–324.

Zhan, Z.H., Zhang, J., Li Y. & Shi, Y.H. (2011). Orthogonal learning particle swarm optimization. *IEEE Transactions on Evolutionary Computation* 15(6), 832–847.