An Ejection Chain Approach for the Quadratic Multiple Knapsack Problem

Bo Peng, Mengqi Liu, Zhipeng Lü, Gary Kochengber, Haibo Wang

PII: S0377-2217(16)30096-0
DOI: 10.1016/j.ejor.2016.02.043
Reference: EOR 13548

To appear in: *European Journal of Operational Research*

Received date: 7 July 2014
Revised date: 17 February 2016
Accepted date: 27 February 2016

Please cite this article as: Bo Peng, Mengqi Liu, Zhipeng Lü, Gary Kochengber, Haibo Wang, An Ejection Chain Approach for the Quadratic Multiple Knapsack Problem, *European Journal of Operational Research* (2016), doi: 10.1016/j.ejor.2016.02.043

Highlights

- This paper presents an ejection chain approach (ECA) to tackle the quadratic multiple knapsack problem (QMKP).
- To our knowledge, this is the first paper to employ the ejection chain to solve the QMKP.
- A powerful neighborhood construction phase based on greedy and random operators is embedded in our ECA procedure.
- ECA improves upper bounds for 34 instances and matches the best known results for the remaining ones except 6 cases.

An Ejection Chain Approach for the Quadratic Multiple Knapsack Problem

Bo Peng[a], Mengqi Liu[b,*], Zhipeng Lü[a], Gary Kochengber[c], Haibo Wang[d]

[a]*SMART, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, P.R. China*
[b]*Business School, Hunan University, Changsha, 410082, P.R. China*
[c]*Business School, University of Colorado Denver, Denver, USA*
[d]*Sanchez School of Business, Texas A&M International University, Laredo, TX 78041, USA*

Abstract

In an algorithm for a problem whose candidate solutions are selections of objects, an ejection chain is a sequence of moves from one solution to another that begins by removing an object from the current solution. The quadratic multiple knapsack problem extends the familiar 0-1 knapsack problem both with several knapsacks and with values associated with pairs of objects. A hybrid algorithm for this problem extends a local search algorithm through an ejection chain mechanism to create more powerful moves. In addition, adaptive perturbations enhance the diversity of the search process. The resulting algorithm produces results that are competitive with the best heuristics currently published for this problem. In particular, it improves the best known results on 34 out of 60 test problem instances and matches the best known results on all but 6 of the remaining instances.

Keywords: Ejection chain; Quadratic multiple knapsack problem; Adaptive perturbation; Metaheuristics

*Corresponding author.
Email addresses: `liumengqi163@126.com` (Mengqi Liu), `zhipeng.lv@hust.edu.cn` (Zhipeng Lü)

1. Introduction

The quadratic multiple knapsack problem (QMKP) (Hiley and Julstrom, 2006) extends the well-known 0-1 knapsack problem in two aspects. First, each knapsack possesses its own capacity, and each object can be assigned to at most one knapsack. Second, in addition to their individual values, objects have values in pairs that accrue to the total objective value when both objects in a pair are assigned to the same knapsack. The objective of QMKP is to fill the knapsacks with objects of maximum total value without exceeding the capacity of any knapsack. As a generalization and a combination of the multiple knapsack problem (Hung and Fisk, 1978) and the quadratic knapsack problem (Gallo et al., 1980), QMKP is known to be NP-hard (Hiley and Julstrom, 2006).

Meta-heuristic algorithms (Hiley and Julstrom, 2006; Sundar and Singh, 2010; García-Martínez et al., 2013, 2014) are powerful tools for handling the QMKP problem. Among these algorithms, local search is one of the most well-known techniques. However, local searches based on simple neighborhood moves may easily fall into the local optima. To overcome this drawback, a variable depth method called ejection chain approach examines a large search space by generating a sequence of interrelated simple moves to create compound moves. In the past two decades, ejection chain methods have been widely used to tackle a variety of challenging optimization problems (see Section 2.2). The current work is motivated by these applications to employ ejection chain methods for the QMKP.

Different from most local search heuristics that directly move from one solution to another, the ejection chain approach first moves to intermediate structures, called reference structures, before moving to another solution. During these procedures, a certain amount of infeasibility is imposed on the initial solution, which has to be ejected to obtain a new feasible solution. The ejection of infeasibility can be delayed to create a chain by moving to other reference structures. At each step of the chain, feasible solutions can be obtained by ejecting the infeasibility. Hence, the approach is termed ejection chain algorithm (ECA). The ejection chain approach can explore much larger search spaces in a compact manner than traditional local search heuristics based on simple neighborhood moves.

The main contributions of this paper are summarized as follows:

- To our knowledge, this work is the first to employ the ejection chain method to solve the QMKP. In addition, this technique has never been used to address other knapsack problems.
- Both greedy and random operators are embedded in the proposed ejection chain local search. This study also proposes an effective perturbation phase based on two specialized perturbation operators and an adaptive management mechanism.
- The performance of the ECA is tested on 60 benchmark instances that were extensively used in previous studies. The outcomes show the efficacy of this algorithm in terms of both solution quality and robustness. In particular, the ECA generates results competitive with those of state-of-the-art approaches presented in literature by improving the best known results on 34 instances and matching the best known results on all but 6 of the remaining instances.
- The effects of some important parameter settings and components of the proposed algorithm are analyzed.

The remainder of the paper is organized as follows: Section 2 presents the mathematical formulation of the QMKP and the related works. Section 3 describes the main components of the ECA. Section 4 presents the comprehensive computational results and comparisons between the ECA and some other best-performing algorithms in literature. The effects of several important components and the parameter settings of the proposed algorithm are analyzed in Section 5. Finally, Section 6 concludes this study and gives suggestions for future research directions.

2. Problem Formulation and Related Works

2.1. Mathematical formulation of QMKP

The QMKP involves assigning a set of objects into knapsacks, such that the total profit of all objects in the knapsacks is maximized without violating the capacity constraint of any knapsack. It includes a set $N = \{1, \dots, n\}$ of n objects and a set $M = \{1, \dots, m\}$ of m knapsacks. Each object $i \in N$ has a profit value v_i and a weight w_i. Each pair of objects $i \in N$ and $j \in N$ $(i \neq j)$ has a profit value v_{ij}, while each knapsack $k \in M$ possesses a capacity C_k. Each object should be assigned to at most one knapsack k such that the total weight of the objects in each knapsack k does not exceed its capacity C_k. The value of an assignment of objects N to knapsacks M is the sum of the linear values of the included objects and the quadratic values of the object pairs that fall into the same knapsack. In the QMKP, the objective is to maximize the total profit value V_{sum}. The decision variable x_{ik} is 1 if object i is assigned to knapsack k; otherwise, the value is 0. Thus, QMKP can be formulated as follows:

$$Max \quad V_{sum} = \sum_{i=1}^{n}\sum_{k=1}^{m} x_{ik}v_i + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{k=1}^{m} x_{ik}x_{jk}v_{ij}, \tag{1}$$

subject to

$$\sum_{k=1}^{m} x_{jk} \leq 1; \qquad j = 1, \dots, n, \tag{2}$$

$$\sum_{j=1}^{n} x_{jk}w_j \leq C_k; \qquad k = 1, \dots, m, \tag{3}$$

$$x_{jk} \in \{0, 1\}; \qquad j = 1, \ldots, n; \qquad k = 1, \ldots, m. \qquad (4)$$

In the above formulation, objective function (1) aims to maximize the total profit value. Constraint (2) guarantees that each object can be assigned to at most one knapsack. Constraint (3) ensures that the total weight of any knapsack does not exceed its capacity. Constraint (4) imposes binary restrictions on decision variables.

2.2. Related works

In this section, related works on the algorithms for solving the QMKP and applications of the ejection chain approach are briefly reviewed.

Hiley and Julstrom (2006) presented the first study on QMKP in literature. The authors introduced three heuristic methods, namely, greedy heuristic, stochastic hill-climber method, and genetic algorithm. Greedy heuristic method filled one knapsack with one object at a time by choosing an unassigned object with the maximum profit/weight ratio. Hill-climber method removed some objects from the knapsacks, and then refilled the knapsacks by applying the afore mentioned greedy heuristic method. Genetic algorithm encoded candidate solutions as strings with lengths equal to the number of objects and employed the hill-climber method as its mutation operator. Singh and Baghel (2007) presented a new steady-state grouping genetic algorithm for QMKP. Saraç and Sipahioglu (2007) proposed another genetic algorithm to solve QMKP. They developed a specialized crossover operator to generate feasible solutions and presented two distinct mutation operators. Sundar and Singh (2010) introduced an artificial bee colony (ABC) algorithm based on the swapping of unassigned objects with already assigned ones. Experimental results demonstrated the superiority of the approach over several reference algorithms in terms of solution quality. The computational results obtained by Wang et al. (2012) indicated that the branch and cut method can effectively solve the quadratic knapsack problem with multiple knapsack constraints.

Recently, García-Martínez et al. (2014) combined a novel local search procedure with an iterated greedy approach based on a tabu mechanism for QMKP. They extended the local search method proposed by Sundar and Singh (2010) to exchange any two objects assigned to different knapsacks. The tabu-based destruction mechanism stores the components that were recently removed from the incumbent solution via short-term memory and prevents these components from being added into the partial solution again. García-Martínez et al. (2013) also addressed the QMKP by using the strategic oscillation (SO) method. They defined critical levels for QMKP and designed strategies to exploit the constraint structure by effectively exploring solutions in the feasible and infeasible regions close to the constraint boundaries.

For applications of the ECA, Glover (1996) originally designed an ejection chain strategy to generate neighborhoods of compound moves with attractive properties for the traveling salesman problem. Rego and Roucairol (1996) employed an ejection chain procedure to generate compound moves to solve the vehicle routing problem. Yagiura et al. (2004) embedded the ejection chain approach into neighborhood construction combined with tabu search to address the generalized assignment problem. Other successful and recent applications of this methodology are detailed in Burke and Curtois (2010); Rego et al. (2010); Lozano et al. (2012); Kingston (2012); Sevaux et al. (2013).

3. Ejection Chain Algorithm

The ECA is initiated by selecting elements to undergo a change of state (e.g., to remove one object from its knapsack) (Glover, 1996). Then, it explicitly identifies a so-called reference structure, which is similar to but slightly different from a solution, for example violating some constraints or missing some elements. On the basis of several predefined transition rules, moves are generated from one reference structure to another, and back from reference structures to solutions. The transition rules, together with the reference structures, define the ejection neighborhood moves.

In general, the framework of the ECA consists of three phases: initial solution construction, ejection chain local search, and adaptive perturbation. More precisely, a greedy constructive algorithm first produces a promising solution as the initial solution. Then, it iteratively alternates between an ejection chain local search phase (to perform intensive search) and a perturbation phase (to discover new promising search spaces) to obtain a successful tradeoff between intensification and diversification.

3.1. Main framework

Algorithm 1 Framework of the ECA for the QMKP

```
1: Input: The benchmark instance for the QMKP, and the maximum computing time
2: Output: The best solution $S^*$ found so far
   /*Initial solution construction phase*/
3: $S^0 \leftarrow$ Init_Solution()                                /* Section 3.2 */
4: $S^* \leftarrow S^0$, no_improv_iter $\leftarrow 0$
5: while the maximum computing time is not reached do
6:     /*Ejection chain local search phase*/
7:     $S' \leftarrow$ Ejection_Chain($S^0$)                        /* Section 3.3 */
8:     /*Updating the best solution and the intermediate parameter*/
9:     if $S'$ is better than $S^*$ then
10:        $S^* \leftarrow S'$, no_improv_iter $\leftarrow 0$
11:    else
12:        no_improv_iter $\leftarrow$ no_improv_iter $+ 1$
13:    end if
14:    /*Adaptive perturbation phase based on both random and greedy strategies*/
15:    $S^0 \leftarrow$ Adaptive_Perturbation($S'$, no_improv_iter)    /* Section 3.4 */
16: end while
17: return $S^*$
```

The proposed ejection chain approach for the QMKP is outlined in Algorithm 1. First, *Init_Solution* is used to generate an initial solution S^0 by following a greedy constructive heuristic (line 3). Then, an ejection chain local search procedure *Ejection_Chain* employs a first-improvement strategy to obtain a local optimum S' (line 7). If the solution quality cannot be further improved, a perturbation phase *Adaptive_Perturbation* will adaptively perturb the incumbent solution S' to diversify the search (line 15). These two procedures are repeated until the stopping criterion (i.e., maximum computing time) is satisfied. During this process, S^* records the best solution found so far and *no_improv_iter* denotes the number of iterations without improving the best solution S^* (lines 9-13). The following sections describe the main components of the ECA.

3.2. Initial solution

Hiley and Julstrom (2006) presented a greedy constructive heuristic that can produce initial solutions for the ECA. The greedy constructive heuristic examines the relative value densities of objects to be assigned to knapsacks. Initially, all the knapsacks are empty. Then, an unassigned object is assigned to a knapsack with the highest value density. Each assignment of an object to a knapsack mandates the updating of value densities of the remaining unassigned objects with respect to that knapsack.

To facilitate precision, García-Martínez et al. (2014) defined $\Delta(i,k)$, which is denoted as the profit value associated with assigning object i to knapsack k in Eq. 5, and $D(i,k)$, which is defined as its division by the weight of object i in Eq. 6. This study uses this greedy constructive heuristic to iteratively assign an unassigned object to a knapsack if this operation generates the maximal value of $D(i,k)$. After each assignment, the objective function is calculated by Eq. 7.

$$\Delta(i,k) = V_i + \sum_{j \in N} x_{jk} V_{ij}, \qquad i \in N,\ k \in M, \tag{5}$$

$$D(i,k) = \Delta(i,k)/w_i, \qquad i \in N,\ k \in M, \tag{6}$$

$$f(S') = f(S) + \Delta(i,k), \qquad i \in N,\ k \in M. \tag{7}$$

3.3. Ejection chain local search phase

The proposed local search procedure uses a first-improvement based hill climbing algorithm with an ejection chain neighborhood. To form an ejection chain, ejection moves and trial moves are alternately executed. One type of ejection move is to remove an object j from its knapsack to generate an incomplete solution, where object j remains free. Such a solution is called the reference structure. Another type of ejection move is to shift an object to the knapsack from which another object has just been ejected in the previous ejection move. This type of ejection move is applied to reference structures, to form a chain effect. Finally, a trial move attempts to assign the free object j into the most profitable knapsack to make a complete solution at each chain level.

For example, if an object is ejected (as an ejection move) and immediately added into another knapsack (as a trial move), this is considered a shift move. That is, a shift move is a special case of ejection chain move. Similarly, a swap move is also a special case of ejection chain move.

Figure 1 illustrates an example. Object j_0 is first ejected from its knapsack k_0 to generate a reference solution. Given that the weight of object j_0 exceeds the excess weight of any knapsack, a trial solution that assigns object j_0 to any knapsack does not exist. Then, let j_2 be the object whose assignment to knapsack k_0 is the most suitable among the objects that satisfy the capacity constraint. j_2 is assigned to knapsack k_0, that is, the ejection move of j_2 is triggered by the ejection of j_0. At this point, a trial solution can be obtained through a trial move that assigns object j_0 to knapsack k_2. Otherwise, object j_0 is still free and object j_1 is selected according to the same rule and is assigned to knapsack k_2. Another trial solution is generated by performing a trial move that assigns object j_0 to knapsack k_1. This process is repeated until the stopping condition is met.

The ejection chain local search is presented in Algorithm 2. In general,

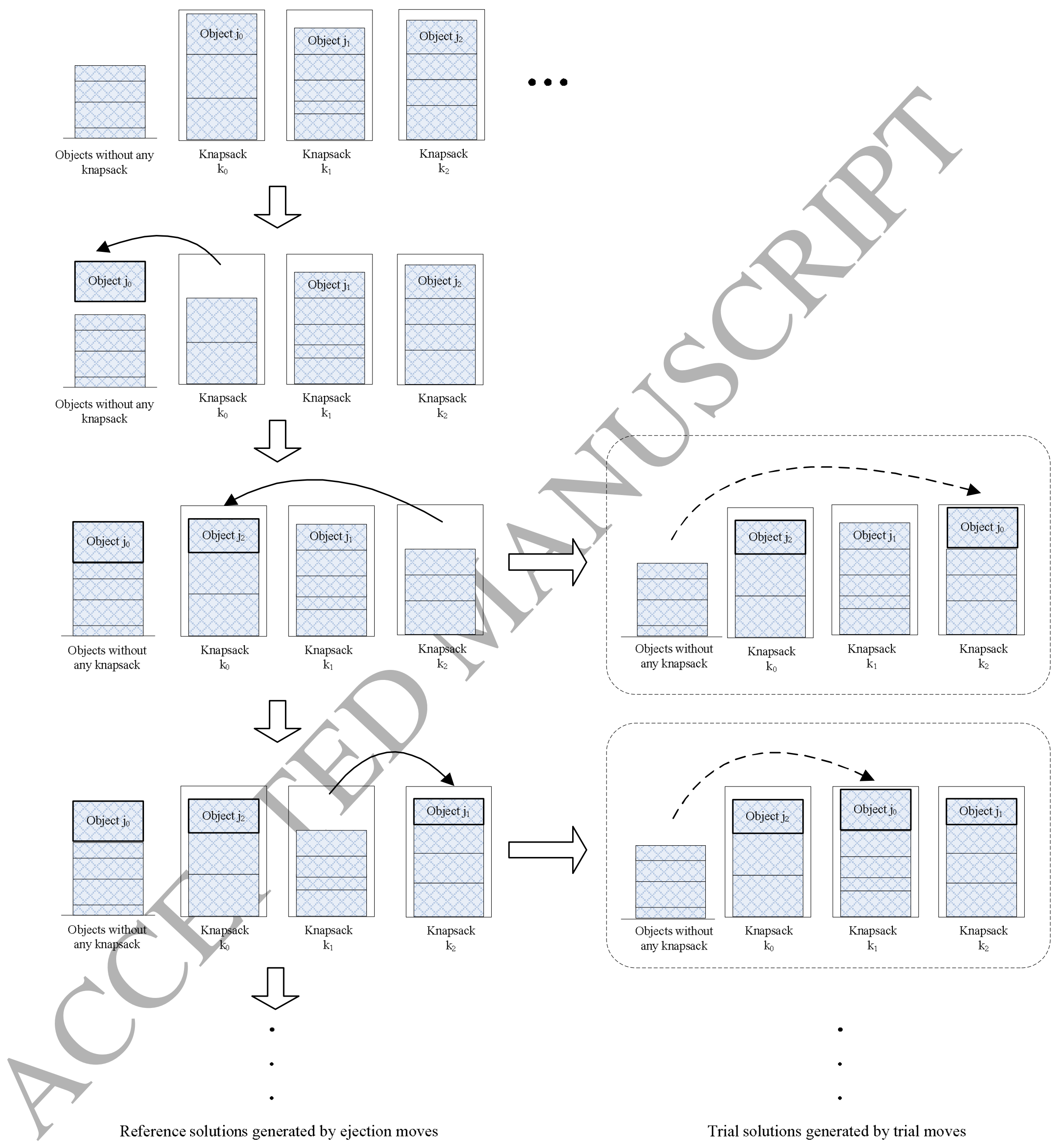
Object j0
Object j1
Object j2
Objects without any knapsack
Knapsack k0
Knapsack k1
Knapsack k2
Reference solutions generated by ejection moves
Trial solutions generated by trial moves

Figure 1: Procedure of ejection chain local search

Algorithm 2 Ejection chain local search phase

```
1: **Input**: Starting solution $S^0$
2: **Output**: The best solution $S'$ found in the local search
3: *len*: the length of the current constructing chain
4: $S$: the current solution, $S' \leftarrow S^0$, $S \leftarrow S^0$
5: $S^R$: the current reference solution, $S^T$: the current trial solution
6: while (the best solution $S'$ can be improved further) do
7:     $H \leftarrow$ the set of assigned objects in the current solution $S$
       //One iteration of the first-improvement based ejection chain local search (lines 8-26)
8:     for $j \in H$ do
9:         $S^R \leftarrow$ Ejection_Move1($S$, $j$), len $\leftarrow 1$
10:        while (len $\leq cl$) do
11:            //Constructing a trial solution and updating the current solution $S$ (lines 12-15)
12:            $S^T \leftarrow$ Trial_Move($S^R, j$)
13:            if $S^T$ is better than $S$ then
14:                $S \leftarrow S^T$, goto line 27
15:            end if
               //Choosing a reference solution based on greedy and random strategies (lines 16-23)
16:            for $l \in N$ and $l$ is not in the recently ejected object's knapsack do
17:                $S_l^R \leftarrow$ Ejection_Move2($S^R, l$)
18:            end for
19:            if rand[0,1) $\leq \alpha$ then
20:                $S^R \leftarrow$ arg best $\{S_l^R\}$
21:            else
22:                $S^R \leftarrow$ random $\{S_l^R\}$
23:            end if
24:            len $\leftarrow$ len $+ 1$
25:        end while
26:    end for
       //Updating the best solution (lines 27-29)
27:    if $S$ is better than $S'$ then
28:        $S' \leftarrow S$
29:    end if
30: end while
31: return $S'$
```

this local search is based on a first improvement strategy, which indicates that, for each iteration, once a trial solution is better than the current solution, the incumbent solution becomes the trial solution. This process is repeated until the local optimum is reached. Specifically, for current solution S, *Ejection_Move1* is applied to construct the first reference solution S^R by removing object j from its knapsack (line 9). If no better trial solution is obtained, then all assigned objects will be candidates for removal in constructing the initial reference solution (line 7). This phase then transforms into another type of process *Ejection_Move2* to iteratively generate subsequent candidate reference solutions (lines 10-25). As previously mentioned, two strategies based on greedy and random heuristics are employed to generate the reference solution (lines 19-23). Then, a trial solution S^T is generated by reassigning the first removed object j into the most profitable knapsack (i.e., maximum objective increment) by the process *Trial_Move* (line 12).

During the ejection chain local search procedure, the ECA employs two strategies to select the object in each ejection move: a greedy strategy based on the maximum objective increment value and a random strategy to randomly select a reference solution to enhance search diversification. At each iteration, the probability of selecting the greedy mechanism is α, whereas that of the random mechanism is $1 - \alpha$ (lines 19-23). The trial solution does not always exist if the weight of the first ejected object j exceeds the excess weight of any knapsack. In this case, the trial solution is the current reference solution, i.e., $S^T \leftarrow S^R$. In addition, the length of the chain is denoted as *len* and its value is set to 1 after the first reference solution is generated. In the whole procedure, the length of the chain cannot exceed the maximum value *cl*.

Important points of the ejection chain local search procedure are highlighted below:

- Each object can be selected at most once in one ejection chain search process starting by ejecting each assigned object.

- The set of objects without any knapsack is regarded as a group in one special knapsack whose capacity constraint is neglected. Objects without knapsacks are also candidates in constructing reference solutions.
- In an extreme situation of an ejection move, if the excess weight of the current knapsack is smaller than the weight of any object, the knapsack is replaced with another knapsack with the maximum excess weight to continue the procedure.

3.4. Adaptive perturbation phase

Adaptive perturbation phase aims to jump out of the search region of the incumbent solution and diversify the search to a new starting solution, while inheriting certain elite information from the current solution. As described in Algorithm 3, the proposed perturbation phase alternates between *Random_Remove* and two construction strategies: *Greedy_Construct* and *Greedy_and_Random_Construct*. First, *Random_Remove* randomly chooses a number of n_r assigned objects and removes them from their knapsacks. Then, the solution is iteratively constructed until no object can be further assigned to any knapsack. The selection of the two construction strategies *Greedy_Construct* and *Greedy_and_Random_Construct* depends on whether the number of iterations without improving the best solution (*no_improv_iter*) exceeds a predetermined value β. These construction mechanisms are illustrated as follows:

- *Greedy_Construct*: This strategy iteratively chooses one object i with the maximum $D(i, k)$ value from a set of unassigned objects, including those recently removed, to refill knapsack k without violating the capacity constraint. This mechanism is similar to the greedy constructive heuristic introduced in Section 3.2.

- *Greedy_and_Random_Construct*: At each iteration of this construction strategy, the best z candidate moves in terms of the values of $D(i,k)$ are selected and ranked in a non-descending order. Then, the probability of choosing the hth object-knapsack pair is $h/\sum_{i=1}^{z} i$, $(h = 1, \ldots, z)$.

Algorithm 3 Adaptive perturbation phase

```
1: Input: The starting solution $S'$ and the number of iterations no_improv_iter
2: Output: The constructed solution $S^0$
3: $S \leftarrow$ Random_Remove($S'$)
4: if no_improv_iter $< \beta$ then
5:     $S^0 \leftarrow$ Greedy_Construct($S$)
6: else
7:     $S^0 \leftarrow$ Greedy_and_Random_Construct($S$)
8:     no_improv_iter $\leftarrow$ 0;
9: end if
10: return $S_0$
```

4. Computational Results

The ECA was coded in C++ and run on a PC with a Quad-Core AMD Athlon 3 GHz CPU and 2 Gb RAM under a Windows 7 operating system. The experiments were conducted on two sets (d = 0.25 and 0.75) of 60 benchmark instances[1],[2]. These problem instances are characterized by the proportion d of non-zero profit v_{ij}, number of objects n, number of knapsacks m, and capacities of the knapsacks c (80% of the sum of the weights of all objects divided by the number of knapsacks).

4.1. Parameter setting

Table 1 presents the descriptions and settings of important parameters used in the ECA. The last column denotes the values of these parameters

[1]http://www.optsicom.es/qmkp/

[2]http://www.uco.es/grupos/kdis/kdiswiki/index.php/TIG-QMKP

for all the QMKP instances. Given that different groups exhibit varying characteristics, these parameters can be tuned with respect to each benchmark group. To demonstrate the robustness and effectiveness of the proposed ECA, a set of parameter values is fixed for all the benchmark instances, with the exception of threshold β.

Table 1: Settings of some important parameters in the ECA

Parameter	Description	Value
cl	The length of the longest allowable chain	6
α	The probability of employing greedy strategy in the ejection chain local search	$\frac{2}{3}$
β	The threshold used to alternate between *Greedy_Construct* and *Greedy_and_Random_Construct* in the adaptive perturbation phase	5 (for d = 0.25) and 2000 (for d = 0.75)
z	The number of elite candidates in the *Greedy_and_Random_Construct* process	3
n_r	The number of objects removed in the *Random_Remove* process	$\frac{n}{10}$

To evaluate the effectiveness of the proposed ECA algorithm and facilitate future comparisons, this study reports the detailed results of the ECA (best and average objective values, standard deviation (*SD*), and time required to generate the results).

The ECA is compared with the ABC algorithm by Sundar and Singh (2010), tabu-enhanced iterated greedy algorithm (TIG) by García-Martínez et al. (2014), and strategic oscillation algorithm (SO) by García-Martínez et al. (2013). ABC algorithm was run on a Linux based 3.0 GHz Core 2 Duo system with 2 GB RAM, whereas both SO and TIG were performed on a computer with a 2.8 GHz Intel Core i7903 processor with 12 GB RAM. The computer used to run the ABC is very similar to ours, whereas the computers employed by both SO and TIG are slightly faster than ours. Nonetheless, this study merely reports the actual running time on the computers without time conversion given that all tested computers are very similar to one another.

4.2. Computational results of the QMKP instances

The proposed ECA is tested on each instance 40 times, and each run stops when the time meets the average time *Time* as reported in literature.

As depicted in Tables 2 and 3, *Best*, *Avg*, and *SD* denotes the best objective values, the average objective values, and the standard deviations of the objective values, respectively. The last column *Tav* represents the average running time to obtain the best values by the ECA (in seconds).

For all 60 benchmark instances, the ECA does not detect solutions that are worse than the previous best known solutions except in six cases. The performance of this algorithm is also competitive in terms of runtime in comparison with other state-of-the-art algorithms. Specifically, the ECA improves the best known results on 34 out of 60 instances (18 instances with $d = 0.25$ and 16 instances with $d = 0.75$).

Table 2: Computational results and comparisons of QMKP instances with $d = 0.25$

Instances					ABC			TIG			SO			ECA			
n	m	I	c	Time	*Best*	*Avg*	*SD*	*Best*	*Avg*	*SD*	*Best*	*Avg*	*SD*	*Best*	*Avg*	*SD*	*Tav*
100	3	1	688	3.8	29139	28753	209.68	29138	28894.6	120.74	29234	29159.25	**48.43**	**29286**	**29261.03**	52.83	**1.28**
100	3	2	738	3.69	28443	28004	285.91	28473	28224.35	60.87	**28491**	28467	32.27	**28491**	**28478.13**	**28.07**	**1.5**
100	3	3	663	2.87	26901	26585.33	192.9	27013	26905.4	77.47	**27179**	**27155.35**	**31.79**	**27179**	27021.9	66.53	**1.41**
100	3	4	804	3.74	28568	28109.03	342.19	**28593**	**28573.6**	**10.56**	**28593**	28570.4	13.50	**28593**	28564.55	21.26	**2.05**
100	3	5	723	3.57	27849	27073.67	274.77	**27892**	27721.68	174.68	**27892**	27811.53	**20.56**	**27892**	**27826.2**	49.59	**1.84**
100	5	1	413	2.7	22390	22117.12	141.17	22264	22126	91.03	22509	22337.43	84.13	**22581**	**22413.05**	**68.29**	**1.41**
100	5	2	442	2.93	21584	21224.03	188.87	21580	21430.38	84.20	**21678**	21540.35	87.18	**21678**	**21541.73**	**64.81**	**1.76**
100	5	3	398	2.4	21093	20771.12	215.23	21100	21015.03	**34.45**	21188	**21104.75**	57.49	**21239**	21056	60.90	**1.02**
100	5	4	482	3.26	22178	21767.5	194.69	22180	22043	98.19	**22181**	**22136**	**56.43**	**22181**	22113.53	73.35	**2.19**
100	5	5	434	3.18	21301	20875.47	199.51	**21669**	21397.58	126.71	**21669**	21462.88	**70.51**	**21669**	**21540.68**	79.47	**1.58**
100	10	1	206	1.42	15953	15573.65	170.84	16118	15863	120.50	16065	15886.58	92.51	**16169**	**15992.68**	**74.38**	**0.87**
100	10	2	221	1.84	15487	14896.35	192.44	15525	15398.43	64.66	15510	15359.95	65.19	**15645**	**15459**	**60.77**	**0.82**
100	10	3	199	1.58	14339	14027.83	191.24	**14773**	14554	106.90	14663	14568.38	**63.75**	14771	**14635.78**	70.48	**1.04**
100	10	4	241	2.1	15807	15397	244.28	**16181**	16089.95	71.61	16159	16013	77.94	**16181**	**16105.05**	**49.61**	**1.14**
100	10	5	217	1.82	14719	14376.8	177.96	15150	15023.45	84.78	15130	15021.33	64.63	**15326**	**15114.4**	**63.47**	**0.95**
200	3	1	1381	23.99	100662	100103.02	283.79	100218	100056.23	**92.62**	101100	100653.5	181.44	**101128**	**100916.15**	145.36	**14.69**
200	3	2	1246	18.61	**107958**	107545.2	240.77	107787	107644.98	61.42	107805	107607.15	86.29	**107958**	**107860.1**	**56.22**	**7.53**
200	3	3	1335	29.85	104521	104006.98	311	104479	104251.5	79.93	**104538**	104271.68	91.71	**104538**	**104523.7**	**28.3**	**13.19**
200	3	4	1413	38.93	98791	98344.32	268.1	98896	98557.4	**142.53**	**99559**	**99003.63**	223.48	99327	99002.13	144.68	**21.83**
200	3	5	1358	29.22	102049	101406.48	344.51	101973	101635.43	93.83	102041	101667.73	160.54	**102256**	**102064.55**	**70.39**	**16.5**
200	5	1	828	19.88	74922	74132.95	519.19	74239	73977.78	**118.20**	74559	74237.4	169.65	**75095**	**74867.23**	138.94	**11.1**
200	5	2	747	16.75	79506	79073.32	278.65	79480	79234.28	107.32	79400	79153.55	100.12	**79745**	**79523.48**	**87.33**	**11.5**
200	5	3	801	22.86	77607	77069.52	244.68	77700	77420.5	179.10	77632	77452.25	150.44	**77897**	**77745.8**	**78.62**	**14**
200	5	4	848	28.07	73181	72607.25	372.38	73173	72477.65	247.03	73327	72884.03	182.63	**73654**	**73449.23**	**109.1**	**16.19**
200	5	5	815	20.74	76022	75455.98	248.11	75884	75693.18	116.06	75996	75751.38	125.09	**76538**	**76189.63**	**98.89**	**12.76**
200	10	1	414	10.37	49883	49079.47	425.35	51413	50845.78	193.36	51323	50862.7	156.42	**51674**	**51382.18**	**111.02**	**6.01**
200	10	2	373	8.48	53298	51831.55	459.79	54116	53608.45	169.08	53975	53649.03	139.02	**54330**	**53989.4**	**132.33**	**5.42**
200	10	3	400	11.15	52281	51324.28	359.07	52735	52456.28	143.13	52841	52337.73	**140.19**	**53127**	**52763.15**	150.78	**5.77**
200	10	4	424	12.83	49210	48190.6	466.33	50221	49656.4	204.40	50190	49802.43	**163.80**	**50870**	**50466.05**	187.95	**8.54**
200	10	5	407	10.99	51921	51437.97	296.41	52651	52328.38	164.68	52470	52211.58	**126.1**	**53294**	**52991.3**	149.92	**6.56**

The results of Tables 2 and 3 are also summarized in Table 4, where columns *Value* and *Number* represent the average values and the number of best results in terms of *Best*, *Avg*, and *SD* over all the QMKP instances, respectively.

Table 3: Computational results and comparisons of QMKP instances with $d = 0.75$

Instances					ABC			TIG			SO			ECA			
n	m	I	c	Time	*Best*	*Avg*	*SD*	*Best*	*Avg*	*SD*	*Best*	*Avg*	*SD*	*Best*	*Avg*	*SD*	*Tav*
100	3	1	669	2.07	69721	69373	231.72	**69977**	**69936.05**	**6.64**	69935	69925.73	37.77	69935	69758.1	113.42	**0.76**
100	3	2	714	1.86	69462	69041	236.38	**69504**	**69442.78**	**39.75**	**69504**	69442.48	49.40	**69504**	69386.43	69.66	**0.92**
100	3	3	686	1.86	68635	67960.05	406.74	68811	68811	**0**	**68832**	**68812.58**	5.6	**68832**	68804.05	14.12	**0.68**
100	3	4	666	1.88	69986	69687.68	217.28	**70028**	70019.88	32.91	**70028**	**70028**	**0**	**70028**	69948.95	61.62	**0.98**
100	3	5	668	2.12	69679	69136.4	246.88	**69692**	69638.48	24.32	69653	**69640.53**	**12.95**	**69692**	69603.65	64.96	**0.96**
100	5	1	401	2.07	4922	48937.47	196.48	49345	**49218.1**	**29.81**	49363	49197.53	55.7	**49397**	49213.88	47.03	**1.07**
100	5	2	428	1.96	49313	48908.05	202.63	**49316**	49081.53	81.88	49305	**49137.38**	82.1	49245	49029.98	**63.68**	**0.92**
100	5	3	411	1.71	48472	47874.5	380.3	**48495**	48327.48	85.51	**48495**	48287.88	77.08	**48495**	**48359.38**	**64.97**	**0.95**
100	5	4	400	1.83	50199	50017.93	197.12	49866	49866	**0**	**50246**	50025.03	97.69	**50246**	**50210.68**	22.49	**0.87**
100	5	5	400	2.01	48710	48409.75	133.12	**48752**	48619.6	65.78	**48752**	**48653.18**	**62.88**	48678	48568.25	71.27	**0.99**
100	10	1	200	1.22	29875	29429.2	208.12	29877	29548.68	102.80	29931	29788.48	**54.53**	**30000**	**29829.83**	85.65	**0.69**
100	10	2	214	1.45	30939	30697.8	134.28	30980	30832.15	83.73	30973	30829.05	**70.25**	**31050**	**30885.5**	74.34	**0.91**
100	10	3	205	1.3	29465	28983.78	246.93	29695	29439.95	142.78	29730	29519.48	112.25	**29837**	**29628.63**	**92.1**	**0.72**
100	10	4	200	1.4	31663	31218.85	176.33	31550	31333.45	**67.95**	31587	**31392.48**	71.15	**31696**	31385.83	91.45	**0.8**
100	10	5	200	1.42	30219	29736.47	213.33	30096	29895.4	**75.08**	30229	29918.7	94.01	**30348**	**30049.05**	89.65	**0.74**
200	3	1	1311	14.11	269736	267117.92	1070.76	**270718**	270525.9	204.46	**270718**	**270617.48**	**166.63**	**270718**	270520.73	196.88	**6.91**
200	3	2	1414	16.27	256195	253916.75	896.46	257090	256764.98	95.68	257026	256852.3	88.06	**257288**	**257102.95**	**80.86**	**8.09**
200	3	3	1342	11.87	268890	267079.03	1124.57	**270069**	269974.03	85.82	**270069**	269955.03	91.51	**270069**	**269999.8**	**81.78**	**7.37**
200	3	4	1565	30.64	246205	244618.4	1022.02	246704	246356.53	**101.17**	246882	246473.13	157.1	**246993**	**246546.6**	110.53	**14.59**
200	3	5	1336	10.46	279490	276605	1443.72	**279598**	**279572.3**	34.82	**279598**	279562.43	33.9	**279598**	279542.18	**27.46**	**5.96**
200	5	1	786	12.34	184448	183046.65	735.6	184909	184500.8	150.78	184822	184529	**107.67**	**184914**	**184576.5**	156.22	**7.14**
200	5	2	848	12.34	173575	171738.85	735.6	**174682**	174239.48	245.75	**174682**	174267	**139.36**	**174682**	**174304.93**	157.25	**6.84**
200	5	3	805	12.1	185107	185059.52	469.38	186443	186170.68	172.71	186526	186216.75	**123.93**	**186675**	**186308.55**	164.98	**5.44**
200	5	4	939	27.03	165273	164042.2	777.57	166358	166159.55	**97.56**	166584	166165.38	137.84	**166641**	**166307**	120.65	**15.83**
200	5	5	801	14.06	192764	190474.27	1021.33	193084	**192712.25**	118.40	193053	192702.25	**104.88**	**193097**	192680.08	155.6	**7.48**
200	10	1	393	7.64	111000	109624.73	714.45	112262	111889.75	169.32	112354	112043.68	**150.05**	**112537**	**112114.03**	190.58	**4.42**
200	10	2	424	9.96	103540	102603.18	522.59	105092	104669.83	169.32	**105151**	104781.5	162.83	105113	**104785.75**	**143.11**	**5.43**
200	10	3	402	8.04	112509	111388.2	509.42	113868	113510.55	194.64	113869	113563.08	**140.04**	**114071**	**113601.2**	163.22	**4.54**
200	10	4	469	14.81	96859	95681.7	545.54	98252	97807.73	155.00	98028	97747.55	**106.18**	**98321**	**98031.4**	171.40	**7.98**
200	10	5	400	8.21	115125	113909.6	590.96	116513	115856.3	240.19	116298	115807.53	**168.32**	**116551**	**116052.78**	234.12	**5.14**

Table 4: Summarized performance comparison with the best-performing reference algorithms

Algorithm	*Best*		*Avg*		*SD*	
	Value	*Number*	*Value*	*Number*	*Value*	*Number*
ABC	82325.6	1	82291.3	0	399.12	0
TIG	83404	16	83163.8	6	108.58	14
SO	83452	20	83233.7	11	97.08	23
ECA	83576.1	54	83366.6	43	95.91	23

Based on Table 4, the ECA outperforms other reference algorithms in terms of both *Value* and *Number* for all the evaluation criteria related to *Best*, *Avg*, and *SD*. The proposed ECA achieves better results than other reference algorithms in terms of the best and average results on 54 and 43 out of 60 instances respectively. The ECA also performs better than SO (and is significantly superior to other algorithms) in terms of standard deviation (ECA's 95.91 versus SO's 97.08), although both ECA and SO generate the best results on 23 instances.

In column 5 of Tables 2 and 3, *Time* represents the total running times of the reference algorithms for each specific instance. Although the average computing time *Tav* of the ECA to obtain the best results cannot be directly compared with *Time*, *Tav* is less than *Time* on all instances. If the total running time is set as *Time* in column 5, the ECA can still obtain the same results reported in the table, and outcome indicates that the ECA is comparable with the reference algorithms in terms of computational efficiency. In addition, previous best performing algorithms outperform CPLEX under the same time limit.

In summary, these experimental results demonstrate the competitiveness of the ECA in terms of solution quality, computational efficiency, and robustness.

5. Analysis and Discussion

5.1. Effect of ejection chain length

As mentioned in Section 3.3, computational efficiency of the ejection chain local search is related to the maximum chain length *cl*. To determine the effect of different *cl* values, various versions of the ECA are re-run with *cl* values of 2, 6, and 12. For each *cl* value, each problem instance is run 40 times. Other components and parameters are fixed as described in Section 4.1.

Table 5: Summarized performance comparison of the ECA with different *cl* values over all 60 instances

Algorithm	*Best*		*Avg*		*Tav*	
	Value	*Number*	*Value*	*Number*	*Value*	*Number*
ECA(cl = 2)	83516.5	24	83309.8	19	292.56	33
ECA(cl = 6)	83576.1	49	83366.6	41	318.57	21
ECA(cl = 12)	83534.6	22	83286.8	0	337.33	7

Table 5 summarizes the overall performance comparison among different versions of the ECA with varying *cl* values (2, 6, and 12). The notations are as provided above. *ECA*(cl = 6) obtains the best results for 49 out of 60 instances, unlike the other two versions. *ECA*(cl = 6) also yields the best results in terms of *Avg* for 41 problem instances, and outperforms *ECA*(cl = 12) on all the instances. Furthermore, *ECA*(cl = 12) consumes considerably more computational time than the other two variants. Specifically, the average time *Tav* for different values of *cl* over the 60 instances are 292.56, 318.57, and 337.33 for $cl = 2$, $cl = 6$ and $cl = 12$, respectively.

To summarize, *ECA*($cl = 6$) can facilitate a good tradeoff between solution quality and computational efficiency.

5.2. Effect of the adaptive perturbation operators

The proposed ECA relies on two perturbation operators, that use parameter β to adaptively control the probability of selecting each operator. To analyze the effect of different combinations of the two perturbation operators, the value of β is set to 2,000, 5, and 0, where $\beta = 0$ indicates that only the *Greedy_and_Random_Construct* strategy is used. The other two versions uses the combinations of these two strategies.

Table 6 summarizes the overall performance comparison of the different versions of the ECA given varying β values. The notations are as provided above. Given the problem set with $d = 0.25$, *ECA*(β = 5) yields better

Table 6: Summarized performance comparison of the ECA with different β values over all 60 instances

Algorithm	$d = 0.25$						$d = 0.75$					
	Best		*Avg*		*Tav*		*Best*		*Avg*		*Tav*	
	Value	*Number*	*Value*	*Number*	*Value*	*Number*	*Value*	*Number*	*Value*	*Number*	*Value*	*Number*
$ECA(\beta = 2000)$	49555.8	9	49350.7	1	186.09	5	117475	23	117238	22	126.12	0
$ECA(\beta = 5)$	49677.1	25	49495.3	20	192.45	7	117131	1	116745	0	101.17	0
$ECA(\beta = 0)$	49622.4	12	49376.3	6	178.05	15	116327	0	115579	0	39.59	30

results than the other two variants in terms of best and average solution quality. However, this variant consumes slightly more computational time than the other two variants.

For the problem set with $d = 0.75$, $ECA(\beta = 2000)$ dominates the other two variants on almost all instances (with only one exception) in terms of both *Best* and *Avg*. $ECA(\beta = 0)$ performs the best in terms of *Tav* but displays the lowest solution quality.

These observations show the importance of the adaptive perturbation mechanism and provide insight into the selection of appropriate β values for different QMKP instances.

6. Conclusion

This work presents the ECA to solve the QMKP. Key features of the proposed approach include a powerful chain local search procedure based on random and greedy strategies and an adaptive mechanism to enhance search diversification.

Experimental results on a set of benchmark instances show that the proposed approach competes favorably with the best-performing algorithms for the QMKP. In particular, the ECA can generate the current best solutions for most QMKP instances, and the best known results are improved on 34 out of the 60 tested instances.The ECA also displays a competitive performance in terms of both average results and standard deviation compared with the best reference algorithms presented in literature. The experiments also demonstrate the effectiveness of important parameters and the beneficiary of the proposed strategies in the perturbation phase.

This study can be extended in several directions. First, a powerful tabu search method can be employed to improve search capability of the ejection chain search phase. Second, outcomes may be enhanced further by investigating other neighborhood search operators and perturbation strategies. Finally, given that the various ideas introduced in this paper have been used to successfully solve the QMKP, the effectiveness of these concepts may be tested on other knapsack and assignment problems.

Acknowledgment

We greatly appreciate the anonymous referees for their helpful comments, which helped significantly improve the paper. We also thank Dr. Zhuo Wang for his kind assistance in facilitating the discussion on the ejection chain algorithm and Dr. Chance Carraway for his valuable suggestions to improve the paper. This research was supported in part by the National Natural Science Foundation of China under grant numbers 61370183 and 71471057, as well as by the program for New Century Excellent Talents in University (NCET 2013).

References

Burke, E. K., Curtois, T., 2010. An ejection chain method and a branch and price algorithm applied to the instances of the first international nurse rostering competition. In: Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling PATAT. Vol. 10. p. 13.

Gallo, G., Hammer, P. L., Simeone, B., 1980. Quadratic knapsack problems. In: Combinatorial Optimization. Springer, pp. 132–149.

García-Martínez, C., Glover, F., Rodriguez, F. J., Lozano, M., Martí, R., 2013. Strategic oscillation for the quadratic multiple knapsack problem. Computational Optimization and Applications, 1–25.

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

Glover, F., 1996. Ejection chains, reference structures and alternating path methods for traveling salesman problems. Discrete Applied Mathematics 65 (1), 223–253.

Hiley, A., Julstrom, B. A., 2006. The quadratic multiple knapsack problem and three heuristic approaches to it. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation. pp. 547–552.

Hung, M. S., Fisk, J. C., 1978. An algorithm for 0-1 multiple-knapsack problems. Naval Research Logistics Quarterly 25 (3), 571–579.

Kingston, J. H., 2012. Repairing high school timetables with polymorphic ejection chains. Annals of Operations Research, 1–16.

Lozano, M., Duarte, A., Gortázar, F., Martí, R., 2012. Variable neighborhood search with ejection chains for the antibandwidth problem. Journal of Heuristics 18 (6), 919–938.

Rego, C., James, T., Glover, F., 2010. An ejection chain algorithm for the quadratic assignment problem. Networks 56 (3), 188–206.

Rego, C., Roucairol, C., 1996. A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In: Meta-Heuristics. Springer, pp. 661–675.

Saraç, T., Sipahioglu, A., 2007. A genetic algorithm for the quadratic multiple knapsack problem. In: Advances in Brain, Vision, and Artificial Intelligence. Springer, pp. 490–498.

Sevaux, M., Rossi, A., Soto, M., Duarte, A., Martí, R., 2013. Grasp with ejection chains for the dynamic memory allocation in embedded systems. Soft Computing, 1–13.

Singh, A., Baghel, A. S., 2007. A new grouping genetic algorithm for the quadratic multiple knapsack problem. In: Evolutionary Computation in Combinatorial Optimization. Springer, pp. 210–218.

Sundar, S., Singh, A., 2010. A swarm intelligence approach to the quadratic multiple knapsack problem. In: Neural information processing. theory and algorithms. Springer, pp. 626–633.

Wang, H., Kochenberger, G., Glover, F., 2012. A computational study on the quadratic knapsack problem with multiple constraints. Computers & Operations Research 39 (1), 3–11.

Yagiura, M., Ibaraki, T., Glover, F., 2004. An ejection chain approach for the generalized assignment problem. INFORMS Journal on Computing 16 (2), 133–151.