

Minimum Complete Pareto Front of the Biobjective 0-1 Knapsack Problem

Lasko M. Laskov

0000-0003-1833-818

Informatics Department

New Bulgarian University

21 Montevideo Str., 1618 Sofia, Bulgaria

Email: llaskov@nbu.bg

Marin L. Marinov

0009-0003-9544-819X

Informatics Department

New Bulgarian University

21 Montevideo Str., 1618 Sofia, Bulgaria

Email: mlinar@nbu.bg

Abstract—We propose an exact algorithm that constructs the minimum complete Pareto front of the biobjective knapsack problem with one linear objective function (that represents profit) and one nonlinear objective function (that represents risk). The algorithm is composed by two stages. The first stage is using a dynamic programming scheme that constructs a Pareto optimal solution that maximizes the profit objective function, and it is applied in the standard setup of the knapsack problem in which the total weight of the items is bigger than the capacity of the knapsack. In the trivial setup of the problem, in which the total weight of the items is less or equal to the capacity, we apply a greedy algorithm that constructs the remaining solutions that are not constructed in the first stage.

We prove mathematically the correctness, the computational complexity and the memory complexity of the proposed algorithm. We demonstrate its execution using comprehensive numerical examples, and we demonstrate its applicability for large-scale instances in the provided experiments results.

I. INTRODUCTION

THE knapsack problem is a classical combinatorial optimization problem that is well known and studied in the literature. The 0-1 knapsack problem and its variants gets much attention because of its numerous applications in financial decision problems, securitization, cryptosystems, combinatorial auctions, and many more (see for example Chapter 15 of [1]).

As in the case of many optimization problems, in the presence of more than one objective function, we cannot find a single optimal solution, but we look for a set of non-dominated solutions also known as Pareto optimal solutions [2]. The set of Pareto optimal solutions is referred to as *Pareto optimal set* and also as *Pareto front*. For complex large-scale problems, an efficient way to describe the Pareto front is a minimum complete Pareto front, which is composed by a single Pareto optimal solution from each class of equivalent solutions. In this paper we show that the problem for construction of the minimum complete Pareto front for the biobjective maximum profit minimum risk 0-1 knapsack problem is pseudo-polynomial.

We propose an exact algorithm that constructs the minimum complete Pareto optimal set of biobjective 0-1 knapsack problems with one objective function being linear (profit) and the other being non-linear (risk). We prove mathematically the correctness of the described procedures and we

show analytically their computational complexity. We also demonstrate their execution using comprehensive numerical examples. Besides that, we provide extensive experiments that prove their applicability for large-scale input instances.

This paper is organized as follows. In the next section II we provide a short literature survey. In section III we present the terms and definitions that are used in the paper. In section IV we describe our algorithm that constructs a Pareto optimal solution that maximizes the linear objective function. Section V describes our algorithm that constructs a minimum complete Pareto front of the biobjective maximum profit minimum risk 0-1 knapsack problem. In section VI we provide our experiments results and finally in section VII we give our conclusions.

II. RELATED WORKS

Multiobjective and particularly biobjective versions of the 0-1 knapsack problem have been studied during last decades and still are subject of extensive research and both heuristic methods and exact algorithms are used [3].

The paper [4] considers the capital budgeting problem with two objective functions: maximizing the value of the accepted project and minimizing their risk. The proposed method is based on a heuristic procedure and it searches for the discrete efficient frontier of the problem.

In [5] the author solves biobjective 0-1 knapsack problem with both objective functions being linear. The proposed exact algorithm is based on a layered network representation that is used to construct parametric solution. The network representation of the problem is used to define objective value. The dominant solutions are constructed on a subnetwork that is a result of the objective value.

Another work that solves a 0-1 knapsack problem with two linear objective functions is [6]. The described exact algorithm is based on branch-and-bound design paradigm and discovers the entire Pareto optimal set. The efficiency of the branch-and-bound procedure depends on the order of the items according to their value to weight ratio, separately ranked according to both criteria.

In [7] two two-phase exact algorithms are proposed which are based on branch-and-bound paradigm. Both algorithms

share the same first phase in which the set of supported optimal solutions is determined by solving a parametrized version of the single-objective knapsack problem. The two second phases are branch-and-bound procedures, where the first visits in breadth-first and the second one visits in depth-first search manner a tree with nodes associated with solutions. The work presents extensive experiments with biobjective knapsack problems with number of items up to 500.

In [8] is described the biobjective 0-1 knapsack problem with one linear objective function and one nonlinear objective function from the type min-max, which according to authors show additional properties of the set of Pareto optimal solutions. The authors discuss that the heuristic approach to use linear combination of both objective functions lead to construction of a number of Pareto optimal solutions that is not close to the full set, and even decreases as the size of the problem grows. They propose an exact algorithm based on branch-and-bound paradigm that discovers the minimum complete Pareto front of the problem. Also, they describe their experiment results for random parameters of the problem in three different ranges $(0, 20]$, $(0, 100]$, $(0, 500]$ for number of items in the range $[10, 100]$.

An improvement of the dynamic programming approach in the case of multiobjective 0-1 knapsack problem is described in [9]. The improvement is based on complementary dominance relations that are used to discard partial solutions that cannot lead to new Pareto optimal solutions. Experiments with two and three linear objective functions are reported with instance sizes up to 4000. Latter, the work [10] proposes three alternative approaches to improve the work [9] in the case of two linear objective functions. Experiments with instances up to 4000 show improved computational time and reduced memory requirements.

A 0-1 knapsack problem with two linear objective functions is considered in [11]. The authors compare three standard multiobjective evolutionary algorithms and the point out that none of them actually result in the complete Pareto front of the problem. The reported experiments are with instances up to size 500.

In [12] three variants of the biobjective 0-1 knapsack problem are examined: unconstrained, bounded cardinality and fixed cardinality. The authors propose a local search metaheuristic method based on Pareto Local Search (PLS) method [13]. Also, they point out that the experiment results show that connected efficient sets are observed frequently without being caused by the size and structure of the input problem. The experiments are conducted on problem instances up to 1000 items and described method is reported to find the minimum complete Pareto front in less than an hour.

One of the state-of-the-art metaheuristic methods in the specialized literature is Ant Colony Optimization (ACO). An extensive reference about ACO and its applications can be found in the monograph [14]. The work contains a detailed description of the ACO method in the context of the multiple knapsack problem. It discusses in details the heuristic model, the possibilities for placing a pheromone and four variants

for transition probability model. Also, the recent work [15] describes an ACO method for multiple knapsack problem that adopts a local search procedure. The local search procedure is applied at the end of each iteration of the ACO method and is used to overcome the strong constraints of the problem. The authors report that the proposed solution is tested on the standard library of freely accessible examples Operational Research Library *OR-Library*¹.

In the recent article [16] the authors describe a biobjective 0-1 knapsack problem with conflicts which is a variant of the 0-1 knapsack problem in which the objective is to maximize the profit and to minimize the penalty of mutually conflicting choices. They propose an exact algorithm which is based on an improvement of ε -constrained method called AUGMECON2 and constructs the complete Pareto set. In the paper are also given extensive experimental results verifying the proposed algorithm.

The examples mentioned above and also many other references in the literature pay a special attention to the computability of the multiobjective and particularly to the biobjective 0-1 knapsack problems and also of the other versions of the knapsack problem, such as the subset sum problem [17]. Often the problems are reported to be NP-hard or even NP-complete and therefore hard to compute for large instances. The goal of our research is to show that in many cases the biobjective 0-1 knapsack problem is weakly NP-hard and pseudo-polynomial exact algorithm can find its solution even for large examples. In the case of the construction of a minimum complete Pareto front, we propose a pseudo-polynomial exact algorithm.

III. NOTATIONS AND DEFINITIONS

Given the set of n number of items $I = \{1, 2, \dots, n\}$ labeled with the first n natural numbers, we are given the vector of items profits $p = (p_1, p_2, \dots, p_n)$, the vector of items risks $r = (r_1, r_2, \dots, r_n)$ and the vector of items weights $w = (w_1, w_2, \dots, w_n)$. For all items $i \in I$ the corresponding numbers p_i , r_i and w_i are integers and are strictly bigger than 0. Also, by the positive integer c we denote the capacity of the knapsack which is also an integer.

For each ordered binary n -tuple $x = (x_1, x_2, \dots, x_n)$, $x_i \in \{0, 1\}$ for all $i \in I$, we define the following three functions. The function $f(x)$ that is the *profit* of the solution x :

$$f(x) = \sum_{i \in I} p_i x_i. \quad (1)$$

The function $g(x)$ that is the *risk* of the solution x :

$$g(x) = \max\{r_i x_i : i \in I\}. \quad (2)$$

The function $h(x)$ that is the *weight* of the solution x :

$$h(x) = \sum_{i \in I} w_i x_i. \quad (3)$$

¹OR-Library currently can be found on the following Internet web-address: <https://www.brunel.ac.uk/~mastjjb/jeb/info.html>.

Then the biobjective knapsack problem $KP_n(c)$ with n items I and capacity c is given by (4).

$$KP_n(c) \begin{cases} \text{maximize } f(x) \\ \text{minimize } g(x) \\ \text{subject to } h(x) \leq c, \\ x = (x_1, x_2, \dots, x_n), x_i \in \{0, 1\} \forall i \in I \end{cases} \quad (4)$$

Let both $x = (x_1, x_2, \dots, x_n)$ and $y = (y_1, y_2, \dots, y_n)$ be ordered binary n -tuples.

Definition 1. When for the binary n -tuple x the inequality $h(x) \leq c$ is fulfilled, we call x a **feasible solution** of the biobjective knapsack problem $KP_n(c)$.

Let us denote by $h(I)$ the total weight of all items in I , in other words:

$$h(I) = \sum_{i \in I} w_i. \quad (5)$$

Then if $h(I) > c$ we say that $KP_n(c)$ is in a *standard setup*, and if $h(I) \leq c$ we say that $KP_n(c)$ is in a *trivial setup* in which all items can fit in the capacity of the knapsack.

Definition 2. When $f(x) \geq f(y)$, $g(x) \leq g(y)$ and at least one of the inequalities is strict, we say that the feasible solution x **dominates** the feasible solution y , denoted by $y \prec x$.

Definition 3. When both $f(x) = f(y)$ and $g(x) = g(y)$, we say that the feasible solution x is **equivalent** to the feasible solution y , denoted by $x \sim y$.

Definition 4. When for the feasible solutions x and y at least one of the relations $y \prec x$ and $x \sim y$, we will denote it by $y \preceq x$.

Definition 5. A feasible solution x is a **Pareto optimal solution**, when there does not exist any other feasible solution y , such that $x \prec y$.

We denote by P the set of all Pareto optimal solutions of the biobjective knapsack problem $KP_n(c)$. The set P is decomposed into K number of disjoint subsets $P_1, P_2, \dots, P_K$ by applying the equivalence criterion given in Definition 3:

$$P = \bigcup_{j=1}^K P_j. \quad (6)$$

We call each of the subsets P_j , $j = 1, 2, \dots, K$ a *class of equivalent Pareto optimal solutions*, and the following two statements hold for them:

- all Pareto optimal solutions in P_j , $j = 1, 2, \dots, K$ are equivalent among each other;
- if $j \neq k$, there does not exist a solution $x \in P_j$ and a solution $y \in P_k$ for which $x \sim y$.

It is easy to verify that if $j \neq k$, then for $x \in P_j$ and $y \in P_k$ one of the following statements is true:

- $f(x) > f(y)$ and $g(x) > g(y)$ or
- $f(y) > f(x)$ and $g(y) > g(x)$.

Also, in this paper we use a numeration of the classes of equivalent Pareto optimal solutions, such that

$$l_j > l_{j+1} \text{ and } t_j > t_{j+1}, \quad (7)$$

where $l_j = f(x)$ and $t_j = g(x)$ for each Pareto optimal solution $x \in P_j$ and each $j = 1, 2, \dots, K - 1$.

Now let x be some feasible solution. Then there exists at least one $j = 1, 2, \dots, K$, for which the inequalities hold:

$$l_j \geq f(x) \text{ and } t_j \leq g(x). \quad (8)$$

If x is not a Pareto optimal solution, then at least one of inequalities (8) is strict. Particularly, this means that there does not exist a feasible solution for which at least one of the following inequalities holds:

$$f(x) > l_1 \text{ and } g(x) < t_K, \quad (9)$$

where l_1 is the profit of the solutions in the class P_1 and t_K is the risk of the solutions in the class P_K .

We denote by X the set of all feasible solutions of $KP_n(c)$ and for each $j = 1, 2, \dots, K$ we define the corresponding subset of X :

$$X_j = \{x \in X : g(x) < t_j\}. \quad (10)$$

Also, we denote by X_f is the set of all feasible solutions x , for which $f(x) = \max\{f(y) : y \in X\}$.

Definition 6. A **minimum complete Pareto front** is each ordered K -tuple $P^* = (x^{(1)}, x^{(2)}, \dots, x^{(K)})$ of Pareto optimal solutions $x^{(j)}$, such that $x^{(j)} \in P_j$ for all $j = 1, 2, \dots, K$.

We define the following three relations and an operation over the set of ordered pairs of numbers. Let $a = (a_1, a_2)$ and $b = (b_1, b_2)$ be two ordered pairs of numbers.

Definition 7. We say that the ordered pair a **dominates** the ordered pair b when at least one of the following statements is true:

- $b_1 < a_1$;
- $b_1 = a_1$ and $a_2 < b_2$.

We denote that a dominates b by $b \prec a$.

Definition 8. We say that two ordered pairs a and b are **equivalent** when $a_1 = b_1$ and $a_2 = b_2$. We denote that a and b are equivalent by $b \sim a$.

Definition 9. When $b \prec a$ or $b \sim a$ we will denote it by $b \preceq a$.

Definition 10. We define the operation **direct sum** of two ordered pairs of numbers a and b denoting it by $a \oplus b$, where:

$$a \oplus b \sim (a_1 + b_1, \max\{a_2, b_2\}). \quad (11)$$

Proposition 1. Let $a = (a_1, a_2)$, $b = (b_1, b_2)$, $y = (y_1, y_2)$ and $z = (z_1, z_2)$ be ordered pairs of numbers. Then the following properties hold.

- 1) For a and b exactly one of the following relations hold:
 - a) a dominates b , $b \prec a$;
 - b) a and b are equivalent, $b \sim a$;
 - c) b dominates a , $a \prec b$.

- 2) The relation dominance given in Definition 7 is transitive, namely if $z \prec y$ and $y \prec a$, then $z \prec a$.
- 3) The operation direct sum given in Definition 10 is associative, namely $(a \oplus b) \oplus y \sim a \oplus (b \oplus y)$.
- 4) If $y \prec a$ and $z \prec b$, then $y \oplus z \prec a \oplus b$.

Proof: The proof of the proposition follows directly from the corresponding definitions given above. ■

IV. MAXIMUM PROFIT PARETO OPTIMAL SOLUTION

In this section we describe our algorithm that constructs a Pareto optimal solution that maximizes the linear criterion of the biobjective knapsack problem $KP_n(c)$. Formally, we solve the following Problem 1.

Problem 1. Given a biobjective maximum profit minimum risk knapsack problem $KP_n(c)$ defined by (4) with the vectors p , r and w , find a Pareto optimal solution x^* of $KP_n(c)$, such that $f(x^*) = \max\{f(x) : x \in X\}$.

To solve Problem 1 we extend the bottom-up dynamic programming algorithm for standard single-objective knapsack problem [1]. The proposed procedure constructs the table of optimal solutions of the subproblems v and simultaneously finds the profit $f(x^*)$ and the risk $g(x^*)$ of a feasible solution x^* for which

$$(f(x), g(x)) \preceq (f(x^*), g(x^*)), \text{ for all } x \in X. \quad (12)$$

For this purpose we modify the Bellman recurrent relation [18] for the biobjective knapsack problem $KP_n(c)$. Let $b_i(j)$ denotes the optimal solution value for a capacity j and newly selected item i . If the values of $b_{i-1}(j)$ are already calculated for all $j = 0, 1, \dots, c$, then the new item i can be selected according to the Bellman recurrence relation:

$$b_i(j) = \begin{cases} b_{i-1}(j) & \text{if } j < w_i, \\ \max\{b_{i-1}(j), b_{i-1}(j - w_i) \oplus (p_i, r_i)\} & \text{if } j \geq w_i, \end{cases} \quad (13)$$

where in this case we consider the operator $\max\{a, b\}$ in the terms of the dominance relation between two ordered pairs of numbers a and b given in Definition 9:

$$\max\{a, b\} = \begin{cases} a & \text{if } b \preceq a, \\ b & \text{otherwise.} \end{cases} \quad (14)$$

The base case of the recurrence relation (13) is $b_i(j) = (0, 0)$ when either of i or j is equal to 0.

Then, the selected items that form the optimal solutions of the subproblems are stored in the table v according to the relation

$$v[i, j] = \begin{cases} 1 & \text{if } b_{i-1}(j) \prec b_{i-1}(j - w_i) \oplus (p_i, r_i), \\ 0 & \text{otherwise,} \end{cases} \quad (15)$$

where 1 denotes that the item is selected and 0 denotes that the item is not selected.

The set of items is represented by an array $I = [1, 2, \dots, n]$ of integer indexes. We need this representation latter in section

Algorithm 1 Procedure POS-MAX-PROF(I, p, r, w, c, b, v)

Input: item array I , vectors p, r, w , capacity c , arrays b, v

Output: Pareto optimal solution x^* , its profit and risk $b[c]$

```

1: set all values of  $b[0 : c]$  to  $(0, 0)$ 
2: for all  $i \in I$  do
3:   for  $j \leftarrow c$  down to  $w_i$  do
4:     if  $b[j] \prec b[j - w_i] \oplus (p_i, r_i)$  then
5:        $b[j] \leftarrow b[j - w_i] \oplus (p_i, r_i)$ 
6:        $v[i, j] \leftarrow 1$ 
7:     else
8:        $v[i, j] \leftarrow 0$ 
9:     end if
10:  end for
11: end for
12: let  $x^*[1 : n]$  be an array initialized with 0
13:  $m \leftarrow c$ 
14: for  $i \leftarrow |I|$  down to 1 do
15:   if  $v[i, m] = 1$  then
16:      $x^*[i] \leftarrow 1$ 
17:      $m \leftarrow m - w_i$ 
18:   end if
19: end for
20: return  $x^*, b[c]$ 
```

V to determine the subset of items that are considered in the calculations. The Bellman recurrent relation (13) that stores the optimal values of the subproblems in the objective space is represented by an array $b[0 : c]$ of profit-risk ordered pairs. The table v that stores the optimal solutions of the subproblems in the decision space is represented as a two-dimensional Boolean array $v[1 : n, 0 : c]$ with n rows and $c + 1$ columns. Note that both b and the columns of v use zero-based indexing. To enhance the efficiency of the procedure described in section V, both arrays b and v are declared outside the Algorithm 1 and are passed to it as arguments.

The first part of Algorithm 1 (lines 1 – 10) calculates the values of the optimal solutions of the subproblems in the decision space following the logic of the recurrence relation (13) and at the same time it constructs the table of the optimal solutions in the decision space directly implementing (15). The second part (lines 11 – 18) uses the calculated table v to construct the Pareto optimal solution x^* .

Theorem 1. Algorithm 1 constructs a Pareto optimal solution x^* which belongs to the class of equivalent Pareto optimal solutions P_1 . The ordered pair of its profit and risk $(f(x^*), g(x^*))$ calculated by the algorithm is stored in $b[c]$.

Proof: The proof of the theorem consists by three parts.

- 1) Algorithm 1 follows the dynamic programming scheme given in [1]. Using Proposition 1 we prove that there exists a feasible solution x^* for which the following

relations hold:

$$\begin{aligned} (f(x^*), g(x^*)) &\sim b[c] \\ \text{and } (f(x), g(x)) &\preceq (f(x^*), g(x^*)), \\ &\text{for all } x \in X. \end{aligned} \quad (16)$$

- 2) Let x^* be such a feasible solution, so that $(f(x), g(x)) \preceq (f(x^*), g(x^*))$ holds for all $x \in X$. We proof that in this case it holds that

$$x^* \in X_f \text{ and } g(x^*) = \min\{g(x) : x \in X_f\}, \quad (17)$$

where X_f is the set of all feasible solutions x for which $f(x) = \max\{f(y) : y \in X\}$.

- 3) We prove that the feasible solution x^* belongs to the set P_1 exactly when (17) holds. ■

Proposition 2. *The computational complexity of Algorithm 1 is $O(cn)$. The memory requirement of the algorithm is also $O(cn)$.*

Proof:

- 1) The relation dominance given in Definition 7 and the operation direct sum given in Definition 10 introduce only a small constant factor in Algorithm 1. Since it follows the dynamic programming scheme [1], its computational complexity is $O(cn)$.
- 2) The memory allocated by the algorithm differs from the dynamic programming scheme [1] only with the double size needed for the array b which in our case stores ordered pairs of numbers. The memory requirements are dominated by the two dimensional array v , therefore the memory complexity of the algorithm is $O(cn)$. ■

We will demonstrate the Algorithm 1 using the following Example 1.

Example 1. Consider $KP_9(182)$ with profit vector p , risk vector r and weight vector w given in (18).

$$\begin{aligned} p &= (32, 12, 37, 25, 32, 28, 39, 27, 32) \\ r &= (30, 26, 25, 25, 20, 15, 8, 7, 6) \\ w &= (29, 11, 37, 24, 29, 24, 36, 26, 30) \end{aligned} \quad (18)$$

We will construct a Pareto optimal solution that maximizes the linear objective function of $KP_9(182)$ using Algorithm 1 and that represents the class P_1 .

The algorithm applies a bottom-up dynamic programming scheme to calculate the two-dimensional Boolean array v with 9 rows and 183 columns with columns indexing starting from 0. The elements $v[i, j]$ of the array store whether the corresponding item i is selected for a subproblem with capacity j . For example, the algorithm calculates that $v[2, 11] = 0$, which means that the item 2 is not selected for a subproblem with capacity 11 and $v[9, 182] = 1$, which means that the item 9 is selected for the problem capacity 182.

The array b contains profit-risk pairs, where $b[182] = (195, 25)$, which means that the profit $l_1 = 195$ and the risk

$t_1 = 25$, where (l_1, t_1) are the profit and risk of the class of equivalent Pareto optimal solutions P_1 .

Finally, the algorithm outputs a Pareto optimal solution x^* that corresponds to P_1 :

$$x^* = (0, 0, 1, 0, 1, 1, 1, 1, 1), \quad (19)$$

together with its corresponding profit and risk $(195, 25)$.

Since the example is relatively small, we can verify the result using an exhaustive search method. The exhaustive search shows that the knapsack can contain at most 7 of the given items, otherwise the capacity constraint is violated. Therefore, there exist at most 501 feasible solutions. By direct verification, we construct a list X of all feasible solutions to the problem $KP_9(182)$. All feasible solutions in the list X are plotted in the objective space in Fig. 1 represented by red circles. It turns out that X has exactly 469 elements. We verify that there are exactly 6 feasible solutions in X that have a maximum profit of 195 and they are listed in (20).

$$\begin{aligned} &(1, 0, 1, 0, 1, 1, 1, 1, 0) \\ &(1, 1, 0, 1, 1, 1, 1, 1, 0) \\ &(1, 0, 1, 0, 0, 1, 1, 1, 1) \\ &(1, 1, 0, 1, 0, 1, 1, 1, 1) \\ &(0, 0, 1, 0, 1, 1, 1, 1, 1) \\ &(0, 1, 0, 1, 1, 1, 1, 1, 1) \end{aligned} \quad (20)$$

Among the discovered solutions with maximum profit (20), exactly one has minimum risk, namely $(0, 0, 1, 0, 1, 1, 1, 1, 1)$ which coincides with the Pareto optimal solution x^* (19) found by Algorithm 1.

V. MINIMUM COMPLETE PARETO FRONT

In this section we solve the main problem considered in this paper which is formally stated in Problem 2.

Problem 2 (Main problem). *Given a biobjective maximum profit minimum risk knapsack problem $KP_n(c)$ defined by (4) with the vectors p , r and w , find a minimum complete Pareto optimal front P^* and a list s composed by ordered pairs (l_j, t_j) , where l_j and t_j , $j = 1, 2, \dots, K$, are the profit and risk values defined by (7).*

Before we construct Algorithm 2 that solves Problem 2, we need to prove the following lemma.

Lemma 1. *We suppose that for some $j < K$ are calculated the profit l_j and the risk t_j of any Pareto optimal solution that belongs to the class of equivalent Pareto optimal solutions P_j . Then, necessary and sufficient condition for a feasible solution x to belong to the class P_{j+1} is given by following equations:*

$$\begin{aligned} f(x) &= \max\{f(y) : y \in X_j\} \\ \text{and } g(x) &= \min\{g(z) : z \in X_{jf}\}, \end{aligned} \quad (21)$$

where X_{jf} is the set of all feasible solutions z , for which $f(z) = \max\{f(y) : y \in X_j\}$.

Proof: We will prove the necessary and sufficient statements.

$\Rightarrow$ We will prove that if $x \in P_{j+1}$, then (21) are fulfilled.

Let $x \in P_{j+1}$. Then $g(x) = t_{j+1} < t_j$ and $x \in X_j$.

For any $y \in X_j$, $g(y) < t_j$ holds. Then y cannot be dominated by an element of the set $P_1 \cup \dots \cup P_j$. Therefore, there exists a feasible solution $z \in P_k$ for some $k \in \{j+1, \dots, K\}$ such that $y \preceq z$. Then $f(z) = l_k \geq f(y)$ and $g(z) = t_k \leq g(y)$. Since $k \geq j+1$, then $f(y) \leq l_k \leq l_{j+1} = f(x)$. Therefore

$$f(x) = \max\{f(y) : y \in X_j\}, \quad (22)$$

i.e. $x \in X_{jf}$. Let $z \in X_{jf}$. Then $f(z) = f(x)$. Therefore, $g(z) \geq g(x)$, because otherwise $x \prec z$, which contradicts the fact that x_0 is a Pareto optimal solution. Then (21) holds.

$\Leftarrow$ We will prove that if $x \in X_{jf}$ and it fulfills (21), then $x \in P_{j+1}$.

Since $P_{j+1} \subseteq X_j$, then $l_{j+1} \leq f(x)$. Moreover, $l_{j+1} \geq f(x)$, because every feasible solution $y \in X_j$ is dominated by an element of $P_{j+1} \cup \dots \cup P_K$. Then $l_{j+1} = f(x)$ and

$$X_{jf} = \{z \in X_j : f(z) = l_{j+1}\}. \quad (23)$$

Then for $y \in P_{j+1}$, from (21) it follows that $g(x) \leq g(y) = t_{j+1}$. Moreover, it is not possible $g(x) < t_{j+1}$, because otherwise we get that the Pareto optimal solution y is dominated by the feasible solution x . Therefore $g(x) = t_{j+1}$ and then $x \in P_{j+1}$, because $f(x) = l_{j+1}$. ■

Algorithm 2 is composed by two stages. The first stage (lines 1 – 10) constructs the lists P^* and s in the nontrivial setup of the biobjective knapsack problem $KP_n(c)$ in which the sum of all weights of the items is bigger than to the capacity of the knapsack. The **while** loop consecutively applies Algorithm 1 that finds a Pareto optimal solution x^* with maximum profit among items that are currently included in the lookup table I , based on the dynamic programming scheme described in section IV. It appends x^* in the list P^* and its corresponding profit-risk pair into the list s . Then, before the next iteration, it keeps in the lookup table I only those items that have risk strictly less than the risk of the latest discovered x^* .

The second stage of the algorithm (lines 11 – 23) covers the trivial setup of $KP_n(c)$ in which the total weight of all items in I is less or equal to the capacity c . In this case the dynamic programming procedure is not needed and we can apply the greedy algorithmic paradigm. All unique values of the risks of items in the lookup table I are sorted in descending order and consecutively are used to construct the solutions that represent the remaining classes of equivalent Pareto optimal solutions.

The following Theorem 2 proves the correctness of Algorithm 2.

Theorem 2. Algorithm 2 constructs a minimum complete Pareto front P^* and the list s of ordered pairs (l_j, t_j) for $j = 1, 2, \dots, K$ defined in (7), where l_j is the profit and t_j is the risk of any Pareto optimal solution $x \in P_j$.

Proof:

- 1) We prove the theorem for the first stage of the algorithm. Let us denote by k the number of the iterations of the

Algorithm 2 Procedure MIN-COMPLETE-PF(p, r, w, c)

Input: Vectors p, r, w and capacity c

Output: List containing a minimum complete Pareto front P^* and list of corresponding prices and risks s

```

1: let  $P^*$  and  $s$  be empty lists
2: let  $I[1 : n]$  be an array initialized with  $[1, 2, \dots, n]$ 
3: let  $b[0 : c]$  be an array of ordered pairs
4: let  $v[1 : n, 0 : c]$  be a Boolean array
5: while  $I \neq \emptyset$  and  $h(I) > c$  do
6:    $(x^*, (l, t)) \leftarrow \text{POS-MAX-PROF}(I, p, r, w, c, b, v)$ 
7:    $P^* \leftarrow P^* \cup \{x^*\}$ 
8:    $s \leftarrow s \cup \{(l, t)\}$ 
9:    $I \leftarrow \{i \in I : r_i < t\}$ 
10: end while
11: if  $I \neq \emptyset$  and  $h(I) \leq c$  then
12:    $r' \leftarrow \{r_j : j \in I \text{ and } r_j \notin r'\}$ 
13:    $r' \leftarrow \text{SORT}(r') \text{ \{descending order\}}$ 
14:   for  $k \leftarrow 1$  to  $|r'|$  do
15:     let  $x[1 : n]$  be an array initialized with 0
16:     for all  $i \in I$  do
17:       if  $r_i \leq r'_k$  then
18:          $x[i] \leftarrow 1$ 
19:       end if
20:     end for
21:      $P^* \leftarrow P^* \cup \{x\}$ 
22:      $s \leftarrow s \cup \{(f(x), g(x))\}$ 
23:   end for
24: end if
25: return  $P^*, s$ 

```

while loop of the first stage of the algorithm. Using Lemma 1 we prove that for the currently calculated P^* and s the following statements hold.

- a) The list P^* contains a Pareto optimal solution for each of the classes P_j for $j = 1, 2, \dots, k$.
- b) The list s contains the profit-risk pairs for the corresponding elements of P^* .

The proof is composed using the method of mathematical induction. For the base case $i = 1$ it follows directly from Theorem 1.

- 2) We prove the theorem for the second stage of the algorithm, in the case in which $I \neq \emptyset$ and $h(I) \leq c$. The poof is based on mathematical induction over the iterations of the **for** loop (lines 14 – 23). ■

Corollary 1. Let $KP_n(c)$ defined by (4) is a biobjective maximum profit minimum risk knapsack problem with the vectors p, r and w . Then the number of classes of equivalent Pareto optimal solutions K defined in (6) is not more than the number of unique risk values in the vector r .

Proposition 3. The computational complexity of Algorithm 2 is $O(cn^2)$. The memory allocation required by the algorithm is $O(cn)$.

TABLE I

AFTER k -TH ITERATION OF THE SECOND STAGE, A PARETO OPTIMAL CLASS P_j IS CONSTRUCTED CONTAINING SINGLE SOLUTION x WITH PROFIT l AND RISK t

k	P_j	r'_k	x	l	t
1	P_2	20	(0, 0, 0, 0, 1, 1, 1, 1, 1)	158	20
2	P_3	15	(0, 0, 0, 0, 0, 1, 1, 1, 1)	126	15
3	P_4	8	(0, 0, 0, 0, 0, 0, 1, 1, 1)	98	8
4	P_5	7	(0, 0, 0, 0, 0, 0, 0, 1, 1)	59	7
5	P_6	6	(0, 0, 0, 0, 0, 0, 0, 0, 1)	32	6

Proof:

- 1) Proposition 2 shows that the complexity of Algorithm 1 is $O(cn)$. This procedure cannot be triggered more than n times by the **while** loop of the first stage of Algorithm 2. Therefore, the complexity of the first stage can be evaluated by $O(cn^2)$. Since the complexity of the second stage can be evaluated by $O(|r'|n)$, in the general case the complexity of the first stage dominates, and therefore the entire running time is $O(cn^2)$.
- 2) The dominating memory allocation of the algorithm is the two-dimensional array v that has n rows and $c + 1$ columns. Since both lists P^* and s store K elements, in analogy to Proposition 2 it follows that the memory complexity of the algorithm is $O(cn)$.

■

Example 2. We will construct the minimum complete Pareto front P^* of the biobjective knapsack problem $KP_9(182)$ with parameter vectors given in (18) in Example 1.

Before the execution of the first stage of the algorithm, the lookup table of item indexes contains all items and its is $I = [1, 2, \dots, 9]$. The dynamic programming scheme executes as described in Example 1 and it finds one Pareto optimal solution x^* given in (19) that represents the class P_1 with its corresponding profit and risk (195, 25).

The risk of the class P_1 $t_1 = 25$ is used to keep in I only those items that have risks strictly less than its value. The resulting lookup table of items becomes $I = [5, 6, 7, 8, 9]$ and since their total weight is 145, the first stage of the algorithm completes.

The second stage of the algorithm starts. The greedy strategy begins by sorting the unique risk values of the items in I in descending order. The resulting vector is $r' = [20, 15, 8, 7, 6]$. Each of the risks in r' are consecutively used to construct the remaining Pareto optimal solutions that represent the classes $P_2, P_3, \dots, P_6$ respectively. The classes, represented by the corresponding Pareto optimal solutions are given in Table I.

Fig. 1 shows the minimum complete Pareto front of the problem $KP_9(182)$ together with all feasible solutions X that have been calculated by an exhaustive search method, as described in Example 1. The plot clearly shows that there is no feasible solution that dominates any of the Pareto optimal solutions in P^* , constructed by Algorithm 2.

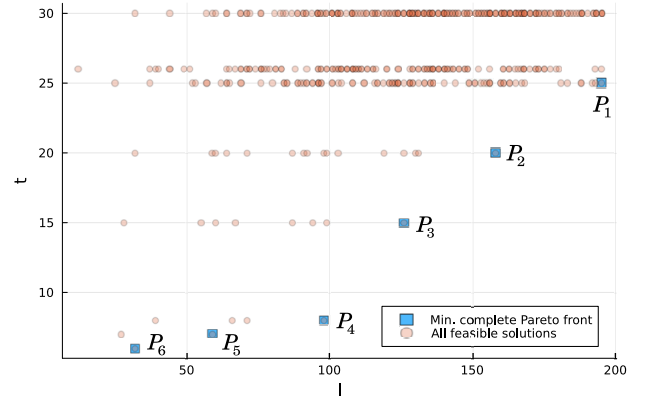


Fig. 1. Minimum complete Pareto front P^* of the knapsack problem $KP_9(182)$ with parameter vectors given in (18). P^* contains $K = 6$ number of classes of equivalent Pareto optimal solutions with corresponding solutions shown as blue squares. The red circles show all feasible solutions of the problem found by an exhaustive search

VI. EXPERIMENTS RESULTS

We have implemented the prototype version of the proposed algorithms in Wolfram language, running in Wolfram Mathematica 14.3 [19]. It was used to verify the correctness of the algorithm, also using examples that are small enough to be solved by an exhaustive search method. The final implementation² of the algorithms is written in Julia programming language [20] and it is used to test the proposed algorithms with large random knapsack problems.

The experiments are conducted on a PC with AMD Ryzen 7 CPU with 32 GiB RAM memory running 64-bit Linux 6.8.0-106. The knapsack problems are generated with random parameter vectors p , r and w . Since even a single-objective knapsack problem is harder to solve when the ratio between the capacity c and the sum of the weights of all items is close to 0.5 (see [7]), we set the capacity to:

$$c = \lfloor h(I)/2 \rfloor. \quad (24)$$

We provide four different set of experiments E_1 , E_2 , E_3 and E_4 (see Table II) that aim to show both the running time efficiency of the implementation of Algorithm 2 and its memory requirements. Since the first stage of the algorithm is based on the dynamic programming scheme used in Algorithm 1, the required memory can be considered a critical resource. Detailed results of four sets of random experiments are given in the appendix as Table III-VI respectively. The growth of the running time in seconds and the memory allocations in bytes are illustrated in Fig. 2a and Fig. 2b respectively.

For the set of random experiments E_1 we fix all the intervals from which p_i , r_i and w_i are randomly selected, and we increment consecutively only the size n of the random knapsack problem $KP_n(c)$, starting from $n = 100$ up to $n = 5000$ with step 100. The execution time of the implementation grows with polynomial complexity (see Fig. 2a)

²Open source implementation of Algorithm 2 in Julia programming language is published in <https://github.com/laskov/Biobjective-KP-Pareto-Front>.

TABLE II

THE SETUP OF THE SETS OF EXPERIMENTS E_1 , E_2 , E_3 AND E_4 . FOR EACH EXPERIMENT THE VALUES PARAMETER VECTORS ARE SELECTED RANDOMLY FROM THE GIVEN INTERVAL. FOR E_1 THE INTERVALS FOR RANDOM VALUES OF p , r AND w ARE FIXED AND FOR EACH CONSECUTIVE EXPERIMENT n GROWS WITH 100. FOR E_2 , E_3 AND E_4 THE SIZE OF THE RANDOM $KP_n(c)$ IS FIXED, ONE OF THE INTERVALS OF THE RANDOM PARAMETER VECTOR GROWS WITH 100, WHILE THE OTHERS ARE FIXED

Experiment	Size	Profit	Risk	Weight
E_1	$n = 100, 200, \dots, 5000$	$p_i \in [1, 100]$	$r_i \in [1, 100]$	$w_i \in [1, 100]$
E_2	$n = 1000$	$p_i \in [1, 100], \dots, p_i \in [1, 5000]$	$r_i \in [1, 100]$	$w_i \in [1, 100]$
E_3	$n = 1000$	$p_i \in [1, 100]$	$r_i \in [1, 100], \dots, r_i \in [1, 5000]$	$w_i \in [1, 100]$
E_4	$n = 1000$	$p_i \in [1, 100]$	$r_i \in [1, 100]$	$w_i \in [1, 100], \dots, w_i \in [1, 5000]$

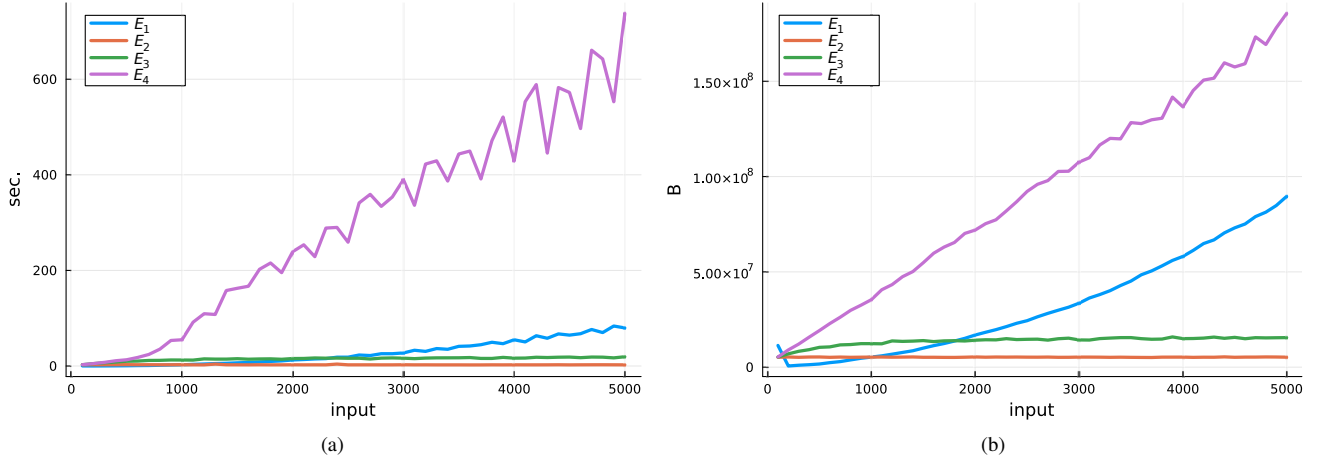


Fig. 2. The running time in seconds (a) and the memory allocations in bytes (b) of the implementation of Algorithm 2 for random biobjective knapsack problems $KP_n(c)$ in the sets of experiments: E_1 when n grows, E_2 when the interval of the profit values grows, E_3 when the interval of the risk values grows and E_4 when the interval of the weight values grows

with relatively small variations due to the random nature of the experiments: for example for $n = 4200$ a local peak is observed that is relatively small compared to the neighboring running times (see Table III). The curve that represents the growth of the allocated memory is more stable, except a peak for the first experiment with $n = 100$, where the used memory is 11404656 bytes, which is a result from the compilation process of Julia.

For the remaining sets of experiments E_2 , E_3 and E_4 , the size of the random $KP_n(c)$ is fixed to $n = 1000$.

Random experiments set E_2 fixes all intervals of the random vectors, except the interval for the random profit which starts from $[1, 100]$ up to $[1, 5000]$, where the size of the interval grows with 100 for each consecutive experiment. Both the running time of the implementation and the memory used in Fig. 2a and 2b remain in the same interval with minor changes, which is observed also in the results in Table IV. The set of experiments E_2 show that the change of the profit parameter has nearly no impact on the performance.

Random experiments set E_3 grows the interval for the random risk starting from $[1, 100]$ up to $[1, 5000]$ with step 100 for each consecutive experiment. The experiments show that the risk parameter has higher impact on the execution time and memory allocation than the profit parameter. Both running time and memory used grow in a way similar to a logarithm

function in the interval between 0 and 2000 and after that the curve stabilizes to nearly parallel to the abscissa of the plots, which also can be seen in the results in Table V.

Analogously, the set of random experiments E_4 grows the interval for the random weight starting from $[1, 100]$ up to $[1, 5000]$ with step 100 for each consecutive experiment. It is obvious from both Fig. 2a and 2b that the weight parameter has the highest impact on the running time and memory allocations of the implementation of Algorithm 2 due to the rapid growth of the capacity and (24). In this case the running time of the algorithm grows polynomially with local minima and maxima due to the random nature of the experiments. However, none of the local peaks do not show an exponential nature. The random knapsack problem $KP_n(c)$ with a random weight interval $[1, 5000]$ has size $n = 1000$ and capacity $c = 1303379$ (see Table VI). In this case the program completes in 737.568382 sec. (or 12 min. and 17 sec.) and allocated 185568856 bytes (or 185.57 MB).

The above experiments show that using Algorithm 2 large instances of the biobjective knapsack problem can be solved in both feasible running time and memory complexity. For example a random $KP_n(c)$ with $n = 10000$, $c = 250813$ and random intervals fixed $[1, 100]$, the proposed algorithm completes in 8 minutes and 52 seconds and requires 380.102 MB memory allocations. The constructed minimum complete

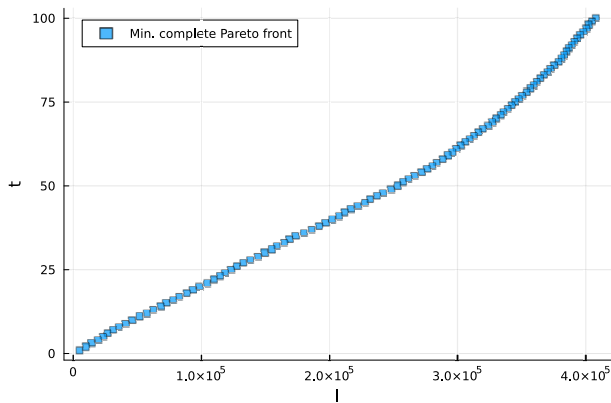


Fig. 3. Minimum complete Pareto front P^* of a random $KP_n(c)$ with $n = 10000$, $c = 250813$ and random intervals fixed $[1, 100]$. P^* contains $K = 100$ number of classes of equivalent Pareto optimal solutions

Pareto front P^* is composed by $K = 100$ number of classes of equivalent Pareto optimal solutions. The plot of P^* in the objective space is given in Fig. 3.

VII. CONCLUSION

In this work we examine in details the biobjective maximum profit minimum risk 0-1 knapsack problem. We propose an exact algorithm for construction of the minimum complete Pareto front of the problem. The algorithm is composed by two stages: the first one is based on dynamic programming scheme, while the second one implements a greedy strategy. We prove mathematically the correctness of the algorithm and we show that in this case the minimum complete Pareto front P^* can be constructed with pseudo-polynomial complexity, namely $O(cn^2)$, where n is the size of the problem, and c is the capacity of the knapsack, which shows that the construction of P^* is a weakly NP-hard problem.

It is a well-known fact in the literature [7] that the hardest instances of knapsack problems are when the ratio between the capacity c and the sum of the weights of all items is close to 0.5. Our experiments show that in this case the memory complexity of the implementation of the proposed algorithm grows with polynomial complexity, which allows large instances to be solved.

As it can be expected, the computational complexity of the proposed algorithm is dominated by its first stage which is based on dynamic programming. We improve its efficiency by adopting the dominance relation and open sum between ordered pairs of profit and capacity, which allows our dynamic scheme to generate Pareto optimal solutions directly, without finding all optimal solutions with respect to the linear objective function.

REFERENCES

- [1] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*, 1st ed. Berlin Heidelberg: Springer-Verlag, 2004. ISBN 978-3-540-24777-7
- [2] K. Miettinen, *Nonlinear multiobjective optimization*, 1st ed., ser. International Series in Operations Research & Management Science. Boston, USA: Springer New York, NY, 1999, vol. 12. ISBN 978-0-7923-8278-2. doi: 10.1007/978-1-4615-5563-6
- [3] V. Cacchiani, M. Iori, A. Locatelli, and S. Martello, “Knapsack problems – an overview of recent advances. part i: Multiple, multidimensional, and quadratic knapsack problems,” *Computers & Operations Research*, vol. 143, p. 105693, 2022. doi: 10.1016/j.cor.2021.105693
- [4] M. J. Rosenblatt and Z. Sinuany-Stern, “Generating the discrete efficient frontier to the capital budgeting problem,” *Operations Research*, vol. 37, no. 3, pp. 384–394, 1989. doi: 10.1287/opre.37.3.384
- [5] M. Eben-Chaime, “Parametric solution for linear bicriteria knapsack models,” *Management Science*, vol. 42, no. 11, pp. 1565–1575, 1996. doi: 10.1287/mnsc.42.11.1565
- [6] U.-E. Lukata and J. Teghem, “Solving multi-objective knapsack problem by a branch-and-bound procedure,” in *Multicriteria Analysis (Proceedings of the 6th International Conference on MCDM)*, J. Clímaco, Ed. Berlin, Heidelberg: Springer, 1997. doi: 10.1007/978-3-642-60667-0_26 pp. 269–278.
- [7] M. Visée, J. Teghem, M. Pirlot, and E. L. Ulungu, “Two-phases method and branch and bound procedures to solve the bi-objective knapsack problem,” *Journal of Global Optimization*, vol. 12, no. 2, pp. 139–155, 1998. doi: 10.1023/A:1008258310679
- [8] I. I. Melamed, I. K. Sigal, and N. Y. Vladimirova, “Study of the linear parametrization of criteria in the bicriteria knapsack problem,” *Computational Mathematics and Mathematical Physics*, vol. 39, no. 5, pp. 711–714, 1999.
- [9] C. Bazgan, H. Hugot, and D. Vanderpooten, “Solving efficiently the 0-1 multi-objective knapsack problem,” *Computers & Operations Research*, vol. 36, no. 1, pp. 260–279, 2009. doi: 10.1016/j.cor.2007.09.009
- [10] J. R. Figueira, L. Paquete, M. Simões, and D. Vanderpooten, “Algorithmic improvements on dynamic programming for the bi-objective 0,1 knapsack problem,” *Computational Optimization and Applications*, vol. 56, pp. 97–111, 03 2013. doi: 10.1007/s10589-013-9551-x
- [11] R. Kumar and P. Singh, “Assessing solution quality of biobjective 0-1 knapsack problem using evolutionary and heuristic algorithms,” *Applied Soft Computing*, vol. 10, no. 3, pp. 711–718, 2010. doi: 10.1016/j.asoc.2009.08.037
- [12] A. Liefooghe, L. Paquete, and J. R. Figueira, “On local search for bi-objective knapsack problems,” *Evolutionary Computation*, vol. 21, no. 1, pp. 179–196, 03 2013. doi: 10.1162/EVCO_a.00074
- [13] L. Paquete, T. Schiavinotto, and T. Stützle, “On local optima in multi-objective combinatorial optimization problems,” *Annals of Operations Research*, vol. 156, pp. 83–97, 03 2007. doi: 10.1007/s10479-007-0230-0
- [14] S. Fidanova, *Ant colony optimization and applications*, 1st ed., ser. Studies in Computational Intelligence. Cham, Switzerland: Springer, 2021, ch. 3, pp. 9–19. ISBN 978-3-030-67380-2
- [15] S. Fidanova and K. Atanassov, “Ant algorithm with local search procedure for multiple knapsack problem,” in *Large-Scale Scientific Computations. LSSC 2023*, ser. Lecture Notes in Computer Science, I. Lirkov and S. Margenov, Eds., vol. 13952. Cham: Springer Nature Switzerland, 2024. doi: 10.1007/978-3-031-56208-2_24. ISBN 978-3-031-56208-2 pp. 246–252.
- [16] D. Granata and A. Raiconi, “Bi-objective knapsack problem with conflicts,” *Annals of Operations Research*, vol. 357, pp. 979–1001, 2026. doi: 10.1007/s10479-025-06792-5
- [17] C. F. Bernardes, P. Castellucci, D. Gonçalves, E. Duzzioni, and A. Mucherino, “Adiabatic quantum computing for the subset sum problem: Preliminary studies,” in *Proceedings of the 20th Conference on Computer Science and Intelligence Systems (FedCSIS)*, ser. Annals of Computer Science and Information Systems, M. Bolanowski, M. Ganzha, L. Maciaszek, M. Paprzycki, and D. Ślęzak, Eds., vol. 43. IEEE, 2025. doi: 10.15439/2025F5496 pp. 635–639.
- [18] R. Bellman, *Dynamic programming*, ser. Rand Corporation research study. New Jersey: Princeton University Press, 1957. ISBN 0-691-07591-X
- [19] W. R. Inc., “Mathematica, Version 14.3,” champaign, IL, 2025. [Online]. Available: <https://www.wolfram.com/mathematica>
- [20] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, “Julia: A fresh approach to numerical computing,” *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017. doi: 10.1137/141000671

TABLE III

THE EXPERIMENT RESULTS FOR THE SET E_1 . FOR EACH EXPERIMENT ARE GIVEN THE SIZE n OF THE RANDOM $KP_n(c)$, THE CAPACITY c , THE NUMBER OF ITERATIONS OF THE FIRST STAGE OF THE ALGORITHM DP , THE NUMBER OF CLASSES K , THE RUNNING TIME IN SECONDS AND THE ALLOCATED MEMORY IN BYTES

n	c	DP	K	sec.	bytes
100	2370	27	60	0.192361	11404656
200	5189	43	84	0.074955	621248
300	7571	46	91	0.186477	1012200
400	10149	45	95	0.325061	1292864
500	12416	48	99	0.514735	1664144
600	15154	50	99	0.862284	2368640
700	17299	47	100	1.051775	2918000
800	20549	52	100	1.601669	3814992
900	23283	54	100	2.105216	4665744
1000	25269	51	100	2.526908	5294712
1100	27325	49	100	3.404243	5938320
1200	29835	51	100	4.228739	6809304
1300	32280	53	100	5.139709	7723992
1400	34803	50	100	5.689914	8617592
1500	38544	49	100	6.659498	9993552
1600	40539	51	100	7.410178	11283656
1700	42528	50	100	8.547013	12394552
1800	44170	53	100	9.637717	13693784
1900	47583	49	100	10.718795	15111920
2000	50512	53	100	12.661393	16864728
2100	53201	52	100	13.577578	18304408
2200	55395	50	100	14.768672	19681520
2300	58227	48	100	15.488323	21274592
2400	60640	50	100	18.472667	23032288
2500	61987	51	100	18.536955	24379104
2600	65523	50	100	23.024740	26418752
2700	68232	49	100	21.930178	28219896
2800	69976	50	100	25.763486	29873208
2900	71810	50	100	25.750629	31463608
3000	75178	48	100	27.333955	33545288
3100	79142	50	100	32.979915	36286200
3200	80220	51	100	30.612038	38088456
3300	82265	50	100	36.519294	40136808
3400	85397	51	100	35.012538	42822160
3500	87572	51	100	41.173826	45156744
3600	91550	51	100	41.879265	48465704
3700	93393	50	100	44.571282	50512568
3800	95658	52	100	49.684219	53151352
3900	98709	50	100	46.740300	55989888
4000	100022	51	100	54.906478	58093504
4100	103285	51	100	50.395597	61262096
4200	106700	52	100	63.375236	64853000
4300	107728	51	100	58.054848	66773336
4400	111353	53	100	67.188842	70446224
4500	113775	50	100	64.630784	73094544
4600	114024	52	100	67.565599	75123832
4700	118634	49	100	76.437761	79003144
4800	119389	51	100	69.963581	81326344
4900	122633	49	100	83.680103	84791936
5000	127007	51	100	79.683188	89594792

TABLE IV

THE EXPERIMENT RESULTS FOR THE SET E_2 . FOR EACH EXPERIMENT ARE GIVEN THE INPUT SIZE OF THE PROFIT INTERVAL OF THE RANDOM $KP_n(c)$, THE CAPACITY c , THE NUMBER OF ITERATIONS OF THE FIRST STAGE OF THE ALGORITHM DP , THE NUMBER OF CLASSES K , THE RUNNING TIME IN SECONDS AND THE ALLOCATED MEMORY IN BYTES

input	c	DP	K	sec.	bytes
100	25537	48	100	2.497104	5202568
200	25292	53	100	2.563865	5338776
300	25189	49	100	2.615696	5170576
400	25655	51	100	2.677847	5323160
500	25326	53	100	2.854040	5343448
600	24746	50	99	2.651212	5163032
700	25480	50	100	2.724162	5302712
800	24837	50	100	2.588680	5170984
900	25293	49	100	2.611297	5238280
1000	24891	53	100	2.779131	5283544
1100	25758	50	100	2.511845	5325520
1200	25467	48	100	2.414479	5234648
1300	24962	52	100	4.070327	5282288
1400	25713	52	100	2.618404	5352352
1500	25059	50	100	2.517577	5185280
1600	24704	51	100	2.439173	5191256
1700	24783	48	100	2.599391	5137048
1800	24426	50	100	2.450656	5115496
1900	24968	52	100	2.480711	5222264
2000	25596	52	100	2.587080	5369904
2100	25191	50	100	2.465151	5228648
2200	25949	51	100	2.496546	5323784
2300	25957	49	100	2.435280	5278416
2400	25482	50	99	4.067429	5309544
2500	24947	52	100	2.474779	5256456
2600	24818	52	100	2.481539	5240200
2700	25682	49	100	2.503077	5300936
2800	24827	51	100	2.438800	5171080
2900	24790	50	100	2.455589	5188624
3000	25361	51	100	2.660996	5283864
3100	25414	50	100	2.464787	5276304
3200	24941	51	100	2.487061	5245504
3300	24455	52	100	2.402717	5165856
3400	25863	48	100	2.390888	5232352
3500	24579	50	100	2.387666	5158800
3600	24604	48	100	2.308801	5111960
3700	24206	48	100	2.322524	5078272
3800	25517	49	100	2.484438	5256224
3900	25860	48	100	2.412311	5266672
4000	25418	51	99	2.503650	5272288
4100	24044	53	100	2.455724	5146992
4200	25429	49	100	2.427618	5223928
4300	24593	52	100	2.472394	5206664
4400	26073	52	100	2.617775	5396064
4500	25023	49	99	2.383496	5186064
4600	25103	53	100	2.526726	5275592
4700	25212	52	100	2.595549	5282080
4800	25193	54	100	2.636268	5363976
4900	25077	53	100	2.552560	5328576
5000	25013	51	100	2.468907	5235864

TABLE V

THE EXPERIMENT RESULTS FOR THE SET E_3 . FOR EACH EXPERIMENT ARE GIVEN THE INPUT SIZE OF THE RISK INTERVAL OF THE RANDOM $KP_n(c)$, THE CAPACITY c , THE NUMBER OF ITERATIONS OF THE FIRST STAGE OF THE ALGORITHM DP , THE NUMBER OF CLASSES K , THE RUNNING TIME IN SECONDS AND THE ALLOCATED MEMORY IN BYTES

input	c	DP	K	sec.	bytes
100	25398	52	100	2.786150	5301152
200	25654	101	197	5.031842	6983304
300	25276	139	287	6.823429	8318472
400	25403	168	350	8.238025	9183360
500	24907	203	421	9.934682	10405688
600	25365	215	459	10.344359	10637496
700	24989	242	507	11.677036	11688760
800	24858	250	521	11.839637	11860280
900	24807	266	560	12.652127	12337760
1000	24656	260	581	12.538190	12337072
1100	24642	260	597	12.371560	12255888
1200	25652	305	625	14.952819	13769904
1300	25151	295	655	14.456647	13525168
1400	25003	295	662	14.428137	13673392
1500	25764	307	666	15.308496	13945344
1600	25216	288	668	14.265012	13414792
1700	25118	307	684	14.784321	13794216
1800	25066	311	686	15.009936	13894328
1900	24749	294	703	14.117298	13732608
2000	25790	320	695	15.657935	14145720
2100	25804	319	717	15.842302	14378800
2200	25508	316	732	16.874359	14322968
2300	25376	335	750	16.380485	14956192
2400	25317	325	728	16.605351	14473136
2500	25703	336	747	16.214019	14617792
2600	25621	325	748	16.292390	14685744
2700	23766	319	752	14.661402	14185384
2800	25399	343	761	16.498662	14915920
2900	25113	358	764	16.940552	15211688
3000	25229	320	752	16.416953	14287552
3100	24952	321	753	15.434881	14224776
3200	25279	337	775	16.595661	14980928
3300	26007	346	777	17.078169	15265032
3400	25057	363	777	17.127836	15503424
3500	25034	363	790	17.205967	15515280
3600	25443	346	777	17.727352	14956032
3700	25902	324	765	15.915364	14640360
3800	24787	339	783	15.877983	14745320
3900	25331	371	801	18.073476	15865856
4000	24681	345	779	16.467634	14976240
4100	25158	343	783	16.653660	15033008
4200	25478	340	797	18.335916	15246768
4300	25738	363	801	17.751761	15773696
4400	26203	336	797	18.431640	15138728
4500	25130	363	804	18.722976	15637280
4600	24538	338	798	17.426693	14920136
4700	25201	356	815	18.759530	15472184
4800	24785	355	800	18.532486	15359752
4900	24963	357	808	17.041350	15438184
5000	25956	356	801	19.361000	15508688

TABLE VI

THE EXPERIMENT RESULTS FOR THE SET E_4 . FOR EACH EXPERIMENT ARE GIVEN THE INPUT SIZE OF THE WEIGHT INTERVAL OF THE RANDOM $KP_n(c)$, THE CAPACITY c , THE NUMBER OF ITERATIONS OF THE FIRST STAGE OF THE ALGORITHM DP , THE NUMBER OF CLASSES K , THE RUNNING TIME IN SECONDS AND THE ALLOCATED MEMORY IN BYTES

input	c	DP	K	sec.	bytes
100	26072	49	100	2.738086	5350920
200	51964	50	100	5.061265	9037816
300	74405	48	100	7.658738	12113328
400	99296	51	100	10.999880	15726400
500	124928	47	100	13.094396	19220776
600	150659	49	100	17.937637	22881528
700	173259	53	100	24.156577	26207576
800	199794	49	100	35.242871	29810808
900	218575	51	100	53.456656	32521752
1000	239405	49	100	54.943613	35410400
1100	276205	49	100	91.633238	40610200
1200	295203	49	100	109.425386	43320104
1300	324734	50	100	107.839264	47463776
1400	343109	53	100	157.949509	50161744
1500	376922	49	100	162.632602	54805080
1600	411693	51	100	166.892711	59773560
1700	434902	50	100	202.345560	62997024
1800	451882	50	100	215.557550	65412360
1900	485290	52	100	195.286042	70166400
2000	498153	50	100	239.238663	71930568
2100	521734	50	100	253.587633	75277680
2200	536301	51	100	228.716402	77322640
2300	569168	50	100	288.623522	81971888
2400	604098	48	100	289.971004	86821008
2500	641868	49	100	259.025881	92185152
2600	668576	50	100	341.268801	95926584
2700	681821	52	100	359.158963	97883520
2800	715439	54	100	333.958604	102684864
2900	717371	49	100	353.628299	102790192
3000	751106	50	100	389.744548	107586400
3100	766811	53	100	336.190976	109896464
3200	814778	49	100	422.540977	116564224
3300	839208	50	100	429.297346	120037296
3400	836905	52	100	386.947595	119764264
3500	898336	48	100	443.548021	128308688
3600	894877	49	100	449.686748	127795912
3700	908138	52	100	391.330431	129789440
3800	914510	49	100	471.682794	130601688
3900	992897	51	100	520.979711	141687976
4000	956969	50	100	428.886785	136627080
4100	1016895	52	100	553.004495	145149416
4200	1055508	54	100	588.860083	150618008
4300	1063256	49	100	445.450118	151599424
4400	1120860	48	100	582.607360	159674344
4500	1104969	49	100	572.351532	157500392
4600	1117635	49	100	496.746135	159208328
4700	1216461	53	100	660.719896	173275176
4800	1188451	51	100	642.319103	169281872
4900	1250260	51	100	552.844951	178034720
5000	1303379	54	100	737.568382	185568856