Proposal of OptDG Algorithm for Solving the Knapsack Problem

Matej Arlović, Tomislav Rudec, Josip Miletić, Josip Balen
Faculty of Electrical Engineering-Computer Science and Information Technology
J. J. Strossmayer University of Osijek, Osijek, Croatia

***Abstract*—In a computational complexity theory, P, NP, NP-complete and NP-hard problems are divided into complexity classes which are used to emphasize how challenging it is to solve particular types of problems. The Knapsack problem is a well-known computational complexity theory and fundamental NP-hard optimization problem that has applications in a variety of disciplines. Being one of the most well-known NP-hard problems, it has been studied extensively in science and practice from theoretical and practical perspectives. One of the solution to the Knapsack problem is the Dantzig's greedy algorithm which can be expressed as a linear programming algorithm which seeks to discover the optimal solution to the knapsack problem. In this paper, an optimized Dantzig greedy (OptDG) algorithm that addresses frequent edge cases, is suggested. Furthermore, OptDG algorithm is compared with the Dantzig's greedy and optimal dynamically programmed algorithms for solving the Knapsack problem and performance evaluation is conducted.**

Keywords*—Dynamic programming; Dantzig algorithm; greedy algorithm; knapsack problem; linear programming; NP-Problem; optimization; OptDG

I. Introduction

In computational complexity theory P, NP, NP-complete and NP-hard terminology is used to denote how difficult it is to find solutions to a specific type of problem. To begin, it has to be determined whether the problems are P, NP, NP-complete, or NP-hard. A problem is P if it can be resolved in polynomial time [1]. If the suggested solution can be checked in polynomial time, the problem is NP [2]. A problem is NP-complete if it is in NP, and all other NP problems can be reduced to it in polynomial time [1]. This implies that if a solution is discovered for an NP-complete problem, it can be applied to all other NP problems. A problem is NP-hard if it is at least as difficult as the most difficult NP problems. This implies that if a solution is discovered for an NP-hard problem, it can be applied to all NP-complete problems [3]. The Knapsack problem is a well-known computational complexity theory and fundamental NP-hard optimization problem that has applications in a variety of disciplines, including operations research, cryptography, and combinatorial optimization. If a collection of items is given, each of which has a weight and a profit, and a knapsack that can only carry a certain quantity of weight, the Knapsack problem is present. The objective is to determine which subset of items can enhance the total profit of the knapsack without exceeding its carrying capacity. The 0-1 Knapsack is a variant of the Knapsack problem in which a knapsack with a specific capacity c and n items with weights $t_1, t_2, ..., t_n$ and profits $p_1, p_2, ..., p_n$ is given. The main goal is to find the item combination that maximizes the total profit of the knapsack while adhering to the weight limit, with the additional constraint that each item can only be turned on or off once, i.e., each item can only be used once. The theory can be formulated and Integer Linear Program (ILP) shown below:

$$maximize \sum_{i=1}^{n} p_i x_i \tag{1a}$$

$$subject\,to \sum_{i=1}^{n} t_i x_i \leq c \tag{1b}$$

$$x_i \in \{0,1\}, \forall i \in \{1,2,3,...,n\} \tag{1c}$$

The n binary variables $x_1, x_2, ..., x_n$ are decision variables that determine whether or not an item is placed in the knapsack. As part of the input, the problem parameters $c, n, t_1, t_2, ..., t_n$ and $p_1, p_2, ..., p_n$ are given. The 0-1 knapsack problem is NP-hard, indicating that exact solutions to problems with enormous inputs may be intractable. To tackle the NP-hard knapsack problem, several algorithms, including dynamic programming, branch and bound, and genetic algorithms, have been proposed. However, these algorithms only perform well with smaller quantities, i.e., in particular types of problems. As the numbers increase, it becomes nearly impossible to solve the problem.

The 0-1 Knapsack problem is a major optimization problem strongly related to a number of other major optimization problems. For example, by solving an equivalent 0-1 Knapsack problem instance, instances of the binary integer programming, and the bounded and unbounded knapsack problems can be solved. Furthermore, the 0-1 Knapsack problem arrises as a column generation subproblem of the cutting stock problem and is a particular instance of a variety of problems such as the knapsack problem with conflicts [4], [5], the traveling thief problem [6] and the multidimensional knapsack problem [7].

II. Related Work

The Knapsack problem is among the most actively researched topics in combinatorial optimization. In recent years, a vast number of the Knapsack problem variants have been addressed. The 0-1 Knapsack is the most well-known Knapsack problem, and it has been intensively studied for decades. These studies have yielded an abundance of theoretical, practical, and algorithmic findings that have, to some extent, saturated this particular field [8]. According to research conducted by study [9], the Knapsack problem is listed as one of the most popular algorithmic problems, as well as the and the second most popular problem in the NP-hard category. In this section, a brief summary of the literature regarding the 0-1 Knapsack problem is presented.

The most effective algorithms for solving the 0-1 Knapsack problem employ branch-and-bound, dynamic programming, or hybrid approaches that combine the both. Over time, a sequence of advancements resulted in the creation of MT1 [10], MT2 [11], Expknap [12], Minknap [13], and Combo [14] algorithms. The Combo algorithm is the most significant algorithm among several others. Despite being published over twenty years ago, it remains the best-known and most effective algorithm. It can solve the majority of problem instances within a few seconds. Although it shares similarities with the Minknap algorithm, Combo introduces new techniques when faced with a large number of dynamic programming states. The Combo algorithm uses a number of interesting methods, one of which is adding valid inequalities (cardinality constraints) to the formulation of integer programming, which are then relaxed to make more accurate dual bounds [15]. Previous research [12], [14], [16], [17] consistently demonstrates that Combo is typically the fastest algorithm among the five algorithms studied. In this paper, an empirical hardness model, using Combo's running time as an indicator of problem instance difficulty, is adopted. Given Combo's ability to efficiently solve most large problem instances within seconds and the (weak) NP-hardness of the Knapsack problem, researchers have expressed interest in identifying more challenging instances. While it is generally believed that such hard instances exist, the process of discovering them and understanding the key features contributing to their difficulty remains unclear. Research on instances of the 0-1 Knapsack problem often focuses on instances with considerably large coefficients, such as those exhibiting exponential growth relative to the number of items (n). Evaluating the practical difficulty of these instances does not involve using existing algorithm implementations like Combo, as most implementations support only 32-bit or 64-bit integers. Instead, these instances are examined against hypothetical sets of algorithms, considering different assumptions about the strength of the bounds employed by these algorithms. Noteworthy papers in this category include [18], [19], and [20], which describe challenging problem instances with large coefficients designed for various hypothetical algorithms. [21] and [22] introduced new branching strategies for Branch-and-Bound (B&B) methods. The analysis of how item profits or weights respond to perturbations was explored by [23], [24] and [25]. [26] focused on a specific sensitivity analysis called tolerance analysis, which can be performed in amortized time $O(c\, log\, n)$ for each item. Recent advancements in enhancing existing Fully Polynomial Time Approximation Schemes (FPTAS) were made by [27] and [28]. A sensitivity study of greedy heuristics was provided by [29] for the 0-1 knapsack problem and the subset sum problem. [30] addresses the discounted 0-1 knapsack problem (DKP), an extension of the classical knapsack problem where items are grouped into threes, and only one can be picked from each group. They suggest preprocessing and reducing the problem size before using dynamic programming using exact and heuristic fixation methods. These strategies greatly reduce DKP instance solution time, and the authors provide a new collection of more difficult instances for further evaluation.

III. Methods to Solve 0-1 Knapsack Problem

Dynamic programming is a potent methodology and optimization technique in mathematics and computer programming that allows problems to be broken down into smaller subproblems and their solutions to be saved to prevent redundant computations [31]. Dynamic programming yields accurate results for locating optimal solutions, but its computational complexity frequently renders it impractical, necessitating the use of heuristics as an alternative method. In experiments conducted in this research, adapted [32] and Dantzig's greedy algorithm, are used.

A. Dantzig's Greedy Algorithm

The Dantzig's greedy algorithm can be used to solve the 0-1 Knapsack problem. The simplex method is a method for resolving linear programming issues, and the Dantzig's greedy algorithm is a particular kind of linear programming algorithm based on it. The algorithm, which can be expressed as a linear program, seeks to discover the optimal solution to the knapsack problem. The fundamental idea behind Dantzig's method is to build a table with columns that indicate the weight capacity of the knapsack and rows that represent items. The profit of each cell in the table represents the maximum profit that can be obtained for a specific weight capacity using the items utilized up until that point. The table is filled from bottom to top. The initial step of the technique is to add the profit of the first item to the table's first row. The algorithm chooses the maximum profit for each cell based on a comparison between the weight capacity profits of the knapsack with the current item and the knapsack without it for each subsequent row. The maximum possible profit for the specified weight restriction and item count is the last number in the last row of the table. Using a straightforward dynamic programming technique, Dantzig's greedy algorithm can be implemented with a time complexity of $O(nC)$, where n is the number of items, and C is the maximum weight the knapsack can carry. Due to the need to store a table with n rows and C columns, the technique has an $O(nC)$ time complexity. Due to its assumption that each item can only be used once, Dantzig's approach can only be used to solve the 0-1 Knapsack problem. As a result, the unbounded knapsack problem cannot be resolved using it [33]. Algorithm 1 displays Dantzig's greedy algorithm.

B. OptDG Algorithm

According to Danzig's heuristic approach, it has been assumed that items from largest to smallest are placed in a knapsack, based on the $\frac{p_i}{t_i}$ (profit-over-weight) ratio. The main challenge is to increase the total profit of placing items into the knapsack in this manner by replacing a specific item or more items, if possible. The following theorem is demonstrated: if an item with profit p_i and weight t_i is removed and then one or more items are inserted to the right of the removed item in the sorted sequence, a higher weight than the deleted item should be chosen in order to to attain a higher profit. In other words, if an item is removed and replaced with another item to its right while maintaining or decreasing the total weight in the knapsack, the overall worth will either remain the same or it will drop. This theorem will be useful in two situations: first while looking for a better profit, it will be obvious that either one of the previously skipped items must be included

Algorithm 1 Dantzig's greedy algorithm for solving 0-1 Knapsack problem

```
Require: A set of items $items$, and a knapsack capacity $capacity$
Ensure: A knapsack that is packed to its full carrying capacity with items that have the greatest possible profit

  function SolveKnapsackGreedy(items, capacity)
      Sort $items$ by item $p/t$ in decreasing order
      $knapsack\_items \leftarrow []$
      $total\_p \leftarrow 0$
      $total\_w \leftarrow 0$

      for each $item$ in $items$ do
          if $total\_w + item\_weight \leq capacity$ then
              $total\_p \leftarrow total_p + item\_val$
              $total\_w \leftarrow total_w + item\_weight$
              Add $item$ to $knapsack\_items$
          end if
      end for
  return $knapsack\_items$, $total\_p$, $total\_w$
  end function
```

(i.e., the item to the left of the deleted item in the sorted list) or items with a weight greater than the weight of the removed item. Furthermore, when developing algorithms to find the optimal profit or searching for a heuristic that provides a better profit than a pre-determined one when the knapsack is full, replacements involving items with lower profit-to-weight ratios and of equal or lower total weight should not be considered. In other words, there is no need to investigate these item replacement possibilities.

Theorem 1: If $\alpha, \beta, \gamma, and \delta$ are all greater than zero, and if $\frac{\alpha}{\beta} \geq \frac{\gamma}{\delta}$, then the following holds: $\frac{\alpha}{\beta} \geq \frac{\alpha+\gamma}{\beta+\delta} \geq \frac{\gamma}{\delta}$.

Proof:

Given equation $\frac{\alpha}{\beta} \geq \frac{\gamma}{\delta}$ gives:

$$\frac{\alpha}{\beta} = \frac{\frac{\alpha}{\beta}\cdot(1+\frac{\delta}{\beta})}{1+\frac{\delta}{\beta}} = \frac{\frac{\alpha}{\beta}+\frac{\alpha\delta}{\beta^2}}{1+\frac{\delta}{\beta}} = \frac{\left(\frac{\alpha}{\delta}+\frac{\alpha}{\beta}\right)\cdot\frac{\delta}{\beta}}{\left(\frac{\beta}{\delta}+1\right)\cdot\frac{\delta}{\beta}} =$$
$$\frac{\frac{\alpha}{\delta}+\frac{\alpha}{\beta}}{\frac{\beta}{\delta}+1} \geq \frac{\frac{\alpha}{\delta}+\frac{\gamma}{\delta}}{\frac{\beta}{\delta}+1} = \frac{\frac{\alpha+\gamma}{\delta}}{\frac{\beta+\delta}{\delta}} = \frac{\alpha+\gamma}{\beta+\delta},$$

$$\frac{\alpha+\gamma}{\beta+\delta} = \frac{\frac{\alpha+\gamma}{\beta}}{\frac{\beta+\delta}{\beta}} = \frac{\frac{\alpha}{\beta}+\frac{\gamma}{\beta}}{1+\frac{\delta}{\beta}} \geq \frac{\frac{\gamma}{\delta}+\frac{\gamma}{\beta}}{1+\frac{\delta}{\beta}} = \frac{\left(\frac{\gamma}{\delta}+\frac{\gamma}{\beta}\right)\cdot\frac{\beta}{\delta}}{\left(1+\frac{\delta}{\beta}\right)\cdot\frac{\beta}{\delta}} =$$
$$\frac{\frac{\gamma\beta}{\delta^2}+\frac{\gamma}{\delta}}{\frac{\beta}{\delta}+1} = \frac{\frac{\gamma}{\delta}\cdot\left(\frac{\beta}{\delta}+1\right)}{\frac{\beta}{\delta}+1} = \frac{\gamma}{\delta}.$$

■

Theorem 2: If in the Danzig sequence, the term $\frac{p}{t}$ is to the left of the terms $\frac{p_1}{t_1}, \frac{p_2}{t_2}, ..., \frac{p_n}{t_n}$, that is, $\frac{p}{t} \geq \frac{p_i}{t_i}$ for every $i = 1, 2, ..., n$, and if $t = t_1+t_2+...+t_n$ (the weight of the item inserted in the knapsack is equal to the weight of those that will be inserted instead), then it follows that $v \geq p_1 + p_2 + ... + p_n$. That is, the profit of the item initially placed in the knapsack and then removed is greater than the total profit of the items that will be placed in its place.

Proof: If instead of an item with profit p and weight t, we insert an item to its right in the sequence of items sorted in descending order by the profit-to-weight ratio, that is, if we insert an item with ratio $\frac{p_1}{t_1}$ where $\frac{p}{t} \geq \frac{p_1}{t_1}$, then it is evident that if the denominators are equal, $p \geq p_1$. That is, $\frac{p}{t} \geq \frac{p_1}{t_1}$, and $t = t_1$ implies $p \geq p_1$. Hence $\frac{p}{t} \geq \frac{p_1}{t_1}$ and $t = t_1$ implicates $p \geq p_1$. ■

Assume that instead of an item with profit p and weight t, n items are inserted to its right in the sequence of items sorted in descending order by the profit-to-weight ratio, i.e., items with ratios $\frac{p_1}{t_1}, \frac{p_2}{t_2}, \cdots, \frac{p_n}{t_n}$, and $t = t_1 + t_2 + \cdots + t_n$ are inserted, which implies:

$$\frac{p}{t} \geq \frac{p_1}{t_1}, \frac{p}{t} \geq \frac{p_2}{t_2}, \cdots, \frac{p}{t} \geq \frac{p_n}{t_n}$$

Without loss of generality, it can be assumed that the ratios $\frac{p_i}{t_i}$ are sorted in descending order. Due to lemma 1 (all weights and profits are positive), it holds that:

$$\frac{p_1}{t_1} \geq \frac{(p_1+p_2)}{(t_1+t_2)} \geq \frac{p_2}{t_2} \geq \frac{p_3}{t_3}$$
$$\frac{p_1}{t_1} \geq \frac{p_1+p_2+p_3}{t_1+t_2+t_3} \geq \frac{p_3}{t_3} \geq \frac{p_4}{t_4}$$
$$\cdots$$
$$\frac{p}{t} \geq \frac{p_1}{t_1} \geq \frac{p_1+p_2+\cdots+p_n}{t_1+t_2+\cdots+t_n} \geq \frac{p_n}{t_n}$$

that is,

$$\frac{p}{t} \geq \frac{p_1+p_2+\cdots+p_n}{t_1+t_2+\cdots+t_n}\, and\, t = t_1+t_2+\cdots+t_n$$

which implies that Eq. 4, is true, as claimed.

$$p \geq p_1 + p_2 + \cdots + p_n \tag{4}$$

In particular, it follows that when replacing an item of weight t and profit p with a set of items that have a total weight smaller than the removed item ($t > t_1 + t_2 + \cdots + t_n$), and satisfy the condition that the ratio $\frac{p}{t} \geq \frac{p_i}{t_i}$ for every $i = 1, 2, ..., n$, then it also holds that $p \geq p_1 + p_2 + \cdots + p_n$. By employing the Dantzig's greedy algorithm to fill the knapsack initially and subsequently attempting to improve the situation by replacing one item with others, an item that is heavier than the one being removed or an item that was previously overlooked or skipped during the initial filling, has to be selected. Specifically, if the knapsack has been filled to its maximum capacity C using the Danzig method, it ensures the optimal selection of items. To enhance the results further, OptDG algorithm is proposed. The algorithm removes the heaviest item from the knapsack and adds other items until equation 5 is met.

$$\sum_{n}^{i=1} t_i \leq C \tag{5}$$

Algorithm 2 OptDG algorithm for solving the 0-1 Knapsack problem

```
Require: A set of items $items$, total knapsack capacity
    $capacity$, current $total\_profit$, and current $total\_weight$
Ensure: A knapsack that is packed to its full carrying
    capacity with items that have the greatest possible profit

  function SolveKnapsackOptimized(items, C,
  knapsack_items, total_p, total_w)
      Sort $items$ by item's weight in decreasing order
      $unique\_items \leftarrow$ Items not contained in the knapsack
      $heavy\_item \leftarrow$ Heaviest item in a knapsack

      $total\_w \leftarrow total\_w - heavy\_weight$
      $total\_p \leftarrow total\_p - heavy\_profit$
      Remove heaviest item in a knapsack

      for $item$ in $unique\_items$ do
          if $total\_w + item\_weight \leq C$ then
              $total\_p \leftarrow total\_p + item\_val$
              $total\_w \leftarrow total\_w + item\_weight$
              Add $item$ to $knapsack\_items$
          end if
      end for
      return $knapsack\_items$, $total\_p$, $total\_w$
  end function

  function SOLVE_KNAPSACK(items, capacity)
      $knapsack\_items$, $total\_p$, $total\_w$, $new\_items \leftarrow$
  SolveKnapsackGreedy($items$, $C$)
      if size of $knapsack\_items$ == 0 then
          return $[], 0, 0$
      end if
      $opti\_knapsack$, $opti\_p$, $opti\_w \leftarrow$
  SolveKnapsackOptimized($new\_items$, $C$,
  $knapsack\_items$, $total\_p$, $total\_w$)
      if $opti\_p > total\_p$ then
          return $opti\_knapsack$, $opti\_p$, $opti\_w$
      else
          return $knapsack\_items$, $total\_p$, $total\_w$
      end if
  end function
```

The resulting knapsack must meet an Eq. 6, and have a higher profit than the previous knapsack. If the improvement in knapsack profit is not achieved, the algorithm reverts to the Dantzig's solution. The optimization aims to maximize the knapsack's profit and approach the outcome of the optimal algorithm. OptDG algorithm, depicted in Algorithm 2, has a time complexity of $O(nC)$, where n represents the number of items and C represents the maximum weight capacity of the knapsack.

$$C - \sum_{i=1}^{n} t_i \geq 0 \tag{6}$$

IV. Experimental Results

This section provides an evaluation of the OptDG algorithm's performance in relation to Datzing's greedy algorithm (implemented using dynamic programming). Table I to Table VII present results of an experiment that benchmarked both Datzing's greedy algorithm and OptDG algorithm with different item generation strategies.

V. Discussion

This section discusses research outcomes on the optimal (using dynamic programming) algorithm, Datzing's greedy algorithm, and newly proposed OptDG algorithm.

The maximum number of items that can be placed in the knapsack remains at 20, but the total capacity of the knapsack, as well as the profits and weights of the items, vary depending on the case. In order to observe the speed of execution of a specific test case, the number of iterations was set to 100,000. This means that we generated new items 100,000 times and attempted to fill the knapsack. The number of cases and the duration of the algorithm are mean averages. In the first test case, we generated random items with weights ranging from 1 to the knapsack's carrying capacity C, and their profits ranged from 1 to 100. The outcomes for the first test case are displayed in Table I. In 4% of cases, the OptDG algorithm outperformed Dantzig's greedy algorithm, but it took longer to execute in those cases than Dantzig did in other situations. Table II shows the results of the methods that were tested, with weights equal to the knapsack's total capacity and random profits between 1 and 100. Because the heaviest item in the knapsack could not be found when all things were of the same weight, the optimized procedure in this observed scenario was never better than conventional Dantzing. Table III displays how items are organized in a knapsack, with weights equal to 5 and profits chosen at random from 1 to 100. Even in this test case, Dantzing's greedy algorithm was still superior because, like in the previous observation, it is unable to identify the heaviest item in the knapsack when all of the profits are equal. The improved approach takes twice as long as the standard Dantzig's greedy algorithm due to the time spent looking for the heaviest item, which cannot be discovered. The configuration of items in the knapsack is shown in Table IV, where the profits are equal to 5, and the weights of the items are distributed at random from 1 to the knapsack's maximum capacity. Because all items have fixed profits and the OptDG algorithm also sought the items with the highest profit, it was unable to determine the best arrangement of the items in the knapsack and did not outperform Dantzing's greedy algorithm in this case. Table V shows the combination of items when their squared profits and randomly generated weights between 1 and T are used. The OptDG algorithm exhibited the highest efficiency in the observed case in point, but since it must place items in the knapsack twice, it is frequently slower than the standard Dantzig's greedy algorithm. Table VI displays how items are arranged when the weights are created at random between the ranges of 1 and T and the profits are powers of 10. In this instance, the Dantzig's greedy algorithm and the optimal algorithm were nearly identical, and the OptDG algorithm also added to the Dantzig's greedy algorithm's strengths. As a result, the accuracy of the algorithm has increased to 100%. If the OptDG algorithm cannot find a better result, it will

TABLE I. CASE 1: THE ITEMS IN THE KNAPSACK ARE RANDOMLY GENERATED, WHERE BOTH THE PROFITS AND THE WEIGHTS ARE RANDOMLY GENERATED IN THE RANGE OF 1 TO 100

Capacity [C]	Items	Iters.	Dantzing closer to optimal	OptDG better than Dantzing	Optimal timings [ns]	Danzing timings [ns]	OptDG timings [ns]
50	20	100000	60.264%	4.509%	231050	4700	4830
100	20	100000	60.127%	4.342%	478580	3780	5830
150	20	100000	60.261%	4.380%	686380	4880	4560
200	20	100000	60.197%	4.316%	897600	4230	6540
250	20	100000	60.182%	4.293%	1108070	2480	5950
300	20	100000	60.183%	4.400%	1343440	3270	5470
350	20	100000	60.216%	4.344%	1566680	3400	7210
400	20	100000	60.246%	4.318%	1818600	4580	5620

TABLE II. CASE 2: THE ITEMS IN THE KNAPSACK ARE RANDOMLY GENERATED WHERE THE PROFITS ARE RANDOMLY GENERATED IN THE RANGE FROM 1 TO 100, AND THE WEIGHTS ARE EQUAL TO THE TOTAL CAPACITY OF THE KNAPSACK C

Capacity [C]	Items	Iters.	Dantzing closer to optimal	OptDG better than Dantzing	Optimal timings [ns]	Danzing timings [ns]	OptDG timings [ns]
50	20	100000	100%	0%	171210	2820	4860
100	20	100000	100%	0%	342810	2170	4180
150	20	100000	100%	0%	527060	3440	4490
200	20	100000	100%	0%	688170	4390	4850
250	20	100000	100%	0%	833560	2970	4010
300	20	100000	100%	0%	999790	3270	5760
350	20	100000	100%	0%	1183110	3920	3270
400	20	100000	100%	0%	1323160	3580	4950

TABLE III. CASE 3: THE ITEMS IN THE KNAPSACK ARE RANDOMLY GENERATED WHERE THE PROFITS ARE RANDOMLY GENERATED IN THE RANGE OF 1 TO 100, AND THE WEIGHTS ARE EQUAL TO THE NUMBER 5

Capacity [C]	Items	Iters.	Dantzing closer to optimal	OptDG better than Dantzing	Optimal timings [ns]	Danzing timings [ns]	OptDG timings [ns]
50	20	100000	100%	0%	254390	3590	6240
100	20	100000	100%	0%	503960	4210	7220
150	20	100000	100%	0%	718510	4090	6660
200	20	100000	100%	0%	950920	3760	7660
250	20	100000	100%	0%	1191520	4040	5180
300	20	100000	100%	0%	1444620	5890	5300
350	20	100000	100%	0%	1706450	5930	8460
400	20	100000	100%	0%	2001300	5530	6250

TABLE IV. CASE 4: THE ITEMS IN THE KNAPSACK ARE RANDOMLY GENERATED WHERE THE PROFITS ARE EQUAL TO THE NUMBER 5, AND THE WEIGHTS ARE RANDOMLY GENERATED RANGING FROM 1 TO 100

Capacity [C]	Items	Iters.	Dantzing closer to optimal	OptDG better than Dantzing	Optimal timings [ns]	Danzing timings [ns]	OptDG timings [ns]
50	20	100000	100%	0%	195300	2190	5010
100	20	100000	100%	0%	430430	3140	4540
150	20	100000	100%	0%	626020	3570	4490
200	20	100000	100%	0%	849670	4090	5870
250	20	100000	100%	0%	1060400	3730	5580
300	20	100000	100%	0%	1283480	5000	5750
350	20	100000	100%	0%	1529420	6100	5010
400	20	100000	100%	0%	1786890	4240	3960

TABLE V. Case 5: The Items in the Knapsack are Randomly Generated Where the Profits are Squared, and the Weights are Randomly Generated Ranging from 1 to 100

Capacity [C]	Items	Iters.	Dantzing closer to optimal	OptDG better than Dantzing	Optimal timings [ns]	Danzing timings [ns]	OptDG timings [ns]
50	20	100000	64.006%	5.439%	226020	4740	4530
100	20	100000	64.037%	5.148%	452930	4830	5590
150	20	100000	64.065%	5.077%	680830	5470	3720
200	20	100000	63.641%	5.249%	897590	4460	5100
250	20	100000	63.872%	5.182%	1118750	4090	5170
300	20	100000	63.959%	5.097%	1347190	4030	6530
350	20	100000	64.040%	4.990%	1597040	3780	5000
400	20	100000	63.969%	5.115%	1864040	4820	7990

TABLE VI. Case 6: Items in the Knapsack are Randomly Generated Where the Profits are Powers of 10, and the Weights are Randomly Generated Ranging from 1 to C

Capacity [C]	Items	Iters.	Dantzing closer to optimal	OptDG better than Dantzing	Optimal timings [ns]	Danzing timings [ns]	OptDG timings [ns]
50	20	100000	99.869%	0.027%	234560	6570	6420
100	20	100000	99.847%	0.029%	473490	5280	6400
150	20	100000	99.834%	0.031%	692280	4900	6760
200	20	100000	99.843%	0.042%	918950	4990	7000
250	20	100000	99.840%	0.021%	1147130	5170	8740
300	20	100000	99.844%	0.025%	1407410	4990	7480
350	20	100000	99.841%	0.028%	1666750	5780	7180
400	20	100000	99.809%	0.041%	1902380	7320	7960

TABLE VII. Case 7: The Items in the Knaspack are Randomly Generated Where the Profits are Randomly Generated Ranging from 1 to 100, and the Weights are Powers of 10

Capacity [C]	Items	Iters.	Dantzing closer to optimal	OptDG better than Dantzing	Optimal timings [ns]	Danzing timings [ns]	OptDG timings [ns]
50	20	100000	100.000%	0.000%	187830	6880	4840
100	20	100000	98.609%	1.351%	375880	7030	5960
150	20	100000	100.000%	0.000%	556060	6560	7840
200	20	100000	99.989%	0.002%	723580	8410	5520
250	20	100000	100.000%	0.000%	915100	7980	9490
300	20	100000	100.000%	0.000%	1118070	7950	5790
350	20	100000	100.000%	0.000%	1279060	8640	8800
400	20	100000	100.000%	0.000%	1492240	7160	6690

return the Dantzing result. Table VII displays the results when the weights are powers of 10, and the profits are randomly generated in the range of 1 to 100.

VI. Conclusion

In this paper, the binary knapsack (KP01) problem is addressed, which is one of the most complex problems in the theory of computational complexity. Although dynamic programming yields an exact answer, it is computationally expensive for large numbers. Heuristic algorithms are used to quickly approximate results in order to speed up the process at the expense of solution precision. In this research, we looked at Dantzig's greedy algorithm, which efficiently places items in the knapsack but falls short of the dynamic programming approach's outcome in some circumstances.

In order to address this issue, OptDG algorithm is proposed. The proposed algorithm, after arranging the knapsack using the Dantzig's greedy algorithm, removes the heaviest item and attempts to add other items that are not currently in the knapsack. If the total profit of the knapsack increases, the function returns the new solution, otherwise it returns the original solution. Theoretical foundations of the proposed algorithm is described and mathematically presented. Furthermore, a benchmark for performance evaluation of the speed and accuracy metrics of the proposed OptDG algorithm, optimal dynamically programmed algorithm and Dantzig's greedy algorithm in various scenarios is conducted. In the majority of instances, the proposed OptDG algorithm outperformed the conventional Dantzig's greedy algorithm. According to the performance evaluation results, due to the additional knapsack filling, the proposed OptDG algorithm requires an additional minor time for execution which is not considered as a drawback in majority of possible applications.

The aim of our forthcoming research is to integrate further benchmark studies to enhance our comprehension of how to optimize the OptDG algorithm in regard to Dantzing's greedy model.

Acknowledgment

This work is supported by Croatian Science Foundation under project number DOK-2021-02-7509.

References

[1] W. Tian, G. Li, X. Wang, Q. Xiong, and Y. Jiang, "Transforming NP to P: An Approach to Solve NP Complete Problems," Apr. 2015.

[2] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, ser. STOC '71. New York, NY, USA: Association for Computing Machinery, May 1971, pp. 151–158.

[3] M. Sipser, *Introduction to the Theory of Computation*, third edition, international edition ed. Australia Brazil Japan Korea Mexiko Singapore Spain United Kingdom United States: Cengage Learning, 2013.

[4] S. Coniglio, F. Furini, and P. San Segundo, "A new combinatorial branch-and-bound algorithm for the Knapsack Problem with Conflicts," *European Journal of Operational Research*, vol. 289, no. 2, pp. 435–455, Mar. 2021.

[5] U. Pferschy and J. Schauer, "The Knapsack Problem with Conflict Graphs," *Journal of Graph Algorithms and Applications*, vol. 13, no. 2, pp. 233–249, 2009.

[6] M. R. Bonyadi, Z. Michalewicz, and L. Barone, "The travelling thief problem: The first step in the transition from theoretical problems to realistic problems," in *2013 IEEE Congress on Evolutionary Computation*, Jun. 2013, pp. 1037–1044.

[7] A. Fréville, "The multidimensional 0–1 knapsack problem: An overview," *European Journal of Operational Research*, vol. 155, no. 1, pp. 1–21, May 2004.

[8] V. Cacchiani, M. Iori, A. Locatelli, and S. Martello, "Knapsack problems — An overview of recent advances. Part I: Single knapsack problems," *Computers & Operations Research*, vol. 143, p. 105692, Jul. 2022.

[9] S. S. Skiena, "Who is interested in algorithms and why?: Lessons from the Stony Brook algorithms repository," *ACM SIGACT News*, vol. 30, no. 3, pp. 65–74, Sep. 1999.

[10] S. Martello and P. Toth, "An upper bound for the zero-one knapsack problem and a branch and bound algorithm," *European Journal of Operational Research*, vol. 1, no. 3, pp. 169–175, May 1977.

[11] ——, "A New Algorithm for the 0-1 Knapsack Problem," *Management Science*, vol. 34, no. 5, pp. 633–644, May 1988.

[12] D. Pisinger, "A minimal algorithm for the Bounded Knapsack Problem," in *Integer Programming and Combinatorial Optimization*, ser. Lecture Notes in Computer Science, E. Balas and J. Clausen, Eds. Berlin, Heidelberg: Springer, 1995, pp. 95–109.

[13] ——, "A Minimal Algorithm for the 0-1 Knapsack Problem," *Operations Research*, vol. 45, no. 5, pp. 758–767, 1997.

[14] S. Martello, D. Pisinger, and P. Toth, "Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem," *Management Science*, vol. 45, no. 3, pp. 414–424, 1999.

[15] J. Jooken, P. Leyman, and P. De Causmaecker, "Features for the 0-1 knapsack problem based on inclusionwise maximal solutions," *European Journal of Operational Research*, Apr. 2023.

[16] K. Smith-Miles, J. Christiansen, and M. A. Muñoz, "Revisiting where are the hard knapsack problems? via Instance Space Analysis," *Computers & Operations Research*, vol. 128, p. 105184, Apr. 2021.

[17] J. Jooken, P. Leyman, and P. De Causmaecker, "A new class of hard problem instances for the 0–1 knapsack problem," *European Journal of Operational Research*, vol. 301, no. 3, pp. 841–854, Sep. 2022.

[18] V. Chvátal, "Hard Knapsack Problems," *Operations Research*, vol. 28, no. 6, pp. 1402–1411, Dec. 1980.

[19] Z. Gu, G. L. Nemhauser, and M. W. P. Savelsbergh, "Lifted Cover Inequalities for 0-1 Integer Programs: Complexity," *INFORMS Journal on Computing*, vol. 11, no. 1, pp. 117–123, Feb. 1999.

[20] S. Jukna and G. Schnitger, "Yet harder knapsack problems," *Theoretical Computer Science*, vol. 412, no. 45, pp. 6351–6358, Oct. 2011.

[21] F. A. Morales and J. A. Martínez, "Analysis of Divide-and-Conquer strategies for the 0–1 minimization knapsack problem," *Journal of Combinatorial Optimization*, vol. 40, no. 1, pp. 234–278, Jul. 2020.

[22] Y. Yang, N. Boland, and M. Savelsbergh, "Multivariable Branching: A 0-1 Knapsack Problem Case Study," *INFORMS Journal on Computing*, vol. 33, no. 4, pp. 1354–1367, 2021.

[23] M. Hifi, H. Mhalla, and S. Sadfi, "Sensitivity of the Optimum to Perturbations of the Profit or Weight of an Item in the Binary Knapsack Problem," *Journal of Combinatorial Optimization*, vol. 10, no. 3, pp. 239–260, Nov. 2005.

[24] ——, "An adaptive algorithm for the knapsack problem: Perturbation of the profit or weight of an arbitrary item," *European Journal of Industrial Engineering*, vol. 2, no. 2, pp. 134–152, Jan. 2008.

[25] T. Belgacem and M. Hifi, "Sensitivity analysis of the optimum to perturbation of the profit of a subset of items in the binary knapsack problem," *Discrete Optimization*, vol. 5, no. 4, pp. 755–761, Nov. 2008.

[26] D. Pisinger and A. Saidi, "Tolerance analysis for 0–1 knapsack problems," *European Journal of Operational Research*, vol. 258, no. 3, pp. 866–876, May 2017.

[27] T. M. Chan, "Approximation Schemes for 0-1 Knapsack," p. 12 pages, 2018.

[28] C. Jin, "An Improved FPTAS for 0-1 Knapsack," Apr. 2019.

[29] D. Ghosh, N. Chakravarti, and G. Sierksma, "Sensitivity analysis of a greedy heuristic for knapsack problems," *European Journal of Operational Research*, vol. 169, no. 1, pp. 340–350, Feb. 2006.

[30] C. Wilbaut, R. Todosijevic, S. Hanafi, and A. Fréville, "Heuristic and exact reduction procedures to solve the discounted 0–1 knapsack problem," *European Journal of Operational Research*, vol. 304, no. 3, pp. 901–911, Feb. 2023.

[31] T. H. Cormen, Ed., *Introduction to Algorithms*, 3rd ed. Cambridge, Mass: MIT Press, 2009.

[32] J. Dong, "Dynamic Programming — 0/1 Knapsack (Python Code)," Sep. 2020.

[33] G. B. Dantzig, "Discrete-Variable Extremum Problems," *Operations Research*, vol. 5, no. 2, pp. 266–277, 1957.