Performance Comparison of Unbounded Knapsack Problem Formulations

Oksana Pichugina[1,2], Olha Matsiy[3] and Yuriy Skob[2]

[1] *University of Toronto, 27 King's College Circle, Toronto, M5S 1A1, Canada*
[2] *National Aerospace University "Kharkiv Aviation Institute", 17 Chkalova Street, Kharkiv, 61070 Ukraine*
[3] *V. N. Karazin Kharkiv National University, 4 Svobody Sq., Kharkiv, 61022, Ukraine*

Abstract

This study investigates various linear discrete formulations of the unbounded knapsack problem, including binary and bounded forms. Two variations of binary knapsack problem reformulations are examined, each with distinct linear constraints. Gurobi and CPLEX solvers are used to compare the performance of these models. The computational experiments are conducted on randomly generated instances of sizes 10-100. The results revealed that, on average, Gurobi was 20% faster than CPLEX. Integer, bounded, and 0/1 knapsack problem formulations had comparable mean run times. Incorporating specific constraints in the 0/1 formulation yielded superior results compared to the basic 0/1 knapsack problem model. Overall, this research contributes to understanding efficient and effective formulations for solving the unbounded knapsack problem and choosing a better solver.

Keywords 1

Knapsack problem, binary knapsack problem, unbounded knapsack problem, bounded knapsack problem, Gurobi, CPLEX, Integer Programming, Linear Binary Optimization, Combinatorial Optimization, Euclidean Combinatorial Optimization

1. Introduction

Combinatorial optimization (CO) is a branch of optimization that deals with finding the best solution among a finite set of possible solutions [1,7]. It has wide-ranging applications in various fields, including Computer Science, Operations Research, Engineering, Finance, and many others. Overall, CO is highly important because it provides powerful tools for solving complex problems and making better decisions in a wide range of applications [1,7].

Euclidean Combinatorial Optimization (ECO) is a subfield of CO dealing with images of combinatorial sets in Euclidean space and their convex hulls (combinatorial polytopes) [6,11,14,16].

Integer programming (IP) is a subdomain of ECO that deals with optimization problems where the variables are constrained to take on integer values. IP is an important tool in CO because many CO problems can be formulated as integer programming problems, such as the travelling salesman problem, Knapsack Problem (KP), the graph colouring problem etc. [18]. IP plays a crucial role in combinatorial optimization by providing a powerful tool for solving many problems involving integer decision variables.

This paper focuses on KP, a classical CO problem in which a set of items must be packed into a knapsack subject to capacity constraints. The objective is to maximize the total value of the items packed into the knapsack, subject to the constraint that the total weight of the items does not exceed the knapsack capacity [3,10].

COLINS-2023: 7th International Conference on Computational Linguistics and Intelligent Systems, April 20–21, 2023, Kharkiv, Ukraine
EMAIL: oksanapichugina1@gmail.com (O. Pichugina); olga.matsiy@gmail.com (O. Matsiy); yuriy.skob@gmail.com (Yu. Skob)
ORCID: 0000-0002-7099-8967 (O. Pichugina); 0000-0002-1350-9418 (O. Matsiy); 0000-0003-3224-1709 (Yu. Skob)

Binary optimization (Binary Programming, BP) is a special case of IP that deals with optimization problems where the decision variables are restricted to take on binary values, i.e., 0 or 1. BP problems arise in many applications, such as network flow problems, scheduling problems, and Boolean function optimization. Here, the decision variables can represent binary decisions, such as whether to include or exclude a certain item in a set or schedule a job at a particular moment. Binary optimization problems have many applications in various fields. They can be solved using various techniques, including IP, dynamic programming, and heuristic algorithms such as simulated annealing, genetic algorithms, and tabu search.

Due to the discreteness of the search domain, CO problems allow multiple formulations. The performance of a solver can depend greatly on the formulation of the problem being solved. Different problem formulations can result in different solution times, solution quality, and solution methods used by the solver. This is particularly true for the KP, which can be formulated in various ways [3,10], including as an IP problem [18], a dynamic programming problem [11], a constraint satisfaction problem, the Quadratic Unconstrained Binary Problem (QUBO) [15] etc.

For example, in KP, the choice of the objective function can significantly impact the solver's performance. A simple objective function that only considers the value of items being packed can result in a suboptimal solution, particularly when the weights of the items are not uniformly distributed. In contrast, a more complex objective function that balances the value and weight of the packed items can result in a better solution but may require more computational resources. Similarly, the choice of solver used to solve the problem can also affect performance. Different solvers may be better suited to particular problem formulations or problem instances and can have different strengths and weaknesses. For example, some solvers may be better at solving linear programming problems, while others may be better at solving mixed-integer or nonlinear programming problems.

In general, it is highly important to choose a problem formulation that is well-suited to a specific problem being solved and a solver that is well-suited to the problem formulation. Experimenting with different problem formulations and solvers may also be useful to determine which combination yields the best performance for a particular problem instance.

KP is a well-known problem in computer science and optimization that has important applications in many fields, including finance, resource allocation, and logistics [3,5].

In this paper, we aim to investigate the solution time of the Unbounded KP, where each item placed in the knapsack has an unlimited number of copies depending on its formulation and solver, where solvers are limited to well-known Gurobi and CPLEX.

2. Knapsack problem (KP)

The **knapsack problem (KP)** [5,9,17,18] is a classic problem in CO that involves selecting a subset of items with the goal of maximizing the total value while satisfying a constraint on the total weight or volume.

2.1. KP variations

There are several variants of the knapsack problem, including:

- **0/1 KP (binary KP)**: This is the classic problem variant, where each item can be included in the knapsack either completely (i.e., with weight equal to its capacity) or not at all. The goal is to maximize the total value of the selected items (**Objective 1**) subject to the constraint on the total weight (**Constraint 1**).
- **Bounded KP (BKP)**: In this variant, each item has a limited number of copies available, and the goal is to select a subset of items to maximize Objective 1 subject to Constraint 1 and the limit on the number of copies of each item that can be included (**Constraint 2**).
- **Unbounded KP (UKP, integer KP):** In this special case, items have an unlimited number of copies, and the goal is to select a subset of them to maximize Objective 1 subject to Constraint 1 and the integrity of the number of copies of each item (**Constraint 3**).

- **Fractional knapsack problem (FKP):** In this version, items can be included in the knapsack partially, i.e., fractions of an item can be utilized. The goal is to maximize Objective 1 subject to Constraint 1.
- **Subset sum problem (SSP):** This is a special case of the knapsack problem where each item has the same weight and aims to select a subset that sums to a specific target value.

These knapsack problems possess different properties and may require different algorithms. Also, formulations of the same problem can be different. Respectively, appropriate algorithms vary depending on the formulations

2.2. KP applications

KP has practical applications in many fields [1,2,3,4,8,10,12,13], including:

- **Resource allocation**: In manufacturing and logistics, the KP can be used to determine which orders to fulfill, which items to stock, and how to allocate resources such as vehicles, workers, and machines.
- **Finance**: The KP can be used in financial portfolio optimization, where an investor must choose a set of investments to maximize returns while respecting constraints on available capital and risk tolerance.
- **Project selection**: In project management, the KP can be utilized to select a set of projects that maximize the expected value of the portfolio while respecting constraints such as budget and resource availability.
- **Cutting stock problem**: In cutting stock problems, KP is used to find the optimal way to cut large items, such as sheets of metal, into smaller pieces with minimum waste.

Also, the KP has various applications in natural language processing (NLP) [2,12]. Here are some examples:

- **Named entity recognition:** Named entity recognition (NER) identifies named entities (such as people, organizations, and locations) in text. It is a problem of selecting the most relevant named entities can be formulated as a KP, where each named entity is represented by a weight (indicating its relevance) and a value (indicating its frequency or importance).
- **Sentence compression**: Sentence compression involves generating shorter sentences that preserve the meaning of the original sentences. This can be formulated as a KP, where the goal is to select the most important words or phrases from the original sentence to include in the compressed sentence.
- **Text summarization**: KP can be used in text summarization, where the goal is to select a subset of sentences that represent the most important content of a document. The problem can be formulated as a KP, where each sentence is represented by a weight (indicating its importance) and a value (indicating its length).
- **Keyword selection**: In keyword selection, the goal is to choose a set of keywords that best represent the content of a document or a set of documents. This can be formulated as a KP, where each keyword is represented by a weight (importance) and a value (frequency or relevance).

Thus, KP is an important problem in computer science and optimization, and its applications extend to a wide range of fields.

2.3. Literature on KP

The KP is a well-studied problem in the CO, and many books, articles, and research papers exist on the topic. Among them are books:

- Books [3,10] provide a comprehensive overview of the KP and its variants, including exact and heuristic solution methods, complexity analysis, and applications in various fields.
- The source [5] is a classic book covering fundamental computer science algorithms, including the KP and dynamic programming.
- The book [18] provides an in-depth treatment of integer programming, including the KP, mixed-integer programming, and branch-and-bound methods for solving IP problems.

- [9] is the book dedicated to the dynamic programming approach to solving the KP and other optimization problems. It covers the theory and application of dynamic programming algorithms with examples and exercises.
- The book [18] covers approximation algorithms for optimization problems, particularly the KP, with a focus on algorithm design and analysis.

Many more resources are available, including research papers, conference proceedings, and online tutorials.

2.4. KP benchmark libraries

Several benchmark libraries are available for the KP that can be used to evaluate the performance of different algorithms and solvers. Here are some of the most commonly used benchmark libraries:

- **OR-Library** [19]: The OR-Library is a collection of test problems for various optimization problems, including KP instances. It includes a range of problem sizes and characteristics, such as varying numbers of items and capacities, and can be used to evaluate the performance of different algorithms and solvers.
- **DIMACS Implementation Challenge** [21]: The DIMACS Implementation Challenge includes a set of benchmark instances for several combinatorial optimization problems, including the KP. The instances are designed to be challenging and can be used to evaluate the performance of different algorithms and solvers.
- **MIPLIB [20]**: The Mixed IP Library (MIPLIB) includes a set of benchmark instances for mixed IP problems, including the KP. It contains instances of varying sizes and characteristics and can be used to evaluate the performance of different solvers and algorithms.

These benchmark libraries can be useful for evaluating the performance of different algorithms and solvers and comparing the effectiveness of different approaches to solving the KP.

2.5. KP solution techniques

Various methods are known to solve the KP, ranging from exact methods providing optimal solutions to heuristic approaches that provide approximate solutions [3,10]. Among them are exact and approximate methods

2.5.1. Exact methods

- **Linear IP (ILP):** This mathematical optimization technique can solve the KP by formulating the problem as a linear program with integer variables and using an ILP solver to find the optimal solution.
- **Branch and Bound**: This method enables solving quite large instances by exploring a search tree to find the optimal solution. The search tree is constructed by branching on each item and exploring all possible feasible combinations of items that lead to a better solution.
- **Constraint Programming**: This declarative programming paradigm can solve the KP by expressing the problem's constraints as logical constraints and using a constraint solver to find a feasible solution.
- **Dynamic Programming**: This exact method can solve the KP efficiently for small problem sizes. It works by constructing a dynamic programming table to determine the maximum value that can be obtained for each subproblem of the knapsack packing. Then it uses these subproblems to solve the larger problem.

2.5.2. Approximate methods

- **Greedy Algorithms**: These heuristic methods provide approximate solutions by greedily selecting items based on criteria such as the value-to-weight ratio. Although they do not always provide optimal solutions, they are computationally efficient and can solve the KP with large problem sizes.

- **Metaheuristic Algorithms**: These heuristic methods use search and optimization techniques to find good solutions to Knapsack problems. Examples include genetic algorithms, simulated annealing, and tabu search.

The choice of method depends on the problem size, the required level of optimality, and the computational resources available. Exact methods are preferred for small problem sizes, while heuristic and metaheuristic methods are preferred for larger problem sizes or when an approximate solution is acceptable. Also, a set of relevant methods depends on the selected formulation of the KP.

2.6. Gurobi and CPLEX solvers

Gurobi and CPLEX are two of the most popular optimization solvers researchers and practitioners use to solve complex mathematical optimization problems. Here are some brief characteristics and comparisons between the two:

Gurobi:

- The developer is Gurobi Optimization, LLC;
- uses state-of-the-art algorithms and heuristics to solve a wide range of optimization problems, including linear programming, quadratic programming, mixed-integer programming, and convex optimization;
- has a user-friendly interface and can be easily integrated with programming languages such as Python, C++, and Java;
- offers advanced features, such as parallel processing, customized callbacks, and automatic tuning for optimal performance;
- known for its speed and efficiency and is often used for solving large-scale optimization problems in finance, energy, and logistics industries.

CPLEX:

- Developer is IBM;
- offers a variety of optimization algorithms for linear programming, mixed-integer programming, quadratic programming etc.;
- is highly customizable and can be easily integrated with programming languages such as C++, Java, and Python;
- offers advanced features, such as parallel processing, customized callbacks, and warm-start capabilities;
- is known for its robustness and reliability.

Docplex is a Python package developed by IBM that provides an easy-to-use interface for modeling and solving optimization problems using the IBM CPLEX solver. Docplex allows users to formulate and solve linear programming (LP), integer programming (IP), mixed-integer programming (MIP), quadratic programming (QP), and mixed-integer quadratic programming (MIQP) problems using a high-level, object-oriented modeling language.

When it comes to a comparison between Gurobi and CPLEX, the choice depends on the specific problem being solved.

Some points of comparison are:

- **Performance**: Both solvers are known for their speed and efficiency, but Gurobi is often considered slightly faster and more memory-efficient for certain types of problems.
- **User interface**: Gurobi is often considered more user-friendly and easier to set up, while CPLEX offers more customization options for advanced users.
- **Licensing**: Both solvers offer free academic licenses, but Gurobi is generally more expensive for commercial use.

Thus, Gurobi and CPLEX are excellent optimization solvers with their own strengths and weaknesses. The choice between them depends on the user's specific needs and the problem being solved.

Gurobi and CPLEX are both suitable for solving knapsack problems, as they offer powerful optimization algorithms for integer programming problems, which the KP can be formulated as.

Solving the KP using Gurobi or CPLEX can be formulated as a binary or integer linear programming problem, where the binary decision variables represent whether an item is packed in the knapsack or not. Respectively, the objective function is the sum of the values of the packed items, and the capacity constraint is represented as a linear inequality.

Gurobi and CPLEX both offer powerful algorithms for solving MILP problems, including branch-and-bound, cutting-plane, and heuristics, which can be used to find the optimal or near-optimal solutions to the KP. Additionally, Gurobi and CPLEX offer some features that can be used to speed up the optimization process and improve the quality of the solutions.

In summary, both Gurobi and CPLEX are well-suited for solving the KP and offer powerful algorithms and features for finding optimal or near-optimal solutions. The choice between them may depend on factors such as the specific problem formulation, the size and complexity of the problem, and personal preferences.

Among other solvers to solve the KP is another popular Integer programming solver SCIP, constraint programming solvers, Choco and Gecode, solvers supporting evolutionary and local search algorithms.

3. Unbounded KP formulations

The UKP is a classic optimization problem in which a set of items, each with a weight w_i and a value v_i, must be packed into a knapsack of limited capacity C in such a way as to maximize the total value of the items packed.

One way to formulate the unbounded knapsack problem mathematically is as follows:

Let n be the number of items, and let w_i and v_i denote the weight and value of the i-th item, respectively. Let C be the maximum capacity that the knapsack can hold. Then the goal is to find the maximum possible value that can be packed into the knapsack.

Let $J_n = \{1,...,n\}$, $J_n^0 = J_n \cup \{0\}$, decision variables form a vector $(x_1,...,x_n)$, where x_i is a decision variable for each item i represents the number of times that item i is included in the knapsack. Then the objective function is:

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

subject to constraints:

$$\sum_{i=1}^{n} w_i x_i \le C \tag{2}$$

$$x_i \in Z_+,\ i \in J_n. \tag{3}$$

To the model (1)-(3) further we will refer to as **UKP.IP**.

Let us reformulate it as a Bounded KP. For that, we will define the maximum number $k_i \in Z_+$ of copies of the item i that can be used in the knapsack. Clearly,

$$k_i = \left[\frac{C}{w_i}\right],\ i \in J_n. \tag{4}$$

In this notation, we obtain the Bounded KP (1), (2),

$$x_i \in J_{k_i}^0,\ i \in J_n, \tag{5}$$

where k_i is given by (4) (further we referred to as **UKP.BKP)**. Evidently, the models UKP.IP and UKP.BKP are equivalent.

Now, we can reduce UKP.BKP to 0/1.KP of higher dimension. For that, we introduce multisets of weights and values

$$W=\left\{w_1^{k_1},...,w_n^{k_n}\right\},\ V=\left\{v_1^{k_1},...,v_n^{k_n}\right\}, \tag{6}$$

where the notation $w_i^{k_i}$ means that the multiplicity of w_i in W is k_i for $i\in J_n$. Thus the cardinality of the multisets (6) is

$$N=\sum_{i=1}^{n}k_i \tag{7}$$

and enumerating elements of W,V yields

$$\begin{aligned} W'&=\left\{w_j'\right\}_{j\in J_N}=\left\{w_1,...,w_1,w_2,...,w_2,...,w_n,...,w_n\right\},\\ V'&=\left\{v_j'\right\}_{j\in J_N}=\left\{v_1,...,v_1,v_2,...,v_2,...,v_n,...,v_n\right\}. \end{aligned} \tag{8}$$

In these notations, a binary equivalent of **UKP.BKP is**

$$maximize\ \sum_{j=1}^{N}v_j'y_j \tag{9}$$

subject to constraints:

$$\sum_{j=1}^{N}w_j'y_j\le C, \tag{10}$$

$$y_j\in\{0,1\},\ j\in J_N, \tag{11}$$

where N is given by (7). $y=(y_1,...,y_N)$ is another vector of decision variables where y_j is a binary decision variable that takes a value of 1 if an item j is selected for packing, and 0 otherwise.

To the model (9)-(11) we will refer to it as **UKP.0/1KP.1.**

Note that, in contrast to the previous model, in UKP.0/1KP.1 some items can be identical, i.e., they are represented by the same tuple $\left\langle\begin{matrix}w\\v\end{matrix}\right\rangle$, $w\in W,\ v\in V$, where $W=\{w_i\}_{i\in J_n},V=\{v_i\}_{i\in J_n}$.

Without loss of generality, we can assume that all tuples in $\left\langle\begin{matrix}W\\V\end{matrix}\right\rangle=\left\{\left\langle\begin{matrix}w_i\\v_i\end{matrix}\right\rangle\right\}_{i\in J_n}$ are ordered lexicographically. More specifically, under the ordering $\left\langle\begin{matrix}w_i\\v_i\end{matrix}\right\rangle\preceq\left\langle\begin{matrix}w_{i+1}\\v_{i+1}\end{matrix}\right\rangle, i\in J_{n-1}$, we mean that

$$w_i\le w_{i+1}, i\in J_{n-1}, \tag{12}$$

if $i'\in J_{n-1}$ such that

$$w_{i'}=w_{i'+1},\ \text{then}\ v_{i'}\le v_{i'+1}. \tag{13}$$

From the ordering (12), (13) of the coordinates of decision variables, it follows that W',V' are also ordered. Namely.

$$w_j'\le w_{j+1}',\ j\in J_N\ \text{ and }\ v_j'\le v_{j+1}',\ j\in I_i. \tag{14}$$

where I_i is a set of indices of weights from W' equal to $w_i, i\in J_n$.

Now, we can strengthen the model UKP.0/1KP.1 by adding the following constraints:

$$y_j\le y_{j+1},\ j\in J_{I_i}, i\in J_n\ \text{ and }\ v_j'\le v_{j+1}',\ j\in I_i. \tag{15}$$

To the model (9)-(11), (15) we will refer to as **UKP.0/1KP.2.**

4. Computational experiment

We solved randomly generated KP instances of dimensions 10-100 in the form of the models UKP.IP, UKP.BKP, UKP.0/1KP.1, UKP.0/1KP.2 in order to compare their performance in two selected popular solvers – CPLEX and Gurobi, the software implementation was done in Python. For CPLEX, the package DOCPLEX was used.

Generator of instance outline:

- Set n;
- Set the number prn of problems generated in the series;
- Set ranges $[w_{\min}, w_{\max}], [v_{\min}, v_{\max}]$ for generating weights and values, respectively;
- Set m - the upper bound for items copies;
- Generate sets $W \subset U([w_{\min}, w_{\max}]), V \subset U([v_{\min}, v_{\max}])$;
- Generate a constant $C \in m \cdot c \cdot [0.8, 1]$, where the value $c = \min_{i \in J_n} w_i$ such that $C \geq \max_{i \in J_n} \mathrm{w}_i$ is fulfilled.

We generated 100 instances in each dimension $10, 20, ..., 100$ and experimented with them. The main conclusion is that Gurobi performs better, and, on average, it is nearly 20% faster than CPLEX. The runtime of models UKP.IP, UKP.BKP, UKP.0/1KP.1 is very similar, and we cannot single out a top model. At the same time, a comparison of the binary models UKP.0/1KP.1 and UKP.0/1KP.2 shows that the strengthening UKP.0/1KP.1 by introducing the constraint (15) is fruitful as a performance of UKP.0/1KP.2 is around 15% better than UKP.0/1KP.1.

5. Further work

The classical KP is a well-studied problem, but there are several of its generalizations and extensions that are of interest in the research community [3,8,10,13]:

- **Multiple KP**: This is a KP generalization where multiple knapsacks are available for packing items. Each item has a weight and value, and the goal is to pack the items into the knapsacks while maximizing the total value subject to a capacity constraint on each knapsack.
- **Time-dependent KP**: This is an extension of the KP where the items have a time-dependent value. Each item has a value that varies over time, and the goal is to pack the items into the knapsack at the optimal time to maximize the total value.
- **Stochastic KP**: In this problem, the item values and/or weights are uncertain. The goal is to pack the items into the knapsack to maximize the expected value subject to the capacity constraint.
- **Multiple-choice knapsack problem**: Each item is associated with a set of options. The goal is to choose one option for each item to maximize the total value subject to the capacity constraint on the knapsack.

These generalizations and extensions of the classical KP provide additional complexities to the problem, making it more challenging to find an optimal solution. Researchers intensively work on developing general and specialized algorithms and heuristics for each of these variants of the KP, but still, many open problems remain in this research field.

The knapsack problems arise as a subproblem in many nonlinear binary optimization problems

$$\textit{maximize}\ f_0(y) \tag{16}$$

subject to the weight constraint (10), the binary constraint (11) and any others linear or nonlinear constraints:

$$f_i(x) \leq 0,\ i \in J_m, \tag{17}$$

where $f_i : B_N \to B_1,\ i \in J_m$. Functions present in (17) can be incorporated into the penalty function, and we come to the equivalent optimization problem to (10), (11), (16), (17) having the form of:

$$\textit{minimize}\left\{ \mathrm{F}_\lambda(y) = -f_0(y) + \sum_{i=1}^{m} \lambda_i f_i(y) \right\}, \tag{18}$$

where $\lambda = (\lambda_1, ..., \lambda_m) \in R^m_{>0}$ is a vector of penalty parameters chosen large enough to ensure that optimizers of the above problem and the problem (18) coincide. The function $F_\lambda(y)$ can always be convexified with the help of a regularization term $\left(y - \frac{\mathbf{1}}{\mathbf{2}}\right)^2 - \frac{n}{4}$, where $\frac{\mathbf{1}}{\mathbf{2}} = \left(\frac{1}{2}, ..., \frac{1}{2}\right)$. As a result, we come to a problem

$$\textit{minimize} \left\{ \Phi_{\lambda,\mu}(y) = F_\lambda(y) + \mu\left(\left(y - \frac{\mathbf{1}}{\mathbf{2}}\right)^2 - \frac{n}{4}\right)\right\}, \tag{19}$$

subject to constraints (10), (11) equivalent to the above two. Here $\mu > 0$ is a regularization parameter chosen large enough to guarantee convexity of $\Phi_{\lambda,\mu}(y)$. Also, we can assume that $\Phi_{\lambda,\mu}(y)$ is differentiable. If not it can be replaced by its convex differentiable extension from B_n onto R^n since a set B_n is vertex located [16]. Now, the conditional gradient method can be applied to solving (19) with constraints (10), (11) where an auxiliary problem of minimizing a gradient of $\Phi_{\lambda,\mu}(y)$ on the correspondent polytope is solved on each iteration. These auxiliary problems are, in fact, binary knapsack problems. In such a way KPs can be used in nonlinear optimization over the binary domain.

In future, we plan to extend the experimental part onto other formulations of the classical KP, such as QUBO, and consider nonlinear optimization generalizations and the listed and many more extensions and generalizations of KP.

6. Conclusion

This study explored different linear discrete formulations for solving the unbounded knapsack problem, including transforming the problem into binary and bounded forms. Two variations of binary knapsack reformulations were examined, each with distinct linear constraints. The performance of these models was compared using Gurobi and CPLEX solvers, and the implementation was carried out in Python. The experiments were conducted on randomly generated instances of sizes ranging from 10 to 100, with the weights and values of items following uniform discrete distributions within specific ranges. Our findings indicate that on average, Gurobi is approximately 20% faster than CPLEX. The IKP, BKP, and 0/1 KP formulations had similar mean run times. Additionally, the inclusion of specific constraints in the 0/1 formulation demonstrated superior performance over the underlying 0/1 KP model.

7. References

[1] N. Christofides, Combinatorial optimization, Wiley, 1979. URL: http://www.gbv.de/dms/hbz/toc/ht000072474.pdf, OCLC: 4495250.

[2] T. Hirao, Y. Yoshida, M. Nishino, N. Yasuda, M. Nagata, Single-document summarization as a tree knapsack problem, in: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2013-10, pp. 1515–1520. URL: https://aclanthology.org/D13-1158.

[3] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, softcover reprint of hardcover 1st ed. 2004 ed., Springer, 2010-12-07.

[4] L. Kirichenko, T. Radivilova, V. Ryzhanov, Applying visibility graphs to classify time series, in: S. Babichev, V. Lytvynenko (Eds.), Lecture Notes in Computational Intelligence and Decision Making, Lecture Notes on Data Engineering and Communications Technologies, Springer International Publishing, 2022, pp. 397–409. doi:10.1007/978-3-030-82014-5_26.

[5] D. Knuth, Art of Computer Programming, The: Fundamental Algorithms, Volume 1, 3rd ed., Addison-Wesley Professional, 1997-07-07.

[6] L. Koliechkina, O. Pichugina, O. Dvirna, Horizontal method application to multiobjective combinatorial optimization over permutations, in: 2022 IEEE 3rd International Conference on System Analysis & Intelligent Computing (SAIC), IEEE, 2022-10-04, pp. 1–5. doi:10.1109/SAIC57818.2022.9923018.

[7] B. Korte, J. Vygen, Combinatorial Optimization: Theory and Algorithms, 6th ed. 2018 ed., Springer, 2018-03-23.

[8] S. Laabadi, M. Naimi, H. E. Amri, B. Achchab, The 0/1 multidimensional knapsack problem and its variants: A survey of practical models and heuristic approaches 8 (2018-09-04) 395–439. doi:10.4236/ajor.2018.85023, number: 5 Publisher: Scientific Research Publishing.

[9] A. Lew, H. Mauch, Dynamic Programming: A Computational Tool, softcover reprint of hardcover 1st ed. 2007 ed., Springer, 2010-11-18.

[10] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, revised ed., John Wiley & Sons, 1990-11-30.

[11] O. B. Matsiy, A. V. Morozov, A. V. Panishev, A recurrent algorithm to solve the weighted matching problem 52 (2016-09-01) 748–757. doi:10.1007/s10559-016-9876-4.

[12] H. Nishikawa, T. Hirao, T. Makino, Y. Matsuo, Text summarization model based on redundancy constrained knapsack problem, in: Proceedings of COLING 2012: Posters, The COLING 2012 Organizing Committee, 2012-12, pp. 893–902. URL: https://aclanthology.org/C12-2087.

[13] O. Pichugina, L. Koliechkina, The constrained knapsack problem: Models and the polyhedral-ellipsoid method, in: A. Strekalovsky, Y. Kochetov, T. Gruzdeva, A. Orlov (Eds.), Mathematical Optimization Theory and Operations Research: Recent Trends, Communications in Computer and Information Science, Springer International Publishing, 2021, pp. 233–247. doi:10.1007/978-3-030-86433-0_16.

[14] O. Pichugina, S. Yakovlev, New classes of unconstrained permutation-based problems and their solutions, in: 2022 IEEE 3rd International Conference on System Analysis & Intelligent Computing (SAIC), IEEE, 2022-10-04, pp. 1–5. doi:10.1109/SAIC57818.2022.9922919.

[15] A. P. Punnen, The Quadratic Unconstrained Binary Optimization Problem: Theory, Algorithms, and Applications, 1st ed. 2022 ed., Springer Nature, 2022-07-13.

[16] Y. G. Stoyan, S. V. Yakovlev, Theory and methods of Euclidian combinatorial optimization: Current status and prospects 56 (2020-05-01) 366–379. doi:10.1007/s10559-020-00253-6.

[17] V. V. Vazirani, Approximation Algorithms, softcover reprint of hardcover 1st ed. 2001 ed., Springer/Sci-Tech/Trade, 2010-12-08.

[18] L. A. Wolsey, Integer Programming, 2nd ed., Wiley, 2020-10-20.

[19] OR-LIBRARY. URL: http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

[20] MIPLIB - mixed integer problem library. URL: https://miplib2010.zib.de/.

[21] DIMACS :: Implementation challenges. URL: http://dimacs.rutgers.edu/programs/
challenge/.