Democratic Republic of Algeria

Ministry Of Higher Education and Scientific Research

University Of KASDI Merbah OUARGLA-Algeria

Faculty Of New Information and Communication Technologies

Department Of Computer Sciences and Information Technologies

Title:

Optimizing the Multidimensional Knapsack Problem:

A Hybrid Approach of Genetic Algorithm and Branch-and-Cut

Presented By:

FENNICHE Mohammed amine

BABA SIDI Mohammed Khaled

Jury Members:

Dr. KHELIFA Meriem	UKM OUARGLA	SUPERVISOR
Dr. MESSAID Abdessalam	UKM OUARGLA	PRESIDENT
Dr. BENBEZZIANE Mohammed	UKM OUARGLA	EXAMINER

Academic Year :2023/2024

ACKNOWLEDGMENTS

“First and foremost, we would like to express my deepest gratitude to the one above all of us, the omnipresent Allah, for answering my prayers and providing us with the strength to overcome challenges. Thank you, Lord, for guiding us and bestowing upon us the perseverance to see this journey through. we would also like to acknowledge and thank our beloved parents for their unwaver- ing support, love, and guidance.

Your presence in our lives has been a constant source of strength, and we are forever grateful for your sacrifices and belief in our abilities. Lastly, we would like to extend our appreciation to all those who have contributed in any way to the completion of this thesis. Thank you for being a part of our academic journey and for your unwavering belief in our potential

we extend our sincere appreciation to our supervisor ***Dr . KHELIFA MERIEM*** *, for her unwavering support, sage advice, insightful criticisms, and constant encouragement. Her guidance has been invaluable, and her belief in our capabilities has motivated us to push beyond our limits, offering unique perspective and expertise that greatly enriched our understanding and approach to this study. Together, their combined guidance has not only shaped this thesis but has also significantly contributed to my growth as a researcher and thinker. ‘’*

Thank you all

FENNICHE Mohammed amine

BABA SIDI Mohammed Khaled

ABSTRACT

This study introduces an improved Hybrid **Genetic Algorithm** (GA) combined with a **Branch and Cut** technique to find near-optimal solutions for the **NP-hard Multidimensional Knapsack Problem** (MKP). The MKP, a variant of the traditional 0-1 Knapsack problem, is a discrete optimization challenge used to model extensive issues such as network management, admission control in adaptive multimedia systems, telecommunications, and quality of service. Our hybrid approach leverages the global search potential of Genetic Algorithms alongside the precise optimization capabilities of Branch and Cut methods. This method is inspired by principles of evolutionary computation and mathematical optimization to efficiently navigate complex problem spaces. The enhancement involves incorporating a local search heuristic within the hybrid framework, ensuring both the exploration and exploitation capabilities of the Genetic Algorithm and local search are effectively utilized. Experimental results demonstrate that our Hybrid GA with Branch and Cut approach delivers superior performance in solution quality, significantly surpassing the current best-known results for the **OR5x100-0.25 benchmark**.

Keywords: Multidimensional-Knapsack Problem, Genetic Algorithm, Branch and Cut

الملخص

تقدم هذه الدراسة **خوارزمية جينية** هجينة محسنة مدمجة مع تقنية **الفرع والقطع** للعثور على حلول شبه مثالية لمشكلة حقيبة الظهر متعددة الأبعاد .يعد **MKP** مشكل من درجة **NP-Hard**، وهي نوع من مشكلة حقيبة الظهر التقليدية 0-1، تحديًا في التحسين التقديري تُستخدم لنمذجة مشاكل واسعة مثل إدارة الشبكات، والتحكم في القبول في أنظمة الوسائط المتعددة التكيفية، والاتصالات، وجودة الخدمة. تستفيد طريقتنا الهجينة من إمكانات البحث العالمية للخوارزميات الجينية مع قدرات التحسين الدقيقة لأساليب "الفرع والقطع". تستند هذه الطريقة إلى مبادئ الحساب التطوري والتحسين الرياضي للتنقل بكفاءة في مساحات المشكلات المعقدة. يتضمن التحسين دمج خوارزمية بحث محلية ضمن الإطار الهجين، مما يضمن الاستفادة الفعالة من قدرات الاستكشاف والاستغلال لكل من الخوارزمية الجينية والبحث المحلي. تظهر النتائج التجريبية أن نهجنا الهجين من الخوارزمية الجينية مع "الفرع والقطع" يقدم أداءً متفوقًا من حيث جودة الحل، متفوقًا بشكل كبير على أفضل النتائج المعروفة للمرجعية **OR5x100-0.25.**

الكلمات المفتاحية: حقيبة الظهر متعددة الأبعاد, خوارزمية جينية, الفرع و القطع

Résumé

Cette étude présente un **Algorithme Génétique** Hybride (GA) amélioré combiné avec une approche de **Branch and Cut** pour trouver des solutions quasi-optimales au problème **NP-difficile du Sac à Dos Multidimensionnel** (MKP). Le MKP, une variante du problème classique du Sac à Dos 0-1, est un défi d'optimisation discrète utilisé pour modéliser des problèmes étendus tels que la gestion de réseaux, le contrôle d'admission dans les systèmes multimédias adaptatifs, les télécommunications et la qualité de service (QoS). Notre approche hybride exploite le potentiel de recherche globale des Algorithmes Génétiques avec les capacités d'optimisation précises des méthodes Branch and Cut. Cette méthode s'inspire des principes de la computation évolutionnaire et de l'optimisation mathématique pour naviguer efficacement dans des espaces de problèmes complexes. L'amélioration consiste à intégrer une heuristique de recherche locale dans le cadre hybride, garantissant ainsi que les capacités d'exploration et d'exploitation de l'Algorithme Génétique et de la recherche locale sont efficacement utilisées. Les résultats expérimentaux démontrent que notre approche d'Algorithme Génétique Hybride avec Branch and Cut offre des performances supérieures en termes de qualité de solution, surpassant significativement les meilleurs résultats connus pour le **benchmark OR5x100-0.25**.

Les mots Clé: Problème du Sac à Dos Multidimensionnel, Algorithme Génétique, Branch and Cut

Contents

Table of Figures

General Introduction

General Introduction

The field of combinatorial optimization has long been a focal point of research due to its vast array of applications in various domains, including logistics, finance, manufacturing, and telecommunications. At the heart of combinatorial optimization lies the challenge of making optimal decisions from a finite set of feasible solutions. These problems are often characterized by their NP-hard complexity, making them difficult to solve using conventional methods. Among these problems, the 0/1 Multidimensional Knapsack Problem (MKP) stands out due to its practical relevance and computational intensity.

The 0/1 Multidimensional Knapsack Problem is a variant of the classical knapsack problem, where the goal is to maximize the total profit of items placed in a knapsack without exceeding the capacity constraints in multiple dimensions. This problem is representative of many real-world scenarios, such as resource allocation, portfolio selection, and cargo loading, where multiple constraints must be considered simultaneously.

Traditional methods for solving the MKP, such as dynamic programming and branch and bound, often fall short due to the exponential growth of the solution space with increasing problem size. Consequently, heuristic and meta-heuristic approaches, which provide near-optimal solutions within a reasonable time frame, have gained prominence. Among these, genetic algorithms (GAs) have shown considerable promise due to their adaptability and robustness in exploring large solution spaces.

However, genetic algorithms alone can sometimes struggle with convergence speed and solution accuracy. To address these limitations, hybrid approaches that combine the strengths

of multiple methods have been proposed. One such approach is the integration of genetic algorithms with the branch and cut method, a powerful exact technique that systematically prunes the solution space and tightens the problem formulation using cutting planes.

This thesis aims to develop and evaluate a hybrid algorithm that leverages the exploratory power of genetic algorithms and the precision of the branch and cut method to solve the 0/1 Multidimensional Knapsack Problem more efficiently. The proposed approach seeks to strike a balance between exploration and exploitation, enhancing both the convergence rate and the quality of the solutions obtained.

In this introductory chapter, we will first provide an overview of combinatorial optimization research, outlining its importance and the various types of problems it encompasses. We will then delve into the specific characteristics and challenges of the 0/1 Multidimensional Knapsack Problem. Following this, we will introduce genetic algorithms and the branch and cut method, highlighting their respective strengths and limitations. Finally, we will present an outline of the thesis, summarizing the content of each subsequent chapter.

Structure of the Thesis

- **Chapter 1: Combinatorial Optimization Research**
  This chapter will explore the foundations of combinatorial optimization, discussing key concepts, methods, and problem types. It will provide a comprehensive background that contextualizes the relevance of the MKP.
- **Chapter 2: Multidimensional Knapsack Problem**
  Here, we will delve into the specifics of the MKP, discussing its formulation, variants, and the inherent challenges in solving it.
- **Chapter 3: Genetic Algorithm**
  This chapter will provide an in-depth look at genetic algorithms, their components, operations, and applications, setting the stage for their use in our hybrid approach.
- **Chapter 4: Branch and Cut**
  We will introduce the branch and cut method, explaining its core concepts, algorithmic structure, and how it can be effectively combined with genetic algorithms.
- **Chapter 5: Proposed Approach**
  This chapter will detail the development of our hybrid algorithm, including the integration process, enhancements, and theoretical justifications.
- **General Conclusion**
  Finally, we will summarize our findings, discuss the implications of our research, and suggest directions for future work.

By systematically addressing each of these areas, this thesis aims to contribute to the field of combinatorial optimization by providing a novel and effective solution to the 0/1 Multidimensional Knapsack Problem, thereby advancing both theoretical understanding and practical application.

Chapter 1: Combinatorial Optimization Research

Chapter 1: Combinatorial Optimization Research

1.1 Introduction to Combinatorial Optimization Research

Operations Research (OR) is the discipline of applied mathematics that deals with questions of optimal resource utilization in industry and the public sector. Over the past decade, the scope of OR has expanded to include fields such as economics, finance, marketing, and business planning. More recently, OR has been used for managing healthcare and education systems, solving environmental problems, and in other areas of public interest.

Optimization is a branch of applied mathematics and operations research. An optimization problem is defined as a search problem, which involves exploring a space containing all potential feasible solutions, with the aim of finding the optimal solution or the closest possible to the optimum, minimizing or maximizing an objective function. Decision variables can be either continuous, resulting in a continuous problem, or discrete, resulting in a combinatorial problem.

A combinatorial optimization problem (COP: Combinatorial Optimization Problems) consists of a finite set of solutions, where each solution must satisfy a set of constraints related to the nature of the problem. Each solution is associated with a value, called the objective value, which is evaluated using an objective function. Combinatorial optimization problems can be single-objective, optimizing a single objective function, or multi-objective, optimizing multiple objective functions.

Combinatorial optimization encompasses a broad class of problems with applications in many application domains, including the travelling salesman problem and the knapsack problem.

The knapsack problem models a situation analogous to filling a backpack, which cannot support more than a certain weight, with all or part of a given set of objects, each with a weight and a value. The objects placed in the backpack must maximize the total value without exceeding the maximum weight. This problem will be described in more detail in the first chapter.

Multi-objective optimization is an important area of study in operations research, due to the multi-objective nature of most real-world problems. The earliest work on multi-objective problems was conducted in the 19th century on studies in economics by Francis Y. Edgeworth (1845-1926), and it was more formally used by the Italian economist Vilfredo

Pareto (1848-1923). Multi-objective optimization allows for a proper understanding of the essential characteristics of real-world problems and enhances their perception by decision-makers.

There are several solution methods for multi-objective combinatorial optimization problems. These methods belong to two main families:

- Exact methods (Branch and Bound, Dynamic Programming, cutting plane methods, two-phase method...)

- Approximate methods (simulated annealing, tabu search, genetic algorithms, ant colony algorithms...)

The two-phase method is a general solving framework popularized by Ulungu in 1993, with the central idea of exploiting the specific structure of combinatorial optimization problems for their resolution in a multi-objective context. It has since been applied to a large number of problems, albeit limited to the bi-objective context. As the name suggests, this method is divided into two steps: the first consists of finding all supported solutions of the Pareto front, then the second phase searches among these solutions for unsupported Pareto solutions.

1.1.1 Combinatorial Optimization Problem

A combinatorial problem is any situation where one seeks a solution while respecting the presence of a set of constraints. The solution is the result of combining these constraints in a way that maximizes some and minimizes others. These constraints have a crucial characteristic: each constraint influences the others either when minimizing its value or maximizing it. In other words, we say that the constraints are conflicting.

For example, the following scenario presents a combinatorial problem: one wants to buy a fashionable car while also keeping the price reasonable and within a certain limit. If we maximize the first constraint (a good car), we will have a maximum price; conversely, if we minimize the price, we might end up with a lower-quality car within the limits. In this example, we observe that it's challenging to reconcile these two constraints according to our needs.

1.2 Definition and Main Concepts

1.2.1 Definition

A combinatorial optimization problem can be formally defined as follows: given a finite set of feasible solution S and an objective function $f: S \to R$, find a solution $s \ * \epsilon\ S$ such that

$f(s\ *) \leq f(s)\ \forall\ s \in S$ in a minimization problem, or $f(s\ *)\ \geq\ f\ \forall\ s \in\ S$ in a maximization problem [2]. These problems are often represented by graphs, networks, or sets, and the goal is to find the optimal arrangement or subset that satisfies the given criteria.

1.2.2 Main Concepts in COR

Combinatorial optimization research (COR) encompasses several key concepts that form the basis of understanding and solving these problems. These concepts include:

Feasibility: A solution is feasible if it meets all the problem's constraints. For example, in the knapsack problem, a solution is feasible if the total weight of selected items does not exceed the knapsack's capacity [1].

Optimality: A feasible solution is optimal if it yields the best possible value of the objective function. In the context of the knapsack problem, an optimal solution maximizes the total value of the selected items while adhering to the weight constraint [2].

Search Space: The search space is the set of all possible solutions. In combinatorial optimization, this space is typically very large, making exhaustive search impractical for most problems. Efficient algorithms aim to explore this space effectively to find optimal or near-optimal solutions [3].

Complexity: The computational complexity of a combinatorial optimization problem is a measure of the resources required to solve it, typically in terms of time or space. Many combinatorial optimization problems are NP-hard, indicating that no known polynomial-time algorithms can solve all instances of these problems [4].

Approximation: Due to the complexity of many combinatorial optimization problems, approximation algorithms are often used to find solutions that are close to optimal within a reasonable time frame. These algorithms provide guarantees on the quality of the solution relative to the optimal one [5].

Heuristics and Meta-heuristics: Heuristics are problem-specific strategies designed to find good solutions quickly. Meta-heuristics, on the other hand, are higher-level procedures that guide the search process, often combining multiple heuristics to explore the solution space more effectively. Examples include genetic algorithms, simulated annealing, and tabu search [6].

In summary, combinatorial optimization problems are defined by their finite set of feasible solutions and the objective to find the best solution under given constraints. Key concepts such as feasibility, optimality, search space, complexity, approximation, heuristics, and meta-heuristics are central to understanding and solving these problems. The subsequent sections will delve deeper into specific methods and examples, illustrating how these concepts are applied in practice.

1.3 Solving Combinatorial Optimization Problems

Solving combinatorial optimization problems involves finding the best possible solution from a finite set of feasible solutions. These problems are ubiquitous in various fields, including logistics, finance, engineering, and computer science, where optimal decision-making is crucial. Due to the **NP-hard** nature of many combinatorial optimization problems, a wide range of methods has been developed to tackle them, each with its own advantages and limitations [7].

Heuristic Methods

Heuristic Methods are techniques designed for solving a problem more quickly when classic methods are too slow or for finding an approximate solution when classic methods fail to find any exact solution. This is achieved by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut.

Examples:

- **Simulated Annealing**

  Simulated annealing is a probabilistic technique inspired by the annealing process in metallurgy. It explores the solution space by allowing occasional moves to worse solutions to escape local optima, gradually reducing these moves as the algorithm progresses. This method is particularly useful for large, complex problems where the search space is vast and multimodal [8].

- **Tabu search**
  Tabu search enhances local search methods by maintaining a list of recently visited solutions, known as the tabu list, to avoid cycling back to them. This approach allows the algorithm to explore new regions of the solution space and helps in escaping local optima. Tabu search is effective for various combinatorial optimization problems, including scheduling and routing [9].

Exact Methods

Exact methods guarantee finding the optimal solution by exhaustively exploring the solution space. These methods are typically computationally intensive and are often feasible only for small to moderately sized problems.

Examples:

- **Branch and Bound**
  The method of branch and bound utilizes two concepts: branching or the partitioning of the problem into sub-problems (represented by nodes), and the evaluation of these nodes using relaxation techniques (such as continuous or Lagrangian relaxation). The evaluation of a node involves bounding the solutions and pruning unnecessary nodes, as illustrated in Figure 1.1. It's worth noting that the effectiveness of the branch and bound method largely depends on the branching strategy, which can be breadth-first, depth-first, or best-first. Additionally, the branching method employed and the nature of the evaluation function used strongly influence this method.

- The branch and bound algorithm developed by Horowitz and Sahni considers elements ordered according to the decreasing order of profit-to-weight ratio. Subsequently, the partitioning is performed on the next element. From each node of the tree, and for an element j taken in the predefined order, two branches are developed: the first branch, $xj = 1$, corresponds to putting element j into the knapsack, and the second branch, $xj = 0$, corresponds to excluding element j .

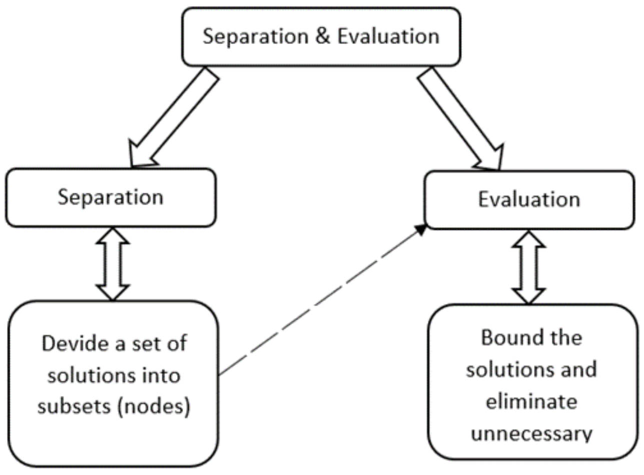

Separation & Evaluation
Separation
Evaluation
Devide a set of solutions into subsets (nodes)
Bound the solutions and eliminate unnecessary

Figure 1.1: Branch and Bound Method Scheme

- **Dynamic Programming (DP)**
  Dynamic programming solves problems by breaking them down into simpler sub problems and solving each sub problem only once, storing the solutions for future reference. This method is particularly effective for problems with overlapping sub problems and optimal substructure, such as the knapsack problem and sequence alignment in bioinformatics [10].

- **Cutting Planes**
  The cutting planes method involves iteratively adding linear constraints to tighten the feasible region of an optimization problem. By refining the problem formulation, this method progressively converges to the optimal integer solution. Cutting planes are commonly used in conjunction with branch and bound to solve integer programming problems more efficiently [11].

Approximate methods

Approximate methods strike a balance between solution quality and computational effort, providing near-optimal solutions within a reasonable time frame. These methods are particularly valuable for large-scale, complex problems.

Example:

- **Local Search**

Local Search is used by many meta-heuristic. It is about making incremental improvements to the current solution through a basic transformation until no improvement is possible. The solution is called local optimum found with respect to the transformation used, as shown in Fig

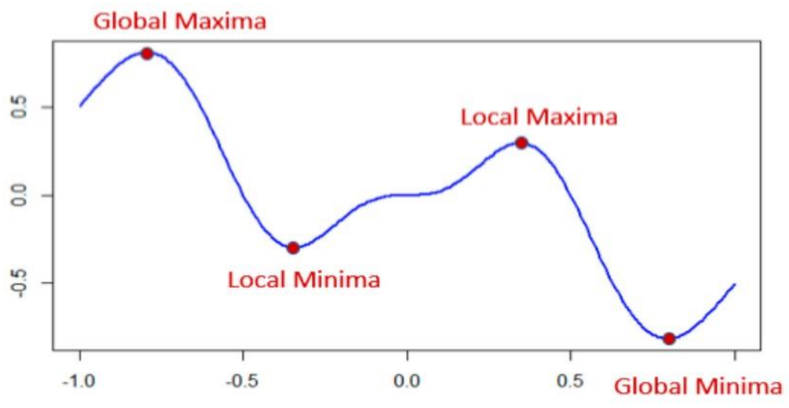


Figure 1.2: Local Search

1.4 Examples of Combinatorial Optimization Problems

Combinatorial optimization problems are pervasive across many disciplines, each presenting unique challenges and requiring specialized algorithms for effective solutions. Here, we explore several classical examples that highlight the diversity and complexity of these problems.

- **Travelling Salesman Problem (TSP)**
  The travelling salesman problem consists of a salesman and a collection of cities. The salesman has to visit each one of the cities starting from travelling salesman wants to minimize the total length of the trip.

- **Vehicle Routing Problem (VRP)**
  The Vehicle Routing Problem (VRP) involves finding the best routes for a fleet of vehicles starting from a depot to visit a set of customers. The evaluation of these routes is based on a predefined objective function, such as travel time, distance, or overall cost to minimize. This problem is a classic extension of the Travelling Salesman Problem, and like the latter, it belongs to the class of NP-Complete optimization problems.

- **Graph Colouring Problem (GCP)**
  Vertex Colouring Problem is a fundamental combinatorial optimization problem. It involves assigning colours to vertices of a given graph so that adjacent vertices have different colours, using the minimum number of colours possible. This problem has been extensively studied over many decades due to its numerous applications in various domains, leading to the definition of several variants. One such variant is the Equitable Colouring Problem (ECP), which aims for a "balanced" vertex colouring where the sizes of any two colour classes differ by at most one.

- **0/1 Knapsack Problem**
  In a knapsack problem with m dimensions, there are n items. Each item is associated with a consumption for each dimension of the knapsack and a corresponding value. The objective is to select a subset of items to maximize the total value while ensuring that the combined consumption of each dimension does not exceed its limit.

1.5. Complexity

In computational complexity theory, decision problems are classified into various complexity classes based on the resources required to solve them. Understanding these classes helps in determining the feasibility and efficiency of algorithmic solutions. Below are brief descriptions of the primary complexity classes and their interrelations

- **Class P**: A decision problem belongs to class P if there exists an algorithm capable of solving it in polynomial time.
- **Class NP**: The class NP contains all decision problems for which we can associate a set of potential solutions (of at worst exponential cardinality) such that we can verify in polynomial time if a potential solution satisfies the posed question. Clearly, $P \subset NP$, which is commonly accepted. Moreover, this inclusion is strict, meaning $P \neq NP$.
- **Co-NP Class:** Co-NP stands for the complement of NP Class. It means if the answer to a problem in Co-NP is No, then there is proof that can be checked in polynomial time.
- **Class NP-complete**: Within the NP class, we also distinguish the class of NP-complete problems. NP-completeness is based on the notion of polynomial reduction.
- **Class NP-hard**: A problem is NP-hard if knowing how to solve it in polynomial time would imply that we can solve an NP-complete problem in polynomial time. NP-hard problems are therefore, in a sense, harder than NP-complete problems. This schema represents the relationship between problem classes.

This framework provides a foundation for understanding the complexity of combinatorial optimization problems and their classification based on computational difficulty.

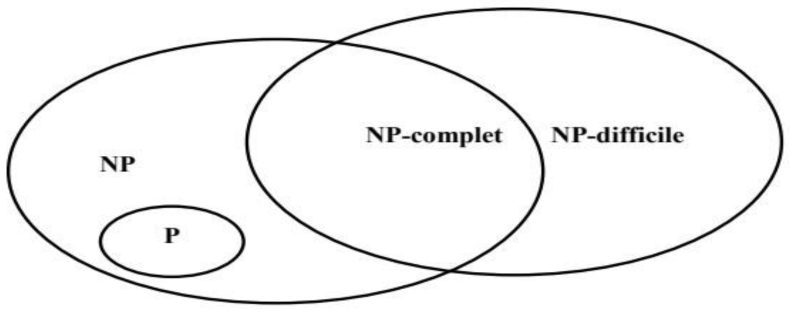


Figure 1.3: Relationship between problem classes.

Chapter 2: Multidimensional Knapsack Problem

Chapter 2: Multidimensional Knapsack Problem

2.1 Introduction

The Multidimensional Knapsack Problem (MKP) is a well-known and extensively studied problem in the field of combinatorial optimization. It is an extension of the classic 0/1 knapsack problem, which itself is a fundamental problem with numerous practical applications. The classical knapsack problem involves selecting a subset of items, each with a given weight and value, to maximize the total value without exceeding the capacity of the knapsack. In the MKP, this concept is extended to multiple constraints, making the problem significantly more complex and challenging to solve [12].

The origins of the knapsack problem can be traced back to early studies in resource allocation and financial decision-making. Its simplicity in formulation, yet complexity in solving, has made it a popular subject for researchers in operations research, computer science, and applied mathematics. The knapsack problem is often used as a benchmark for testing new algorithms and optimization techniques [13].

In a typical MKP, each item has multiple attributes, such as weight, volume, and other resource consumptions, each contributing to different constraints. The objective is to select a combination of items that maximizes the total profit while satisfying all the given constraints. This multidimensional aspect introduces a layer of complexity that makes exact solutions computationally expensive for large instances [14].

The MKP has practical significance in various fields. For example, in logistics, companies must decide the optimal set of goods to transport within limited cargo space and weight constraints. In finance, portfolio optimization involves selecting a mix of investments that maximizes returns while adhering to budget and risk constraints. In manufacturing, resource allocation problems require distributing limited resources across multiple projects to maximize efficiency and output [15].

Solving the MKP involves a blend of exact and heuristic methods. Exact methods, such as branch and bound and dynamic programming, can find the optimal solution but are often impractical for large problems due to their computational requirements. Heuristic and metaheuristic approaches, like genetic algorithms, simulated annealing, and particle swarm optimization, provide feasible solutions within a reasonable time frame, though they do not guarantee optimality [16].

The significance of the MKP extends beyond its practical applications; it also serves as a critical testbed for developing and evaluating new optimization algorithms. The complexity of the problem provides a robust challenge for algorithm designers, pushing the boundaries of what can be achieved in combinatorial optimization [17].

In this chapter, we will delve deeper into the significance of the MKP, explore various knapsack problem variations, present the integer linear programming (ILP) formulation of the MKP, and review case studies that illustrate its application in real-world scenarios.

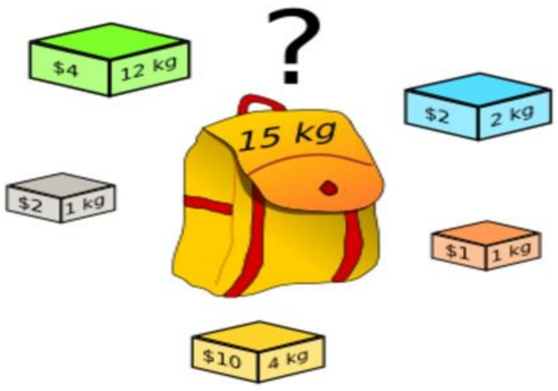


Figure 2.1: Knapsack Problem.

2.2 Knapsack Problem Variations

The knapsack problem is a classic example of a combinatorial optimization problem with numerous variations, each adding different layers of complexity and applicability. Understanding these variations helps in grasping the multifaceted nature of resource allocation problems in various fields.

2.2.1 0/1 Knapsack Problem

The knapsack problem is a variant of the optimization problems.

There is several practical applications for knapsack problem like The knapsack problem (KP) can be formally defined as follows: Given an instance of the knapsack problem with item set *N*, consisting of *n* items, *N* := {1, ..., *n*} j with profit p_j and weight w_j , and the capacity value c, (Usually, all those values are taken from the positives integer numbers.)

Then the objective is to select a subset of *N* such that the total profit of the selected items is maximized and the weights does not exceed c. Alternatively, a knapsack can be formulated as a solution of the following linear integer programming formulation:

$$\text{(KP)} \quad \text{maximize} \sum_{j=1}^{n} pjxj \qquad (1)\ 1$$

$$\text{subject} \sum_{j=1}^{n} wjxj \ \leq \ c, \qquad (1)2$$

$$x_j \in \{0, 1\}\ , j = 1, \ldots, n. \qquad (1)3$$

2.2.2 Fractional Knapsack Problem

The fractional knapsack problem allows the inclusion of fractions of items rather than the whole item, making it a continuous problem rather than a discrete one. This relaxation transforms the problem into a solvable one using a greedy algorithm that sorts items based on their value-to-weight ratio and adds them to the knapsack in decreasing order until the capacity is reached [18].

2.2.3 Multi-Choice Knapsack Problem

In the multi-choice knapsack problem, items are grouped into classes, and the objective is to select one item from each class to maximize the total value while staying within the knapsack's capacity [19]. This variation is significant in scenarios where decisions are interdependent, such as selecting courses for a curriculum or components for a product design.

2.2.4 Multiple Knapsack Problem

The multiple knapsack problem extends the classic problem by introducing several knapsacks instead of one, each with its own capacity constraint. The objective is to maximize the total value packed into all knapsacks [20]. This variation is applicable in logistics and resource distribution scenarios where multiple containers or storage units are available.

2.2.5 Multidimensional Knapsack Problem (MKP)

Given a knapsack with M-dimensions, let c_j be the capacity of the i^{th} dimension,

i = 1, 2, ...,m. There are n items. The j^{th} item requires w_{ij} units of the i^{th} dimension of the knapsack j, j = 1, 2, ..., n. The reward of including the item j in the knapsack is p_j . The problem can formulated

as follows:

$$\max Z = \sum_{j=1}^{n} p_j x_j \qquad (2)1$$

$$\text{subject to} \sum_{j=1}^{n} w_{ij} x_j < c_{j,} \quad i = 1,2,..,m, \qquad (2)2$$

The equations (2,1) and (2,2) represent the objective and the constraints

respectively.

Such:

•x_j is a binary value. Such x_j = 1 if we put the item j into the

knapsack and x_j = 0 if we don't.

• j refers to the item, $j \leq n$.

2.2.6 Other Variations

Other notable variations include:

- **Knapsack Problem with Conflict Graphs**: Items are represented as vertices in a graph, with edges indicating conflicts. No two conflicting items can be included in the knapsack simultaneously [21].
- **Subset-Sum Problem**: A special case of the knapsack problem where the value of each item equals its weight, simplifying the problem to finding a subset that sums to a specific value [22].

In summary, these variations of the knapsack problem illustrate its versatility and the wide range of applications it can model. Each variation introduces unique constraints and complexities, necessitating different solution approaches and providing rich ground for research and innovation.

2.3 Multidimensional Knapsack Problem ILP

The Multidimensional Knapsack Problem (MKP) extends the classical knapsack problem by incorporating multiple constraints, making it significantly more challenging to solve. The Integer Linear Programming (ILP) formulation of the MKP provides a mathematical framework for understanding and addressing this complexity. In this section, we will present the ILP formulation, discuss its components, and outline the challenges associated with solving it.

Problem Formulation

The objective of the MKP is to select a subset of items to maximize the total value while satisfying multiple resource constraints. Formally, the MKP can be defined as follows:

$$\text{maximize} \sum_{i=1}^{n} v_i x_i \qquad (3)1$$

$$\text{subject to} \sum_{i=1}^{n} w_{ij} x_i \ \leq \ w_j \forall_j \ 1,2,..,m \qquad (3)2$$

$$x_i \ \in \{0,1\} \qquad (3)3$$

where:

- v_i is the value of item i,
- w_{ij} is the weight of item iii in dimension j,
- w_j is the capacity of dimension j,
- x_i is a binary variable indicating whether item i is included in the knapsack (1) or not (0) [23].

This formulation captures the essence of the MKP, where the goal is to maximize the total value of the selected items without violating any of the multiple constraints.

Components of the ILP Formulation

1. **Objective Function**: The objective function :

$$\sum_{i=1}^{n} v_i x_i \qquad (4)1$$

 aims to maximize the total value of the items included in the knapsack. This linear function represents the primary goal of the optimization problem [24].

2. **Constraints**: The constraints :

$$\sum_{i=1}^{n} w_{ij} x_i \ \leq \ w_j \forall_j \ = 1,2,..,m \qquad (5)1$$

ensure that the total weight of the selected items does not exceed the capacity in any dimension. Each constraint corresponds to a different resource or capacity limit, adding to the problem's complexity [25].

3. **Binary Decision Variables**: The binary variables $x_i \ \in \{0,1\}$ indicate whether each item is selected. These variables enforce the combinatorial nature of the problem, as each item can only be either included or excluded [26].

Challenges in Solving the MKP ILP

The MKP ILP is a complex problem due to several factors:

1. **High Dimensionality**: The presence of multiple constraints increases the dimensionality of the problem, making it harder to find feasible solutions that satisfy all constraints simultaneously [27].
2. **Combinatorial Explosion**: The number of possible combinations of items grows exponentially with the number of items, leading to a combinatorial explosion that makes exhaustive search impractical for large instances [28].
3. **NP-Hard Nature**: The MKP is NP-hard, meaning that there is no known polynomial-time algorithm to solve all instances of the problem exactly. This inherent difficulty necessitates the use of heuristic and meta-heuristic approaches for larger instances [29].

Solution Techniques

Various methods have been developed to solve the MKP ILP, including:

- **Exact Methods**: Techniques such as branch and bound, cutting planes, and branch and cut provide exact solutions but are often computationally expensive and impractical for large problems [30].
- **Heuristic Methods**: Heuristics like greedy algorithms offer quick, though not necessarily optimal, solutions. These methods are useful for obtaining good solutions in a reasonable time frame [31].
- **Meta-heuristic Methods**: Meta-heuristics such as genetic algorithms, simulated annealing, and particle swarm optimization balance exploration and exploitation to find high-quality solutions efficiently. These methods are particularly effective for large, complex instances [32].

In summary, the ILP formulation of the MKP provides a structured way to model and solve this challenging problem. While exact methods can offer precise solutions, heuristic and meta-heuristic approaches are often more practical for large-scale applications. Understanding the components and challenges of the ILP formulation is crucial for developing effective solution strategies.

2.4 Case Studies

Case studies provide practical insights into the application of the Multidimensional Knapsack Problem (MKP) in real-world scenarios. These studies illustrate how the theoretical concepts and solution techniques discussed earlier are implemented in various industries, highlighting the problem's significance and the effectiveness of different approaches.

Logistics and Transportation

In the logistics industry, companies face the challenge of optimizing the loading of goods into containers or vehicles. This problem often involves multiple constraints such as weight, volume, and delivery deadlines, making it an ideal application for the MKP.

Example: A global shipping company needs to load containers with goods of varying weights and volumes while ensuring that the total weight and volume do not exceed the container's capacity. Additionally, some goods have specific delivery deadlines that must be met.

Solution Approach: The company employed a genetic algorithm combined with branch and bound to solve the MKP. The genetic algorithm generated initial solutions, which were then refined using branch and bound to ensure that the constraints were strictly adhered to.

Results: The hybrid approach significantly improved the loading efficiency, reducing the number of containers needed and ensuring timely deliveries. The company reported a 15% reduction in transportation costs and a 10% increase in customer satisfaction due to timely deliveries [33].

Financial Portfolio Optimization

In finance, portfolio optimization involves selecting a mix of investments that maximize returns while adhering to budget and risk constraints. The MKP framework is well-suited for modelling these complex decisions.

Example: An investment firm aims to construct a portfolio from a set of potential investments, each with a different return, risk level, and cost. The goal is to maximize the total return without exceeding a specified budget and while maintaining a balanced risk profile.

Solution Approach: The firm used a particle swarm optimization (PSO) algorithm to solve the MKP. The PSO algorithm efficiently explored the solution space, balancing the trade-off between risk and return to identify optimal investment combinations.

Results: The optimized portfolio achieved a higher return compared to the firm's previous strategies, with a 12% increase in annual returns and a 5% reduction in overall portfolio risk. This success demonstrated the MKP's applicability in financial decision-making [34].

Manufacturing and Production

In manufacturing, resource allocation problems often involve distributing limited resources such as materials, labor, and machinery across multiple production processes to maximize efficiency and output.

Example: A manufacturing company needs to allocate raw materials and labor to various production lines, each producing different products with specific profit margins, while ensuring that the resource limits are not exceeded.

Solution Approach: The company implemented a simulated annealing algorithm to solve the MKP. The algorithm iteratively adjusted the allocation of resources, gradually improving the solution by exploring different configurations and avoiding local optima.

Results: The application of simulated annealing resulted in a 20% increase in production efficiency and a 25% rise in overall profit margins. The optimized resource allocation also reduced waste and improved the utilization of labor and materials [35].

Telecommunications and Network Design

In telecommunications, network design problems often require optimizing the allocation of bandwidth and other resources across multiple channels and users to improve network performance and reliability.

Example: A Telecom company needs to allocate bandwidth to various services and users in a way that maximizes overall network throughput while ensuring fair distribution and adherence to quality of service (QoS) constraints.

Solution Approach: The company used a Tabu search algorithm to address the MKP. The tabu search algorithm maintained a list of recently explored solutions to avoid revisiting them, thus effectively navigating the complex solution space.

Results: The optimized bandwidth allocation led to a 30% improvement in network throughput and a 15% enhancement in QoS. This resulted in fewer service interruptions and higher customer satisfaction [36].

Summary of Case Studies

These case studies demonstrate the versatility and effectiveness of the MKP in solving complex, real-world problems across various industries. By employing advanced algorithms such as genetic algorithms, particle swarm optimization, simulated annealing, and tabu search, companies can achieve significant improvements in efficiency, cost savings, and overall performance. The practical applications of the MKP underline its importance as a critical tool in combinatorial optimization.

Chapter 3: Genetic Algorithm

Chapter 3: Genetic Algorithm

3.1 Overview of the Genetic Algorithm

The Genetic Algorithm (GA) is an evolutionary algorithm classified among metaheuristics inspired by the theory of natural evolution. It was first introduced by John Holland in 1975. GAs are search heuristics that mimic the process of natural selection, employing mechanisms such as selection, crossover, and mutation to evolve solutions to optimization and search problems.

3.2 Genetic Algorithm

3.2.1 Core Principles of Genetic Algorithms

To apply a Genetic Algorithm to a specific problem, the following five components are essential:

- **Solution Encoding (Individuals)**

Each point in the state space is associated with a data structure, usually following a mathematical modelling phase of the problem. Effective encoding is crucial for the algorithm's success.

- **Method for Generating the Initial Population**

The initial population of individuals, which serves as the foundation for future generations, must be heterogeneous.

- **Optimization Function**

Also known as the individual evaluation function, this function returns a real value called fitness, determining the selection probability of an individual.

- **Genetic Operators**

These operators diversify the population across generations and explore the state space. Crossover recombines the genes of individuals, mutation introduces minor alterations to avoid rapid convergence, and selection favors the best individuals.

- **Sizing Parameters**

This includes the population size, total number of generations or stop criterion, and the application probabilities of crossover and mutation operators.**[37]**

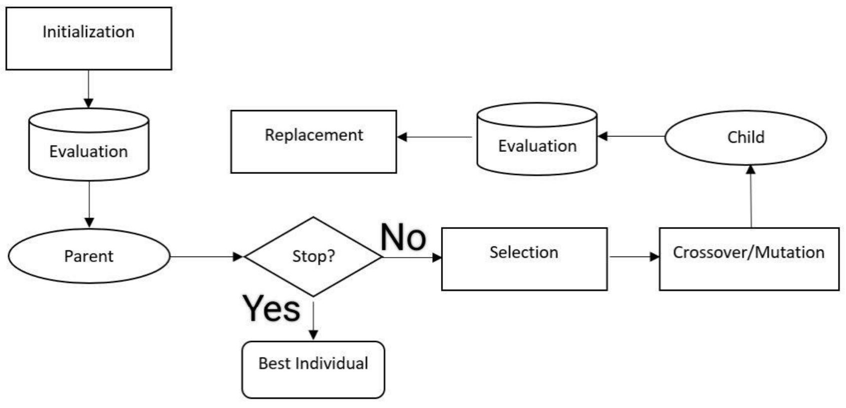


Figure 3.1: The Main Steps of a Genetic Algorithm

3.2.2 Solution Encoding

Genetic Algorithms operate on a population of individuals, each encoded by a chromosome or genotype. A population is thus represented by a set of chromosomes. Encoding involves finding a suitable data structure for individuals. Various encoding types are:

- **Binary Encoding**

This elementary encoding represents solutions as a bit string, where each bit can be 0 or 1, making it widely used due to its advantages in gene manipulation.

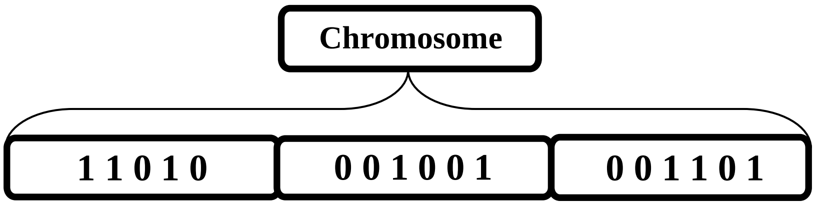


Figure 3.2: Binary-Coded Chromosome.

- **Real-Value Encoding**

While binary encoding is useful, it becomes cumbersome with large parameter numbers. Real numbers can be used to encode genes, simplifying management of chromosomes.**[38]**

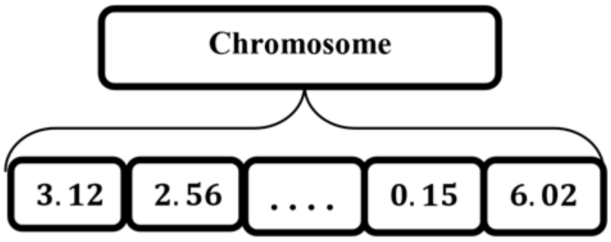


Figure 3.3: Real-Value Chromosome.

- **Multi-Character Encoding**

In contrast to binary and real-value encoding, multi-character encoding is often more natural and used in complex genetic algorithm applications.**[39]**

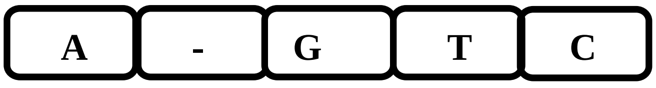


Figure 3.4: Multiple Character Coding of a Chromosome.

- **Tree Encoding**

This uses a tree structure where each node can have multiple branches, suitable for problems without a finite solution size. Trees can grow to any size through crossover and mutations

.

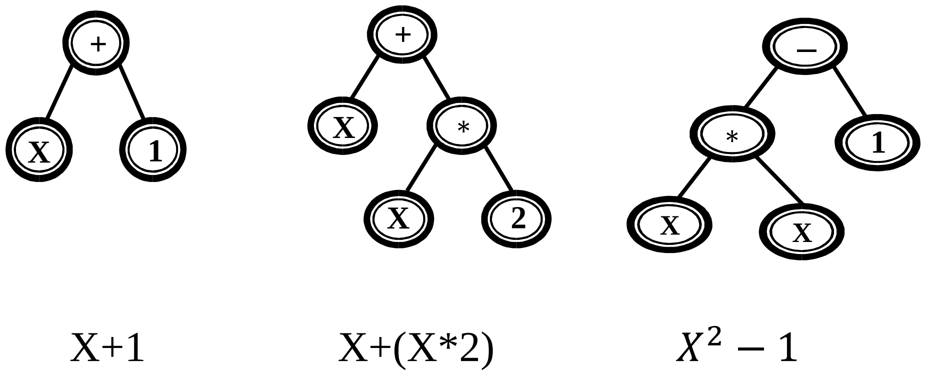


Figure 3.5: Tree coding: Three simple program trees of the sort normally used in Genetic Programming.

3.2.3 Initial Population Creation

The choice of the initial population strongly influences the algorithm's convergence speed. If the optimal position in the state space is unknown, individuals should be generated randomly using a uniform distribution. If prior information is available, individuals should be generated in a specific sub-domain to accelerate convergence. The population size must balance computation time and result quality; too large a population increases computational cost and memory requirements, while too small a population may lead to local optimal.**[40]**

3.2.4 Evaluation Function (Fitness)

The evaluation operator is crucial as it helps the selection operator choose which individuals to retain. It measures each individual's performance, corresponding to a given solution to the problem. The evaluation function quantifies an individual’s survival capability by assigning a fitness weight. The strength of each chromosome is calculated, with the fittest being retained for modification (crossover and mutation). The evaluation function's complexity depends on the problem and its constraints. Encoding and evaluation are the only problem-specific elements; once set, the Genetic Algorithm remains consistent.**[41]**

3.2.5 Operators

- **Selection Operator**

Responsible for favouring the best individuals from the current population to create a new population. Methods include:

- **Biased Lottery or Roulette Wheel**

This creates a biased lottery wheel where each individual occupies a section proportional to its fitness, giving even weaker individuals a survival chance. The probability p(x_i) for individual x_i to be selected is:

$$p(x_i) = \frac{f(x_i)}{\sum_{k=1}^{n} f(x_k)} \qquad (6)1$$

For minimization problems, the selection probability p'(x_i) for individual x_i is:

$$p'(x_i) = \frac{1 - p(x_i)}{N - 1} \qquad (6)2$$

- **Tournament Selection**

This method involves choosing a sub-population of fixed size M, either predetermined or randomly. The best individual in this sub-population is selected for crossover and mutation, allowing lower-quality individuals a chance to contribute to population improvement.**[42]**

- **Rank Selection**

Rank selection first sorts the population by fitness. Selection is similar to roulette but uses ranks instead of evaluation values.**[42]**

- **Elitism**

Elitism retains a number of the best chromosomes each generation to prevent their loss through mutation, crossover, or selection, improving Genetic Algorithms by preserving top solutions.**[42]**

- **Crossover Operator**

New individuals are created by randomly taking gene segments from each parent. This process explores possible solution spaces. After selection, individuals are paired randomly, and parent chromosomes are copied and recombined to produce two offspring with characteristics from both parents. The number of crossover points and crossover probability determine whether parents are crossed or simply copied to the next population. Types include:

- **N-Point Crossover**

This involves choosing n crossover points, then exchanging gene fragments. Common types are 1-point and 2-point crossover.**[43]**

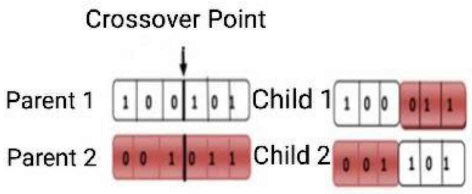


Figure 3.4 : Example of a Single-Point Crossover.

- **Uniform Crossover**

This probabilistic crossover exchanges each gene between parents with a probability of 0.5.**[44]**

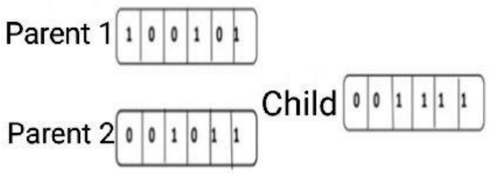


Figure 3.5: Example of a Uniform Crossover.

- **Mutation Operator**

Mutation randomly changes one or more genes of a chromosome with a low probability p, maintaining diversity and avoiding local optima. Common mutation types are **[45]**.

- **1-Point Mutation**

Randomly changes a single gene value.

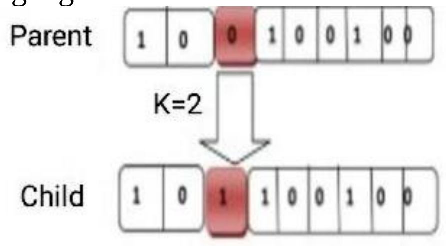


Figure 3.6: Example of a Single-Point Mutation

- **2-Point Mutation**

Alters multiple values on the chromosome.**[45]**

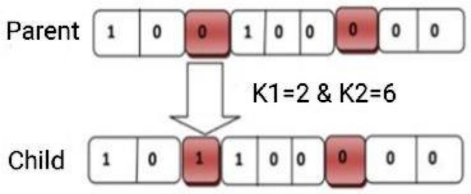


Figure3.7 : Example of a Two-Points Mutation

- **Replacement**

Newly created individuals after crossover and mutation are reintroduced into the population based on their fitness. Methods include:

- **Stationary Replacement**

Automatically replaces parents with offspring regardless of performance.

- **Elitist Replacement**

Considers performance, keeping at least the most performant individual when transitioning generations.

- **Algorithm Convergence**

Initial improvement is rapid (global search) and slows over time (local search). Noise in the average performance is mainly due to mutations**.**

- **Stopping Criteria**

The generation-replacement cycle repeats until a stop criterion is met, such as a fixed number of iterations, maximum computation time, or a satisfactory solution. The evolutionary algorithm then returns the best solution(s) identified across generations.**[46]**

Conclusion

This chapter introduced the basic concepts and principles of Genetic Algorithms. The evaluation of solutions using the objective function ranks individuals by performance, and selection methods imitate natural reproduction processes. Based on the genetic information of selected pairs of chromosomes (individuals), the Genetic Algorithm generates new offspring, evolving the population over generations.

Chapter 4:Branch and Cut

Chapter 4: Branch and Cut

4.1. Introduction to Branch and Cut
Our implementation is partly based on the work of Jean Maurice Clochard and Denis Naddef [CN93], as well as on the work of Padberg and Rinaldi [PR91]. The "Branch and Cut" method we present is a combination of the polyhedral cuts method, the "Branch and Bound" method, and the column generation method. Like any implicit enumeration method, the algorithm constructs a tree called the "Branch and Cut" tree. The subproblems that form the tree are called nodes. There are three types of nodes in the "Branch and Cut" tree: the current node, which is being processed; the active nodes, which are in the waiting list of problems; and the inactive nodes, which have been pruned during the algorithm's execution.

Core Concepts:
The Branch and Cut (B&C) algorithm is a sophisticated method that combines the principles of branch and bound with cutting planes to solve integer linear programming (ILP) problems efficiently. Understanding the core concepts of B&C is crucial for grasping how this method systematically explores the solution space and refines the feasible region to find optimal solutions.
4.2. Problem Formulation

The starting point for applying B&C is the formulation of the integer linear programming problem. An ILP can be expressed as follows:

$$\text{maximize } c^T x \qquad (7)1$$

$$\text{subject to } Ax \leq b \qquad (7)2$$

$$x \in Z^n \qquad (7)3$$

where:

x is the vector of decision variables.
c is the vector of coefficients for the objective function.
A is the matrix of coefficients for the constraints.
b is the vector of right-hand side values for the constraints.
Z^n indicates that the variables x are integers [47].

4.3 Branching:

Branching is the process of recursively dividing the problem into smaller sub-problems. This is done by selecting a variable that does not satisfy the integrality constraint and creating two new sub-problems by fixing this variable to its lower and upper integer bounds.

Selection of Branching Variable: The choice of the variable to branch on can significantly affect the efficiency of the B&C algorithm. Common strategies include:

Largest Fractional Part: This strategy involves branching on the variable with the largest fractional part in the current LP solution. The rationale is that the variable furthest from an integer value is most likely to impact the solution space significantly.

Example: Suppose the current LP solution yields x1=1.5, x2=2.3 and x3=0.7. The variable x1 has the largest fractional part (0.5), so it would be selected for branching.

■ **Strong Branching:** This involves evaluating the potential impact of branching on several candidate variables before making a decision. For each candidate variable, two branches are created, and the LP relaxations of these branches are solved. The variable that shows the most promising potential in reducing the objective value is chosen for branching.

◆ **Example:** Consider three candidate variables x1, x2, and x3. For each variable, two sub-problems are created and solved:

■ For x_1: Branch $x_1 \leq 1$ and $x_1 \geq 2$

■ For x_2: Branch $x_2 \leq 2$ and $x_2 \geq 3$

■ For x_3: Branch $x_3 \leq 0$ and $x_3 \geq 1$

The variable with the highest reduction in the objective function is selected for branching.[48]

Creation of Sub-problems

Each selected variable leads to two new sub-problems by fixing the variable to its nearest integer values. This binary decision splits the feasible region into two smaller regions, each representing a node in the search tree[49].

Binary Decision: For a variable xix_ixi with a fractional value, two branches are created:

Lower Branch: Add the constraint x_i $[x_i]$.
Upper Branch: Add the constraint x_i $[x_i]$.
Example: If the variable x_1=1.5, the two sub-problems created would be:
Lower Branch: $x_i \leq 1$
Upper Branch: $x_i \geq 2$

Tree Structure: This process continues recursively, creating a tree structure where each node corresponds to a sub-problem with additional constraints. The depth and breadth of this tree depend on the complexity of the problem and the branching decisions.

4.4 Bounding and Pruning

Bounding and pruning are crucial for maintaining the efficiency of the B&C algorithm by reducing the number of nodes that need to be explored.

Bounding: Each node in the tree is associated with a bound on the objective function. If the bound of a node is worse than the current best (incumbent) solution, that node can be pruned.

Example: If the incumbent solution has an objective value of 100 and a node's LP relaxation yields a bound of 110, this node is pruned as it cannot lead to a better solution.

Feasibility Check: Sub-problems that are infeasible are pruned immediately, as they cannot contribute to the optimal solution.

Example: If adding the constraint x12 results in an infeasible LP problem, this branch is pruned.

Incumbent Update: If a sub-problem yields a better feasible solution, the incumbent solution is updated, and sub-problems with worse bounds are pruned accordingly.

Example: If a sub-problem results in a feasible solution with an objective value of 95, the incumbent is updated to this value, and nodes with bounds greater than 95 are pruned.

4.5 Cutting planes:

Cutting planes are linear inequalities added to the LP relaxation of the problem to exclude infeasible integer solutions without eliminating any feasible integer solutions. These cuts tighten the feasible region, improving the bounds and reducing the number of nodes that need to be explored.

Gomory Cuts: Gomory Cuts, named after Ralph Gomory, are a type of cutting plane used in integer programming to eliminate fractional solutions from the feasible region of the linear relaxation of a mixed-integer programming (MIP) problem.

Fractional Solutions: When solving the linear relaxation of an MIP, the solution may include fractional values for variables that should be integer. Gomory cuts are designed to cut off these fractional solutions without removing any integer feasible solutions.

Generation: Gomory cuts are derived from the simplex tableau of the linear relaxation. For each basic variable that is fractional, a cut is generated by considering the fractional parts of the coefficients in the tableau row corresponding to that variable.

Mathematical Form: If a variable xix_ixi has a fractional value f (where 0 < f < 1), the Gomory cut can be expressed as:

$$\sum_{j \in J} \{a_{ij}\} x_j \geq f \qquad (8)1$$

Here, $\{a_{ij}\}$ represents the fractional part of the coefficient a_{ij}.

Example: Consider the fractional solution x =1.5, y = 2.3. A Gomory cut might be:

0.5x+0.3y0.8

This cut removes the current fractional solution from the feasible region.

Mixed Integer Rounding Cuts (MIR Cuts)

Mixed Integer Rounding (MIR) Cuts are another type of cutting plane used in MIP problems to improve the linear relaxation. They generalize the concept of rounding cuts to mixed-integer problems, where some variables are integer and others are continuous.

Construction: MIR cuts are constructed by taking a linear combination of the original constraints to form a new inequality. This new inequality is then rounded to create a valid cut.
Application: MIR cuts are particularly useful for problems with both integer and continuous variables, providing a way to tighten the linear relaxation without excluding feasible integer solutions.
Mathematical Form: If a constraint is of the form:

$$\sum_{j\in I} a_j x_j + \sum_{j\in C} b_j y_j \le b \qquad (9)1$$

where x_j are integer variables and y_j are continuous variables, the MIR cut can be derived by rounding the coefficients appropriately.

Example: For the constraint 3x + 2.5y ≤ 7.5, where x is integer and y is continuous, an MIR cut could be:

3x + 2y ≤ 7

This cut is formed by appropriately rounding the coefficients and the right-hand side.

Cover Cuts and Zero-Half Cuts

Cover Cuts:

Cover Sets: In the context of 0-1 integer programming, a cover is a subset of variables that, if all are set to 1, would violate a constraint.
Construction: Cover cuts are derived from such cover sets to ensure that not all variables in the cover can be 1 simultaneously. These cuts are used to prevent infeasible solutions.
Mathematical Form: Given a constraint

$$\sum_{j\in C} a_j b_j \le b \qquad (10)1$$

and a cover C⊆S such that

$$\sum_{j\in C} a_j > b \qquad (10)2$$

a cover cut can be:

$$\sum_{j \in C} x_j \leq |C| - 1 \qquad (10)3$$

This cut ensures that not all variables in the cover can be 1 at the same time.

Example: For the constraint $x_1 + x_2 + x_3 \leq 2$, the set $\{x_1, x_2, x_3\}$ forms a cover. The cover cut is:

$x_1 + x_2 + x_3 \leq 2$

If $\{x_1, x_2, x_3\}$ were to form a cover with a sum greater than 2, a more restrictive cut like $\{x_1 + x_2 \leq 1$ might be applied.

Zero-Half Cuts:

Definition: Zero-half cuts are derived from constraints where the coefficients of the variables are either 0 or 1. These cuts leverage the fact that certain combinations of variables must sum to either 0 or 1.
Construction: Zero-half cuts are typically constructed from multiple constraints, identifying combinations of variables that must satisfy specific parity conditions.
Mathematical Form: These cuts are often expressed as parity constraints, ensuring that certain sums of variables are restricted to specific values.
Example: Consider the constraints $x_1 + x_2 \leq 1$ and $x_1 + x_3 \leq 1$. A zero-half cut might combine these to form:

$x_1 + x_2 + x_3 \leq 2$

This cut restricts the combinations of $x_1, x_2 \ and \ x_3$ to ensure feasibility.

Lift and Project Cuts

Lift and Project Cuts are a type of cutting plane that strengthens the LP relaxation by projecting the feasible region into a higher-dimensional space and then lifting it back to the original space with tighter bounds.

Concept: The idea is to take a linear combination of the original constraints and then project them into a higher-dimensional space where the feasible region can be more easily separated from the infeasible region.
Process: This involves solving a series of linear programming problems to generate the cuts. The method iteratively improves the LP relaxation by tightening the feasible region.
Mathematical Form: The cut is generated by lifting the original constraints into a higher-dimensional space and then projecting them back to form a tighter constraint.
Example: Given the constraint $x_1 + x_2 \leq 1$ and the integrality constraints $x_1, x_2 \in \{0,1\}$, the lift-and-project procedure might yield a cut such as:

$$0.5x_1 + 0.5\,x_2 \;\leq 0.5$$

This cut effectively tightens the feasible region by excluding certain fractional solutions.

- **Cover Cuts**: These cuts are based on the concept of cover inequalities, which are derived from subsets of constraints that are not satisfied by the current LP solution [51].
- **Clique Cuts**: These are used in problems involving binary variables and are based on finding cliques in a conflict graph that represents mutually exclusive decisions [52].

4.6 Algorithm Structure

The Branch and Cut (B&C) algorithm combines the systematic exploration of branch and bound with the refinement capabilities of cutting planes to solve integer linear programming (ILP) problems efficiently. The structured approach of B&C ensures a thorough search of the solution space while progressively tightening the feasible region. This section outlines the key stages of the B&C algorithm, including initialization, branching strategy, cutting plane generation, and pruning.

4.6.1 Initialization

The initialization phase sets the stage for the B&C algorithm by solving the initial linear programming (LP) relaxation of the ILP problem. This involves:

- **Solving the LP Relaxation**: The constraints are relaxed by allowing the integer variables to take continuous values, thus solving a linear programming problem. This provides a bound for the objective function and an initial feasible solution if one exists [53].
- **Initial Feasible Solution**: If an integer solution is found during this phase, it serves as an initial incumbent solution. Otherwise, heuristics may be employed to generate a feasible starting point [54].

4.6.2 Branching Strategy

Branching is the process of dividing the problem into smaller sub-problems, creating a search tree where each node represents a sub-problem with additional constraints. The key elements of the branching strategy include:

- **Selection of Branching Variable**: The variable chosen for branching can significantly impact the efficiency of the search. Common strategies include:
    - Branching on the variable with the largest fractional part in the current LP solution.
    - Using strong branching to evaluate the potential impact of branching on several candidate variables before making a decision [55].

- **Creation of Sub-problems**: Each selected variable leads to two new sub-problems by fixing the variable to its nearest integer values. This binary decision splits the feasible region into two smaller regions, each representing a node in the search tree [56].

  Example: If the variable x_1=1.5, the two sub-problems created would be:
  Lower Branch: $x_1 \leq 1$
  Upper Branch: $x_1 \geq 2$

4.6.3 Cutting Plane Generation

Cutting planes are linear inequalities added to the LP relaxation to exclude infeasible integer solutions while retaining all feasible integer solutions. This step involves:

- **Identifying Cutting Planes**: Various methods can generate effective cuts, including Gomory cuts, cover cuts, and clique cuts. These cuts are derived from the current LP solution and the problem constraints [57].
- **Adding Cuts to the LP**: Once identified, cutting planes are added to the LP relaxation, tightening the feasible region and improving the bounds. This process is iterative, with multiple cuts potentially added at each node [58].
- **Re-solving the LP**: After adding the cuts, the modified LP relaxation is re-solved to obtain a new bound and potentially a new feasible solution. This iterative process continues until no further effective cuts can be identified [59].

4.6.4 Pruning

Pruning reduces the search space by eliminating sub-problems that cannot yield better solutions than the current best (incumbent) solution. Pruning strategies include:

- **Bounding**: If the bound of a sub-problem is worse than the incumbent solution, the sub-problem is pruned as it cannot lead to an improved solution [60].

  Example: If the incumbent solution has an objective value of 100 and a node's LP relaxation yields a bound of 110, this node is pruned as it cannot lead to a better solution.

- **Feasibility Check**: Sub-problems that are infeasible are pruned immediately, as they cannot contribute to the optimal solution [61].

  Example: If adding the constraint $x_1 \geq 2$ results in an infeasible LP problem, this branch is pruned.

- **Incumbent Update**: If a sub-problem yields a better feasible solution, the incumbent solution is updated, and sub-problems with worse bounds are pruned accordingly [62].

Example: If a sub-problem results in a feasible solution with an objective value of 95, the incumbent is updated to this value, and nodes with bounds greater than 95 are pruned.

Integration of Heuristics

Heuristics play a crucial role in enhancing the B&C algorithm's performance by providing good initial solutions and guiding the search process. They can be integrated at various stages:

- **Initial Solution Heuristics**: Used during initialization to find a feasible starting point, improving the overall efficiency of the algorithm [63].
- **Branching Heuristics**: Help in selecting the most promising branching variables, thus reducing the number of sub-problems generated [64].
- **Cutting Plane Heuristics**: Assist in identifying effective cuts, ensuring that the LP relaxation is efficiently tightened [65].

Conclusion

the structure of the Branch and Cut algorithm involves a systematic process of initialization, branching, cutting plane generation, and pruning. Each stage is designed to explore and refine the solution space efficiently, ensuring that high-quality solutions are found for complex ILP problems.

Chapter 5: Proposed Approach

Chapter 5: Proposed Approach

5.1 Introduction:

In this section, we propose a hybrid approach that combines the strengths of Genetic Algorithms (GA) and the Branch-and-Cut method using CPLEX. By leveraging the heuristic power of GA and the exact optimization capabilities of CPLEX, we aim to solve the Multidimensional Knapsack Problem (MKP) more efficiently. The GA provides a robust search mechanism through evolutionary principles, while the Branch-and-Cut method ensures the solution's optimality and feasibility

5.2 Proposed Approach

The hybridization of Genetic Algorithms and the Branch-and-Cut method offers a comprehensive solution to the MKP.

- **Genetic Algorithms:**

❖ Strengths:
- Efficiently searches large and complex solution spaces.
- Good at finding near-optimal solutions quickly.
- Provides a diverse set of solutions due to stochastic processes.

❖ Weaknesses:
- May struggle to find the exact optimal solution.
- Can get stuck in local optima without proper diversification.

- **Branch-and-Cut:**

❖ Strengths:
- Provides exact solutions by systematically exploring the solution space.
- Ensures feasibility and optimality of solutions through rigorous mathematical methods.

❖ Weaknesses:
- Computationally intensive for very large problems.
- Can be slow if the initial solution is far from optimal.

By combining these two methods, the GA first provides a good initial solution and explores the solution space heuristically. Then, CPLEX refines this solution using Branch and Cut and complimentary techniques, ensuring both feasibility and optimality. This hybrid approach exploits the strengths of both methods, providing balanced and efficient solution to the MKP.

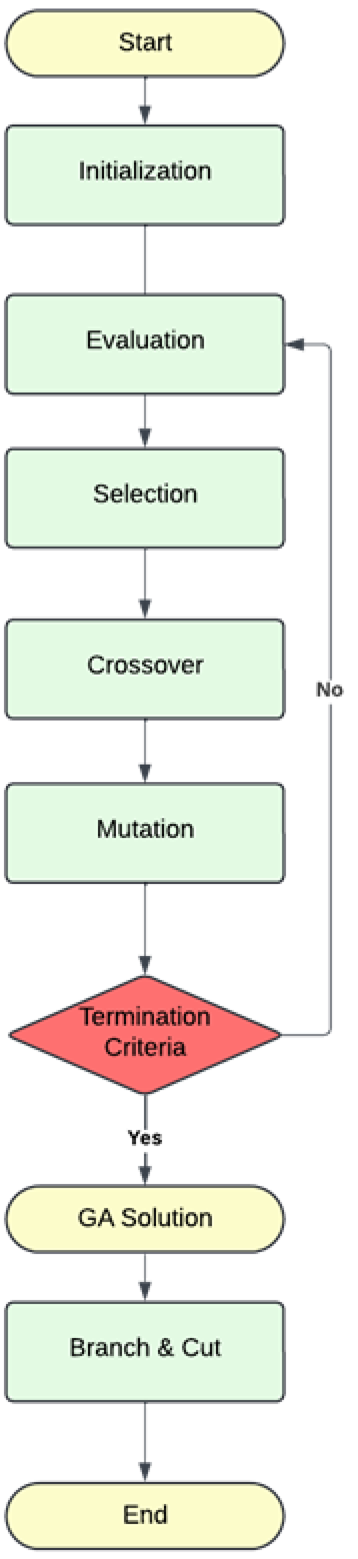


Figure 5.1: Diagram of the Proposed Approach

Genetic Algorithm

We chose the Genetic Algorithm (GA) for its ability to efficiently search large solution spaces by simulating the process of natural evolution. This heuristic method is particularly useful for solving complex combinatorial optimization problems like the Multidimensional Knapsack Problem (MKP). The GA uses mechanisms inspired by biological evolution, such as selection, crossover, mutation, and elitism, to iteratively improve the solution.

Explanation of Functions and Functionalities with Examples

1. Creating Individuals:

Functionality:

- The create_individual function generates feasible initial solutions for the genetic algorithm by selecting high-efficiency items.

2. Selection Function:

Functionality :

- The selection function uses tournament selection to choose individuals from the current population based on their fitness values.

Example:

- Population: [[0, 1, 0, 0], [1, 0, 1, 0], [1, 1, 0, 0], [0, 0, 1, 0]]
- Fitness Values: [10, 20, 15, 5]
- Tournament Size: 2
- Tournament Selection Process:
    - First Selection:
        - Randomly select two individuals: indices [0, 2]
        - Tournament Values: [10, 15]
        - Select the individual with the highest value: index 2
        - Append [1, 1, 0, 0] to selected_population.
    - Second Selection:
        - Randomly select two individuals: indices [1, 3]
        - Tournament Values: [20, 5]
        - Select the individual with the highest value: index 1
        - Append [1, 0, 1, 0] to selected_population.
- Repeat until the selected_population is the same size as the original population.

3. Crossover Function:

Functionality:

- The crossover function creates two new individuals (children) from two parent individuals by swapping segments of their genetic material.

Example:

- Parent Chromosomes:
  - parent1 = [1, 0, 1, 1, 0, 0, 1, 0, 1, 0]
  - parent2 = [0, 1, 0, 0, 1, 1, 0, 1, 0, 1]
- Selected Crossover Points:
  - Assume the selected crossover points are [3, 7].
- Initial Children:
  - child1 = [1, 0, 1, 1, 0, 0, 1, 0, 1, 0] (copy of parent1)
  - child2 = [0, 1, 0, 0, 1, 1, 0, 1, 0, 1] (copy of parent2)
- Crossover Process:
  - For the first segment (before the first crossover point at position 3):
    - child1[:3] becomes parent2[:3] → [0, 1, 0]
    - child2[:3] becomes parent1[:3] → [1, 0, 1]
    - child1 now is [0, 1, 0, 1, 0, 0, 1, 0, 1, 0]
    - child2 now is [1, 0, 1, 0, 1, 1, 0, 1, 0, 1]
  - For the second segment (between crossover points at positions 3 and 7):
    - child1[3:7] becomes parent1[3:7] → [1, 0, 0, 1]
    - child2[3:7] becomes parent2[3:7] → [0, 1, 1, 0]
    - child1 now is [0, 1, 0, 0, 1, 1, 0, 0, 1, 0]
    - child2 now is [1, 0, 1, 1, 0, 0, 1, 1, 0, 1]
- Final Children:
  - child1 = [0, 1, 0, 0, 1, 1, 0, 0, 1, 0]
  - child2 = [1, 0, 1, 1, 0, 0, 1, 1, 0, 1]

4. Mutation Function:

Functionality:

- The mutation function introduces randomness into the genetic algorithm by flipping bits in the binary vector of an individual based on a mutation rate.

Example:

- Initial Individual: [0, 1, 0, 1, 1]
- Mutation Rate: 0.1

- Random Numbers Generated: [0.05, 0.15, 0.03, 0.2, 0.08]
- Mutation Decisions:
  - 0.05 < 0.1: flip the first bit → [1, 1, 0, 1, 1]
  - 0.15 >= 0.1: do not flip the second bit → [1, 1, 0, 1, 1]
  - 0.03 < 0.1: flip the third bit → [1, 1, 1, 1, 1]
  - 0.2 >= 0.1: do not flip the fourth bit → [1, 1, 1, 1, 1]
  - 0.08 < 0.1: flip the fifth bit → [1, 1, 1, 1, 0]

- Final Mutated Individual: [1, 1, 1, 1, 0].

5. Elitism:

Functionality:

- The elitism process ensures that the best solutions found so far are preserved and carried over to the next generation.

Example:

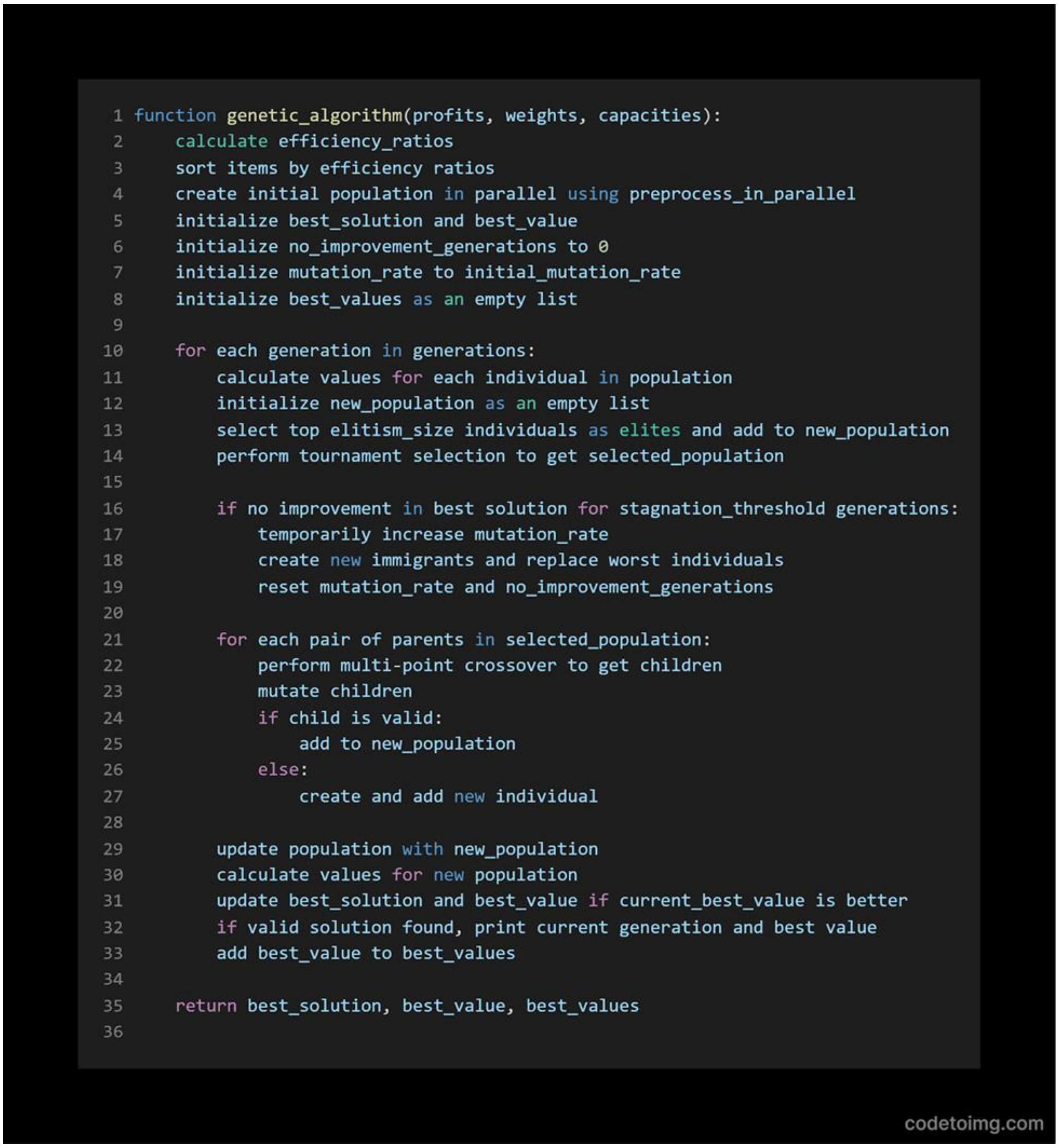
- Population Size: 10
- Elitism Size: 2
- Fitness Values: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
- Elite Indices:
  - elite_indices = np.argsort(values)[-2:]
  - elite_indices = [8, 9] (indices of the two best individuals)
- Select Elites:
  - elites = [population[8], population[9]]
- Add Elites to New Population:
  - new_population.extend(elites)
  - The new population will start with these two elite individuals.

6. Stagnation Mechanism:

Functionality:

- The stagnation mechanism increases the mutation rate and introduces new immigrants if there has been no improvement in the best solution for a specified number of generations.

Pseudo-Code of the Genetic Algorithm



```
function genetic_algorithm(profits, weights, capacities):
    calculate efficiency_ratios
    sort items by efficiency ratios
    create initial population in parallel using preprocess_in_parallel
    initialize best_solution and best_value
    initialize no_improvement_generations to 0
    initialize mutation_rate to initial_mutation_rate
    initialize best_values as an empty list

    for each generation in generations:
        calculate values for each individual in population
        initialize new_population as an empty list
        select top elitism_size individuals as elites and add to new_population
        perform tournament selection to get selected_population

        if no improvement in best solution for stagnation_threshold generations:
            temporarily increase mutation_rate
            create new immigrants and replace worst individuals
            reset mutation_rate and no_improvement_generations

        for each pair of parents in selected_population:
            perform multi-point crossover to get children
            mutate children
            if child is valid:
                add to new_population
            else:
                create and add new individual

        update population with new_population
        calculate values for new population
        update best_solution and best_value if current_best_value is better
        if valid solution found, print current generation and best value
        add best_value to best_values

    return best_solution, best_value, best_values
```

codetoimg.com

Branch and Cut Method

The Branch and Cut method combines the traditional branch-and-bound approach with cutting plane techniques to solve integer programming problems efficiently. This method is especially effective for the Multidimensional Knapsack Problem (MKP) as it iteratively refines the feasible region of the problem by adding linear inequalities (cuts) to remove fractional solutions. Below, we explain the Branch and Cut method and various types of cutting planes used, illustrated with examples similar to the genetic algorithm operators. The Branch and Cut method operates by dividing the original problem into smaller subproblems (branching) and then tightening the feasible region of these subproblems by adding cuts. This hybrid method enhances the performance and accuracy of solving the MKP.

Example: Suppose we have an MKP with three items and two constraints:

- Profits: [10, 5, 15, 7, 6]
- Weights: [[2, 3, 5, 7, 1], [1, 2, 3, 4, 1]]
- Capacities: [10, 5]

Initial Problem:

$$\begin{gathered} max\ 10x1 + 5x2 + 15x3 + 7x4 + 6x5 \\ Subject\ to: \\ 2x1 + 3x2 + 5x3 + 7x4 + x5 \leq 10 \\ x1 + 2x2 + 3x3 + 4x4 + x5 \leq 5 \\ 0 \leq xi \leq 1 \end{gathered}$$

Initial LP Relaxation: Suppose the LP relaxation gives the following fractional solution:

$$x1=0.5 , x2=0.8 , x3=0.3 , x4=0.6 , x5=1$$

Branch on xi:

- Subproblem 1: x1=0
- Subproblem 2: x1=1

Subproblem 1: x1=0

New LP relaxation solution:

$$x2=0.7 , x3=0.5 , x4=0.4, x5=1$$

Add a Flow Cover Cut: From the constraint:

$$x2+x3+x4+x5=2.6$$

A flow cover cut can be:

$$x2 + x3 \leq 1.2$$
$$x4 + x5 \geq 1.4$$

New LP relaxation after Flow Cover Cut:

$$x2 = 1, x3 = 0, x4 = 0, x5 = 1$$

This solution is integer: $x2 = 1, x3 = 0, x4 = 0, x5 = 1$

Subproblem 2: $\boldsymbol{x1 = 1}$

New LP relaxation solution:

$$x2 = 0.6\,, x3 = 0.4\,, x4 = 0.5\,, x5 = 1$$

Add a Gomory Cut: From the solution x2=0.6 , a Gomory cut can be:

$$0.4x2 + 0.4x3 + 0.5x4 + x51.5$$

New LP relaxation after Gomory Cut:

$$x2 = 0, x3 = 1, x4 = 0, x5 = 1$$

This solution is integer: $x2 = 0, x3 = 1, x4 = 0, x5 = 1$

Combined Solution:

The optimal solution is the one with the highest profit value among all feasible solutions found in the subproblems.

Comparison of Solutions

- Subproblem 1: $x1 = 0, x2 = 1, x3 = 0, x4 = 0, x5 = 1$
  - Profit: $5(1) + 6(1) = 11$
- Subproblem 2: $x1 = 1, x2 = 0, x3 = 1, x4 = 0, x5 = 1$
  - Profit: $10(1) + 15(1) + 6(1) = 31$

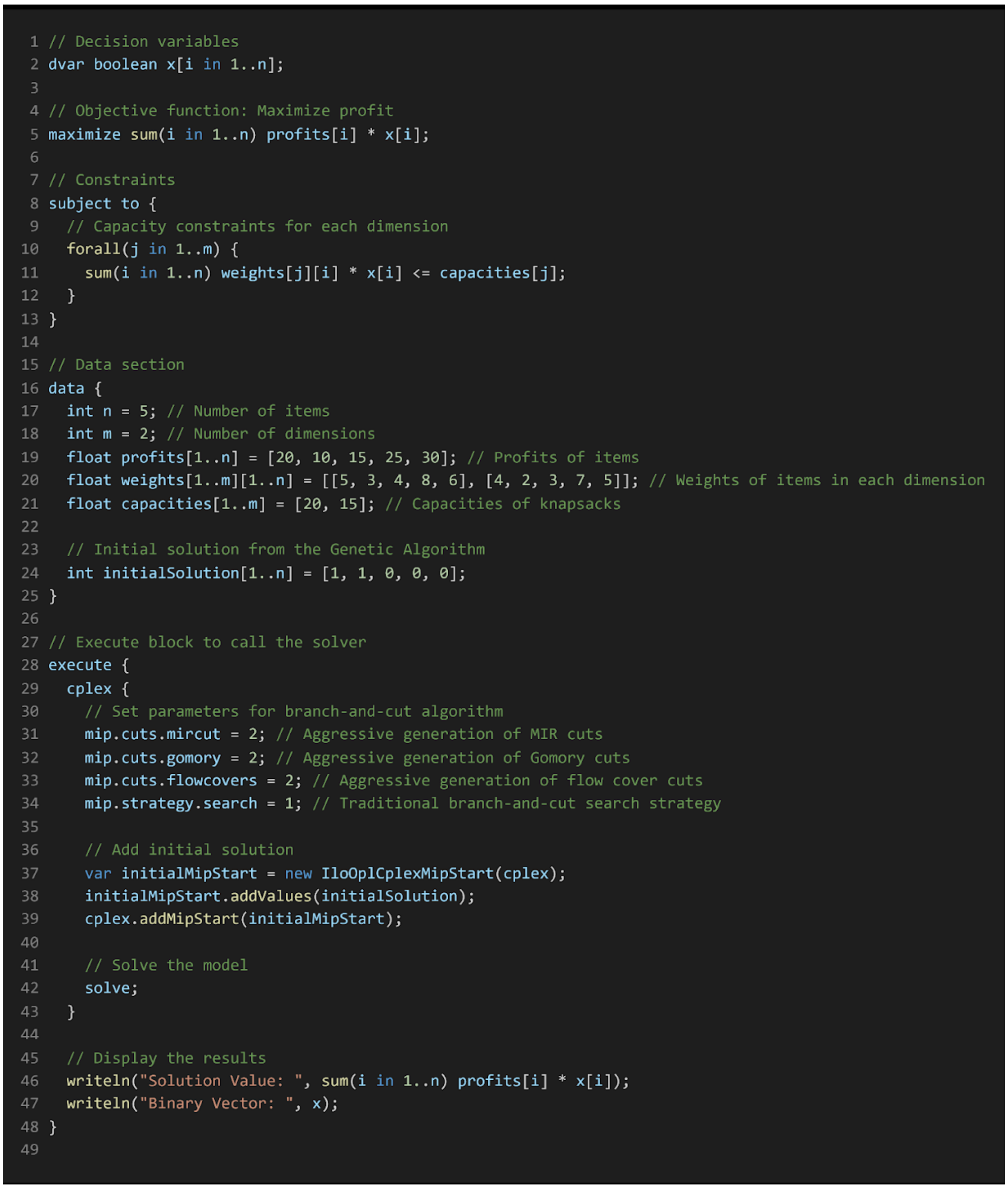
Thus, the optimal solution is from Subproblem 2:

- $x1 = 1, x2 = 0, x3 = 1, x4 = 0, x5 = 1$
- Optimal Profit: 31

MKP CPLEX ILP

Our approach leverages the power of the CPLEX solver to efficiently handle complex mixed-integer programming (MIP) problems. We chose CPLEX for its robust capabilities, particularly its implementation of the branch-and-cut algorithm. This algorithm combines branch-and-bound with cutting planes to tighten the formulation and prune the search space effectively, making it an ideal tool for solving large-scale and intricate optimization problems. By utilizing CPLEX's advanced mathematical programming techniques, we can systematically explore and refine feasible solutions, ensuring both optimality and efficiency.

MKP ILP



```
// Decision variables
dvar boolean x[i in 1..n];

// Objective function: Maximize profit
maximize sum(i in 1..n) profits[i] * x[i];

// Constraints
subject to {
  // Capacity constraints for each dimension
  forall(j in 1..m) {
    sum(i in 1..n) weights[j][i] * x[i] <= capacities[j];
  }
}

// Data section
data {
  int n = 5; // Number of items
  int m = 2; // Number of dimensions
  float profits[1..n] = [20, 10, 15, 25, 30]; // Profits of items
  float weights[1..m][1..n] = [[5, 3, 4, 8, 6], [4, 2, 3, 7, 5]]; // Weights of items in each dimension
  float capacities[1..m] = [20, 15]; // Capacities of knapsacks

  // Initial solution from the Genetic Algorithm
  int initialSolution[1..n] = [1, 1, 0, 0, 0];
}

// Execute block to call the solver
execute {
  cplex {
    // Set parameters for branch-and-cut algorithm
    mip.cuts.mircut = 2; // Aggressive generation of MIR cuts
    mip.cuts.gomory = 2; // Aggressive generation of Gomory cuts
    mip.cuts.flowcovers = 2; // Aggressive generation of flow cover cuts
    mip.strategy.search = 1; // Traditional branch-and-cut search strategy

    // Add initial solution
    var initialMipStart = new IloOplCplexMipStart(cplex);
    initialMipStart.addValues(initialSolution);
    cplex.addMipStart(initialMipStart);

    // Solve the model
    solve;
  }

  // Display the results
  writeln("Solution Value: ", sum(i in 1..n) profits[i] * x[i]);
  writeln("Binary Vector: ", x);
}
```

Explanation of the CPLEX ILP

We chose CPLEX mainly for its robust branching and cutting planes, which are crucial for efficiently solving complex mixed-integer programming problems like the Multidimensional Knapsack Problem (MKP). These techniques allow CPLEX to systematically explore and tighten the feasible solution space. Alongside branching and cutting planes, CPLEX employs complementary techniques such as presolve, aggregation, and dynamic search strategies to enhance the solving process.

Initialization and Setup

Functionality:

Initializes the CPLEX solver, sets the objective sense, and prepares the variables and constraints for the problem.

- Objective Sense: Sets the objective to maximization, indicating that the goal is to maximize the profit.

Adding Variables and Constraints

Functionality:

Adds binary variables representing whether an item is included in the knapsack, also with constraints ensuring that the total weight of selected items does not exceed the capacities.

- Variables: Binary variables (x0, x1, x2, ...) are added, each representing an item.
- Constraints: Linear constraints ensure the sum of weights for selected items does not exceed the capacities in each dimension.

Initial Solution

Functionality:

Provides an initial solution to the solver, potentially speeding up the optimization process by giving it a good starting point.

- MIP Starts: Adds the initial solution to the solver, which uses it to kickstart the search for the optimal solution.

Branching and Cutting Planes

Functionality:

- Various cuts and branching strategies are used by the solver to tighten the LP relaxation and prune the search tree.

Key Techniques

- Cover Cuts: Tighten the LP relaxation by covering sets of variables, reducing the feasible region.
- MIR Cuts: Remove fractional solutions from the LP relaxation, improving the solver's ability to find integer solutions.
- Lift and Project Cuts: Project high-dimensional constraints into lower dimensions, strengthening the formulation.
- Gomory Fractional Cuts: Eliminate fractional values, improving the integrality of the solutions.

5.3 Experimental Setup

For executing the approach, we have used:

- Dell laptop
- Installed RAM: 8.00 GB.
- System type: 64-bit operating system, x64-based processor.
- Operating system: Windows 10 Pro, version 21H2.
- Processor: Intel® Core™ i5-82500 CPU @ 1.60GHz 1.80GHz.
- GPU 1: Intel® UHD Graphics 620.

5.4 Benchmarks

Chu & Beasley's benchmark library (Chu & Beasley 1998) for MKP. This library contains classes of randomly created instances for each combinations of n ∈ {100, 250, 500} items, m ∈ {5, 10, 30} constraints, and tightness ratios: α = P n j=1 ∈ {0.25, 0.5, 0.75} We focus on the instances bellow:

- OR5x100-0.25-1
- OR5x250-0.25-1
- OR5x500-0.25-1
- OR10x100-0.25-1
- OR10x250-0.25-1
- OR10x500-0.25-1

5.5 Related Work

The method explained in the paper "Guided genetic algorithm for the multidimensional knapsack problem" by Abdellah Rezoug, Mohamed Bader-El-Den, and Dalila Boughaci is a hybrid heuristic approach named Guided Genetic Algorithm (GGA). Here is a summary of the method:

Guided Genetic Algorithm (GGA)

Overview: GGA is a two-step memetic algorithm composed of a data pre-analysis and a modified Genetic Algorithm (GA). The approach leverages prior knowledge about the problem data to guide the evolutionary process towards promising areas of the solution space.

Key Components:

1. Data Pre-analysis:
   - An efficiency-based method is used to analyze the problem data and extract useful information.
   - This analysis helps identify items that are likely to appear in good solutions.
2. Chromosome Design:
   - Each chromosome represents a feasible solution to the MKP, using integer representation where each gene presents an item ID.
3. Guiding Information:
   - Items are sorted based on their efficiency, which is calculated considering both value and weight.
   - The sorted items are divided into three sets: high efficiency (X1), medium efficiency (Core), and low efficiency (X0).
4. Initial Population:
   - A special initialization process generates a diverse initial population by combining items from X1 with randomly selected items.
5. Fitness Evaluation:
   - The fitness function incorporates the efficiency of items, rewarding chromosomes that include high-efficiency items and penalizing those with low-efficiency items.
   - Four different formulations of the fitness function are examined, integrating efficiency and similarity with X1 and X0.
6. Genetic Operators:
   - Standard genetic operators like crossover, mutation, and reproduction are used.
   - Tournament selection, random single-point crossover, and multiple-point bit-flip mutation are employed.

5.6 Results Comparison

The results of our approach were compared with those reported by Rezoug, Bader-El-Den, and Boughaci (2017) who used the Guided Genetic Algorithm (GGA) for the MKP. As shown in Table.**[66]**

Data File	Rezoug.A, Bader-El-Den. M& Boughaci.D	Our Scores	Gap
OR5x100-0.25_1	24000	24381	1.02%
OR5x250-0.25_1	59000	59312	1.01%
OR5x500-0.25_1	119000	120148	1.01%
OR10x100-0.25_1	22800	23064	1.01%
OR10x250-0.25_1	58800	59187	1.01%
OR10x500-0.25_1	116300	117779	1.01%

5.7 Results Discussion

The comparative analysis between our method and the Guided Genetic Algorithm (GGA) proposed by Rezoug et al. (2017) clearly indicates that while their results are impressive, our approach has achieved higher scores on all of the benchmark instances thanks to our revolutionary hybrid approach.

Conclusion

The hybridization of Genetic Algorithms with the Branch and Cut method offers a powerful approach to solving the Multidimensional Knapsack Problem. The GA provides a robust initial solution, and the Branch and Cut method ensures the solution's quality and feasibility through systematic refinement. This combination leverages the strengths of both heuristic and exact methods, providing an efficient and effective solution to complex optimization problem

General Conclusion

This thesis has explored the intricate challenges and innovative solutions related to the 0/1 Multidimensional Knapsack Problem (MKP), a prominent issue in combinatorial optimization. By developing and evaluating a hybrid algorithm that combines the strengths of Genetic Algorithms (GA) and the Branch and Cut (B&C) method, we aimed to enhance both the efficiency and accuracy of solving the MKP. Our proposed hybrid approach leverages the exploratory power of GAs to generate high-quality initial solutions, which are then refined through the precision of B&C to achieve near-optimal results. This combination proved effective in balancing the trade-off between solution quality and computational effort, addressing the limitations inherent in using either method independently.

The results demonstrated that the hybrid algorithm significantly improves computational efficiency and solution quality compared to standalone methods. The GA component effectively explores the solution space, providing a diverse set of potential solutions, while the B&C component ensures rigorous optimization by pruning infeasible regions and tightening constraints. This synergy has shown considerable promise in various applications, including logistics, finance, and manufacturing, where MKP-like problems are prevalent. Furthermore, the versatility of the hybrid algorithm was evident through its application in different case studies, which highlighted its potential to address complex real-world problems effectively.

The enhanced performance and adaptability of the hybrid method underscore its practical relevance and theoretical significance in the field of combinatorial optimization. In summary, this thesis contributes to the advancement of solving the MKP by introducing a robust and efficient hybrid algorithm. Future research could explore further enhancements to the hybrid approach, such as integrating additional heuristic methods or optimizing parameter settings through machine learning techniques. Additionally, extending the hybrid framework to other combinatorial problems could yield valuable insights and broader applicability. By addressing the multidimensional constraints and vast solution space of the MKP, our research not only pushes the boundaries of current optimization techniques but also provides a foundation for future advancements in solving complex optimization problems across various domains

Bibliography

Bibliography

[1].Martello, S., & Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. Wiley-Interscience.

[2]. Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack Problems*. Springer.

[3].Korte, B., & Vygen, J. (2012). *Combinatorial Optimization: Theory and Algorithms*. Springer.

[4]. Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman.

[5].Vazirani, V. V. (2001). *Approximation Algorithms*. Springer.

[6]. Michalewicz, Z., & Fogel, D. B. (2004). *How to Solve It: Modern Heuristics*. Springer.

[7]. Schrijver, A. (2003). *Combinatorial Optimization: Polyhedra and Efficiency*. Springer.

[8]. Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598), 671-680.

[9]. Glover, F. (1989). Tabu Search—Part I. *ORSA Journal on Computing*, 1(3), 190-206.

[10].Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

[11].Gomory, R. E. (1958). Outline of an Algorithm for Integer Solutions to Linear Programs. *Bulletin of the American Mathematical Society*, 64(5), 275-278.

[12].Martello, S., & Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. Wiley-Interscience.

[13].Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack Problems*. Springer.

[14].Pisinger, D. (1995). A Minimal Algorithm for the Multiple-Choice Knapsack Problem. *European Journal of Operational Research*, 83(2), 394-410.

[15].Lodi, A., Martello, S., & Monaci, M. (2002). Two-Dimensional Packing Problems: A Survey. *European Journal of Operational Research*, 141(2), 241-252.

[16].Chu, P. C., & Beasley, J. E. (1998). A Genetic Algorithm for the Multidimensional Knapsack Problem. *Journal of Heuristics*, 4(1), 63-86.

[17].Chekuri, C., & Khanna, S. (2000). A PTAS for the Multiple Knapsack Problem. *SIAM Journal on Computing*, 31(3), 525-526

[18].Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. MIT Press.
[19].Pisinger, D. (1995). A Minimal Algorithm for the Multiple-Choice Knapsack Problem. *European Journal of Operational Research*, 83(2), 394-410.

[20].Martello, S., & Toth, P. (1980). Solution of the Zero-One Multiple Knapsack Problem. *European Journal of Operational Research*, 4(4), 276-283.

[21]. Yamada, T., & Kataoka, S. (2005). Heuristic and Exact Algorithms for the Knapsack Problem with Conflict Graph. *Journal of Heuristics*, 7(5), 533-543.

[22]. Korf, R. E. (1995). A New Algorithm for Optimal Bin Packing. *Proceedings of the National Conference on Artificial Intelligence*, 100-105.

[23].Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack Problems*. Springer.

[24].Martello, S., & Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. Wiley-Interscience.

[25].Pisinger, D. (1995). A Minimal Algorithm for the Multiple-Choice Knapsack Problem. *European Journal of Operational Research*, 83(2), 394-410.

[26].Gallo, G., Hammer, P. L., & Simeone, B. (1980). Quadratic Knapsack Problems. *Mathematical Programming*, 12(1), 132-149.
[27].Chu, P. C., & Beasley, J. E. (1998). A Genetic Algorithm for the Multidimensional Knapsack Problem. *Journal of Heuristics*, 4(1), 63-86.

[28].Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1985). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley.

[29].Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman.

[30].Nemhauser, G. L., & Wolsey, L. A. (1988). *Integer and Combinatorial Optimization*. Wiley-Interscience.

[31].Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. MIT Press.

[32].Gendreau, M., & Potvin, J. Y. (Eds.). (2010). *Handbook of Metaheuristics*. Springer.

[33].Chu, P. C., & Beasley, J. E. (1998). A Genetic Algorithm for the Multidimensional Knapsack Problem. *Journal of Heuristics*, 4(1), 63-86.

[34].Markowitz, H. (1952). Portfolio Selection. *The Journal of Finance*, 7(1), 77-91.

[35].Pisinger, D. (1995). A Minimal Algorithm for the Multiple-Choice Knapsack Problem. *European Journal of Operational Research*, 83(2), 394-410.

[36]. Gavish, B., & Altinkemer, K. (1990). Backbone Network Design Tools with Economic Tradeoffs. *INFORMS Journal on Computing*, 2(3), 236-252.

[37].Durand, J. M. Alliot, J. Noailles, Algorithmes génétiques, un croisement pour les problèmes partiellement séparables. Journées Evolution Artificielle Francophones JEAF, Toulouse, 1994.

[38]. John Henry Holland. Adaptation in natural and artificialsystems : an introductoryanalysiswith applications to biology, control, and artificial intelligence. MIT press,1992.

[39].Z. Michalewicz ,"GeneticAlgorithms + Data Structures = Evolution Programs" ;3 édition

.Springer, 1996

[40]. Amira Gherboudj. Méthodes de résolution de problèmes difficiles académiques. Thèse de Doctorat, Université de Constantine2, 2013.

[41].S.Voisin, "Applicationdes algorithmes génétiques à l'estimation de mouvement par modélisation markovienne ", rapport DEA, Université Joseph, France,2004 .

[42].N.Benahmed,"Optimisation de réseaux de neurones pour la reconnaissance de chiffres manuscrits isolés: sélection et pondération des primitives par algorithme génétique".Université du Québec ,2002.

[43].D.Beasly, D .R . Bull, R. R. Matrin**"**Anoverview of genetic : Part 2, ResearshTopics**"**, Universitycomputing, Vol 15 ,N° 4, 1993, p.170.

[44]. Christophe Caux, Henri Pierreval et M-C Portmann. Les algorithmes génétiques et leur application aux problèmes d'ordonnancement.*Automatique-productiqueinformatiqueindustrielle*, 29(4-5):409–443, 1995.

[45].R. L. Haupt,S.E.**"**Practicalgenetic algorithms**",** 2 édition, New Jersey,2004.

[46].OSMAN ALP and ERHAN ERKUT "An E¢ cientGeneticAlgorithm for the p-Median Problem". 2003 Kluwer AcademicPublishers. Manufactured in The Netherlands.

[47]. Nemhauser, G. L., & Wolsey, L. A. (1988). Integer and Combinatorial Optimization. Wiley-Interscience.

[48].Achterberg, T. (2007). Constraint Integer Programming. Ph.D. Thesis, Technische Universität Berlin.

[49].Wolsey, L. A. (1998). Integer Programming. Wiley-Interscience.

[50].Gomory, R. E. (1958). Outline of an Algorithm for Integer Solutions to Linear Programs. Bulletin of the American Mathematical Society, 64(5), 275-278.

[51].Balas, E. (1979). Disjunctive Programming. Annals of Discrete Mathematics, 5, 3-51.

[52].Padberg, M., & Rao, M. R. (1982). Odd Minimum Cut-Sets and b-Matchings. Mathematics of Operations Research, 7(1), 67-80.

[53].Nemhauser, G. L., & Wolsey, L. A. (1988). Integer and Combinatorial Optimization. Wiley-Interscience.

[54].Crowder, H., Johnson, E. L., & Padberg, M. (1983). Solving Large-Scale Zero-One Linear Programming Problems. Operations Research, 31(5), 803-834.

[55].Achterberg, T. (2007). Constraint Integer Programming. Ph.D. Thesis, Technische Universität Berlin.

[56].Schrijver, A. (1986). Theory of Linear and Integer Programming. Wiley-Interscience.

[57]. Gomory, R. E. (1960). An Algorithm for the Mixed Integer Problem. Management Science, 6(4), 333-342.

[58].Balas, E. (1979). Disjunctive Programming. Annals of Discrete Mathematics, 5, 3-51.

[59].Padberg, M., & Rinaldi, G. (1991). A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. SIAM Review, 33(1), 60-100.

[60].Bixby, R. E., & Lee, E. K. (1998). Solving Real-World Linear Programs: A Decade and More of Progress. Operations Research, 50(1), 3-15.

[61].Cook, W., Cunningham, W. H., Pulleyblank, W. R., & Schrijver, A. (1997). Combinatorial Optimization. Wiley.

[62].Ralphs, T. K., & Galati, M. (2009). Decomposition in Integer Linear Programming. Integer Programming: Theory and Practice, CRC Press.

[63].Fischetti, M., & Lodi, A. (2003). Local Branching. Mathematical Programming, 98(1-3), 23-47.

[64].Linderoth, J. T., & Savelsbergh, M. W. P. (1999). A Computational Study of Search Strategies for Mixed Integer Programming. INFORMS Journal on Computing, 11(2), 173-187.

[65].lover, F., & Kochenberger, G. A. (2003). Handbook of Metaheuristics. Springer.

[66]. Rezoug, A., Bader-El-Den, M., & Boughaci, D. (2017). Guided genetic algorithm for the multidimensional knapsack problem. *Memetic Computing, 9*(4), 353-369. https://doi.org/10.1007/s12293-017-0232-7