New results for several rectangle packing and covering problems

Lower bounds and different types of algorithms

Tomas Attila Olaj

DOI:10.18136/PE.2025.967

A dissertation submitted to the Doctoral School of Information Science and Technology for the degree of Doctor of Philosophy (PhD)

PhD Thesis in Computer Science, with a focus on Combinatorial Optimization

Supervisor: Dr. György Dósa

University of Pannonia
Faculty of Information Technology
Department of mathematics
Doctoral School of Information Science and Technology

2025, Hungary

New results for several rectangle packing and covering problems

Lower bounds and different types of algorithms

The thesis was prepared for the award of a doctoral degree (PhD) within the framework of the Doctoral School of Information Science and Technology at the University of Pannonia

in the discipline of **Computer Sciences**

Written by: Tomas Attila Olaj

Supervisor: Dr. György Dósa

I recommend the dissertation for acceptance: **yes** / **no.**

.....
supervisor

I recommend the dissertation for peer review.

.....
chair of the DDHC

The PhD-candidate has achieved ..... % at the public debate.

The composition of the Final Examination Committee:

Chair: .....
Reviewers: .....
Members: .....

Veszprém,

.....
chair of the committee

Qualification of degree: .....
Veszprém,

.....
chair of the UDHC

Abstract

The allocation, packing, and covering problems exhibit strong similarities, to the extent that solution methods developed for one class of problems can often be applied to the others, despite differences in formulation. This thesis focuses specifically on allocation, packing, and covering problems where a combinatorial optimization component plays a central role. In general, given a region (container) and a set of smaller objects (e.g., pieces or items), each object is associated with a profit value. The goal is to identify a subset of objects with maximum total profit, such that all selected items can be placed simultaneously within the region while satisfying specific allocation constraints.

Chapter 2 addresses the *Board Packing Problem* (BoPP), defined as follows: a rectangular board with m rows and n columns is given. Each position $(i, j) \in M \times N$ on the board has an associated integer value $g_{i,j}$, representing the revenue obtained if the position is covered. A set R of rectangles is provided, where each $r \in R$ has a specified height h_r, width w_r, and cost c_r. The objective is to select and place rectangles on the board in order to maximize profit, calculated as the total revenue of covered positions minus the total cost of the rectangles used. All rectangles must be aligned with the sides of the board. Overlaps are permitted, but the revenue from any covered position is counted only once. A mathematical model for this problem is formulated, and optimal solutions for small instances are obtained using the commercial solver CPLEX. For larger instances, where exact methods become inefficient, a metaheuristic algorithm is developed. Several benchmark instances are constructed to evaluate the performance of the proposed methods.

Chapter 3 is dedicated to square packing and covering problems. The input consists of one square of size 1, one of size 2, and so on, up to one square of size n, where n is a natural number. These squares are referred to as items. In the packing variant, the question is: what is the smallest square of size K (also known as the *accommodation square*) into which all items can be packed, without rotation and without overlap? Several lower bounds for K are introduced and applied to determine the smallest possible K for small and medium-sized instances. CPLEX is used to solve these instances. In addition, a simple greedy algorithm is proposed and shown to be asymptotically optimal. In the covering variant, the objective is to cover the largest possible square using the same set of items, allowing overlaps. Small instances are solved to optimality, while for medium and large cases, heuristic algorithms are designed. Finally, an additional asymptotically optimal algorithm is introduced for the covering case.

Tartalmi kivonat (in Hungarian)

Az elhelyezési, pakolási és fedési problémák jelentős hasonlóságokat mutatnak, olyannyira, hogy az egyik problématípushoz kidolgozott megoldási módszerek, a formalizálásbeli eltérések ellenére, gyakran alkalmazhatók más típusokra is. A jelen dolgozatban olyan elhelyezési, pakolási és fedési feladatok kerülnek vizsgálatra, amelyekben a kombinatorikus optimalizálás központi szerepet játszik. Általános esetben, ha adott egy régió és egy kisebb objektumokat (vagy elemeket) tartalmazó halmaz, akkor minden egyes objektumhoz egy profitérték rendelődik. A cél egy olyan részhalmaz meghatározása, amelynek elemei egyidejűleg elhelyezhetők a régión belül, úgy, hogy teljesüljön minden elhelyezési feltétel, és az összprofit maximális legyen.

A 2. fejezetben a *Board Packing Problem* (BoPP) nevű probléma kerül bemutatásra, amely az alábbi módon definiálható: Adott egy m sorból és n oszlopból álló téglalap alakú tábla. A tábla minden $(i, j) \in M \times N$ pozíciójához egy egész szám, $g_{i,j}$ van rendelve, amely a lefedés esetén elérhető bevételt jelöli. Adott továbbá egy R téglalaphalmaz, ahol minden $r \in R$ esetén ismert a magasság h_r, a szélesség w_r, valamint a költség c_r. A cél olyan téglalapok kiválasztása és elhelyezése a táblán, amelyekkel a profit maximalizálható; ez úgy számítandó, hogy a lefedett pozíciókhoz tartozó bevételek összegéből kivonásra kerül a téglalapok költsége. A téglalapokat az oldalaikkal párhuzamosan kell elhelyezni a táblán. Az átfedés megengedett, de minden pozíció után legfeljebb egyszer számolható el bevétel. A problémához egy matematikai modellt írok fel. A kisméretű példákat optimálisan megoldja a CPLEX kereskedelmi megoldó. A nagyobb példák esetén, ahol az egzakt módszerek hatékonysága már nem megfelelő, egy metaheurisztikus algoritmus került kifejlesztésre. A módszerek kiértékeléséhez több benchmark példahalmaz lett kifejlesztve.

A 3. fejezet a négyzetpakolási és négyzetfedési feladatoknak van szentelve. A bemenet egyesével növekvő méretű négyzetekből áll: egy 1×1-es, egy 2×2-es, egészen egy $n \times n$-es négyzetig, ahol n egy természetes szám. A pakolási változatban azt kell meghatározni, hogy melyik az a legkisebb K méretű négyzet (vagy befoglaló négyzet), amelybe az összes elem elhelyezhető forgatás és átfedés nélkül. Több alsó korlátot adok meg a K értékére, és ezek alkalmazásával megállapítható K optimális értéke kis és közepes méretű esetekre. A megoldás során a CPLEX rendszer is felhasználásra kerül. Emellett bemutatok egy egyszerű mohó algoritmust is, amely aszimptotikusan optimális. A fedési változatban a cél a lehető legnagyobb méretű négyzet lefedése ugyanazon elemek felhasználásával, átfedés megengedésével. A kis méretű példák esetén optimális megoldások születtek, míg a közepes és nagyobb példák esetében heurisztikus algoritmusok kerültek kialakításra. Végül bemutatok a fedési feladat esetén is egy aszimptotikusan optimális algoritmust.

Kurzdarstellung (in German)

Die Zuweisungs-, Pack- und Überdeckungsprobleme weisen starke Ähnlichkeiten auf, sodass Lösungsmethoden, die für eine Problemklasse entwickelt wurden, trotz unterschiedlicher Formulierungen häufig auch auf andere Klassen angewendet werden können. In dieser Arbeit werden insbesondere solche Probleme untersucht, bei denen ein kombinatorisches Optimierungselement eine zentrale Rolle spielt. Allgemein gilt: Gegeben ist ein Bereich (Container) sowie eine Menge kleinerer Objekte (z. B. Stücke oder Elemente), denen jeweils ein Gewinnwert zugeordnet ist. Ziel ist es, eine Teilmenge mit maximalem Gesamtgewinn zu bestimmen, wobei alle ausgewählten Objekte gleichzeitig innerhalb des Bereichs platziert werden müssen, unter Einhaltung bestimmter Zuweisungsbedingungen.

In Kapitel 2 wird das *Board Packing Problem* (BoPP) behandelt, das wie folgt definiert ist: Gegeben ist ein rechteckiges Gitter mit m Zeilen und n Spalten. Jede Position $(i, j) \in M \times N$ ist mit einem ganzzahligen Wert $g_{i,j}$ versehen, der den Gewinn angibt, wenn diese Position abgedeckt wird. Eine Menge R von Rechtecken ist gegeben, wobei jedes Rechteck $r \in R$ eine definierte Höhe h_r, Breite w_r und Kosten c_r besitzt. Ziel ist es, Rechtecke auszuwählen und auf dem Gitter zu platzieren, sodass der Profit – d. h. die Summe der abgedeckten Werte abzüglich der Gesamtkosten der verwendeten Rechtecke – maximiert wird. Alle Rechtecke müssen kantenparallel zur Gitterstruktur ausgerichtet werden. Überlappungen sind erlaubt, jedoch darf der Gewinn einer Gitterzelle höchstens einmal gezählt werden. Für das Problem wird ein mathematisches Modell formuliert. Kleine Instanzen werden mit dem kommerziellen Optimierer CPLEX exakt gelöst. Für größere Instanzen, bei denen exakte Methoden nicht effizient genug sind, wird ein metaheuristischer Algorithmus entwickelt. Zur Bewertung der Methoden werden mehrere Benchmark-Instanzen erstellt.

Kapitel 3 ist quadratischen Pack- und Überdeckungsproblemen gewidmet. Der Eingang besteht aus genau einem Quadrat jeder Größe von 1×1 bis $n \times n$, wobei n eine natürliche Zahl ist. Diese Quadrate werden als Objekte betrachtet. Im Packproblem besteht die Fragestellung darin, welches das kleinste Quadrat der Seitenlänge K ist (auch *Unterbringungsquadrat* genannt), in das alle Objekte ohne Drehung und ohne Überlappung passen. Mehrere untere Schranken für K werden eingeführt und genutzt, um für kleine und mittelgroße Instanzen das kleinstmögliche K zu bestimmen. Dabei kommt CPLEX zum Einsatz. Darüber hinaus wird ein einfacher gieriger Algorithmus vorgestellt, der sich asymptotisch als optimal erweist. Im Überdeckungsproblem besteht das Ziel darin, ein möglichst großes Quadrat mit denselben Objekten zu überdecken, wobei Überlappung erlaubt ist. Kleine Instanzen werden exakt gelöst, während für mittelgroße und große Instanzen heuristische Algorithmen entworfen werden. Schließlich wird ein weiterer asymptotisch optimaler Algorithmus für das Überdeckungsproblem präsentiert.

Sammendrag (in Norwegian)

Allokerings-, pakkings- og dekningsproblemer har mange fellestrekk. Løsningsmetoder utviklet for én problemklasse kan ofte anvendes på de andre, til tross for ulikheter i formulering og modellstruktur. Denne avhandlingen fokuserer spesielt på slike problemer der komponenter fra kombinatorisk optimering står sentralt. Generelt antar vi at det er gitt et område (en container) og en mengde mindre objekter (for eksempel enheter eller elementer), hvor hvert objekt tilordnes en gevinstverdi. Målet er å identifisere en delmengde objekter med størst mulig total gevinst, slik at alle valgte objekter kan plasseres samtidig innenfor området, samtidig som gitte allokeringsbetingelser oppfylles.

Kapittel 2 omhandler *Board Packing Problem* (BoPP), som defineres slik: Et rektangulært brett med m rader og n kolonner er gitt. Hver posisjon $(i, j) \in M \times N$ på brettet har en heltallsverdi $g_{i,j}$ som representerer gevinsten ved å dekke posisjonen. Det er gitt en mengde R av rektangler, der hvert $r \in R$ har angitt høyde h_r, bredde w_r, og kostnad c_r. Målet er å velge og plassere rektangler på brettet slik at netto gevinst maksimeres, definert som summen av gevinstene fra de dekkede posisjonene minus de samlede kostnadene for rektanglene. Rektanglene må plasseres kantrett i forhold til brettets sider. Overlapping er tillatt, men gevinst fra en gitt posisjon kan kun telles én gang. En matematisk modell for problemet utvikles, og små instanser løses optimalt ved hjelp av den kommersielle løseren CPLEX. For større instanser, der eksakte metoder er mindre effektive, utvikles en metaheuristisk algoritme. Flere benchmark-instanser konstrueres for å evaluere ytelsen til de foreslåtte metodene.

Kapittel 3 er viet pakkings- og dekningsproblemer for kvadrater. Inndata består av ett kvadrat av størrelse 1, ett av størrelse 2, og så videre opp til ett av størrelse n, der n er et naturlig tall. Disse kvadratene omtales som elementer. I pakkingsvarianten er målet å finne det minste kvadratet med sidelengde K (også kalt *innkvarteringskvadratet*) som gjør det mulig å plassere alle elementene i K uten overlapping og uten rotasjon (kantene må være parallelle med kvadratets sider). Flere nedre grenser for K introduseres og benyttes for å fastslå den minste mulige verdien av K for små og mellomstore instanser. Til dette benyttes CPLEX. I tillegg presenteres en enkel grådig algoritme som viser seg å være asymptotisk optimal. I dekningsvarianten er målet å dekke et størst mulig kvadrat med de samme elementene, der overlapping er tillatt. Små instanser løses optimalt, mens for mellomstore og store tilfeller utvikles heuristiske algoritmer. Til slutt introduseres en ytterligere asymptotisk optimal algoritme.

Acknowledgements

I wish to extend my deepest and most sincere gratitude to my supervisor, Prof. Dr. György Dósa (Gyuri), for his steadfast support, wise advice, and invaluable guidance from even before the early stages of my doctoral journey. His exceptional patience, pedagogical clarity, and tireless dedication have shaped not only the progress of this work, but also my growth as a researcher. Your professional insights and unwavering encouragement have been a constant source of motivation and inspiration, especially in the field of mathematics. I am profoundly grateful for your belief in me, for continually challenging me to reach my potential, and for the countless hours you have devoted to my development throughout this journey. I am also profoundly grateful to Professor Gyuri's family for their warmth, friendship, and unwavering encouragement throughout this journey. Their kindness and support have meant a great deal to me, both personally and emotionally, and I will always cherish the sense of belonging they have extended to me.

I would like to express my sincere gratitude to my opponents, Prof. Dr. József Békési and Dr. Tibor Dulai, for their thorough review, insightful feedback, and valuable suggestions, all of which have contributed significantly to improving the quality of this thesis.

I am deeply thankful to the individuals and research groups in Hungary, Prof. Dr. Zsolt Tuza and his kind-hearted wife Erzsébet, and my ever-busy but always supportive friend Dr. Gyula Ábrahám, and in Norway, Prof. Dr. Lars Magnus Hvattum, and College Professors Peter Grün and Michael Philippe, whose insight, encouragement, and steady support have played a key role in helping me complete this PhD journey.

My sincere thanks go to both current and former colleagues at the Østfold University College (HiOf), and the University of South-Eastern Norway (USN) for their support and collaboration over the years.

My thesis is lovingly and wholeheartedly dedicated to my wife Dr. Mónika Németh (Olaj Tamásné), my daughter Diana Alexia Ilona Olaj, and my family (in Hungary and Norway) for their endless patience with me, incredible emotional support, and unwavering belief in me through every step of this long doctoral quest. Your love, inspiration, and the energy taken in care for me, remind me daily of what truly matters. Thank You for being my ever lasting source of encouragement and for filling my life with happiness and purpose. This is for all of You!

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Structure of the dissertation

The dissertation consists of four main chapters:

1. **Introduction.** This chapter briefly introduces the scientific tools used throughout the thesis, such as mathematical modeling (e.g., linear programming), heuristic, approximation, and metaheuristic algorithms.

2. **The Board Packing Problem (BoPP).** This chapter presents a new model in which rectangles are placed on a large rectangular board. While the problem has several applications, the investigation is carried out from a theoretical point of view. It is proved that the problem is $\mathcal{NP}$-hard, and it is solved by multiple approaches, including a commercial solver and an advanced *evolutionary* algorithm. The efficiency and capabilities of the solution techniques are demonstrated on newly generated benchmark instances. The board may represent, for example, an onshore/offshore area in a factory or warehouse, an estate with valuable or essential resources (e.g., gold, oil, fish, wind), a city, or even a national-scale region; see Figure 1.1.

(a) Mining operation

(b) Wind power facility

(c) Fish farming facility

Figure 1.1: Board packing real life application (images from public sources)

3. **Square Packing and Covering.** Given squares of sizes $1, 2, \ldots, n$ (for a given integer n), a placement into the smallest possible enclosing square is sought for the packing problem, without rotation and without overlap. The covering problem uses the same set of squares to cover the largest possible

square (rotation is not allowed, but overlap is allowed). Several lower bounds are introduced, and algorithms are proposed for both packing and covering. For small values of n, optimal allocations (packing or covering, respectively) are obtained.

4. **New scientific results.** This chapter summarizes the new scientific results established in the thesis.

The dissertation concludes with a **Conclusions** chapter, followed by **Appendices** and a detailed list of **References**.

1.2 Scope and motivation

In connection with pure mathematics, packing and covering often deal with the efficient arrangement of geometric objects. This kind of configuration was the original Greek form of logistics, which was the pure task of skilled mathematicians to understand the physical world. The human nature of acquiring resources needed to be organized, stored, and moved around efficiently so that it can give some effect, profit, and value. The known history of this goes all the way back through many famous scientists to the German mathematician Johannes Kepler [122] in 1611, who proposed a solution how to optimize a stack of cannonballs in such a way to minimize the storage space (uniform sphere packing). However, Kepler only concluded a famous known configuration (used in grocery stores) and conjecture, which were only proved merely 400 years later by Thomas Hales [68] with the influence of László Fejes Tóth (see Figure 1.2).

The principles of logistics were later redefined as a concept in the 20th century to be more concerned about business planning, and the operation of goods and services from one site to another. Today, the most important work in operational analysis within logistics is to make better decisions and solve complex problems to increase profit and maintain sustainability.

Figure 1.2: László Fejes Tóth (picture by Tomas Attila Olaj)

Uniform sphere packing was only one problem to be solved, but there are many more unsolved general problems of optimal way to position three-dimensional objects of different sizes and shapes. This is also true about the basic problems in the two-dimensional case, where there are the determination of densest packing and the thinnest covering with (non-)congruent rectangles into some larger shape (e.g circle or rectangle).

Many problems are classified as $\mathcal{NP}$-hard (non-deterministic polynomial-time hardness). Typically, $\mathcal{NP}$-hard problems are considered difficult optimization problems because there are no known *fast* algorithms that provide an optimal solution for every input. Even with modern computing technology, these problems cannot be solved exactly. Instead, they can only be approximated

to a satisfactory level, which often requires an immense amount of computational power. The time needed to achieve such approximations can be substantial, potentially spanning years, depending on the complexity and number of parameters in the mathematical model.

Despite the old history and research work behind this field where methods from combinatorial optimization are used, the continuous operational usefulness and sustainability of its applications has made this research an active area with many key issues that remained to be solved. In an era where the demand for profitability and environmental sustainability in primary industries (e.g. mining, energy, farming, and fisheries) is increasing, contributing to the facilitation of objects in operational analysis becomes a vital motivator. By combining bin packing with optimization resource allocation, improving profit in the terms of process efficiency, and to reduce waste through advanced modeling and data-driven insights, operational research enables more sustainable practices. It is vital for correct application and business value to know what kind of technique (heuristics, metaheuristics, or exact methods) is suitable for a special logistics problem.

Overall, combinatorial optimization, using mathematical, statistical, and computational methods, is vital for solving large and complex decision-making problems (e.g. facility location) in both operational analysis, logistics, and management science, called the field of Operations Research (OR), where the goal is to identify and model the optimal combination of decisions and actions to achieve maximum profit and efficiency.

Chapter 1 provides a brief introduction to the concepts that are followed or applied. A general overview of optimization is first presented, followed by a more specific introduction to packing problems, as the thesis addresses certain specialized rectangle packing problems.

The thesis is structured into two main chapters following the introduction. Chapter 2 presents the Board Packing Problem (BoPP), a combined problem involving the allocation of rectangles on a given board under specific constraints. Chapter 3 addresses the square packing and square covering problems in a particular and noteworthy case, where the input consists of one square of size 1, one of size 2, and so on, up to one square of size n, with n being a fixed integer.

Taken together, these chapters constitute a novel combined contribution to the fields of operational analysis and combinatorial optimization. At the beginning of Chapters 2 and 3, the publications containing the content of each chapter are listed.

1.3 About combinatorial optimization

1.3.1 Mathematical optimization

Engineers and decision-makers are often faced with increasingly complex problems that arise in a wide range of sectors (e.g. industry, image processing, databases). The problem can be formulated as an optimization problem. Optimizing means choosing among several possibilities the one that best meets certain criteria. Optimization, and more generally operations research, aims to apply mathematical tools to solve such problems. An objective function is defined, which we seek to minimize

or maximize with respect to all the relevant parameters. The definition of the optimization problem is often given by constraints: all relevant parameters must comply with the problem's constraints, otherwise the solutions are not feasible.

Within optimization, we can distinguish two main branches: discrete optimization and continuous optimization, see e.g. Korte, and Vygen [85].

Continuous optimization, means finding an extreme (minimum or maximum) point in a continuous function, defined by real variables. Both constraints and objective function are usually differentiable.

This work is focused on discrete or combinatorial problems that only admit integer solutions, such as the Traveling Salesman Problem (TSP), where a solution is an order of cities to visit, or the Knapsack Problem (KP), where a solution is a set of items. In its most general form, a combinatorial (or discrete) optimization problem consists of finding in a discrete set one among the best subsets (or solutions) achievable, the notion of best solution being defined by an objective function. A combinatorial optimization problem consists of determining:

$$\max\{f(x) \quad : \quad x \in N\} \tag{1.1}$$

where given

- a discrete set N
- a function $f(x)$, called objective function

The set of feasible solutions cannot be described by an exhaustive list, because the difficulty lies precisely in the fact that the number of feasible solutions makes their enumeration impossible, or, on the other hand, it is $\mathcal{NP}$-complete to determine whether a solution exists or not. Therefore, the description of N is implicit; it generally consists of a relatively short list of the properties of feasible solutions.

Finding an optimal solution in a discrete and finite set is, in theory, an easy problem: just evaluate all possible solutions and compare their quality to pick the best one. However, in practice, enumerating and evaluating all solutions can take too much time. The search time for the optimal solution is an important factor, and this is why combinatorial optimization problems are considered so difficult. Complexity theory provides tools to measure this search time. Furthermore, since the set of feasible solutions is defined implicitly, it is sometimes very difficult to find even a single feasible solution.

Many combinatorial optimization problems are considered difficult, meaning that we do not know how to solve them optimally with a polynomial-time algorithm. This class of problems includes the TSP, KP, and many kinds of problems in bin packing, and scheduling [48, 49], optimization in graphs, vehicle routing, facility location, and others.

1.3.2 Applications

In today's globalized market, modern industries must compete on an international scale to generate returns and create value for their stakeholders. With the rapid advancement of digital technology and the ongoing smart information revolution,

industries, particularly those in logistics, face increasingly complex rules and constraints. To thrive, these industries must adapt to new standards in efficiency, sustainability, and technological innovation, navigating challenges that are essential for survival and success in a highly competitive global landscape.

Combinatorial optimization plays a crucial role in numerous industrial applications, one of which is the optimization of production lines. In this scenario, a product composed of numerous parts must be assembled by various machines and human resources, while also consuming additional resources such as water, energy, and materials. The goal is to develop an efficient production plan that not only dictates how to assemble these components but also ensures resource usage is economical and aligns with various constraints.

For instance, one such constraint is the shift-based nature of human labor; each worker's shift typically cannot exceed a specified number of hours, often eight. Additionally, if raw materials are transported to the factory from multiple sources, a transportation subproblem also needs to be addressed to optimize material delivery logistics.

Facility location applications are critical across a wide range of industries, where the primary objective is to determine the optimal placement of facilities in order to enhance efficiency, maximize profit or service quality, while minimizing operational and transportation costs. These problems are often modeled using combinatorial optimization techniques, including variants of the packing and covering problems.

In packing-based formulations, the focus is on fitting facilities or services into a constrained space or network, ensuring that available resources (e.g., land area, capacity, or coverage range) are utilized without overlap or redundancy. For instance, in retail, determining where to place stores or outlets to cover distinct market regions without cannibalizing each other's demand can be interpreted as a spatial packing problem. Similarly, in telecommunications, placing cell towers to provide maximal non-overlapping coverage within a limited frequency spectrum or physical region reflects a packing scenario with interference constraints.

This perspective parallels the structure of the Board Packing Problem (BoPP), introduced in this thesis, where rectangles (representing facilities or service areas) are placed on a revenue-generating grid (analogous to a geographical area). Each placement has a cost and contributes to a spatial objective function, representing profit or coverage. The challenge of maximizing profit under constraints of overlap and board boundaries mimics real-world decisions in urban planning, where multiple service zones compete for limited space and strategic positioning.

On the other hand, covering models aim to ensure that all demand points are adequately served by the placed facilities, typically under constraints related to distance, capacity, or service level. In public services and infrastructure, such as school districts, emergency response units, or utility access, the goal is to guarantee that every region falls within a prescribed service radius. These scenarios are directly relatable to the Square Covering Problem addressed in Chapter 3 of this thesis, where overlapping coverage by a hierarchy of square elements seeks to saturate a larger area efficiently.

Further, the Square Packing Problem, also treated in this work, finds strong analogs in facility layout and manufacturing. For example, configuring a warehouse

or factory floor plan involves placing non-overlapping units (machinery, pallets, storage modules) inside a bounded region, much like packing sequentially-sized squares into the smallest possible containing square. This becomes particularly relevant in the logistics sector, where spatial efficiency translates directly to operational cost savings.

Other domains, such as fisheries, mining, and agriculture, also present hybrid packing and covering challenges. Fishing zones or mining blocks must be spaced out (packing) to avoid resource overlap, while also ensuring that the entire targeted area is economically exploited (covering). This aligns closely with the trade-offs explored in the BoPP model, where overlaps are permitted but benefits from covered areas are only counted once.

1.4 Exact methods

We can distinguish two types of mathematical, statistical, and computational methods in operations research: exact methods and approximate methods. Here is a more precise classification of these methods:

1.4.1 Brute force

Evaluate all possible solutions and compare their quality to pick the best one. However, in practice, enumerating and evaluating all solutions can take too much time. The size of a combinatorial problem can grow exponentially. For example, for the classic TSP, if we want to evaluate all the solutions, the size would be:

- 4 cities: 24 permutations
- 10 cities: 3628800 permutations
- 100 cities: more than 1 trillion permutations

Using this method is normally inefficient (quite often goes above the order of $O(N!)$), and consumes a lot of time mainly because it is slow. So, in the real world, brute force cannot be considered a feasible strategy for combinatorial problems. Still, brute force sometimes can efficiently solve some subproblems of some more complex optimization problems.

1.4.2 Linear programming

Many combinatorial problems can be expressed as linear programs. A linear program in the canonical form is:

$$\max c^t x \tag{1.2}$$

subject to the constraints:

$$Ax \leq b \tag{1.3}$$

$$x \geq 0 \tag{1.4}$$

where $c = (c_1, \ldots, c_q)^T$ and $x = (x_1, \ldots, x_q)^T$ are column vectors with q rows, $A = (a_{ij})$, $1 \leq i \leq p$, $1 \leq j \leq q$, is a matrix with p rows and q columns, and $b = (b_1, \ldots, b_p)^T$ is a column vector with p rows.

$c^T x$ is the objective function, and $Ax \leq b$ and $x \geq 0$ are the constraints of the problem.

$x \in R$ is a feasible solution to the problem if x satisfies the constraints. The set P of all feasible solutions to an optimization problem is called its *feasible set.*

$x \in R$ is an optimal solution to the problem if x is a feasible solution to the problem and, furthermore, for any feasible solution $y \in R$, we necessarily have $c^T y \leq c^T x$. In other words, a feasible solution is optimal if it maximizes the objective function over the feasible set. The main methods to solve a linear program are described in subchapters 1.3.3 to 1.3.6.

1.4.3 Simplex method

The simplex method was invented by George Dantzig [39, 40], and is based on the principle that the optimal value of a linear program, if bounded, is always achieved at a basic feasible solution. Using this foundation, there are two ways to visualize the simplex process (see Figure 1.3). The first way is to see the process as a continuous change: starting with a basic feasible solution, a non-basic variable is gradually increased from zero.

As this variable increases, the values of the current basic variables are adjusted continuously to keep satisfying the system of linear equality constraints. The change in the objective function caused by a unit increase in the non-basic variable, considering the required adjustments to the basic variables, is the relative cost coefficient of that non-basic variable. If this coefficient is negative, the objective value improves as the non-basic variable increases, so it is raised as far as possible until further increase would violate feasibility.

At that point, one of the basic variables becomes zero, switching it to a non-basic variable, while the non-basic variable that was increased becomes basic. The second way to view the process is more discrete: recognizing that only basic feasible solutions need to be considered, various bases are chosen, and the corresponding basic solutions are calculated by solving the associated set of linear equations. The logic for systematically selecting new bases again, involves relative cost coefficients, and is largely derived from the first continuous perspective [90].

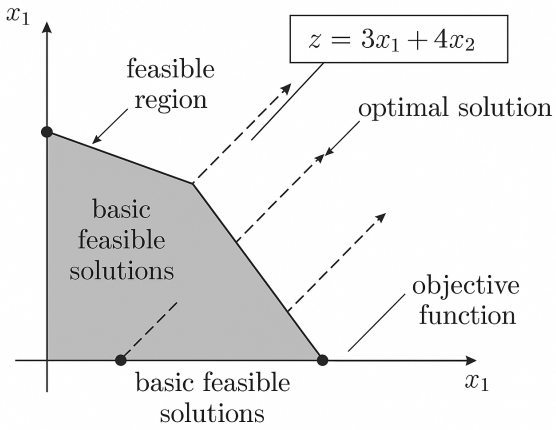


Figure 1.3: Illustration of the Simplex method in two dimensions. The shaded polyhedral region represents the feasible region defined by linear constraints. The vertices of this region correspond to basic feasible solutions. Dashed lines represent level curves of the objective function $z = 3x_1 + 4x_2$, and the arrows indicate the direction in which the objective function increases. The optimal solution lies at the vertex where the last improvement is possible. Image generated using AI tool DALL-E and post-edited for clarity.

The primal simplex method starts with a feasible solution that satisfies all the constraints and then improves the objective function value by moving to neighboring solutions until it finds the optimal one. It focuses on keeping all constraints satisfied while working towards a better objective.

The dual simplex method, on the other hand, starts with an optimal solution in terms of the objective but doesn't initially meet all constraints. It then adjusts to make the solution feasible while maintaining optimality. This method is useful when the goal is to handle adjustments efficiently, especially in situations where constraints change and we want to avoid solving the problem from scratch.

Maros contributed significantly to optimizing the efficiency of the simplex and dual simplex methods, developing algorithms that streamline these processes for practical applications in linear programming [93]. His work emphasized minimizing computational effort, which is key in using the dual simplex method directly from the primal tableau.

Lemke's work on the dual simplex method [87] is an algorithm for finding Nash equilibria in bimatrix games, involving principles similar to those used in the dual simplex method, by iterating through possible solutions and optimizing under specific complementarity conditions. Although the article mainly concerns complementarity problems and game theory, the method is based on many of the same principles of pivoting and iterative solution as dual simplex, and has inspired further use of dual methods in game theory and linear programming.

1.4.4 Interior-point methods

The interior-point method works by introducing a logarithmic barrier function with a weighting parameter. This approach creates a theoretical structure that defines

the analytic center, the central path of solutions as the parameter approaches zero, and the duals of these concepts. This structure is useful for specifying and analyzing different versions of interior-point methods.

The method follows the central path, an analytical curve within the interior of the problem's feasible domain, starting in the middle of the domain and ending in the optimal solution region.

The contributions of Roos, Terlaky, and Vial [108] have been essential in formalizing and advancing the theory of interior-point methods. They have developed key frameworks and algorithms that enhance the practical applications and efficiency of interior-point approaches in solving large-scale optimization problems, making their work foundational in this field.

While interior-point methods are traditionally associated with continuous optimization, their practical relevance extends to combinatorial problems such as bin packing, where LP relaxations serve as a foundation for efficient approximation. The following example illustrates how interior-point techniques can be applied in this context.

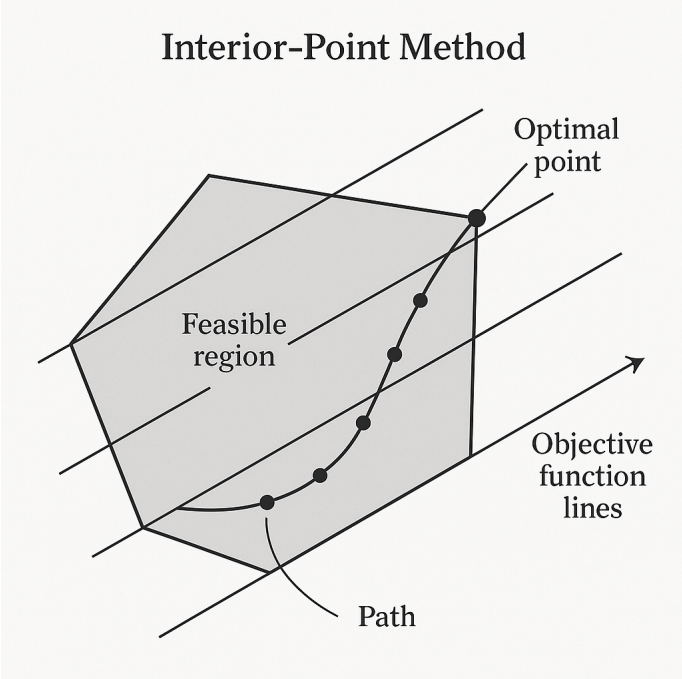


Figure 1.4: Illustration of the image generated using AI tool DALL-E, and post-edited for clarity.

1.4.5 Cutting planes method

If we want to solve an integer program, the simplest methods are not sufficient; we need additional tools. One such tool was first introduced by Gomory in 1958. A Gomory Cut is a linear constraint that is strictly stronger than its parent constraint, yet it does not exclude any feasible integer solutions of the LP problem under consideration [63, 65].

We start by solving the corresponding non-integer linear program through a continuous relaxation of the initial problem (replacing each constraint of the form $x_i \in \{0, 1\}$ with two weaker constraints, $0 \leq x_i \leq 1$). If the obtained optimum is an integer, the process is finished. Otherwise, there exists a linear inequality that separates the optimum from the set of admissible solutions. Finding this inequality to add to the problem's constraints is the *separation problem*, and such an inequality is called a *cut*. Thus, the current non-integer solution is no longer considered admissible. This procedure is repeated until an optimal solution is found.

This principle can also be applied to simplified models of bin packing, where linear relaxations yield fractional solutions that violate the integrality condition. Cutting planes offer a systematic way to eliminate such infeasible solutions without excluding any valid integer configurations. The following example illustrates this idea.

1.4.5.1 Example

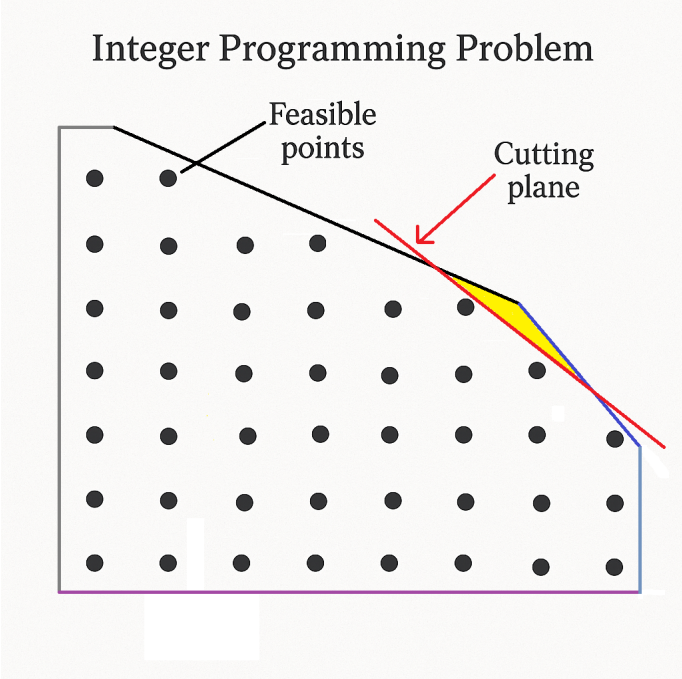


Figure 1.5: Illustration of the image generated using AI tool DALL-E, and post-edited for clarity.

The cutting-plane method eliminates parts of the polytope of the linear relaxation that contain no integer feasible solutions, while preserving all integer-feasible points.

1.4.6 Branch-and-bound

Providing satisfying solutions to large-scale mixed integer programming (MIP) problems requires formulation with linear programming (LP) relaxations that provide a good approximation to the convex hull of feasible solutions. One such formulation

is the branch-and-bound (B&B), another is the branch-and-price method, which is the result of a combination between branch and bound, and column generation methods.

B&B is a method of implicit or intelligent enumeration: all solutions to the problem can be enumerated, but by analyzing the properties of the problem, it is possible to avoid enumerating large classes of bad solutions. The separation phase involves dividing the problem into sub-problems, each with its own set of feasible solutions. By solving all the sub-problems and taking the best solution found, we are guaranteed to have solved the initial problem. The sets of solutions (and their associated sub-problems) thus constructed have a natural hierarchy in the form of a tree, often called a search tree or decision tree.

The evaluation of a node in the search tree aims to determine the optimum of the set of feasible solutions associated with the node in question, or, on the contrary, to mathematically prove that this set contains no interesting solution for solving the problem (i.e., an optimal solution), in which case it is unnecessary to perform the separation (enumeration) of its solution space.

In branch-and-price, sets of columns are left out of the LP relaxation because there are too many columns to handle effectively and, in an optimal solution, the majority of them will have their associated variable equal to zero. To check for the optimality of an LP solution, a subproblem known as the pricing problem is solved to attempt identifying columns to enter the basis. When such columns are found, the LP is reoptimized. When no columns price out to enter the basis and the LP solution does not meet the integrality condition branching occurs. Branch-and-price allows applying column generation throughout the B&B tree [16].

Branch-and-cut methods can also be used to solve combinatorial optimization problems formulated as mixed-integer linear programming. These algorithms consist of a combination of a cutting plane method with a B&B algorithm. Cutting plane methods improve the relaxation of the problem to approximate to a closer level the integer programming problem and B&B algorithms proceed by a rather complex divide and conquer approach. Problems that can be addressed with the branch-and-cut methodology the exploitation of strong cutting planes deriving from polyhedral theory include linear ordering problems, scheduling problems, maximum cut problems, packing problems, network designs, problems, and biological and medical applications, among others.

In the context of bin packing, B&B plays an important role when solving integer programming formulations where items must be assigned to bins using binary variables. Due to the combinatorial nature of the problem, direct LP relaxations often yield fractional solutions that are infeasible in practice. Branching on such variables and systematically exploring the solution space allows us to recover valid integer assignments. The following example illustrates how the method can be applied to a small bin packing instance.

1.4.6.1 Knapsack problem example

Let us consider the following small Knapsack Problem (KP). There are five items, with gain values $15, 8, 37, 25$, and 45, respectively. Their weights are $2, 1, 5, 3$, and

7, respectively. The capacity of the knapsack is $C = 15$. It must be decided what items are packed into the knapsack, so that the total weight is not more than C, and the total gain of the packed items is maximized.

The model of the problem is as follows:

$$\max z = 15x_1 + 8x_2 + 37x_3 + 25x_4 + 45x_5$$

$$\text{s.t. } 2x_1 + x_2 + 5x_3 + 3x_4 + 7x_5 \leq 15$$

$$x_i \in \{0, 1\}, \quad i = 1, \ldots, 5.$$

The relaxed problem is the same, except that all variables are between zero and one, i.e. $0 \leq x_i \leq 1$. The optimal solution of the relaxation is:

$$x_1 = 1,\ x_2 = 1,\ x_3 = 1,\ x_4 = 1,\ x_5 = \tfrac{4}{7} \approx 0.5714.$$

The value of the objective function is ≈ 110.71. This is a fractional solution, which is acceptable here since the variables are allowed to take any value between 0 and 1, if we consider not the KP but its relaxation. It means that 110.71 is an upper bound for the optimal solution of the KP.

1.4.6.2 Branch-and-bound analysis

With the additional constraint $x_5 = 0$, the optimal solution becomes:

$$x_1 = 1,\ x_2 = 1,\ x_3 = 1,\ x_4 = 1,\ x_5 = 0,$$

which yields objective value $z = 85$. With the constraint $x_5 = 1$, the optimal solution under the relaxed 0–1 bounds is:

$$x_1 = 1,\ x_2 = 1,\ x_3 = 0.4,\ x_4 = 1,\ x_5 = 1,$$

which gives $z = 107.8$.

Because on the right branch (where $x_5 = 1$) the estimate is larger, we go to the right for a new branching. Since in the relaxed optimum x_3 is fractional, we branch according to x_3. The next choice for branching is that x_3 is either zero or one. With the constraints $x_5 = 1$ and $x_3 = 0$, the optimal solution is:

$$x_1 = 1,\ x_2 = 1,\ x_3 = 0,\ x_4 = 1,\ x_5 = 1,$$

and the objective is $z = 93$.

With the constraints $x_5 = 1$ and $x_3 = 1$, the optimal solution is:

$$x_1 = 0,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1,\ x_5 = 1,$$

and the objective value is $z = 107$.

This is an integer solution. The value of the solution is greater than all previous estimates on the branches (i.e. 85 and 93). This is the unique optimal solution of the KP. Therefore, the optimal solution of the KP is

$$x_1 = 0,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1,\ x_5 = 1,$$

and the optimum is $z = 107$.

1.4.6.3 Branch-and-bound summary table

Branch	**Constraints**	**Solution** $(x_1, \ldots, x_5)$	**Objective** z
Relaxation	none	$(1, 1, 1, 1, \frac{4}{7})$	110.71 (fractional)
Left branch	$x_5 = 0$	$(1, 1, 1, 1, 0)$	85
Right branch (relaxed)	$x_5 = 1$	$(1, 1, 0.4, 1, 1)$	107.8 (fractional)
Right–Left	$x_5 = 1,\ x_3 = 0$	$(1, 1, 0, 1, 1)$	93
Right–Right	$x_5 = 1,\ x_3 = 1$	$(0, 0, 1, 1, 1)$	107 (integer optimum)

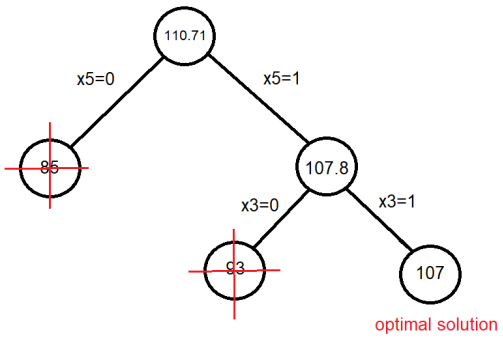


Figure 1.6: Branch-and-bound tree for the knapsack example. Node values indicate relaxed bounds or feasible integer solutions. The optimal solution is obtained at the rightmost leaf with $z = 107$. Illustration of the image generated using AI tool DALL-E, and post-edited for clarity.

1.5 Approximate methods

When confronted with difficult problems, e.g. BoPP as a new problem to be solved, or other $\mathcal{NP}$-complete variants, operations researchers have used approximate techniques to arrive at *a good enough* solution.

Approximate methods rely on heuristics or metaheuristics that reflect a strategy or way of thinking informed by knowledge of the problem. Since they are approximate, they allow finding solutions within a reasonable computation time (often polynomial time).

1.5.1 Greedy algorithm

A greedy algorithm works in the following way: Given an optimization problem, the solution is built step by step, making the locally optimal choice at each step, hoping to find the global optimum. For example, let's consider a simple graph, and we want to make a proper vertex coloring (a coloring so that for each edge the two vertices that are incident to the edge get different colors). We want to use as few colors as possible. This is an $\mathcal{NP}$-hard problem. The greedy algorithm would order the vertices, and then according to the order we give a color to each vertex. (The colors are also ordered, so that we can call them color 1, color 2, etc.). When the color is given to the current vertex, we avoid any color that is already used for the neighbors of the vertex, and from the allowed (not used) colors we give the least-used one in the list. Greedy algorithms are normally very useful for building an initial feasible solution used in metaheuristics.

In the context of a greedy algorithm for vertex coloring, the worst case for the use of color 1 occurs when the graph has a high degree of connectivity, such as in a complete graph (where each vertex is connected to every other vertex). In a complete graph with n vertices, each vertex requires a unique color because every pair of vertices is adjacent. Therefore, in this case, the greedy algorithm will assign each vertex a different color, meaning that color 1 will only be used for the first vertex. As a result, we need n colors to color the graph, and this coloring is optimal.

For other types of graphs, the greedy algorithm might use more colors than the chromatic number (the minimum number of colors required) because it only makes locally optimal choices. For example, in a cycle graph with an odd number of vertices, a proper coloring requires three colors, but depending on the vertex order, a greedy algorithm might use more than three colors.

The greedy algorithm is efficient in terms of time complexity, generally requiring $O(V + E)$ for a graph with V vertices and E edges. This is because each vertex is assigned a color based on the colors used by its adjacent vertices. Checking adjacent vertices and choosing a color is typically done in constant time for each vertex.

The number of colors used by the greedy algorithm can depend heavily on the vertex ordering. In the worst case, it may use as many as $\Delta + 1$ colors, where Δ is the maximum degree of any vertex in the graph [34].

1.5.2 Approximation algorithms

An approximation algorithm always has a guarantee, that its objective value is from a fixed percentage from the optimum value. Let us consider an optimization problem, where the objective function is minimized. For an instance I let $OPT(I)$ mean the optimum value, and for a given algorithm A, let $A(I)$ mean its objective value. Then the asymptotic approximation ratio of algorithm A is the smallest R,

for which it holds for any instance I, that $A(I) \leq R \cdot OPT(I) + C$, where C is a constant, independent from the input. There is also another (slightly different) version of the asymptotic approximation ratio, where instead of C the additive term is $o(OPT(I))$. In case of absolute approximation ratio the term C is zero. The approximation algorithms are introduced in the area of bin packing. For example the famous bin packing algorithm First Fit (FF) works as follows. Given a list of items, we pack the items in the given order, and each item is packed into the first bin where it fits. (Each bin has a capacity, this can be considered to be 1, and each item size is between 0 and 1.) If the item does not fit into any existing bin, we open a new bin to accommodate the new item. It is a classical result, that the asymptotic approximation ratio of FF is 1.7 [76]. It is a recent result that the absolute approximation ratio of FF is also 1.7, see [44]. It is usually assumed that if an algorithm has a small (i.e. close to 1) approximation ratio, then it is efficient. If for any $\varepsilon > 0$ we have an approximation algorithm A_ε for a problem with approximation ratio at most $1 + \varepsilon$, then the collection of such algorithms is called approximation scheme, if the step-number of A_ε is polynomial in $\frac{1}{\varepsilon}$. Such a collection of algorithms is called Polynomial Time Approximation Scheme (PTAS) in the absolute case (if A_ε has an absolute approximation ratio $1+\varepsilon$), or it is called Asymptotic Polynomial Time Approximation Scheme (APTAS) in the asymptotic case (if A_ε has an asymptotic approximation ratio $1 + \varepsilon$) [45].

1.5.3 Metaheuristics

There are algorithms that are considered *meta*, because they can efficiently solve a wide range of problems. Metaheuristics are high-level procedures used to utilize heuristics for finding near-optimal solutions to complex optimization problems. Metaheuristics are often *nature-inspired*, that is, they imitate processes found in natural systems. Metaheuristics can be defined into two classes, namely those that work with populations of solutions and those that don't. The common characteristics of all existing metaheuristics are as follows:

- The goal is to explore the solution space in order to find good (if not optimal) solutions.
- They are stochastic, which helps to deal with the combinatorial explosion of possibilities.
- They are inspired by analogies: with physics (e.g. Simulated Annealing (SA)), with biology (e.g. evolutionary algorithms, Tabu Search (TS)), or with ecology (e.g. Ant Colony Optimization (ACO)).
- They also share the same drawbacks: difficulty in tuning the parameters.
- They use the concepts of intensification and diversification.

Intensification means focusing the search in a local region, assuming that a good current solution is located in that region, and diversification means generating diverse solutions in order to explore the search space on a global scale.

The majority of metaheuristics explore the concept of a solution's neighbourhood. Metaheuristics based on neighbourhood structures start with a solution s and progressively search for another solution s' of better quality. When searching

for good solutions, it is generally undesirable to allow unrestricted movement from a current solution s to any other solution s' in the solution space, as this would lead to an exhaustive (and inefficient) search. Instead, the search is typically restricted to a subset of solutions, called the *neighbourhood* of s, which consists of those solutions that are directly reachable from s through a small, predefined modification.

Formally, for each solution $s \in S$, the set $N(s) \subseteq S$ defines the neighbors of s. A solution $s' \in N(s)$ is said to be reachable from s through a single modification. In general, not all neighbors of a given solution are accessible in practice due to computational or structural constraints. Therefore, an additional function is often defined to restrict the neighbourhood to a subset of acceptable or feasible modifications.

As an illustrative example, if a solution s is represented as a permutation of n elements, a common neighbourhood structure consists of all permutations obtained by swapping two elements in the vector. A local search algorithm explores such a neighbourhood iteratively, seeking an improving solution in $N(s)$, and moving to it when found.

If the entire search space S were known and efficiently enumerable, finding an optimal solution would be straightforward. However, in most real-world problems, the size of S grows exponentially with the input, making exhaustive search infeasible. Moreover, randomly jumping through a large search space without structural guidance is unlikely to yield good solutions in reasonable time. Therefore, neighbourhood-based methods such as local search provide a tractable way to navigate the space through gradual, informed exploration.

In classical local search, only improving solutions are accepted; that is, s' replaces s if it yields a strictly better objective value. However, in more advanced metaheuristics, such as *Simulated Annealing*, *Threshold Accepting*, or *Late Acceptance Hill Climbing*, even worse solutions may be accepted with a small probability. This mechanism promotes exploration and helps the search escape from local optima.

1.5.3.1 Classification of metaheuristics

Metaheuristics can be classified according to the following elements:

- **Nature-inspired or non-nature-inspired**: There are metaheuristics inspired by nature, such as the ant colony optimization, inspired by the movement of ants and their search for food, and the Genetic Algorithm (GA), based on the theory of evolution, etc. There are also algorithms that are not inspired by nature, such as Iterated Local Search (ILS), Greedy Randomized Adaptive Search Procedure (GRASP), Variable Neighbourhood Search (VNS), etc.
- **Single-point or population-based**: A method can process one solution at a time, like GRASP, SA, TS, and more, or process a population of solutions at each iteration, like GA, ACO, and ILS.
- **Single or multiple neighbourhood structures**: The majority of metaheuristics work with a single notion of neighbourhood, but there can also be algorithms like VNS, which uses multiple neighbourhood structures.

Among the most widely recognized and influential metaheuristics are the following algorithms, as discussed by Blum and Roli [24]:

1.5.3.2 Simulated annealing

Simulated Annealing (SA), proposed by Kirkpatrick et al. [80], is a metaheuristic that builds upon local search, but introduces mechanisms to escape local optima. Unlike basic descent-based approaches, SA occasionally accepts worse solutions to allow exploration of the solution space. This acceptance is governed probabilistically and depends on a temperature parameter: at high temperatures, the algorithm is more likely to accept degrading moves, enabling greater exploration, while at lower temperatures, it becomes increasingly selective. The probability of accepting a worse solution decreases as both the degradation and the temperature decrease. This behaviour mimics the annealing process in metallurgy, where a heated material is gradually cooled to reach a stable crystalline structure.

1.5.3.3 Tabu search

Tabu Search (TS) is a metaheuristic originally proposed by Fred Glover [64]. The method enhances local search by introducing adaptive memory structures, most notably the tabu list, which keeps track of attributes of recently visited solutions in order to avoid cycles and encourage diversification. In each iteration, the best admissible neighbor is selected, that is, a solution not prohibited by the tabu conditions, even if it leads to a worsening of the objective value. Unlike Simulated Annealing, Tabu Search does not rely on probabilistic acceptance; instead, it uses strategic memory and aspiration criteria to escape local optima and guide the search toward promising regions of the solution space. Practical implementations and extensions of Tabu Search have been explored by Hvattum and Løkketangen [71], who have applied TS in hybrid metaheuristics and combinatorial optimization, with a focus on adaptive strategies and empirical performance.

1.5.3.4 Genetic algorithm

Genetic algorithms (GAs), originally proposed by John Holland in his book *"Adaptation in Natural and Artificial Systems"* (1975) [70], are inspired by evolutionary principles in nature. These algorithms operate within the configuration space by evolving a population of candidate solutions through iterative processes of recombination (crossover), mutation, and selection. Each solution is treated as an individual, associated with an evaluation or fitness function. A set of such individuals constitutes a population. Crossover refers to the combination of two parent solutions to generate offspring, while mutation introduces random alterations to individual solutions to maintain diversity and avoid premature convergence.

Numerous hybrid forms of GAs have been developed, combining genetic principles with other optimization strategies. Examples include genetic algorithms combined with local search or simulated annealing. Notable implementations include the hybrid GA by Vidal [116] for solving complex vehicle routing problems (VRP), and the grouping genetic algorithm introduced by Falkenauer [53] for bin packing problems.

1.5.3.5 Ant colony optimization

Proposed by [42], the Ant Colony Optimization (ACO) algorithm is inspired by the way ants find food using pheromone trails. In this metaheuristic, artificial ants construct solutions, depositing pheromones proportional to solution quality. These artificial ants are stochastic procedures that generate candidate solutions, guided by artificial pheromone information and heuristic data from previous searches.

Each decision variable in ACO corresponds to a potential path an ant can take. Since paths are discrete, continuous decision variables must be discretized. At each iteration, paths are chosen based on pheromone concentration, often through a biased roulette wheel approach. Probabilities for path selection are calculated using both pheromone levels and local solution preferences, adjusted by user-defined weights. The pheromone update process rewards paths leading to better solutions by increasing pheromone levels on those paths, thus guiding future ants towards better solutions [92].

1.5.3.6 Variable neighbourhood search

VNS was proposed by Mladenovic and Hansen [98]. This is a stochastic metaheuristic that uses a perturbation concept (Shake procedure). A perturbation is characterized by a random exchange move in a solution vector. The procedure in this case is performed according to a number k. In other words, the larger the number of iterations without improvement, the larger the perturbation.

Bentsen and Hvattum [20] explore the application of Variable Neighbourhood Search (VNS) to Binary Integer Programming (BIP) challenges. Their research investigates whether VNS can effectively address BIP instances without relying on large neighbourhoods typically explored by external Mixed Integer Programming (MIP) solvers. The findings suggest that VNS-based methods outperform both exact and heuristic commercial solvers for certain problem types, particularly those that are tightly constrained. However, for other instance types, traditional solvers may still hold an advantage.

1.5.3.7 Greedy randomized adaptive search procedure

The Greedy Randomized Adaptive Search Procedure (GRASP), proposed by Feo and Resende [54], performs a certain number of iterations. At each iteration, a feasible solution is constructed by a randomized constructive heuristic. This solution is then improved by a local search. The best solution obtained is kept at the end of each iteration. First, an initial solution is randomly generated, then local search is applied for each solution. The best result obtained is updated.

1.5.4 Matheuristic

Matheuristic, proposed by Martinez-Sykora et al. [94], refers to a hybrid approach where the mathematical model of a problem is formulated, but solving it directly using a solver may be too hard. In this case, the model is divided into parts. Optimization is applied to one part of the model, while decision variables in other

parts are fixed. After that, we switch to another part of the model, keeping all other variables fixed, and optimize within the newly chosen part. This iterative process can often be very effective. Moreover, there are advanced versions of matheuristics, where the method is combined with metaheuristics for even better results.

1.6 How to define a model for a computational solver

It is common to formulate a mathematical model of the problem under consideration and then submit it to a commercial computational solver. When successful, the solver returns a solution. Examples of such solvers include IBM's CPLEX and the Gurobi Optimizer by Gurobi Optimization, LLC. An academic version is usually available, allowing free use for non-commercial purposes. The solvers differ in efficiency. In this thesis, the academic version of CPLEX, which is part of IBM ILOG CPLEX Optimization Studio, is used as a verification tool for the models.

This section illustrates that, in some cases, a solver can be aided by simple modeling refinements or practical *tricks*. CPLEX is configured as a black-box engine that runs multiple algorithms in parallel and returns the first available solution; its MIP engine typically relies on LP relaxations and branch-and-cut, among other techniques.

As an example, consider the Bin Packing Problem (BPP). A collection of n items is given, where the size of item i satisfies $0 < s_i \leq 1$. A collection of bins is also given, each with capacity 1, and a sufficient number of bins is assumed (e.g., at least n bins). The aim is to pack the items into as few bins as possible, subject to the constraint that the total size of items in any bin does not exceed its capacity. The classical binary linear model is as follows. Let x_{ij} be a binary variable that equals 1 if item i is packed into bin j and 0 otherwise, and let y_j be a binary variable that equals 1 if bin j is used and 0 otherwise. The objective z is the number of used bins, which is minimized. The following binary linear model is obtained:

$$\min z = \sum_{j=1}^{n} y_j \tag{1.5}$$

$$\text{st.} \quad \sum_{j=1}^{n} x_{ij} = 1, \qquad \forall i \tag{1.6}$$

$$\sum_{i=1}^{n} s_i \cdot x_{ij} \leq 1, \qquad \forall j \tag{1.7}$$

$$\sum_{i=1}^{n} x_{ij} \leq M \cdot y_j, \qquad \forall j \tag{1.8}$$

The objective function is given in formula (1.5). Constraint (1.6) ensures that each item is packed into exactly one bin, while constraint (1.7) guarantees that the total size of items in each used bin does not exceed its capacity. Finally, constraint (1.8) ensures that if bin j is used, then the binary variable y_j must be set to 1. The constant M represents a sufficiently large value; in practice, it is enough to set

$M \geq 3$ for the instance considered below. Minor variations of this model exist, but the core formulation remains essentially the same.

It is worth noting that some instances may be too difficult for a solver to handle directly. However, if the solver is assisted, for example, by exploiting certain structural properties of the instance—the problem may then be solved successfully.

Let us consider the following instance, which is the *Falkenauer_t120_00* instance [53].

497, 497, 495, 485, 480, 478, 474, 473, 472, 470, 466, 450, 446, 445, 445, 444, 439, 434, 430, 420,
419, 414, 412, 410, 407, 405, 400, 397, 395, 376, 372, 370, 366, 366, 366, 366, 366, 363, 363, 362,
361, 357, 357, 356, 356, 355, 352, 351, 350, 350, 350, 347, 336, 333, 329, 325, 320, 315, 314, 313,
307, 303, 302, 301, 299, 298, 298, 298, 295, 294, 292, 290, 288, 287, 283, 282, 282, 276, 275, 275,
274, 273, 273, 272, 272, 271, 271, 269, 269, 268, 267, 267, 266, 263, 263, 262, 262, 261, 260, 259,
259, 259, 258, 256, 255, 254, 254, 254, 253, 253, 253, 253, 252, 252, 252, 252, 251, 251, 250, 250.

There are 120 items, each with a size between 250 and 500, and only two items are of size 250. The bin capacity is normalized to 1000. It means that no four item fit into the same bin, and any two items fit into the same bin. It can be known in advance (because the instance is generated in this way) that the items fit into 40 bins. All these bins will be fully packed. Then the problem is to find 40 *good* triplets.

If the solver is run with the model and instance data described above, the problem is found to be too difficult for the solver, as it runs for several minutes. The parameters of the computer used are not material in this context, since the purpose is solely to compare the outcome of running the solver "purely" with the outcome obtained when additional assistance is provided. It can be observed that certain rules can be established for how items may be grouped into valid triplets (or, more precisely, how they cannot be grouped). For example, if the first two items (i.e., the two largest) are placed in the same bin, even the smallest item will not fit, since $497 + 497 + 250 = 1244 > 1000$.

Similarly, it can be observed that no three items among the first 30 can be packed together in the same bin. For instance, $395+376+250 = 1021 > 1000$, which exceeds the bin capacity. However, for the next pair of items, $376+372+250 = 998 \leq 1000$, so these do fit within a bin. This observation implies that the first 30 items must each be placed in separate bins. By incorporating this insight into the model—for example, by enforcing that these items are assigned to different bins—the solver's search space can be reduced significantly. As a result, the optimal solution can be found very quickly, often within just a few seconds.

This means that the formulation of the model matters, and that a problem's model can sometimes be represented in significantly different ways. It is also important that the optimization be simplified where possible, in order to facilitate the solver.

1.7 Regarding $\mathcal{NP}$-hardness

$\mathcal{NP}$ is a class consisting of all problems that can be solved in polynomial time on a nondeterministic finite-state machine (FSM). This means: 1) if the machine *guesses* or arrives at the answer without necessarily finding/generating all possible solutions, then 2) it can be checked/verified in polynomial time whether the answer is correct. However, there is of course no guarantee that the machine *guesses* correctly. If the problem were to be solved systematically/completely, exponential time would be required.

In this thesis, the term $\mathcal{NP}$*-hardness* is encountered (see the intersection of $\mathcal{NP}$-hard and $\mathcal{NP}$ in Figure 1.7). By definition, $\mathcal{NP}$-hard problems form a class that is at least as difficult as the hardest problems in $\mathcal{NP}$, but they do not have to belong to $\mathcal{NP}$ themselves and may even be unsolvable, i.e., without algorithms that solve them completely.

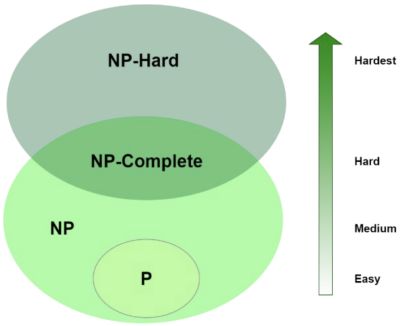


Figure 1.7: Illustration of the complexity classes $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-Complete, and $\mathcal{NP}$-Hard. The classes are arranged by increasing computational difficulty, with $\mathcal{P}$ representing problems solvable in polynomial time, $\mathcal{NP}$ containing problems verifiable in polynomial time, $\mathcal{NP}$-Complete representing the hardest problems in $\mathcal{NP}$, and $\mathcal{NP}$-Hard including problems that are at least as hard as $\mathcal{NP}$-Complete. Adapted from Baeldung [11].

They can be seen as a generalization of $\mathcal{NP}$-complete problems, where an $\mathcal{NP}$-hard problem is at least as demanding as an $\mathcal{NP}$-complete problem. If we could solve an $\mathcal{NP}$-hard problem efficiently, then we could also solve all problems in $\mathcal{NP}$ efficiently. $\mathcal{NP}$-hard problems need not be verifiable in polynomial time and may have solutions that take more than polynomial time both to find and to verify. A problem is classified as $\mathcal{NP}$-hard if any problem in $\mathcal{NP}$ can be reduced to it in polynomial time, making $\mathcal{NP}$-hard problems a broader class than $\mathcal{NP}$ [61].

A naive definition may be stated as follows: if a problem is $\mathcal{NP}$-hard, then it is unlikely that there exists an algorithm that finds an optimal solution for every input in polynomial time. Since all problems considered in this thesis are $\mathcal{NP}$-hard, they are solved using a commercial solver, metaheuristic methods, or heuristic algorithms. This approach does not guarantee that optimal solutions will be obtained for all instances; however, for many instances optimal or near-optimal solutions can still be achieved.

1.8 Generally about packing problems

Packing problems are encountered across a wide range of scientific and industrial domains, including manufacturing, storage and warehousing, transportation, finance, and construction. When solved to optimality, such problems can result in significant cost savings and more efficient resource utilization.

In Chapter 2, a novel variation of the classical two-dimensional bin packing problem is introduced, in combination with principles from facility location theory. This formulation is referred to as the *Board Packing Problem* (BoPP).

In Chapter 3, another class of packing problems is examined, specifically the packing of squares into a larger square. This problem was first studied in the 1950s by Martin Gardner, who brought attention to the question of determining the minimum side length of a square into which the set of squares with sizes $1, 2, \ldots, n$ can be packed without overlap. Since its introduction, continued interest has been shown in the academic community in developing improved solution approaches to this geometric optimization problem.

General packing problems involve assigning objects of varying sizes and/or shapes to limited spaces in a way that minimizes waste or maximizes profit or efficiency. A fundamental variant of these is the one-dimensional bin packing problem, where n items with given weights must be packed into a minimum number of identical bins, each with fixed capacity. The objective is to minimize the number of bins used without exceeding the capacity constraint in any bin. This classical combinatorial optimization problem is known to be $\mathcal{NP}$-hard and has been studied extensively, including in the work of Scheithauer [109].

Being a $\mathcal{NP}$-hard problem, finding an exact solution in reasonable time is generally impractical for large instances. Instead, heuristic algorithms and approximation techniques are often used to find good (but not necessarily optimal) solutions that can be computed efficiently.

Since the simplest version of the bin packing problem is proven to be $\mathcal{NP}$-hard, it implies that all its generalizations are $\mathcal{NP}$-hard as well. Any more complex variation, such as multi-dimensional packing (e.g., 2D or 3D bin packing), variable bin capacities, or additional constraints, extends the fundamental structure of the original $\mathcal{NP}$-hard problem. In complexity theory, if the base problem is $\mathcal{NP}$-hard, any additional constraints or extensions added to the problem only increase its difficulty, making the variations equally or more computationally challenging. Of course, this is also applicable to the BoPP. In the subsections below, the main variations of packing problems will be presented.

1.8.1 One-dimensional packing problems

The first paper on the topic of 1D Bin Packing was written by Ullman [115]. Johnson [76] introduces several approximation algorithms for the BPP. It was the pioneer work in the approximation algorithms field in the literature. Johnson deals with the well known algorithms Next Fit (NF), First Fit (FF), Best Fit (BF). And their ordered versions, when the items are sorted initially into non-increasing order or

weight: First Fit Decreasing (FFD), Best Fit Decreasing (BFD), and other algorithms.

In the early 1970s, tight asymptotic approximation ratios were established for several heuristic algorithms for the bin packing problem: 1.7 for *First Fit* (FF) and *Best Fit* (BF), and 11/9 for *First Fit Decreasing* (FFD). *Next Fit* is one of the simplest algorithms: it maintains a single open bin and assigns items to it sequentially. If the current item does not fit in the open bin, that bin is closed, and a new bin is opened for subsequent items. The running time of Next Fit is $O(n)$, where n is the number of items in the list. For an extensive review of algorithms for the bin packing problem, see Coffman et al. [32] or [33].

Assmann et al. [8] formally introduces what we call the Bin Covering Problem (BCP), that is a dual version of the BPP, but the goal is to cover as many bins as possible. A bin is covered, if the total size of items, assigned to that bin, is at least 1. Similarly to BPP, BCP is also $\mathcal{NP}$-hard. There are many industrial and real-world applications of 1D bin packing, such as:

- Container Loading: load goods of various sizes into shipping containers (trucks, ships, or airplanes) in order to maximize space utilization while adhering to weight or volume constraints.
- Broadcast Channel Allocation: allocate TV or radio programs into schedules, maximizing the number of channels or programs aired without overlapping.
- Portfolio Optimization: choose a mix of investments into a portfolio, balancing risk and return in a way that maximizes profit while adhering to certain constraints (e.g., minimum acceptable return, diversification).

1.8.2 Two-dimensional packing problems

The two-dimensional packing problem refers to the optimal arrangement of items to be packed under the two orthogonal directions, and the packing is loaded into the space with constraints.

In the 2-dimensional bin packing problem, there is a set of rectangles, and the target is to pack all rectangles into squares in a way that no rectangles overlap, and each rectangle must be within the square where it is packed, while minimizing the number of used square bins.

There are two main versions of 2D packing: 2D bin packing without rotation (often called Orthogonal Packing Problem), see Figure 1.8, and 2D bin packing with rotation (often called 2D Rotational Packing Problem), depending whether the rectangles can be rotated or not. Among the rotational packing problem, there are cases when only 90-degree rotation is allowed or rotation is free.

Hochbaum et al. [69] shows a unified technique, referred to as the shifting strategy, that is applicable to numerous geometric covering and packing problems and it is an approximation algorithm. Lodi et al. [89] is a very detailed survey on the 2D bin-packing. It reviews various algorithms, including exact, heuristic, and approximation algorithms for both rotational and orthogonal types.

Caprara et al. [27] focus on the orthogonal 2D problem, considering the natural relaxation of 2KP, which is derived from the one-dimensional KP by assigning item weights equal to the rectangle areas. They prove the worst-case performance

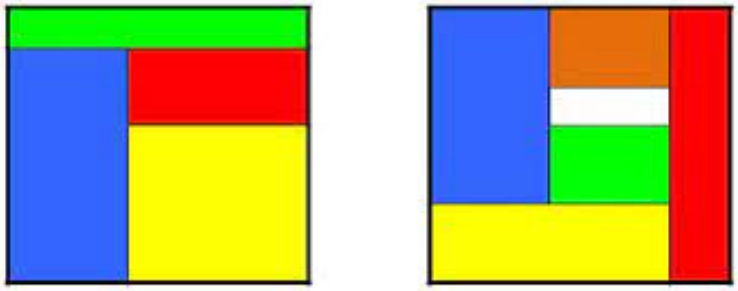


Figure 1.8: 2D bin packing

of the associated upper bound and compare it computationally against four exact algorithms based on this relaxation, demonstrating their effectiveness. Additionally, various other versions of the 2D bin packing problem are discussed. Fowler et al. [55] consider various packing and covering problems in the plane and provide proofs of $\mathcal{NP}$-completeness.

1.8.2.1 2D strip packing

It is a variant of the bin packing problem where the objective is to pack a set of rectangular items into a single strip of fixed width but infinite height, in such a way that the total height used is minimized. Overlap is not allowed. Rotation is either allowed or not allowed Coffman et al. [31]. This variation is frequently found in fabric industry, where strips of fabric are cut from large rolls. The 2D strip packing problem is used to optimize the cutting process by arranging garment pieces (rectangular or irregular shapes) onto strips of fabric with minimal waste.

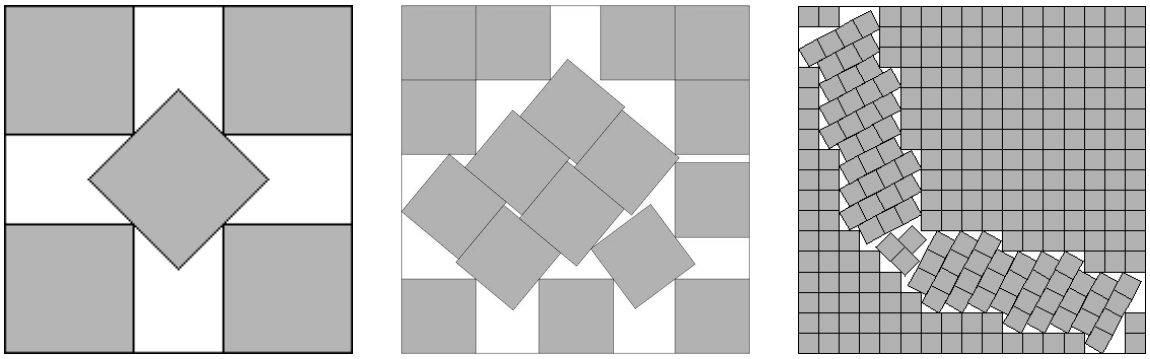
1.8.2.2 Facility location

In the facility location problem, we are given a set of facilities and aim to cover as much demand as possible. Among other approaches, some papers consider facilities as rectangular areas. The objective is to optimally position these rectangles to maximize demand coverage within the feasible region.

The possibly first paper on this topic is by Church and ReVelle (1974) [30], and since then, numerous papers have appeared in the literature addressing various types of location problems, including the work by Blanquero et al. (2016) [22], and the geometric covering model by Megiddo and Supowit (1984) [96]. A more detailed discussion on this topic will be provided in Chapter 2.

1.8.2.3 Packing equal-sized squares into a larger accommodation square

The Erdös-Graham problem [52] is the following: Given n unit squares, rotation is possible (freely), let us pack the items into the smallest possible square. For some small n values the problem is in fact easy. In the Figure 1.9a, is the optimal solution for $n = 5$. In the Figure 1.9b, a feasible solution for $n = 17$. (Source: Internet, searching for square packing). Here we can see that the problem is trivial



(a) 5 squares (b) 17 squares (c) 27 squares

Figure 1.9: Square packing

if no rotation is allowed, but far from easy if rotation is allowed by any degrees, see Figure 1.9c.

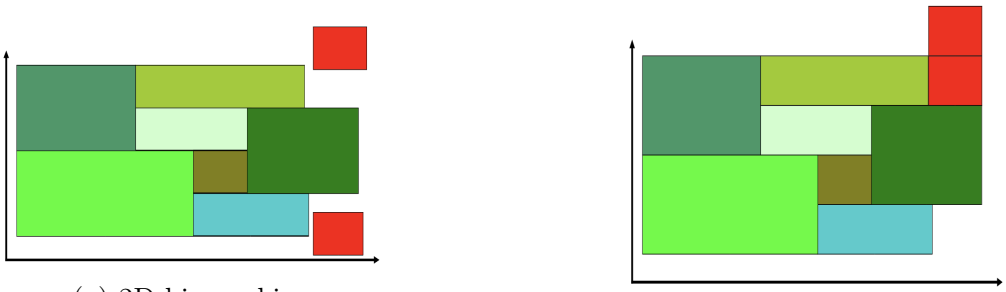
1.8.2.4 Rectangle packing with an application

There are two main versions, whether the rectangles can be rotated or they cannot be rotated. If they can be rotated, this is possible only with 90°.

One application is as follows: Each rectangle corresponds to a job. The horizontal size (width) of the rectangle means the duration of the job, and the vertical size (height) of the rectangle means its resource usage. We suppose that only one resource is needed to complete the job, and this is the needed resource for all jobs. Then one square (where we pack the rectangles) means one working day. The horizontal size (normalized to 1) of the square means one day, and the vertical size of the rectangle means the available amount of resource, that is used by all rectangles (jobs) assigned to this square. Then we want to execute all jobs in the minimum number of working days.

It should be noted that a slight simplification was made in the previous application. In fact, it is not necessary that the rectangles that are assigned to certain square, fit into the square in the vertical direction (but they must fit in the horizontal direction). The reason is, that it is only necessary that the sum of the resource usage must be at most the resource capacity in each time moment. Realizing this phenomenon, we have a relaxed problem, called *Relaxed 2D Bin Packing (2DBP)*. A well-known example for the difference of the two problems can be seen on Figure 1.10 by Mihaly Hujter. In this example there are 8 (or 9) rectangles. The height of each rectangle is either 1 or 2. The width of the two items on the left size are 4 and 6. The width of the not red rectangles with height 1 are as follows from bottom to top: 4, 2, 4 and 6. There is a further rectangle of sizes $4 * 2$, and two small red rectangles of size $2 * 1$.

Now, the nine rectangles fit into a big rectangle of sizes $12 * 4$. But if the two red items are are put together into a $2 * 2$ rectangle, then the eight rectangles cannot be packed into the $12 * 4$ big accommodating rectangle. It means, that the optimal solution of the 2DBP problem (where the accommodation rectangle has normalized sizes $12 * 4$) is 2 bins.



(a) 2D bin packing

(b) Relaxed 2D bin packing

Figure 1.10: Comparison of 2D bin packing and Relaxed 2D bin packing

But the optimal solution of the relaxed 2DBP problem is 1 as can be seen in Figure 1.10b. Here, it does not matter that the used height of the figure is 5 (more than 4), as the total size in each vertical direction is only (at most) 4. Sometimes we want to pack squares into a rectangle, see e.g. Hougardy [72], but this is another problem, see also Novotný [103, 104].

1.9 Generally about covering problems

Covering problems form a fundamental class of combinatorial optimization problems in which the objective is to ensure that a set of demand points, targets, or regions is sufficiently *covered* by a selected subset of available resources.

Covering models are closely related to packing problems, but with a fundamentally different goal: while packing seeks to fit disjoint items into limited space without overlap, covering aims to guarantee that all targets fall within the influence range of deployed resources, possibly allowing overlaps or redundancies. In many applications, a trade-off exists between coverage quality and resource efficiency, often modeled through constraints or multi-objective formulations.

Covering problems appear in numerous domains, including operations research, computer science, telecommunications, and public service planning. A general covering problem is defined over two sets: a set of elements that require coverage (e.g., geographic locations, customers, or data points), and a set of available covering units (e.g., facilities, sensors, or disks) that can be deployed at a cost. The goal is typically to select a minimal subset of covering units such that each element in the ground set is covered by at least one selected unit, according to some feasibility condition (such as distance or connectivity). In weighted versions, the aim is to construct a cover of minimum total weight.

Typical applications include the placement of emergency facilities (e.g., fire stations or ambulance units) [30], sensor networks for surveillance and environmental monitoring [69], mobile base stations in telecommunications [6], public service infrastructure such as schools or polling stations [69], and energy distribution, for example, the optimal siting of charging stations or substations [120].

Several important variants exist. In the *Set Covering Problem (SCP)*, given a ground set (or universe) and a collection of subsets (covering units), the goal is to

find the smallest number of such subsets whose union covers the universe. The classical $\mathcal{NP}$-completeness of SCP and early algorithmic approaches are well known in the literature. In related contexts, Csirik et al. [35] analyzed two simple algorithms for the *bin covering problem*, a dual of bin packing that shares structural similarities with set covering. Their work demonstrates how relatively simple algorithmic strategies can achieve provable performance guarantees, thus contributing to the broader understanding of covering models.

In the *Maximum Covering Location Problem (MCLP)*, given facilities, demand points, and a budget B, the budget constraint limits the number of facilities that may be deployed, and the objective is to maximize the number of covered demand points. Approximation approaches have been developed in related covering contexts. For example, Csirik et al. [36] presented improved approximation algorithms for the *bin covering problem*, which is closely related to maximum covering models. They showed that it is possible to design polynomial-time approximation algorithms with significantly better performance guarantees than previously known methods, thereby contributing to the theoretical foundations of covering optimization.

Other well-known problems include the *Disk Covering Problem* [69], and *k-Covering*, where each element must be covered at least k times (e.g., connected k-coverage in sensor networks) [121].

1.9.1 Bin covering

There are many more papers in the literature considering Bin Packing (BP) than Bin Covering (BC). BC is $\mathcal{NP}$-hard (similarly to BP), and its investigation started with the dual formulation introduced by Assmann et al. [8]. In BC, given items and bins, the goal is to assign the items to bins such that each bin is *covered*, that is, the total size of the items assigned to each bin is at least the bin capacity, while maximizing the number of covered bins. Many bin packing algorithms can be adapted to BC (after some natural modification) [35], and asymptotic PTAS results also exist [36].

1.9.2 Geometric covering problems

In computational geometry, covering problems also arise in determining how a shape or area can be entirely covered by simple geometric figures such as circles, rectangles, or squares.

In Chapter 3 of this thesis, a *square covering problem* [15] is studied, in which a sequence of squares of increasing sizes is used to cover a larger square area. The goal is to determine the largest square that can be completely covered by the items, where overlap is allowed but the items must remain axis-aligned. To the best of my knowledge, such a square covering problem has not been investigated earlier.

1.10 Interface for BoPP

The tool IBM ILOG CPLEX is used for modelling, and is done using CPLEX Optimization Programming Language (OPL). Here I address a geometric optimization problem, specifically placing rectangles on a board. Visualization plays a crucial

role in this process, as observing the placement of items makes it easier to evaluate the effectiveness of my algorithms. To aid in this, I have developed a Python-based interface, since CPLEX offers an application programming interface (API) for Python. This interface serves multiple purposes: it allows for visualizing the solution, running CPLEX models, my own implementations of optimization algorithms, and importing/exporting problems and their solutions.

The Board Packing application also incorporates both heuristic and non-heuristic algorithms; however, the focus of this presentation is placed on CPLEX as the primary solver.

Two mixed-integer programming (MIP) models are considered:

- If rectangle overlap is not allowed, see Figure 1.11
- If rectangle overlap is allowed, (the focus of this thesis), see Figure 1.12

Figure 1.11: A $M \times N$ board with $K = 8$ squares, non-overlapping, and the input cell gain g is defined within the interval $[-9, 9]$. A solution using 6 squares yields a positive profit.

Several aspects of CPLEX's capabilities were tested through the developed interface. The benchmark set was designed from scratch to systematically investigate how different problem characteristics influence instance difficulty and solver performance.

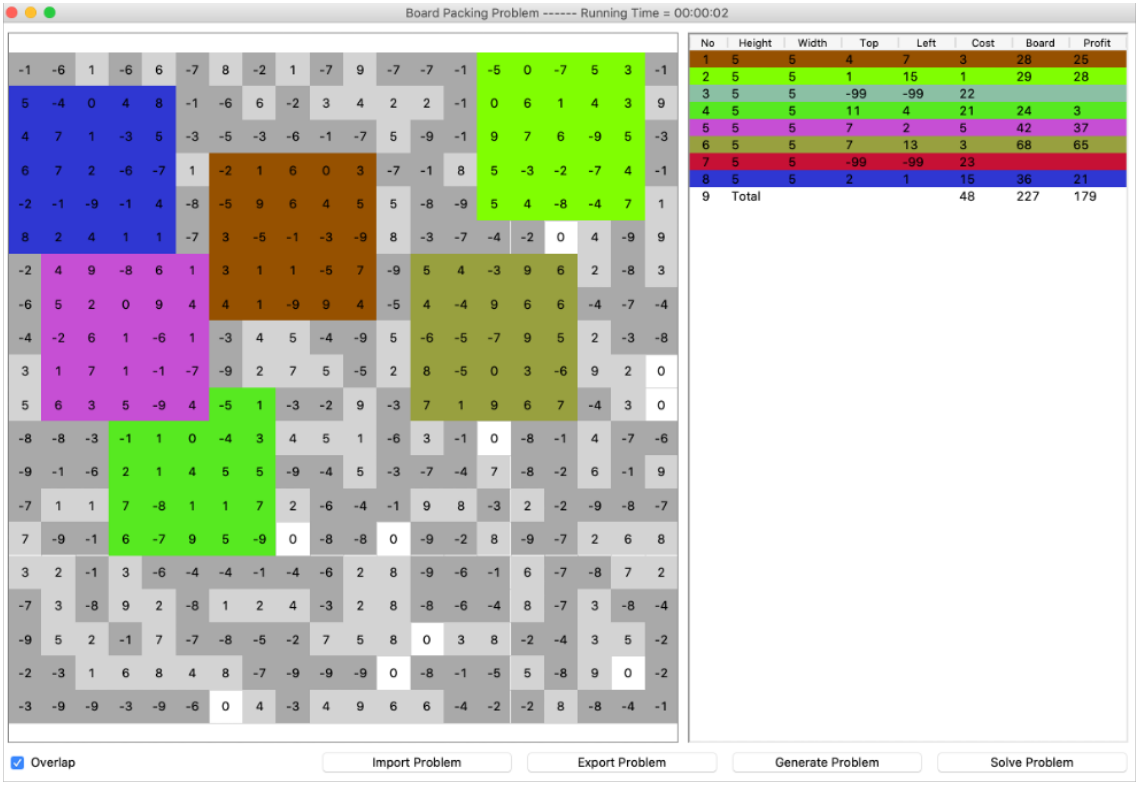


Figure 1.12: Here is the same board with overlapping allowed. A solution using 6 squares is also found to give some positive profit. The total profit is slightly better.

Specifically, the following aspects were examined:

- the effect of increasing the board size,
- the impact of increasing the number of rectangles,
- the use of rectangles with varying sizes and costs,
- and the structure of the gain matrix, including irregular and negative values.

To ensure diversity and generalizability, the benchmark set includes:

- boards of varying dimensions,
- rectangles with heterogeneous characteristics,
- and gain matrices ranging from fully positive to mixed and fully negative.

A real-world application scenario was also included, based on satellite image coverage. In this setting, rectangles represent image tiles, and gain values reflect the coverage utility of each tile. These benchmarks were used to evaluate and compare the performance of the proposed evolutionary algorithm against CPLEX. They provide a structured and reproducible testbed for measuring scalability and performance under increasing problem complexity. The results of these experiments are presented and discussed in Chapter 2. More board figures in Appendix A.

Chapter 2

The Board Packing Problem (BoPP)

Publications

The following publications form the foundation for my research discussed in this chapter and provide essential background and context:

- Abraham Gy., Dosa Gy., Hvattum L. M., **Olaj, T. A.**, Tuza Zs.: The board packing problem. *European Journal of Operational Research*, volume 308, issue 3, pages 1056–1073, `https://doi.org/10.1016/j.ejor.2023.01.030`, **D1 journal, IF=6.365**, 2023, see [2].
- Other related presentations include, [3, 47].

2.1 Introduction

Definition 1. *A rectangular board with m rows and n columns is given. Let $M = \{1, \ldots, m\}$ be the set of rows and $N = \{1, \ldots, n\}$ be the set of columns. Each position $(i, j) \in M \times N$ of the board has an integer value $g_{i,j}$ representing a revenue to be obtained if the position is covered. A set R of rectangles is given, each $r \in R$ with a given height h_r, width w_r, and cost c_r. The objective is to purchase rectangles to place on the board so as to maximize the profit, which is equal to the sum of the revenues of the covered positions, minus the cost of purchased rectangles. The revenues, heights, widths, and costs are expressed as integers. Rectangles must be placed with sides parallel to the sides of the board. They are allowed to overlap, but the revenue from a given board position can only be collected once.*

The previous paragraph describes the *Board Packing Problem* (BoPP), which is the object of study of this chapter. Figure 2.1 illustrates an instance of the problem on a board with $m = 6$ rows and $n = 5$ columns. The revenues $g_{i,j}$, which may be negative, are given for each board position.

Assuming that two rectangles are available for purchase, with $h_1 = w_1 = 2$, $h_2 = 3$, $w_2 = 2$, and $c_1 = 22$ and $c_2 = 33$. Figure 2.1 also illustrates an optimal solution producing a profit of 51. The rectangles are placed with overlap, as other solutions are less profitable. Another example is shown in Figure 2.2.

In this example, the gain matrix contains only entries 0 or 1. Suppose two rectangles (both squares of side length 3) are available, each with zero cost. The

9	13	11	4	13
18	-10	18	1	19
8	-14	4	13	9
4	5	2	18	13
11	8	13	-18	12
8	1	11	13	15

9	13	11	4	13
18	-10	18	1	19
8	-14	4	13	9
4	5	2	18	13
11	8	13	-18	12
8	1	11	13	15

Figure 2.1: Example of a small instance of BoPP to the left, and a solution placing two rectangles to the right.

0	0	0	0	0	0
0	1	1	1	0	0
0	1	1	1	0	0
0	1	1	1	1	0
0	0	1	1	1	0
0	0	1	1	1	0
0	0	0	0	0	0

Figure 2.2: A small example with gain values only 0 or 1.

maximum profit is achieved only when both squares are purchased and placed, and their placements overlap.

The problem is first defined in Dosa et al. [47], where the special case was investigated when rectangles are not allowed to overlap. This first publication in the topic presents a mathematical model and applies a commercial mixed-integer programming solver (CPLEX) to find optimal solutions on a small set of test instances. The problem is studied in greater detail in [2]. The problem and its investigation is also presented in [3]. The present chapter is based on these above mentioned publications, and on further investigations that are not published yet.

2.1.1 Application

An application of the BoPP is when the board represents some area in a city or a country. The rectangles represent facilities that can be built in certain positions. The covered g values represent the profit obtained from the activity of the facilities. In some cases, squares may be considered, and a natural question arises as to why squares, rather than circles, are used to cover the areas. The answer is that in several cases the *circle* means in fact a square, if the considered metric is not the Euclidean norm but the L_1 norm. In a city this can be a natural choice of the norm, a claim supported by the fact that this norm is also called as *Manhattan norm*.

- if $g_{i,j}$ is negative: This means that it is preferable to not cover the point, for example because the location is already inhabited or owned by some person or company. If it is nevertheless covered, a penalty or expropriation cost must be paid.

- the $c_k \geq 0$ purchasing cost of rectangles: This is the cost of installing a new facility (some other research when purchasing a resource is considered: Scheduling with machine cost, see e.g. Dosa and He [43] or [4]).

2.1.2 A classical model in relation to the BoPP

The first model in the literature to address a *similar* problem is likely the Maximal Covering Location Problem (MCLP), defined by Church and ReVelle [30]: A finite set I of users is given, each user $i \in I$ with a given demand $a_i \geq 0$. Also given is a set of J of facilities. The task is to select p facilities from the set J so as to maximize the covered demand. A point i is covered if there is at least one chosen facility so that its *distance* from i is not bigger than R, the fixed positive cover radius. The distance between two points is measured by some norm if the demand points are in the two dimensional plane, or it can be the shortest distance if the demand points are given in a graph. A mathematical programming formulation of the MCLP problem is given by [30]:

$$\max \sum_{i \in I} a_i y_i \tag{2.1}$$

$$\sum_{j \in N_i} x_j \geq y_i, \text{ for all } i \in I \tag{2.2}$$

$$\sum_{j \in J} x_j = p, \tag{2.3}$$

$$x_j \in \{0,1\} \qquad \text{for all } j \in J$$
$$y_i \in \{0,1\} \qquad \text{for all } i \in I$$

The meaning of the notations and constraints is the following: $y_i = 1$ iff the demand point i is covered. Moreover $x_j = 1$ iff facility j is chosen. The number of chosen facilities is exactly p. Let N_i mean the neighborhood of demand point i so that if a facility is placed within this neighborhood then the demand of point i is covered. Then (2.2) enforces that each demand point is served by at least one facility, and (2.1) sums the total covered demand. There are many methods to solve optimally or near-optimally the problem in different settings, see e.g. [107].

The BoPP problem belongs to this class of problems, but there are several differences: In my BoPP model R (the cover radius) is not fixed: it is smaller for a smaller square and bigger for a bigger square, so in my case R depends on the facilities. On the other hand, the BoPP is a special case in the sense that the demand points are in a grid.

From the time of introduction of the model by Church and ReVelle [30] a huge number of concerning papers is published. The introduction of [23] claims that MCLP is known to be $\mathcal{NP}$-hard. This claim is supported by a reference to a paper of Megiddo et al. [97]. In fact, [97] considers a model where the demand points are vertices of a graph, and the facilities should be put into the points of the graph (where a point of a graph can be some vertex or some inner point of some edge also). In this version the model is clearly $\mathcal{NP}$-hard because it is $\mathcal{NP}$-hard to find a minimal dominating set in a graph (let the demand points be the vertices, all a_i

weights are unit, and the covering radius is uniformly $R = 1.5$.) This $\mathcal{NP}$-hardness result holds for general graphs, but not for grid instances in which the board is to be covered by squares. Consequently, the reduction in [97] does not directly establish the computational complexity of the BoPP. This question is addressed in the next section of this chapter.

The paper of Church and ReVelle [30] is also an early adaptation of linear programming with relaxation and branch and bound from 1974. The paper contains two simple algorithms. These are Greedy Adding (GA), which finds a best place for a facility, then finds the best place for the second one, and so on. The other algorithm is Greedy Adding and Substitution (GAS). This is similar to GA, but also tries to repack some previously placed facilities into some better position. Its third method is the following: Write up the MILP model, apply B&B for branching, and apply the LP-relaxation in the decision tree as upper bounds. Nowadays the modern MIP solvers like CPLEX do this method automatically, but it was an early and nice contribution in 1974. The literature relevant to the BoPP is reviewed in the following.

2.1.3 Other related problems in the literature

The BoPP is closely related to several other problems, such as the rectangle blanket problem [41], which has applications in computer vision problems, including template matching and people tracking. When overlap is allowed in the BoPP, the motivation is to represent applications in which rectangular resources can be purchased and applied on a grid to obtain a benefit. For example, the rectangles could represent satellite images [74] that can be taken to obtain useful or profitable information about geographical regions, with uses ranging from post-disaster damage assessment [62] to highway road maintenance [66] or detection of archaeological features [100]. Demiröz [41] also discussed problem variants within covering and partitioning problems as well as cutting and packing problems, to highlight their relationships to the rectangle blanket problem.

Several papers consider the case where partial covering of the customer demands is possible, see e.g. Berman and Kras [19] or Blanquero et al. [23]. Thereafter, the focus is restricted to models in which each demand point is either completely served or not served.

Many papers also consider *similar* problems. Mukherjee and Chakraborty [102] considered a set of given points in the plane that must be covered by a given number of non-intersecting rectangles. The size of each rectangle can be set freely, and the objective is to minimize the area of the rectangles used to cover the points.

In [96], given n demand points in the plane, and the next two, classical problems are considered: the p-center problem is to find p supply points (anywhere in the plane) so as to minimize the maximum distance from a demand point to its respective nearest supply point while the p-median problem is to minimize the sum of distances from demand points to their respective nearest supply points. Both problems are proven $\mathcal{NP}$-hard, no matter whether Euclidean or rectilinear metrics are applied.

Leung et al. [88] consider the problem where given some small squares, and given also a big square, and the question is whether the small squares can be packed

(without overlapping) into the big square. The paper proves that this problem is strongly $\mathcal{NP}$-hard by applying a reduction from 3-partition. This model is connected with BoPP in the following way: The gain value for their model for each position is $g_{i,j} = 1$, any rectangle is in fact a square, and the cost of purchasing any square is zero. If all squares can be packed without overlap then the total gain will be mn otherwise the total gain is strictly smaller. But this problem differs from the board packing model in several respects. In the model of Leung et al., every cell of the large square has a gain value equal to 1, so the encoding of the input is different from the case where each cell may have its own value. Moreover, the covering items are restricted to squares (rather than arbitrary rectangles), and the costs of the items are defined differently. The distinction highlighted above is therefore only one of several differences.

Mahapatra et al. [91] dealt with a problem where the goal is to cover several demand points in the plane, the covering items are a fixed number of disjoint unit squares, and the objective is to maximize the number of covered demand points. The problem is polynomially solvable. A general typology for cutting and packing problems without overlap was given by Wäscher et al. [119]. In bin packing problems, the goal is typically to minimize the number of surrounding containers used [94], rather than to maximize the profit of items placed within a given container.

The focus is restricted to models defined on a finite grid (the *board*) embedded in the two-dimensional plane, with integer coordinates. Each point (cell) has an associated gain value specified individually. The goal is to cover a subset of demand points so as to maximize the total gain, using covering shapes (rectangles or squares, possibly of different sizes), referred to as *items*. Items may be purchased for use in the cover. Depending on the variant, items may be allowed to overlap, or overlap may be disallowed.

Two main lines of research are distinguished from this point onward. The first consists of geometric covering models: given a rectilinear object in the plane, the task is to cover it with smaller items. The second encompasses the broad class of facility location problems.

2.1.3.1 Facility location models closely related to the BoPP

Facility location problems constitute a first direction of related research, where the correspondence to BoPP arises when considering the area covered by a facility as a rectangle, and the board positions as demand to be covered with specified revenues.

There is a huge number of papers that consider some kind of facility location or in other words covering problems. The homepage of Trevor Schuyler Hale lists 3400 such papers [114] (without any comment on them). ReVelle et al. [106] gives also survey of many related papers which are categorized. In case of the p-median problem p facilities are placed in a metric space so that the weighted distances from the demand points should be as small as possible. Clearly, this kind of problem is quite different from my BoPP model.

Some other problem types that are discussed in [106] are referred to as center and covering problems. For some problems it is not an appropriate objective to consider the weighted total distance from the allocated facilities, e.g. in emergency

services, where the worst served demand point should be served as well as possible. One such paper is [50].

The facility location models are divided into *continuous* and *discrete* models (the BoPP model clearly fits into the discrete models).

In certain cases, the rectangles to be allocated are not fixed in shape. As shown in [26], these rectangles can be flexible, provided their area remains constant. Çakir and Wesolowsky [26] studied a continuous facility location problem where facilities are represented as rectangles. The objective is to determine both the location and shape of these rectangles to minimize the cost of expropriated points, see also [18].

Blanquero et al. [22] considered a similar problem, that aims at locating facilities so as to maximize the market share. While that problem is deeply studied in the field of continuous location, they study the problem on networks, formulated as a mixed integer nonlinear programming problem that can be solved by a B&B algorithm. Some researchers have considered the case where a partial covering of customer demands is possible [19], but in BoPP the board positions are either completely covered or not covered.

Another option is to serve *all* demand points, but with as few facilities as possible. This case is similar to the problem of covering a rectilinear object with as few rectangles as possible.

Possibly the model of the paper of Dupont et al. [51] is the most similar to the BPP model. They call their considered problem the generalized covering location problem (GCL). Here a facility has a constant coverage radius. In this model a cost is also taken into account if some facility is built. Moreover, a penalty is incurred if a point is not covered. Additionally, an extra cost or gain may arise if a point is covered by more than one facility. The construction cost of a facility depends on its location, so while all facilities have the same size, their costs may differ. A small example is also given, with four facilities and twenty demand points, where the demand points are certain cells in a grid. The paper gives a B&B algorithm, with two inbuilt greedy methods. Summarizing the similarities and differences between GCL and my model BoPP. In both models, installing a facility incurs a cost, and an uncovered demand point incurs a penalty (negative gain). However, in the GCL model multiple coverage plays a central role, whereas in the BoPP multiple coverage is not accounted for in the objective. In BoPP the facilities may have different sizes while these have the same size in case of GCL. In case of BoPP any cell of a board is a demand point, in case of GCL the demand points are certain cells only. In case of BPP the demand points may have (different) negative gain values also, in case of GCL a demand point has positive gain value, and if it is not covered, a fixed cost should be paid (so the negative gain values are the same). Moreover, in case of the BPP model, the penalty for covering a bad point can be negative infinite also. It means that such a point must not be covered. In case of the GCL model it can happen that some bad point is covered so some amount of penalty is paid, but still the packing of the facility is advantageous because of the covered good points.

The BoPP can be viewed as a special case of other models (e.g., the model of Church and ReVelle [30]). Nonetheless, it remains of interest because the general problem is not solvable to optimality for large inputs with existing methods, and the specialized structure of the BoPP may admit stronger results. Even the compu-

tational complexity of this specialized model is, at this point, an open question.

2.1.3.2 Geometric covering problems: the value of gain is restricted to $-\infty$ or 1

A second direction of related research is in the form of geometric covering problems. These are often studied from a more theoretical perspective, without necessarily focusing on concrete practical applications. Within the main branch of this field, a rectilinear polygon should be covered by certain items, and usually the size of the cover (the number of covering items) is minimized. The sides of the objects are parallel (or perpendicular) to the axes in the two-dimensional space. A covering item cannot extend beyond the rectilinear object, and each cell must be covered (by exactly one or several items). This means, in the language of BoPP, that the value of the gain function is $-\infty$ outside of the rectilinear object, and it is uniformly 1 inside.

Possibly the first mention of such a problem is due to an unpublished manuscript of Masek [95]. Garey and Johnson [61] referred to Masek's problem (page 232, SR25) as rectilinear picture compression and stated that it is $\mathcal{NP}$-complete. The considered problem is the following (according to the description of [61]): Given an n x n matrix of 0's and 1's and a positive integer K. The question is whether there is a collection of K or fewer rectangles that covers precisely those entries in the matrix that are 1's. The rectangles are allowed to intersect each other, so overlap is allowed in this model. A paragraph from the paper by Allender et al. [5] (page 2, first paragraph) is cited below. Only the numbering of the cited papers has been adjusted to align with the references in this thesis, while all other parts of the text remain unchanged.

"In the 1970's, Masek showed that Min-DNF is $\mathcal{NP}$-complete [95]. Masek's result was cited by Garey and Johnson [61] and is widely known, but the proof was never published. Later, Czort presented a modernized, more readable version of Masek's proof [38]. Masek's proof is by direct reduction from Circuit-SAT, using gadget constructions, and even in Czort's version it is long and involved. We present a new, simple $\mathcal{NP}$-completeness proof for Min-DNF, by reduction from 3-Partite Set Cover (or, more particularly, from 3-Dimensional Matching)."

Recall that the situation in which all 1's must be covered, and only the 1's, can also be defined by assigning weight $-\infty$ to the points outside the rectilinear object. The same question (about a possible covering a rectilinear object) can be asked in different words also. In the framework of Culberson and Reckbow [37] given a rectilinear polygon (or object) on the 2-dimensional plane. The question is whether it is possible to cover it by at most K (arbitrarily given) rectangles. The rectangles cannot *go out* from the border of the rectilinear polygon, but can overlap. In this framework it can be supposed (similar to the description of [61]) that the coordinates of the vertices of the rectilinear polygon are all integers, so the rectilinear object can be assumed to lie in the 2-dimensional grid. In other words, it can be divided into 1×1 small cells, and each cell corresponds to a certain row and column in a given matrix.

The answer to the question depends on whether *there are holes* in the polygon or is it simply connected; and also whether the polygon is convex or not. The definition

is stated below.

Definition 2. *A rectilinear polygon is called vertically convex, if for any two cells belonging to the polygon and having the same y coordinates, in this case all cells between these two cells in vertical direction also belong to the polygon. The horizontally convex polygon can be defined analogously. And the object is convex if it is both horizontally and vertically convex.*

Franzblau and Kleitman [56] prove that the minimal covering is polynomial, if the rectilinear object is vertically convex. (Chaiken et al. [28] proves a weaker result, they give a polynomial algorithm for the convex case.) So if the board is convex only from one direction, this is also enough to get polynomiality. They improve a proof of Ervin Gyori [67]. If there may be holes in the rectilinear polygon, then already Masek proves the $\mathcal{NP}$-completeness of the problem. If the polygon is holeless but not convex from any direction, [37] proves the $\mathcal{NP}$-completeness.

Aupperle et al. [9] distinguished between two cases. If there is no hole within the rectilinear polygon, they showed that covering the object by a minimum number of squares can be solved in polynomial time. If, however, there can be holes within the polygon, the problem is $\mathcal{NP}$-hard. The authors also provided some non-trivial applications. Bereg et al. [17] considered a problem where there are both good points and bad points in the plane. All the good points must be covered while no bad point is covered, using the minimum number of rectangles. This problem is also $\mathcal{NP}$-hard, even when the rectangles are substituted by squares. Consider the minimal covering of a rectilinear object and its relationship to the BoPP. The following properties hold:

- Any gain value outside of the polygon is $-\infty$.
- Any gain value inside the polygon is 1.
- The items (used for covering) are not given in advance. Any type and any number of items can be created, provided that they fit into the polygon, and selection from the generated set of items can be made arbitrarily.
- The items can overlap.
- The cost of any item is identical. An appropriately small value $\varepsilon > 0$ can be chosen so that the cost of any item is exactly ε. If the rectilinear object can be enclosed in an $m \times n$ box, then for example $\varepsilon = 1/(mn + 1)$ is a suitable choice. In the worst case, mn items are selected for the cover, and their total cost is less than 1. Thus, covering the whole object with a minimum number of items is equivalent to covering it with minimum cost.

From the above, the differences between the minimal covering of a rectilinear polygon and the BoPP can be identified: in the latter, the gain values are not constrained, the items are given in advance, and each item has its own cost. If overlap is not allowed (but the entire board must be covered), the cover is called a partition (or decomposition). Culberson and Reckhow [37] established that the partitioning of orthogonal polygons into rectangles can be done in polynomial time, even when holes are allowed.

Since many similar models and potential application areas exist in the literature, a better visualization is provided to highlight the similarities and differences between the BoPP and other models, and to emphasize the novelty of the new model. This

is done in the following way. Certain combinations of collections of special features define different types of problems. The features include the following:

a, allocating in metrical space,
b, allocating in graph,
c, allocating in grid,
d, continuous model (or discrete model),
e, total coverage (or not total coverage),
f, allowing overlapping (or not allowing overlapping),
g, constant gain/cost across all covered points (or varying gains/costs),
h, constant cost of items (or varying costs of items),
i, items/facilities of the same size (or different sized items),
j, all items/facilities are selected (or several facilities selected),
k, points either completely covered or not covered (vs. partial coverage possible).

Table 2.1 presents the most relevant models from the literature and their relation to the features listed above. If a feature holds, it is indicated by an X in the appropriate cell; if the feature does not hold (meaning that the alternative stated in parentheses above applies), the corresponding cell remains empty. For example, regarding feature d, an X is placed if the model is continuous, while no X is placed if the model is discrete.

Table 2.1: Overview of existing models and how they relate to the board packing problem.

		Feature										
Authors/model	Reference	a	b	c	d	e	f	g	h	i	j	k
Church and ReVelle	[30]	X			X		X	X	X	X	X	X
Drezner	[50]	X			X	X	X	X	X	X	X	X
Çakir and Wesolowsky	[26]	X			X	X	X		X		X	X
Blanquero et al.	[22]		X		X	X	X		X	X	X	
Berman and Krass	[19]		X		X	X	X		X	X	X	
Dupont et al.	[51]			X			X			X		X
Masek	[95]			X		X	X	X	X			X
Culberson and Reckhow	[37]			X		X	X	X	X			X
Aupperle et al.	[9]			X		X	X	X	X			X
Bereg et al.	[17]			X		X	X	X	X			X
Mukherjee and Chakraborty	[102]	X			X	X	X	X				X
Leung et al.	[88]	X			X			X	X		X	X
Mahapatra et al.	[91]	X			X			X	X	X	X	X
BoPP				X			X					X

It should be emphasized that only some of the main features are listed in the table, and further differences exist among the models. As can be seen, the model most similar to the BoPP is that of Dupont et al. [51]. However, in their model each covering item has the same size (whereas in the BoPP the items may have different sizes), and multiple coverage of a demand point provides additional contributions to the objective (whereas in the BoPP only the distinction between covered and

uncovered points is considered). Moreover, the solution methods differ, and larger instances are addressed in the BoPP. All other models are quite different from the BoPP.

In the next sections, the study of the BoPP is continued by presenting proofs that the problem is $\mathcal{NP}$-hard. Subsequently, a mathematical programming model for the problem is provided.

Then, Section 2.4 describes an evolutionary algorithm that can provide heuristic solutions for large test instances. A computational study is presented in Section 2.5 to evaluate whether certain types of instances can be solved to optimality using commercial software.

2.2 $\mathcal{NP}$-hardness of Board Packing

There are several ways to prove that the BoPP is $\mathcal{NP}$-hard. Such proofs are usually based on reductions. First, the $\mathcal{NP}$-hardness of the BoPP is established by means of reductions from other packing models already known to be $\mathcal{NP}$-hard. Subsequently, a more standalone proof is given, showing that the BoPP is $\mathcal{NP}$-hard even in very special cases, through a reduction from basic $\mathcal{NP}$-hard SAT problems.

2.2.1 Reduction from other $\mathcal{NP}$-hard packing models

In this subsection, the $\mathcal{NP}$-hardness of the BoPP is proved for the following special cases:

- a reduction from the model of Masek, where there are negative gain values with huge absolute value
- a reduction from the model of Bereg et al., in which good and bad points are defined on the plane

The proofs are presented only briefly for several reasons. One reason is the page limit of this thesis. Another reason is that in the next subsection more detailed proofs are provided, where the $\mathcal{NP}$-hardness of the BoPP is established by *direct* reductions from SAT models (rather than from other packing models that themselves rely on reductions from SAT or other problems). The aim here is simply to demonstrate that the $\mathcal{NP}$-hardness of the BoPP can be established in several different ways.

2.2.1.1 Reduction from the model of Masek [95]

First, consider the case in which the gain values $g_{i,j}$ may be negative with very large absolute value. More precisely, in the construction used here each $g_{i,j}$ is restricted to take either a single negative value (of large magnitude) or a fixed small positive value (namely 2). Under this construction, if the number of items $|K|$ is fixed (i.e., not part of the input, or bounded above by a constant), the problem can be solved in polynomial time. Let $p = \max\{m, n\}$. Each item can be placed in at most p^2 distinct positions (a coarse upper bound). Hence, over all items, the number of distinct joint placements is at most $(p^2)^{|K|} = p^{2|K|}$. Note that if overlap is disallowed, some placements become infeasible; however, this only reduces the set of admissible

placements and thus does not affect the stated upper bound. Next, suppose that K is part of the input as well, and consider the variant in which overlap is allowed.

Given an input of Masek's rectilinear covering model, a corresponding input for the BoPP is constructed as follows. In Masek's board, a cell is called a *good* point if it contains a 1, and a *bad* point if it contains a 0. The board of Masek is embedded into a rectangular BOARD of size $M \times N$; let $p = \max(M, N)$. Within this BOARD, define $g(i, j) = 2$ for every $(i, j) \in M \times N$ that corresponds to a good point (i.e., a 1 in Masek's model), and $g(i, j) = -\infty$ for bad points (i.e., where Masek's board has a 0). Outside the BOARD, gains are also set to $-\infty$. In practice, $-\infty$ can be replaced by the finite value $-2p^2$ without loss of generality. Rectangles available for covering are then generated as all sizes $i \times j$ with $1 \leq i \leq M$ and $1 \leq j \leq N$, and $M \cdot N$ copies are provided for each such size. All rectangles are assigned unit cost.

It is asserted that any optimal cover in Masek's model corresponds to an optimal cover in the BoPP instance. An optimal cover in Masek's model covers all 1's, covers no 0's, remains within the board, and uses a minimum number of rectangles. In the BoPP instance, take any optimal solution. No rectangle extends beyond the BOARD, since this would be disadvantageous: any item covers at most p^2 cells, so the maximum gain it can collect from good points is $2p^2$, while covering any cell with $g(i, j) = -2p^2$ plus paying the unit cost yields a net loss (at least $1 + 2p^2$ in penalty versus at most $2p^2$ in gain). By the same reasoning, no bad point is covered.

Consequently, in any optimal BoPP solution all purchased rectangles remain within the BOARD and only good points are covered. Moreover, every good point must be covered: if a good point were left uncovered, purchasing a 1×1 rectangle covering precisely that point would increase the objective (gain 2 minus cost 1). Therefore, the set of covered cells coincides with the set of good points, exactly as in Masek's cover. Since each rectangle has positive cost, an optimal BoPP solution uses a minimum number of rectangles to cover all good points; hence the same number of rectangles is used as in Masek's optimal solution.

Thus, for any input of Masek's model, a corresponding input of the BoPP is produced such that the optimal BoPP solution selects the same number of rectangles as the optimal solution of Masek's model. This establishes a reduction from Masek's rectilinear covering model to the BoPP.

2.2.1.2 Reduction from the red-blue points model of Bereg et al. [17]

Another reduction is obtained by turning to the red–blue points model of Bereg et al. The construction proceeds in a very similar way. Recall that the instance contains "good" and "bad" points, and the objective is to cover all good points while not covering any bad point, using a minimum number of items. This problem is $\mathcal{NP}$-hard, even when the covering items are restricted to axis-aligned squares.

Set the gain of every bad point to $-2p^2$ and the gain of every good point to 2. Any point that is neither good nor bad is assigned gain 0. Thereafter, the reduction proceeds as in the reduction from Masek's model.

2.2.2 Reduction from some $\mathcal{NP}$-hard SAT problems

In this subsection, a direct proof (i.e., not via other packing models) is given that the BoPP is $\mathcal{NP}$-hard, even in very special cases, by means of reductions from appropriate SAT problems. Formally, the following problem is considered, where h and w are positive integers and S is a given set of integers. The latter will put a constraint on the gain values $g_{i,j}$, while the allowed purchasable rectangles $r \in R$ will be restricted to a single size $h \times w$.

> BoardPacking$(h, w; S)$ (BP$(h, w; S)$)
> **Input:** An $M \times N$ rectangular board (matrix) with all entries from S and an integer g.
> **Question:** Does there exist a packing with at most $M * N$ rectangles of size $h \times w$, each having a cost of 1, with total profit at least g?

In fact, a restricted set S will be considered, defined as follows:

> BinaryBoardPacking(h, w) (BBP(h, w)) means BP$(h, w; S)$ with $S = \{0, 1\}$,
> AlmostBinaryBoardPacking(h, w) (ABP(h, w)) means BP$(h, w; S)$ with $S = \{-1, 0, +1\}$.

The result is stated as a three-part theorem. As one can see, its parts (i) and (ii) are independent, while both of them are implied by (iii). The reason for this way of presentation is that the general ideas of the structural reduction are simpler to describe for (i), from which the proofs for (ii) and (iii) are developed by local modifications. In addition, (iii) requires a more complicated generic reduction and topological considerations.

Theorem 3. *The following problems are $\mathcal{NP}$-complete, and their optimization versions are $\mathcal{NP}$-hard:*
(i) BBP$(3, 3)$,
(ii) ABP$(2, 3)$ *and* ABP$(3, 2)$,
(iii) BBP$(2, 2)$.

The proofs are based on reductions from variants of the Boolean Satisfiability Problem. Generally speaking, when a generic instance of satisfiability is considered with p clauses over a set of q variables, the derived instance matrix (board) for BoPP or ABP has size $O(p) \times O(q)$, and to construct such an instance requires $O(pq)$ time. Hence the reduction is quadratic in both time and space.

Proof. The proof has been moved to Appendix B.1 due to space limitations.

2.3 Mathematical programming model

To formulate a binary integer programming model for the problem, let A_r denote the set of possible coordinates (i, j) for the top-left corner of rectangle $r \in R$. Let x_{ijr} be a binary variable taking the value 1 if and only if rectangle r is placed with its top left corner at (i, j). Let B_{ijr} be the set of positions (u, v) for the top-left corner of rectangle r that will result in position (i, j) being covered by that rectangle.

To calculate the revenues from covered board positions, it is defined an additional binary variable y_{ij} which is equal to 1 if and only if position (i, j) of the board is covered. The model becomes:

$$\max \sum_{i \in M} \sum_{j \in N} g_{ij} y_{ij} - \sum_{r \in R} \sum_{(i,j) \in A_r} c_r x_{ijr} \tag{2.4}$$

$$\sum_{(i,j) \in A_r} x_{ijr} \leq 1, \qquad r \in R \tag{2.5}$$

$$y_{ij} \leq \sum_{r \in R} \sum_{(u,v) \in B_{ijr}} x_{uvr}, \qquad i \in M, j \in N : g_{ij} > 0 \tag{2.6}$$

$$|R| y_{ij} \geq \sum_{r \in R} \sum_{(u,v) \in B_{ijr}} x_{uvr}, \qquad i \in M, j \in N : g_{ij} < 0 \tag{2.7}$$

$$x_{ijr} \in \{0, 1\}, \qquad r \in R, (i, j) \in A_r \tag{2.8}$$

$$y_{ij} \in \{0, 1\}, \qquad i \in M, j \in N \tag{2.9}$$

The objective function (2.4) computes the total gain by summing the values of all covered positions, allowing for overlap, while subtracting the total cost incurred by the rectangles used to achieve this coverage. Constraints (2.5) and (2.8) ensure that each rectangle is only used (at most) once.

The role of equation (2.7) ensures that cells with negative gain values $g_{ij} < 0$ are excluded from being counted as covered in the solution unless the model pays the penalty (i.e., takes the loss). It enforces an implication: If a rectangle covers a negatively valued cell (i, j), then the variable y_{ij} must be set to 1. This is captured via a Big-M-style constraint, where $|R|$ (the number of available rectangles) serves as a loose but safe upper bound in the equation (2.7) . This ensures that if any rectangle covers cell (i, j), the left-hand side must also be at least 1, forcing $y_{ij} = 1$, and triggering the loss in the objective function (equation (2.4)).

Equations (2.6) and (2.7) are separated along positive/negative g_{ij} values, because equation (2.6) is used only for positive-gain cells ($g_{ij} > 0$) to make sure that such a cell is counted as covered only if at least one rectangle covers it. Equation (2.7), on the other hand, is used for negative-gain cells ($g_{ij} < 0$), where the model should avoid setting $y_{ij} = 1$, unless it really covers that cell and is willing to accept the associated penalty.

The case $g_{ij} = 0$ is intentionally ignored in both constraints. These cells contribute neither positively nor negatively to the objective, so whether they are covered or not is irrelevant to profit and can be left unconstrained for efficiency. This

separation allows the model to maximize gain while controlling loss, using logical implications tailored to the sign of g_{ij}. The logic is asymmetric:

- **Equation (2.6) for positive profit ($g_{ij} > 0$):** earn only when covered. The solver prefers $y_{ij} = 1$; the constraint allows this only if the cell is actually covered, preventing unearned gains.
- **Equation (2.7) for negative profit ($g_{ij} < 0$):** pay the loss when covered. The solver prefers $y_{ij} = 0$; the constraint forces $y_{ij} = 1$ whenever the cell is covered, so the loss is duly charged. (Cells with $g_{ij} = 0$ are skipped for efficiency.)

2.4 Evolutionary algorithm

An evolutionary algorithm (EA) was implemented to provide heuristic solutions, capable of handling instances that are too large to be solved to optimality using exact methods. The heuristic keeps a set of solutions in memory, referred to as a population. New solutions are generated by combining pairs of existing solutions. Local improvements are performed on the new solutions before they are added to the population. When the population reaches a certain size, it is trimmed down to a target size by selecting the best solutions from the full population. If the search has trimmed down the population a certain number of times without finding any new best solution in between, the search is restarted from scratch. Figure 2.3 illustrates the components of the search, which are described in more detail below.

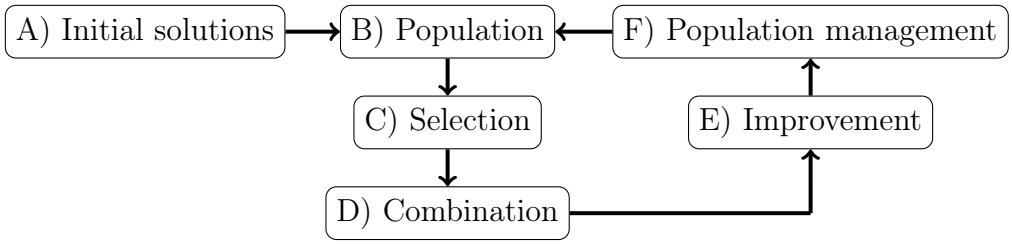


Figure 2.3: An overview of the genetic search developed for the BoPP.

In the evolutionary algorithm, a solution to the BoPP is represented by storing, for each rectangle $r \in R$, the coordinates (i, j) of the top-left corner of the rectangle's position and a Boolean flag to indicate whether the rectangle is purchased. A possible simplification is that storing an additional Boolean flag may be redundant, since the purchase status could be represented directly in the coordinate data (for example, by using $(-1, -1)$ for an unpurchased rectangle).

2.4.1 Initial solutions

The first step of the heuristic is to generate a set of initial solutions. Two mechanisms are used: some solutions are generated randomly, while others are constructed by a greedy heuristic. In the calibrated parameter setting (see Table 2.2), the number of random initial solutions was chosen much larger than the number of constructed ones (e.g., `pop_init_rand` = 190 and `pop_init_con` = 10). This imbalance reflects a

design choice to maximize diversity while still including a smaller number of higher-quality heuristic solutions.

To generate a random solution, each rectangle is selected as purchased with a probability of 50%, and its coordinates are drawn from uniform distributions over the feasible values for the top row and the left-most column. A key motivation for combining random and constructive initialization is to balance diversity and quality in the initial pool. Purely random solutions provide broad coverage of the search space but are usually of low quality, whereas constructive solutions inject more competitive individuals that can serve as anchors for the search process. The relative proportion of random versus constructive solutions therefore controls the trade-off between exploration and exploitation from the outset.

It is well known that the quality of the initial population strongly influences the subsequent performance of evolutionary algorithms. If all initial solutions are poor, many iterations may be required before competitive solutions are found; conversely, if the population is too homogeneous, the search may converge prematurely. The chosen balance between random and constructive initialization was calibrated to avoid both extremes.

Uniform sampling ensures that each feasible coordinate has the same probability of being selected during random initialization. This maintains generality and prevents hidden biases, although it also increases the likelihood of generating unprofitable solutions. The construction heuristic therefore plays a crucial role in ensuring that at least a few good-quality solutions are present from the beginning of the search. Future extensions could explore alternative initialization schemes, such as greedy randomized methods, to further improve the diversity–quality trade-off in the initial pool.

2.4.2 Parameter settings

In the following, when defining and running the heuristic, the following parameter values are used (the meaning of each parameter will be explained in more detail in the following subsections):

Name	**Role**	**Value**
`pop_init_rand`	Random initial solutions	190
`pop_init_con`	Constructed initial solutions	10
`pop_min`	Minimum population size	100
`pop_max`	Maximum population size	200
`pop_max_generated`	New solutions before trimming	500
`pop_max_reductions`	Reductions before restart	25
`pop_elite`	Elite survivors (quality-only)	4
`pop_closest`	Nearest neighbours for diversity	5
`time_limit`	Runtime limit (seconds)	60

Table 2.2: Parameters of the evolutionary algorithm for BoPP.

A parameter `pop_init_rand` is used to control the number of random solutions initially generated. The parameter values used in the evolutionary algorithm were

obtained through a short but systematic calibration on a representative subset of the benchmark instances. The guiding principle was to maximize the average profit within 60 seconds on a standard desktop while keeping the run-to-run variance small. Sensitivity tests (with ±20% variation around each setting) indicated that the EA's average gap changed by less than 1% in absolute terms. Thus, the chosen values are not "magic numbers" but rather a stable compromise between exploration, exploitation, and runtime.

2.4.3 Construction heuristic and main components of the algorithm

Here, I explain the B–D steps of the algorithm. Random solutions are likely to be of poor quality. To increase the likelihood that the initial population contains at least a few decent solutions, a construction heuristic is also used. The idea here is to consider the rectangles in sequence, considering first the rectangles that are cheapest compared to their size. For each rectangle, all possible locations are considered and, if profitable, the rectangle is purchased and inserted into its most profitable position, conditional on the placements of any previously considered rectangles. If it is not profitable to purchase the rectangle, it is left as being unused. By introducing the possibility of changing where in the sorted list of rectangles the insertions should start, several alternative solutions can be constructed. The construction heuristic can be summarized as follows:

1. Given: an instance of the BoPP, and a value `con_shift` to allow constructing several different solutions.
2. Sort the rectangles in increasing order of $c_r/|A_r|$.
3. Restructure the sorted list by taking `con_shift` rectangles from the beginning of the list and moving them to the end of the list.
4. Follow the sequence of rectangles given in the updated list, and insert each of them in their best possible position of the board, or leave them unused if the insertion into the best position does not lead to an increased objective function value.

The construction heuristic is run for values of `con_shift` from 0 to `pop_init_con`-1, yielding in total `pop_init_con` constructed solutions and `pop_init_rand` randomly generated solutions. If this total number is larger than the size of the initial population, `pop_min`, the initial population is determined by taking the solutions with the highest objective function values.

The main iteration of the evolutionary algorithm then consists of steps C, D, and E from Figure 2.3. It starts with the selection of two solutions from the current population. This selection is done by performing two binary tournaments. Each binary tournament is executed by drawing two solutions from the population at random and declaring the better of the two as the winner. The two winners, called $P1$ and $P2$, are then combined to create two new solutions, called $C1$ and $C2$.

The combination method is a form of uniform crossover, where each rectangle is considered independently. Recall that in a solution, each rectangle is described by three values: the coordinates (i, j) of the upper-left corner, and a Boolean value representing whether or not the rectangle is purchased. In the combination method,

each rectangle is considered in turn. With a 50% probability, the placement information about rectangle r in solution $P1$ is assigned to the new solution $C1$ and the placement information in solution $P2$ is assigned to $C2$. Otherwise, the opposite assignment is made.

Each new solution created, including those randomly generated for the initial population, is subjected to local improvements (step E). The local improvement is used instead of a mutation operator, as is common in memetic algorithms [24]. The local improvement starts by considering each rectangle in turn. For a given rectangle, the following modifications are examined, subject to feasibility:

1. If the rectangle is not purchased, try to purchase it and place it at its current position.
2. If the rectangle is currently purchased, try to move it one step up, one step down, one step to the right, or one step to the left.
3. If the rectangle is currently purchased, try to reverse this decision.

When examining the possible improvements, a first-improvement strategy is used: if moving a rectangle one step improves the solution, no further moves are attempted. If no improvements are found after considering all the above changes, one rectangle is selected at random and all feasible placements for that rectangle are evaluated. The rectangle is then inserted at its best location, or is left unused if no profitable placement exists.

2.4.4 Population management

At the end of each iteration, the population is updated (step F). First, the two new solutions $C1$ and $C2$ are added to the population, if admissible. To ensure that the population does not contain duplicated solutions, admissibility is restricted so that new solutions are accepted only if they do not share the same objective function value as a solution already present in the population [105].

The population is associated with both a minimum size, `pop_min`, and a maximum size, `pop_max`. Whenever the current population size exceeds `pop_max`, or if `pop_max_generated` new solutions have been created since the last reduction, the population is trimmed back to `pop_min`. The reduction procedure considers both solution quality and solution diversity, in order to avoid premature convergence [117]. To quantify diversity, the distance between each pair of solutions in the population is calculated. This distance is measured by comparing the placement of each rectangle across the two solutions. For a given rectangle r, let:

$$d_r = \begin{cases} 0 & \text{if both solutions do not use } r, \\ 2 & \text{if one solution uses } r \text{ and the other does not,} \\ \frac{|i_1 - i_2|}{|M|} + \frac{|j_1 - j_2|}{|N|} & \text{if both solutions use } r, \text{ where } (i_1, j_1) \text{ and } (i_2, j_2) \text{ are placements.} \end{cases}$$

The total distance between two solutions is obtained by summing d_r across all rectangles $r \in R$. Each solution is then ranked in two ways:

- **Quality rank (f^{QB}):** solutions are sorted by objective function value, with the best solution given rank 1 and the worst given rank $|P|$, where P is the current population.

- **Diversity rank (f^{DB}):** solutions are sorted according to their average distance to the `pop_closest` most similar solutions, with 1 assigned to the most diverse solution and $|P|$ to the least diverse.

Following Vidal [116], an overall score for each solution is calculated as:

$$f^S = f^{QB} + \Big(1 - \frac{\texttt{pop\_elite}}{|P|}\Big) f^{DB},$$

where `pop_elite` specifies how many solutions should survive solely on the basis of quality. In the trimming process, the solution with the highest f^S is removed, and the procedure is repeated until the population has been reduced to `pop_min`. If the population has been reduced `pop_max_reductions` times without improving the best-found solution, the entire search is restarted from scratch. The evolutionary algorithm terminates after reaching the runtime limit `time_limit`.

2.4.5 Handling identical rectangles in solutions

As the BoPP formulation does not exclude the existence of identical rectangles in a solution, one might ask whether swapping two such rectangles, say r_A and r_B with $h_{r_A} = h_{r_B}$ and $w_{r_A} = w_{r_B}$, creates a misleading notion of diversity. Two solutions that differ only in the placement of r_A and r_B may appear very similar, yet their distance value (based on rectangle coordinates) could be large. Since population diversity in the evolutionary algorithm depends on these distance values, such cases could seemingly have a negative impact on efficiency.

In practice this issue does not arise. If two rectangles have the same height, width, and cost, swapping them produces layouts that are effectively identical, and only one survives in the population because:

1. at most one solution per profit value is kept, and
2. the diversity check is applied only after the profit filter.

Consequently, the swapped variant is normally discarded before the distance metric is consulted. In the rare case both survive (for example, after different local-search moves), the distance measure is used only for occasional tie-breaking and not for parent selection, so any over-estimated distance has negligible impact on population diversity or search efficiency.

This issue, handling identical rectangles in a solution, is analysed in Section 2.5.3 of the thesis. In that experiment, $24/d$ identical rectangles are generated and a subset is chosen to pack; the results are presented in Table 2.4. When $d = 1$ (24 identical rectangles), both the EA and the exact solver reach the optimum. For $d = 2, 3, 4$ (few rectangle types), the solver remains optimal and the EA is near-optimal (within a small gap in Table 2.4). For $d \geq 6$ (many distinct rectangles), the EA outperforms the solver. Overall, the EA performs well for every tested value of d, becoming clearly superior as problem diversity increases, while the solver has an advantage only for very small and highly uniform instances.

2.5 Computational experiments

While the studied problem is $\mathcal{NP}$-hard, I do not know how difficult it is to find optimal solutions in practice. Therefore, I aim to evaluate the ability of a commercial mixed-integer programming solver to identify optimal solutions and obtain a proof of optimality. Furthermore, I want to study the proposed evolutionary algorithm and its ability to find near-optimal heuristic solutions for instances with known optimal solutions as well as its ability to find good feasible solutions when the commercial solver cannot be applied successfully. Note that my goal is not to show that one of the solution methods is better than the other. Rather, I believe that there are certain characteristics of instances that allow each of the solution methods to exhibit a superior performance, and I wish to gain insights regarding these characteristics and their effects on the overall performance.

In the computational experiments, I use CPLEX (version number: 20.1) as the commercial solver, with a running time limit of 600 seconds. The evolutionary algorithm has a time limit of 60 seconds. Both methods are executed on a desktop computer with Intel Core i3-4150 (Dual-Core) CPU at 3.50 GHz and 8.00 GB of RAM, the operating system is Windows 10 Pro 21H1.

However, to validate the observed performance ceiling, I also performed tests on a high-end MacBook Pro with 32 GB of RAM and faster CPU architecture, using extended time limits and CPLEX parameter tuning. These tests confirmed that while some additional progress was achieved, the solver continued to exhibit memory exhaustion or fail to close optimality gaps for instances of moderate to large scale. Even when more memory and time were available, the exponential complexity of the underlying branch-and-bound tree structure and symmetric placements in BoPP led to solver stagnation or extremely slow convergence. These findings are consistent with observations already cited in the thesis, where exact solvers often struggle with packing problems that involve grid-based revenue structures, overlapping items, and variable costs, all of which increase the model's combinatorial complexity.

Additionally, while cloud-based solutions (e.g., IBM Cloud or commercial CPLEX licensing on high-memory virtual machines) could technically allow for more CPU and RAM, such approaches do not fundamentally change the algorithmic limitations. CPLEX may still require impractical runtimes and memory to fully solve BoPP instances, especially as instance size grows. This is precisely why the evolutionary algorithm developed in the thesis becomes essential: it provides consistently high-quality solutions within seconds, even for large-scale instances, and does so using moderate hardware. It also leverages problem-specific knowledge (e.g., local search, trimming, restart mechanisms), which CPLEX does not exploit in its general-purpose formulation. In conclusion, the results confirm that while exact methods are valuable for validating small instances and lower bounds, scalable heuristic approaches are crucial for solving BoPP efficiently in practical contexts.

In Section 2.5.1 I consider the effect of increasing the board size, while keeping other aspects of the problem fixed. Section 2.5.2 analyses the effect of the number of rectangles, while Section 2.5.3 analyses the effect of the variety of different rectangles. Next, Section 2.5.4 considers the effect of the topography of the board, that is, how the profits from covering board positions are related to each other. Section 2.5.5

presents some artificial instances based on covering a given landmass by satellite images. Finally, Section 2.5.6 investigates the performance of the solution methods on sets of instances that are constructed to be very similar to each other. All 142 instances used in the computational experiments are available at `https://home.himolde.no/hvattum/benchmarks/` and were originally introduced by Abraham et al. [2].

2.5.1 The effect of the board size or resolution

In this section I want to evaluate how the size of the board influences the difficulty of solving the problem, while keeping other aspects of the problem fixed. To this end I start with a board of size $|M| \times |N|$, with $|R|$ rectangles of size $h_r \times w_r$ and cost c_r. Then I introduce a scaling factor $p = 1, 2, \ldots$ and generate instances of size $p|M| \times p|N|$, with $|R|$ rectangles of size $ph_r \times pw_r$ and cost $p^2 c_r$. If the original board position (i, j) has a value g_{ij}, then the scaled board has the value g_{ij} for positions $((i-1)p + i', (j-1)p + j')$ with $i' = 1, \ldots, p$ and $j' = 1, \ldots, p$. A base instance is created first, with $|M| = 6$, $|N| = 8$, and $|R| = 12$. The gain values are randomly chosen from $\{1, 2, \ldots, 20\}$, providing the following board:

$$G = \begin{bmatrix} 9 & 15 & 11 & 4 & 13 & 11 & 12 & 12 \\ 18 & 10 & 18 & 1 & 19 & 13 & 10 & 9 \\ 8 & 14 & 4 & 13 & 9 & 14 & 3 & 10 \\ 4 & 5 & 2 & 18 & 13 & 2 & 2 & 9 \\ 11 & 8 & 13 & 8 & 12 & 19 & 18 & 20 \\ 8 & 1 & 11 & 13 & 15 & 6 & 4 & 2 \end{bmatrix}$$

The heights and widths of the rectangles are also chosen at random. Each height is drawn uniformly at random from $\{1, 2, \ldots, |M|/2\}$, and each width from $\{1, 2, \ldots, |N|/2\}$. I generated the costs of the rectangles by first calculating the average gain that a rectangle can cover, i.e. $h_r \cdot w_r \cdot g_{\max}/2$, where the maximum possible gain value is $g_{\max} = 20$. Then, this value is multiplied by a random number drawn from $[\frac{1}{2}, 1]$ and the result rounded to the nearest integer. This process resulted in the following values for the 12 rectangles:

r	1	2	3	4	5	6	7	8	9	10	11	12
h_r	1	3	3	1	2	3	2	1	3	2	3	3
w_r	2	2	3	4	3	2	3	4	1	1	4	3
c_r	18	33	72	27	58	43	31	28	18	14	65	54

CPLEX and the evolutionary algorithm are tested on instances with $p = 1, 2, \ldots, 25$. The smallest of these have $6 \times 8 = 48$ board positions, whereas the largest has $150 \times 200 = 30,000$ positions. For $p \geq 2$, the instances are constructed in a manner that allows us to calculate upper bounds on the optimal objective function value, given that I know the optimal value for $p = 1$. In particular, I solve the instance to optimality for $p = 1$ and obtain an optimal value of 224. Then, for $p \geq 2$, an upper bound for the optimal objective function value is given by $224p^2$.

Table 2.3 shows the results for these instances. For each instance the upper bound (UB) is reported. For CPLEX I report the gap to this value and the total time used in seconds (TT). For the heuristic I report the gap and the time used to find the best solution in seconds (TB). To calculate gaps, I start from a given upper bound (UB) and a given lower bound (LB), and use the formula $Gap = (UB - LB)/LB$ and then round the result to one decimal.

Table 2.3: Varying the size of the board with total time (TT) and time to best (TB) reported in seconds.

		CPLEX		Heuristic	
p	UB	Gap [%]	TT	Gap [%]	TB
1	224	0.00	0.0	0.00	0.6
2	896	0.00	0.2	0.00	4.4
3	2016	0.00	0.5	0.00	5.1
4	3584	0.00	1.2	0.00	30.4
5	5600	0.00	3.1	0.00	5.0
6	8064	0.00	5.8	0.00	12.9
7	10976	0.00	9.3	0.00	7.3
8	14336	0.00	17.5	0.00	13.5
9	18144	0.00	26.7	0.00	13.1
10	22400	0.00	30.4	0.00	1.9
11	27104	0.00	46.3	0.00	9.5
12	32256	0.00	81.3	0.00	14.9
13	37856	0.00	111.5	0.00	32.4
14	43904	0.00	133.4	0.00	46.0
15	50400	0.00	244.6	0.00	6.9
16	57344	Out of memory		0.00	3.1
17	64736	Out of memory		0.00	17.6
18	72576	Out of memory		0.00	25.0
19	80864	Out of memory		0.00	59.0
20	89600	Out of memory		0.45	4.3
21	98784	Out of memory		0.45	53.5
22	108416	Out of memory		0.00	25.0
23	118496	Out of memory		0.45	29.0
24	129024	Out of memory		1.34	34.8
25	140000	Out of memory		0.00	13.9

The first observation is that CPLEX solves the smaller instances to optimality very quickly, and CPLEX is able to solve all instances to optimality for p up to and including 15. However, the time required by CPLEX increases gradually with p. When going from $p = 15$ to $p = 16$, which increases the number of positions on the board from 10,800 to 12,288, CPLEX hits a wall and consistently runs out of memory when trying to solve the problem. Nevertheless, I consider a board size of 90×128, corresponding to $p = 15$, to be a reasonably large board, and could say that the ability of CPLEX to find optimal solutions does not depend too much on just the size of the board.

The evolutionary algorithm was capped at a running time of 60 seconds, but still managed to find optimal solutions for all of the smaller instances, up to $p = 19$. With values of p larger than 19, the gap from the best-found solution to upper bound is still very small and occasionally zero.

2.5.2 The effect of the growing number of rectangles

The size of the board influences the number of locations where a rectangle can be placed, and thus the time required to find the optimal solution. Another factor that influences the number of decisions that must be made is the number of rectangles, $|R|$. In this section I consider a board of size $|M| \times |N|$, with $|R|$ rectangles of random size, with $h_r \in \{1, \ldots, |M|/2\}$ and $w_r \in \{1, \ldots, |N|/2\}$.

I have chosen $|M| = 60$ and $|N| = 80$, which corresponds to $p = 10$ in the previous test, so the board is too big to visualize here. The gain values are chosen randomly between 1 and 20. The cost of each rectangle is determined in the following way: I consider a random position for the rectangle, and sum up the gain values of the cells that are covered by the rectangle. Then this sum is multiplied by a random number, chosen from $[\frac{1}{2}, 1]$.

I create instances with $|R| = 5, 10, 20, 30, 40, 50, 60, 80, 100, 200, 500,$ and 1000 rectangles. For all instances, the board is the same, and the rectangles included in instances with fewer rectangles constitute a subset of the rectangles in instances with more rectangles. In this way I ensure that the optimal objective function value is never smaller when I compare to an instance with more rectangles. This fact is advantageous for estimating the optimum, as the instances in this section are harder to solve for CPLEX.

The results are presented in Table 2.4. For CPLEX I present the best upper bound found within the time limit (UB), the best feasible solution found (LB), and the total running time (TT) which is limited to at most 600 seconds. For the heuristic I provide the best feasible solution found (LB), the time to find the best solution (TB) and the number of rectangles purchased in the best solution (#Rec.).

In this test, the intuition is that ever more rectangles will be purchased as the number of available rectangles increase. Then, gradually, the number of chosen rectangles will stagnate, although some rectangles will be replaced and the optimum value will grow slowly. Thus, the number of chosen rectangles (the number of rectangles that are used in the optimal solution to cover the board) first grows quickly, then slowly, and then the increment stops.

Only the first two instances could be solved to optimality by CPLEX. For $|R| = 20$, CPLEX cannot determine the optimal solution, but still finds a lower bound (14984) and an upper bound (17271) with a gap of 15.54%. For this input the value of the solution provided by the evolutionary algorithm (16623) is already better, moreover the heuristic uses much less time in total. Then, for the next instances, with $30 \leq |R| \leq 100$, CPLEX only provides the trivial feasible solution where no rectangles are purchased and the trivial upper bound consists of the sum of all gain values. The instances appear too large for CPLEX to handle, and most of the time is spent by preprocessing. For $|R| \geq 200$ the memory runs out during the CPLEX solution process.

Table 2.4: Varying the number of rectangles for an instance with $|M| = 60$ and $|N| = 80$.

	CPLEX			Heuristic		
$\|R\|$	UB	LB	TT	LB	TB	#Rec.
5	5349	5349	14.4	5349	0.2	5
10	9287	9287	53.5	9255	1.5	10
20	17271	14984	600.0	16623	2.5	16
30	50690	0	600.0	20780	1.9	13
40	50690	0	600.0	20898	12.3	14
50	50690	0	600.0	23502	5.4	14
60	50690	0	600.0	23441	4.0	15
80	50690	0	600.0	24194	7.5	12
100	50690	0	600.0	24110	2.5	21
200	Out of memory			24351	5.0	18
500	Out of memory			24951	5.6	24
1000	Out of memory			24662	5.1	15

On the other hand, the heuristic provides a solid, reliable performance. As I expected based on the construction of the instances, the objective function value is in general growing gradually, but with diminishing marginal improvements by including additional rectangles. For the heuristic, having many rectangles to choose from also makes the instances difficult, and there are some cases where solving an instance with a higher number of rectangles available leads to worse results. Also, the number of rectangles used does not increase steadily as expected, either because the solutions are not optimal, or because some new rectangle may be used to replace several previously used rectangles.

2.5.3 The effect of rectangle variety

In the previous experiment the rectangles were unique and I varied the number of rectangles included. This section examines whether the problem difficulty changes when the number of different rectangles is varied while keeping the total number of rectangles fixed. To this end, I use the same board as in Section 2.5.2, but consider new rectangles. Since an instance with $|R| = 30$ rectangles is too hard for CPLEX and an instance with $|R| = 20$ rectangles can almost be solved within 600 seconds, I choose to use $|R| = 24$ in the following. This number allows us to consider same-sized groups of rectangles that are identical, and to denote the number of different rectangles by $d = 1, 2, 3, 4, 6, 8, 12, 24$. If I have d different rectangles, then each group of rectangles consists of $24/d$ identical rectangles.

Having several identical rectangles induces a form of symmetry in the mathematical model. However, CPLEX is nevertheless dealing well with this issue and is able to solve the instance with only one type of rectangle ($d = 1$) to optimality within 40 seconds. In general, it seems harder to solve instances with more types of rectangles: CPLEX solves to optimality three of four instances with $d \leq 4$, and only finds a non-trivial feasible solution to one out of four instances with $d \geq 6$. Although

Table 2.5: Varying the number of different types of rectangles for an instance with $|R| = 24$.

	CPLEX			Heuristic	
d	UB	LB	TT	LB	TB
1	23355	23355	39.9	23355	14.1
2	23790	23790	87.0	23534	4.1
3	23616	23557	600.0	23543	25.7
4	22269	22269	315.7	21942	2.1
6	18992	0	600.0	17932	0.9
8	50690	0	600.0	18139	1.4
12	17144	13050	600.0	16329	2.3
24	18132	0	600.0	17354	2.3

the performance of the heuristic is slightly weaker than CPLEX when $d \leq 4$, the heuristic is superior for instances with many ($d \geq 6$) different types of rectangles, and the performance of the heuristic is solid across the range of instances even when given only 60 seconds of running time, see Table 2.5.

2.5.4 The effect of the topography

The size of the board, the number of rectangles, and the number of rectangle types all influence the difficulty of finding optimal solutions in a predictable way. However, the difficulty of an instance may also depend on the structure of the gain values of the board. This is examined in this section, where I generate boards with different topographical features.

2.5.4.1 Varying the maximum gain for randomly generated boards

For the following experiments, a board size of $|M| = 20$ rows and $|N| = 30$ columns was chosen. Each instance has $|R| = 15$ rectangles, which are generated as in Section 2.5.1: the height is a random value from $\{1, \ldots, |M|/2\}$ and the width is a random value from $\{1, \ldots, |N|/2\}$. The costs of the rectangles are generated by assuming that gain values are uniformly distributed between $g_{\text{MAX}}/2$ and g_{MAX}. For example, if $g_{\text{MAX}} = 20$, then the cost of a rectangle is randomly selected between 10 and 20 times its area. In the experiments, several different values of g_{MAX} were considered (see Table 2.5). The randomly generated rectangles can be summarized as follows:

r	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
h_r	7	8	4	9	10	9	4	6	2	7	9	9	4	6	2
w_r	3	7	11	1	3	11	12	3	12	7	13	4	15	11	6
c_r	245	870	576	102	530	1504	621	192	479	623	1507	371	803	874	132

To generate gain values for the board, I chose a value g^{MAX}, and then the gain values are randomly generated in the range $\{1, 2, \ldots, g^{MAX}\}$. I try different values

for g^{MAX} to see how this influences the behaviour of the solvers. Table 2.6 shows the results for experiments where g^{MAX} is varied between 15 and 70.

Table 2.6: Varying the maximum gain on randomly generated boards.

	CPLEX					Heuristic			
g^{max}	UB	LB	TT	#Rec.	Gap	LB	TB	#Rec.	Gap
15	0	0	0.6	0	0.00	0	0.0	0	0.00
20	282	282	0.3	5	0.00	282	0.6	5	0.00
25	788	788	1.3	9	0.00	783	49.2	9	0.63
30	2087	2087	4.2	11	0.00	2083	0.6	11	0.19
35	3219	3219	6.8	11	0.00	3180	11.9	11	1.21
40	4639	4639	438.0	12	0.00	4579	27.3	11	1.29
45	6316	6171	600.0	12	2.36	6196	47.6	13	1.90
50	7488	7170	600.0	11	4.44	7265	56.7	12	2.98
60	9997	9692	600.0	12	3.15	9712	46.3	12	2.85
70	12294	11962	600.0	12	2.78	11931	15.3	12	2.95

For $g^{MAX} = 15$, the costs of the rectangles are too high compared to the gain values, so it is not advantageous to purchase any of the rectangles. This fact is quickly recognized by both solvers. When $g^{MAX} = 20$ the cost of the rectangles is still relatively high. In the optimal solution, which is found by both CPLEX and the evolutionary algorithm, only 5 rectangles are purchased and used to cover selected parts of the board. Finding the optimal solution appears easy for both methods.

CPLEX is also able to prove optimality within 600 seconds for the instances with g^{MAX} equal to 25, 30, 35, and 40. The case with $g^{MAX} = 35$ is the first where the optimal solution has rectangles that overlap, albeit only slightly. For the instances where g^{MAX} is between 25 and 40, CPLEX finds better solutions than the heuristic.

For $g^{MAX} = 40$, the problem is much harder than for the previous cases. The heuristic finds a solution using 11 rectangles, while CPLEX finds an optimal solution using 12 rectangles. Some rectangles in the optimal solution cover each other, as it is still advantageous to buy a rectangle for some cost, even if it is used only for covering an area that is already partly covered by another rectangle. For the instances with larger gain values, with g^{MAX} from 45 to 70, the heuristic in general finds better solutions than CPLEX, and CPLEX is not able to close the optimality gap. It is clear that the difficulty of solving the problem depends on the structure of the board's gain values and whether or not it is beneficial to let rectangles overlap.

I also performed a sensitivity test where the same instances as above were solved after replacing the set of rectangles with a different set of randomly generated rectangles. Overall, the findings point in the same direction, but the instances were somewhat easier for CPLEX, so that the change in performance appeared for a slightly larger value of g^{MAX}. Also, with the alternative set of rectangles, there was less overlap of rectangles in the optimal solution, even for high gain values, which seems to make the instances somewhat easier to solve for CPLEX.

2.5.4.2 Randomly generated boards with some large negative gains

In preparation for the next experiment, I first randomly generate the gain values from the range $\{1, 2, \ldots, g^{MAX}\}$, but then a given number of cells are modified to $g_{ij} = -g^{LARGE} = -1000$. This latter value can in practice be considered as $-\infty$, as it will dominate any profit obtainable from other board cells in the vicinity. To start with a board that is reasonably difficult for CPLEX, I use $g^{MAX} = 40$. Then I put L values of $-g^{LARGE}$ into the board, replacing the original gain values. Values for L vary from 0 to 50 in steps of 5, and the boards are identical except for the cells where $g_{ij} = -g^{LARGE}$. Furthermore, the cells with a value of g^{LARGE} are selected such that when going from L to $L + 5$ cells, the same cells are modified in addition to 5 new cells.

Table 2.7: Instances with randomly added large negative gains.

	CPLEX					Heuristic			
L	UB	LB	TT	#Rec.	Gap	LB	TB	#Rec.	Gap
0	4639	4639	438.0	12	0.00	4579	27.3	11	1.29
5	4424	4424	5.5	11	0.00	4402	16.2	11	0.50
10	4062	4062	11.1	11	0.00	4028	28.7	11	0.84
15	3831	3831	6.8	11	0.00	3831	43.7	11	0.00
20	3620	3620	2.9	10	0.00	3620	50.0	10	0.00
25	3473	3473	2.6	10	0.00	3473	22.6	10	0.00
30	3151	3151	4.4	9	0.00	3120	23.8	9	0.98
35	2912	2912	2.3	10	0.00	2912	46.4	10	0.00
40	2773	2773	1.8	9	0.00	2773	21.8	9	0.00
45	2605	2605	2.5	9	0.00	2605	3.7	9	0.00
50	2292	2292	1.9	8	0.00	2292	2.4	8	0.00

Table 2.7 summarizes the results. The optimal objective function value decreases slowly as the g_{ij} values are reduced while the costs of the rectangles and all other data remain the same. However, the optimal objective function values decrease only slightly, implying that the rectangles still fit into the board while avoiding the large negative values that are introduced. This can happen because the rectangles are small enough to place in between the large negative values. If I keep adding more negative gain values, eventually no rectangles can be beneficially placed and the optimal value will decrease to zero. However, after setting 50 cells to g^{LARGE}, this is still far from happening. For larger values of L, the problem becomes easier to solve, for both CPLEX and the evolutionary algorithm. Already for $L = 5$ the solution time by CPLEX is radically reduced, from 438 seconds for $L = 0$, to 5.5 seconds.

2.5.4.3 Gain values generated based on existing benchmark functions

In the tests above, the values of the board have been assigned randomly. This imposes a certain unstructured topography for the boards, which may influence the difficulty of solving the corresponding instances. In this section I consider instances that are based on benchmark functions for unconstrained non-linear optimization.

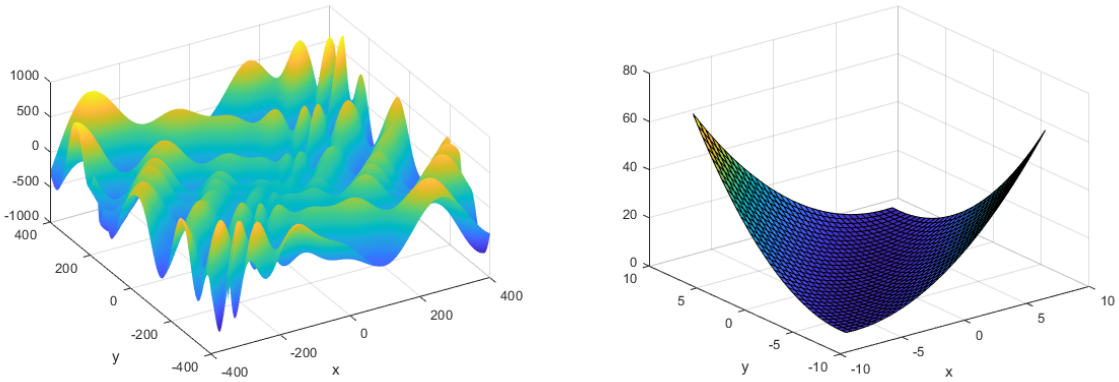


Figure 2.4: Two test functions used to generate instances: to the left the Eggholder function, and to the right Matyas function

In particular I use the eggholder function and the Matyas function [1]. First, let a function $f(x, y)$ be defined on $[x^{MIN}, x^{MAX}] \times [y^{MIN}, y^{MAX}]$. For cell (i, j), let

$$g_{ij} = \left\lceil f(x^{MIN} + (x^{MAX} - x^{MIN})\frac{i-1}{|M|-1}, y^{MIN} + (y^{MAX} - y^{MIN})\frac{j-1}{|N|-1}) \right\rceil .$$

For a given test function, the focal area $[x^{\text{MIN}}, x^{\text{MAX}}] \times [y^{\text{MIN}}, y^{\text{MAX}}]$ is varied to generate several different instances. I choose regions of the function that provide a mix of negative gain values and positive gain values. These instances then have similarities to geographic areas which have deep valleys or high hills. I again choose the board to have 20 rows and 30 columns, and I keep the same collection of $|R| = 15$ rectangles that I applied in Section 2.5.4.1.

Figure 2.4 illustrates the two functions used as a basis of creating instances. The eggholder function is defined for $-400 \leq x, y \leq 400$, and can be stated as:

$$f(x, y) = -(y + 47) \sin \sqrt{\left| \frac{x}{2} + (y + 47) \right|} - x \sin \sqrt{|x - (y + 47)|},$$

whereas the Matyas function, which is shaped like a big valley, is defined for all real x and y values:

$$f(x, y) = 0.26 \left(x^2 + y^2\right) - 0.48xy.$$

Table 2.8 summarizes the instances generated based on the benchmark functions, indicating some of their properties. The instance E3 is illustrated in Figure 2.5, which shows both the board values and the superimposed optimal solution, with each rectangle shown in different colors. Figure 2.6 illustrates two solutions found for the instances based on the Matyas function: an optimal solution for M1, and the best known solution for M3.

Again using a time limit of 600 seconds for CPLEX and 60 seconds for the evolutionary algorithm, Table 2.9 summarizes the results. The solutions found by CPLEX are proven optimal for three of the eggholder instances and two of the Matyas instances. However, when unable to prove optimality, the solutions obtained by CPLEX are still very good, and the optimality gap is small. The solutions

Table 2.8: Description of instances based on benchmark functions.

Instance	Function	x	y	Description
E1	Eggholder	[20, 100]	[30, 70]	Small areas with negative values
E2	Eggholder	[0, 100]	[0, 100]	Two fairly large areas with negative values
E3	Eggholder	[200, 400]	[200, 400]	Quite many negative values, irregular shapes
E4	Eggholder	[300, 500]	[200, 400]	Many valleys with negative values
E5	Eggholder	[300, 400]	[200, 300]	One deep valley, about half of board is negative
M1	Matyas	$[-30, 30]$	$[-30, 30]$	Valley in the middle, high hills on the sides
M2	Matyas	$[-40, 40]$	$[-40, 40]$	Valley in the middle, very high hills on the sides
M3	Matyas	[0, 50]	[50, 100]	Hill-side, with huge gain values
M4	Matyas	[0, 40]	[10, 50]	Hilly area, some low gain values in one corner
M5	Matyas	[0, 40]	[0, 40]	Hilly area, one part of the board with low gains

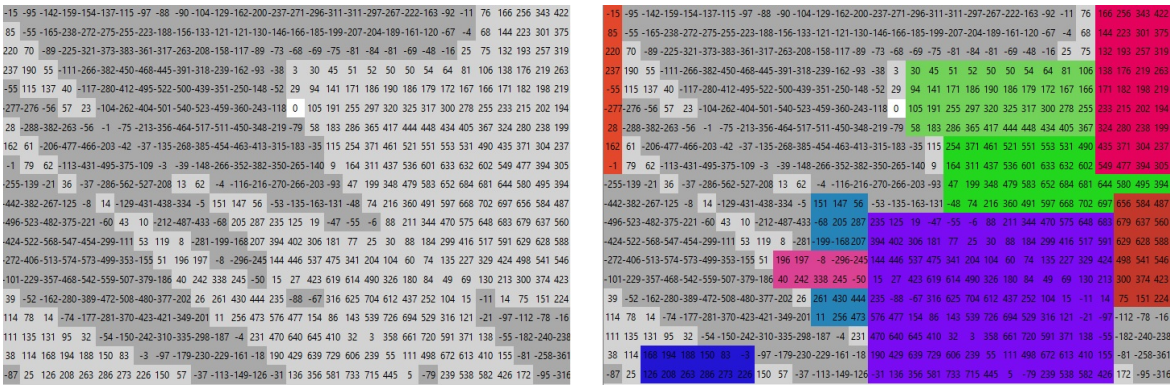


Figure 2.5: Instance E3, with the board values shown on the left and the optimal solution, placing nine rectangles, to the right.

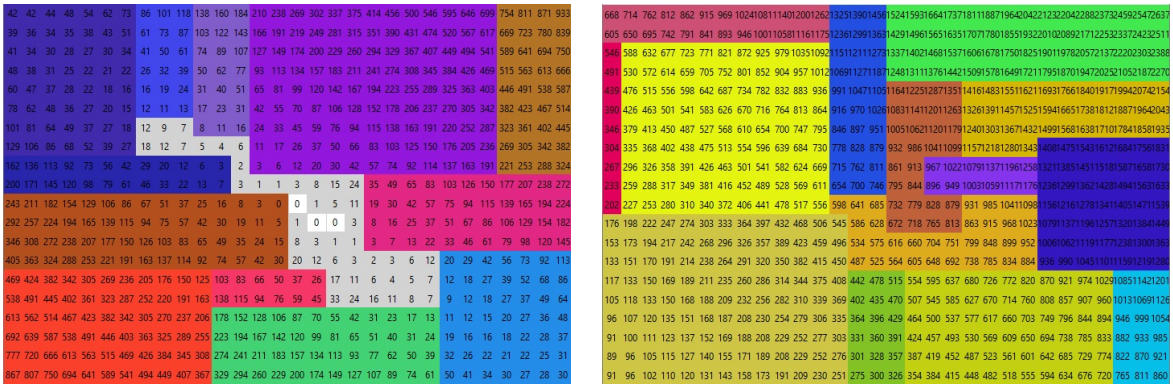


Figure 2.6: Instance M1 with its optimal solution is shown to the left, and instance M3 and its best known solution is shown to the right.

obtained by the heuristic are in general also very good, but not better than CPLEX for any of the instances. Compared to the randomly generated instances, these instances are relatively hard, especially in the cases where many rectangles should be purchased to obtain an optimal solution. As an example, the best solution found for M3 involves covering the entire board using 14 rectangles, yet CPLEX is not able to prove optimality within 600 seconds.

Table 2.9: Results for instances based on the eggholder and Matyas function.

	CPLEX				Heuristic		
Instance	UB	LB	TT	Gap [%]	LB	TB	Gap [%]
E1	33849	33719	600.0	0.39	33349	39.0	1.47
E2	21800	21800	50.7	0.00	21708	59.3	0.42
E3	81089	81089	32.4	0.00	81089	6.6	0.00
E4	108308	108059	600.0	0.23	107829	17.3	0.44
E5	41082	41082	127.1	0.00	41082	43.5	0.00
M1	94118	94118	48.2	0.00	93850	13.8	0.28
M2	173415	173266	600.0	0.09	172977	4.9	0.25
M3	506058	505712	600.0	0.07	505329	1.2	0.14
M4	67421	67108	600.0	0.47	67074	2.7	0.51
M5	45581	45581	489.5	0.00	45435	1.2	0.32

In Section 2.5.1 and Section 2.5.2 I saw that the performance of CPLEX is worse for larger board sizes and with a larger number of rectangles to be placed. An additional test was executed to see if this still holds when the board is not randomly generated, but rather has a structure as imposed by the eggholder and Matyas functions. To this end I generate a second set of instances based on the same areas as indicated in Table 2.8. However, I increase the size of the board from 20×30 to 40×60 and the number of rectangles from 15 to 60. The results are shown in Table 2.10. As can be seen, CPLEX still performs ok for some of the eggholder-instances, but fails to find any non-trivial solution to four of the five Matyas-instances.

2.5.5 Covering by satellite images

The final set of test instances are based on the idea of analyzing a geographical landmass through images captured by satellites. Thus, the board has two types of cells: those representing a body of water, which are considered uninteresting and brings a zero revenue if covered, and those representing land, which are interesting and brings a high revenue if covered. The size of the board is 40×60. There are five types of rectangles, with a height of $h_k = 4$ and widths $w_k \in \{4, 6, 8, 10, 12\}$, with costs $c_k = 10 + w_k$. There are 100 rectangles of each type, so the complete landmass can be covered easily. In total, seven such instances are generated, starting with a small landmass in instance S1, and gradually growing the landmass until reaching S7.

Table 2.11 shows the results. Both CPLEX and the heuristic provides high quality solutions to these instances. CPLEX solves six of the seven instances to optimality, and finds a solution within 0.28% of the optimal solution on the instance

Table 2.10: Results for additional instances based on the eggholder and Matyas functions.

		CPLEX			Heuristic		
Instance	UB	LB	TT	Gap [%]	LB	TB	Gap [%]
E1b	138921	127308	600.0	9.1 %	135858	1.6	2.3 %
E2b	92849	91761	600.0	1.2 %	89634	2.6	3.6 %
E3b	364288	359452	600.0	1.3 %	356970	1.2	2.1 %
E4b	472878	445730	600.0	6.1 %	460735	2.7	2.6 %
E5b	202613	199712	600.0	1.5 %	198903	2.6	1.9 %
M1b	361246	348695	600.0	3.6 %	358724	2.4	0.7 %
M2b	693316	0	600.0	N/A	659870	2.3	5.1 %
M3b	2025950	0	600.0	N/A	1990160	3.1	1.8 %
M4b	292514	0	600.0	N/A	260790	3.1	12.2 %
M5b	207555	0	600.0	N/A	177549	1.5	16.9 %

S6. The heuristic is within 0.24% of the optimal solutions for all instances, and finds a better solution than CPLEX on instance S6. Figure 2.7 shows the solutions found by CPLEX for instances S3 (optimal) and S6 (not optimal).

Table 2.11: Results for instances based on covering a landmass using satellite images.

		CPLEX			Heuristic		
Instance	UB	LB	TT	Gap [%]	LB	TB	Gap [%]
S1	18094	18094	6.8	0.00	18086	26.2	0.04
S2	25036	25036	10.2	0.00	25018	19.4	0.07
S3	46106	46106	40.2	0.00	46028	3.5	0.17
S4	70084	70084	138.9	0.00	69982	7.0	0.15
S5	96386	96386	448.2	0.00	96206	10.2	0.19
S6	105675	105384	600.0	0.28	105500	6.1	0.17
S7	123818	123818	444.7	0.00	123520	10.2	0.24

2.5.6 Robustness of performance

In the previous tests I have evaluated the behaviour of the two solution methods when systematically varying certain aspects of the instances solved, such as the size of the board, the number of rectangles, or the structure of the gain values. In the following I examine whether the behaviour of the solution methods varies substantially when solving instances that are similar in structure. That is, I keep most of the attributes of the instances constant, and only randomly perturb some aspect thought to be insignificant. Five groups of instances are created, with each group consisting of ten perturbed instances.

In the first group of instances, I fix $m = n = 50$ and $|R| = 20$. However, I randomly select each gain value from $\{1, \ldots, 100\}$, and the height and width of each rectangle from $\{1, \ldots, 25\}$. The cost of each rectangle is selected randomly between 0 and the size of the rectangle times the average gain. For the second group of

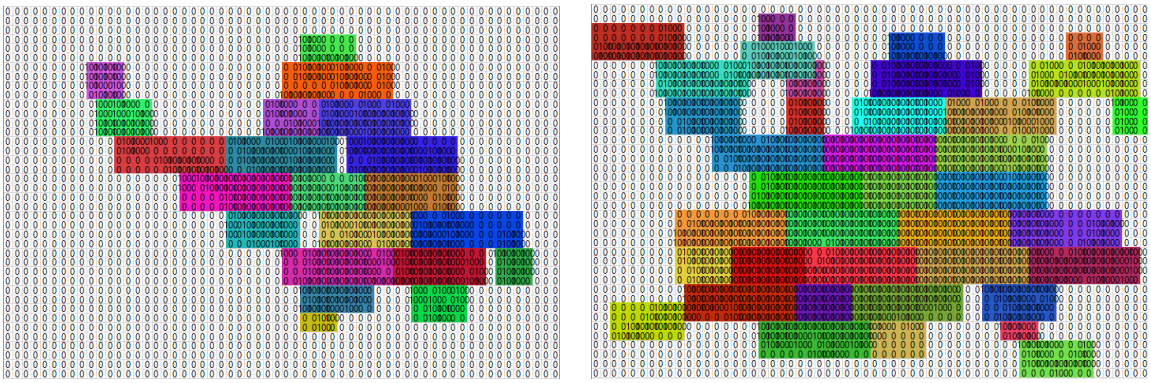


Figure 2.7: Solutions to instances S3 (left), and S6 (right).

instances, the exact same procedure is followed, except that each gain value is either 0 or 1. Instances in group 3 are generated as in Section 2.5.1, with $p = 10$, so again the gains and rectangles vary between instances within the group. In group 4, instances are based on the Eggholder function and E3, but the 15 rectangles are chosen randomly for each new instance. Group 5 follows the same setting as group 4, but based on Matyas and M3.

Table 2.12 shows the results for each group of instances (G). For both CPLEX and the heuristic, the minimum, average, and maximum gaps to the upper bound reported by CPLEX are presented. For CPLEX, I also report the number of instances where optimality was proven (OPT). Furthermore, I report the minimum, average, and maximum total time (TT) for CPLEX and time to best (TB) for the heuristic.

Table 2.12: Results for five groups of instances, each group containing ten instances with similar characteristics.

		CPLEX						Heuristic					
		Gap [%]			TT			Gap [%]			TB		
G	Opt	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
1	1/10	0.0	2.2	3.4	137.0	553.7	600.0	2.3	3.3	4.5	4.0	22.3	47.2
2	7/10	0.0	0.8	4.0	19.3	217.0	600.0	0.7	4.1	6.1	2.2	21.9	49.8
3	10/10	0.0	0.0	0.0	14.6	44.0	81.3	0.0	0.0	0.0	2.5	12.3	50.7
4	10/10	0.0	0.0	0.0	8.8	28.6	55.3	0.0	0.0	0.0	0.4	15.3	45.6
5	1/10	0.0	0.0	0.1	538.6	593.9	600.0	0.1	0.1	0.2	0.4	16.1	44.0

For each of the groups, the performance of the solution methods remains relatively stable. Both CPLEX and the heuristic find the optimal solution for all instances in groups 3 and 4, although the time required to find the best solution (for the heuristic) and to prove optimality (for CPLEX) varies slightly across instances. Similarly, in group 5, the performance is also remarkably consistent. Although CPLEX is only able to prove optimality for one out of ten instances, the optimality gap remains small—never exceeding 0.09% for CPLEX and 0.17% for the heuristic. For groups 1 and 2, the variance in performance is more pronounced: the gaps range from 0% to 4.0% for CPLEX, and from 0.7% to 6.1% for the evolutionary algorithm. In these groups, both the rectangles and the gain values are perturbed. This may indicate

that gain values have a greater impact on the stability of solver performance than the rectangle sizes and costs. Although gain values are also varied in group 3, the instances in that group appear to be easier, and the performance, measured in terms of optimality gaps, remains unaffected.

2.6 Discussion

In this chapter, the Board Packing Problem (BoPP) was addressed. The problem was introduced in [2, 47]. A comparison was made with related packing models in the literature, and it was proved that the BoPP is $\mathcal{NP}$-hard. A MILP formulation of the problem was presented, and an evolutionary algorithm (EA) was introduced as a solution approach.

Benchmark instances were defined. It was observed that CPLEX can solve smaller instances efficiently, whereas for larger instances an optimal solution could not be obtained within a reasonable time limit; moreover, the EA was found to handle such large instances effectively. Here are some characterization of BoPP instances concerning exact solver vs. evolutionary algorithm:

- **Exact MILP solver (CPLEX).** The exact solver dominates on small, sparse, and almost non-overlapping boards. Typical cases include boards of size $\leq 30 \times 30$ with at most 15 rectangles, where a few high-gain hotspots exist and costs are high elsewhere. In these settings, the LP relaxation is already tight, so CPLEX proves optimality within minutes.
- **Evolutionary algorithm (EA).** The EA performs better on large, dense, and highly overlapping instances. Typical cases include boards of size $\geq 40 \times 40$ (or 100×100 in the Industry-L set), with at least 25 rectangles and nearly uniform gains and costs, leading to broad profit plateaus. In such instances, the MILP model explodes into tens of thousands of binaries, while the EA efficiently explores the landscape and finds near-optimal profit within 60 seconds.
- **Rule of thumb.** The exact solver is preferable for *small and sharp* instances, while the EA is preferable for *large and flat* instances, especially when the time budget is short (under one minute).

Additional approaches may also be considered. Several directions that remain for further research are listed below.

1. **Brute Force (BF).** A brute-force approach evaluates all possible configurations and selects the best one. Such an algorithm is feasible only for very small inputs; for example, BF is trivial when there is only a single rectangle. Note that BF can also be employed as a subordinate (*slave*) procedure within a more sophisticated method, as discussed below.
2. **Greedy heuristic.** A simple greedy strategy proceeds as follows. First, the rectangles are ordered (e.g., in nonincreasing order of size), and this order is fixed. The first rectangle in the list is then placed in its best position (considering this rectangle alone for the moment), a step that can be carried out by a brute-force subroutine. After placement, the gains covered by this rectangle are set to zero. The second rectangle is then placed in its best position with respect to the updated gains, followed again by zeroing out the

gains it covers, and so on until all rectangles have been processed.

3. **Local Search (LS).** All K rectangles are first placed in an initial feasible configuration. Then, one rectangle is removed from the current allocation, and the objective value of the remaining configuration is denoted by z. The removed rectangle is reallocated to a position that maximizes the increase in the objective value relative to z. This remove–reallocate step is repeated iteratively.
   Several strategies can be used to select which rectangle to remove. One option is random selection. Alternatively, a contribution-based rule can be applied: for a given allocation, the *contribution* of each rectangle is computed and the rectangle with the smallest contribution is chosen for removal and reallocation. When overlap is disallowed, the contribution is unambiguous. When overlap is permitted, the contribution associated with a cell (i, j) can be shared equally among all rectangles that cover it, so that each rectangle's total contribution is defined as the sum of its shares over all covered cells.
   Modern evolutionary algorithms frequently incorporate such local-search subroutines as embedded *slave* procedures, invoked when no promising new candidates (feasible solutions) are found during the search.
4. **Neighborhood-based local search.** A feasible packing (the *current* or *old* packing) is taken as a starting point, and small modifications are applied to obtain a *neighbor* (the *new* packing). The objective values are then compared. If the new packing yields a better objective value, the old packing is discarded, the new packing is retained, and the step is repeated iteratively. A variety of neighborhood operators can be employed (e.g., single-rectangle shifts, swaps, or localized reinsertions); the choice of operator and the selection policy (first-improvement vs. best-improvement) determine the search dynamics.
5. **Tabu Search (TS).** Tabu Search is another option worth investigating as a potential improvement over the EA, by exploring neighborhoods while using tabu restrictions to avoid cycling and to encourage diversification.
6. **Genetic Algorithm (GA).** In a GA framework, each feasible allocation (packing) constitutes an individual in a population. A typical crossover operator proceeds by combining two parent packings (Packing 1 and Packing 2): the positions of a subset of rectangles are inherited from Packing 1, while the positions of the remaining rectangles are inherited from Packing 2. When the parents do not use the same subset of rectangles or when conflicts arise, a repair or smoothing procedure is applied to restore feasibility and improve coherence. Mutation is performed by moving from the current packing to a neighboring packing, as described in the local-search scheme above (e.g., single-rectangle shifts, swaps, or reinsertions).

Some difficulties may arise when applying the above methods. Consider, for instance, the local search algorithm described in Point 3. Suppose a portion of the board is as follows:

1	1	1	1	2	2	3	3	3	3	2	2	1	1	1

Suppose this is certain part of the board, and the gain value is zero in any other cell. Assume now that two squares, each of side length 4, are available. The first square is placed optimally so that it covers all cells of value 3. The second square is then placed immediately to the left or to the right of the first square. Both squares are now in locally optimal positions, and the local search terminates without finding an improvement. However, the global optimum in this configuration is achieved when the two squares jointly cover all cells with g-values greater than 1 (i.e., cells with gain 2 or 3).

This illustrates that a simple LS can become trapped in a local optimum, whereas the neighborhood-based LS of Point 4 may succeed when at least two squares are removed and reallocated simultaneously. Tabu Search may also outperform the basic LS of Point 3 in such settings.

At this point, the discussion of potential solution methods is concluded; it is simply noted that many avenues remain for further investigation of the BoPP.

2.6.1 Other possible versions of the problem

Finally, some further ways to generalize the BoPP are noted (the present formulation is referred to as **Problem 1**).

Problem 2. Another version is defined as follows. Given the same data as in Problem 1 (i.e., a board with gains $g_{i,j}$ and a set of rectangles with specified heights, widths, and costs), a *knapsack-type* variant is obtained by introducing a budget B: rectangles may be purchased only if the total cost of the purchased set does not exceed B. If B is infinite, or if B is at least the sum of costs of all rectangles, then the budget constraint imposes no restriction and the model reduces to Problem 1. Likewise, if all rectangle costs are zero, the model again reduces to Problem 1 because all items can be purchased.

Problem 3. A slightly different version is a minimization problem rather than a maximization problem. Exactly $L \leq K$ rectangles (from the given K) must be allocated. Two subcases may be considered: overlap permitted and overlap disallowed. Here, the values $g_{i,j}$ represent *cost* (rather than profit), and the task is to place the chosen L rectangles so that the total cost of the covered area is minimized. Any of these variants remains open for further investigation.

Chapter 3

Square Packing and Covering

Publications

The following publications form the foundation for my research discussed in this chapter and provide essential background and context:

- Balogh, J., Dosa, Gy., Hvattum, L.M., **Olaj, T. A.**, Tuza, Zs.: Guillotine cutting is asymptotically optimal for packing consecutive squares. *Optimization Letters*, 16, pages 2775–2785, doi: 10.1007/s11590-022-01858-w, 2022, **Q1 journal, IF=1.502**, see [12].
- Balogh, J., Dósa, Gy., Hvattum, L. M., **Olaj, T. A.**, Szalkai, I., Tuza, Zs.: Covering a square with consecutive squares. *Annals of Operational Research, Springer*, `https://doi.org/10.1007/s10479-025-06633-5`, 2025, **Q1 journal, IF=4.82**, see [15].
- Other related presentations include, [13, 46].

3.1 The problems under investigation

In the present chapter, attention is restricted to a special square packing problem and its dual, a square covering problem. The problem under consideration originates from a question popularized by Martin Gardner in his columns from 1966 and 1975 [58, 59], which in turn was based on a previously unpublished problem posed by R. B. Britton.

Square packing problems represent a classical line of research within discrete optimization. The objective is to place a given set of smaller squares into a larger accommodating square without overlap, often with the goal of minimizing the size of the large square or verifying the feasibility of a given size. This type of problem is closely related to covering variants, where the aim is to ensure that the large square is completely covered by the smaller ones, possibly allowing overlaps. These formulations provide a rich ground for both theoretical investigations and algorithmic approaches, and they motivate the more detailed analysis presented in the following sections.

The considered packing problem is the following:

Definition 4. *Given one square of size* 1*, one square of size* 2*, etc., and finally one square size of size* n *(where* n *is some natural number), these squares are referred to as items. The set of squares will be denoted by* $[1, n]$*. The question is the following: what is the smallest square of size* K *(also called as accomodation square or board), so that all items fit into* K*, without rotation and overlap. This problem corresponds to Problem A005842 in the Online encyclopedia of Integer Sequences [113].*

The setting considered assumes that all squares must be packed axis-parallel; that is, rotations are not allowed. The packing must be non-overlapping, but squares may touch each other along their edges. Further natural assumptions include that squares cannot be split, and that all corner coordinates are integers.

For this problem, Gardner [59] listed the best possible (tight) results up to $n = 17$ based on earlier works. An upper bound (UB) is tight if it coincides with a valid lower bound (LB). The packing of consecutive squares has become a benchmark problem for general square packing problems, e.g. it is used to test more general rectangle packing algorithms [82, 99, 112].

Rectangle packing has been applied in VLSI design, and also can be used to model some real-world problems like scheduling, cutting stock and pallet loading problems. In most of these areas rotation of the rectangles is allowed; however, there are some problems, for example in scheduling, where rotation is forbidden.

I review the current best upper bounds for the packing problem [12], and it can also be found on the webpage [113], searching for the integer sequence A005842. There the best found $UB(n)$ values are listed till $n = 56$ (most of them are tight), where $UB(n)$ is the upper bound for the packing problem with a given parameter n. One of the best lower bounds come from the simple fact that I can get a fairly good estimate by rounding up the square root of the total area of the squares, i.e. in my case, I get the value $LB_0 := \lceil \sqrt{\sum_{k=1}^{n} k^2} \rceil = \lceil \sqrt{n(n+1)(2n+1)/6} \rceil$. Tightness (i.e. when the UB equals a particular LB) has not been proved for the cases $n = 38, 40, 42, 48, 52, 53$ and 55, while the other values of $UB(n)$ given in [113] are proved to be tight for $1 \leq n \leq 56$.

The latest results have been achieved by Hougardy [73]. He showed that $UB(28) = 89$, $UB(32) = 108$, $UB(33) = 113$, $UB(34) = 118$, and $UB(47) = 190$, and proved that these upper bounds are tight; i.e., $LB_0 + 1$ is a matching lower bound in these cases.

In the beginning of the chapter, I will provide several lower bounds for the packing problem, and also I provide some packings from the literature. It turns out, that for any $1 \leq n \leq 24$ (except $n = 18$ and $n = 24$), I can validate the tightness of the best known packing by a simple lower bound.

An interesting special case of the problem is $n = 24$, which was unsolved until 2004. As proved already in [118], $n = 24$ is the unique non-trivial case where $A_n := \frac{n(n+1)(2n+1)}{6}$ is the square of an integer. However, while $LB_0 = 70$, the optimum is 71 as proved by Korf [83]. Korf applied a computer-aided proof: the impossibility of a perfect packing for $n = 24$, $K = 70$ has been demonstrated by computer search, as claimed first in [21] without giving details and later in [83, 84]. However, until now, no proof without use of computer was given. The gap is filled

by [111] recently, giving a complete combinatorial proof. (Some initial steps towards a combinatorial proof were given in [86].)

However, the case $n = 18$ remains open at this point. It is known (from the literature and from computer searches) that the squares $[1, 18]$ cannot be packed into a square of side 46, but can be packed into a square of side $K = 47$. Thus, $K = 47$ is known to be the optimal value; what remains unknown is the reason *why*. See Figure 3.1 for a CPLEX/Python visualization of this example.

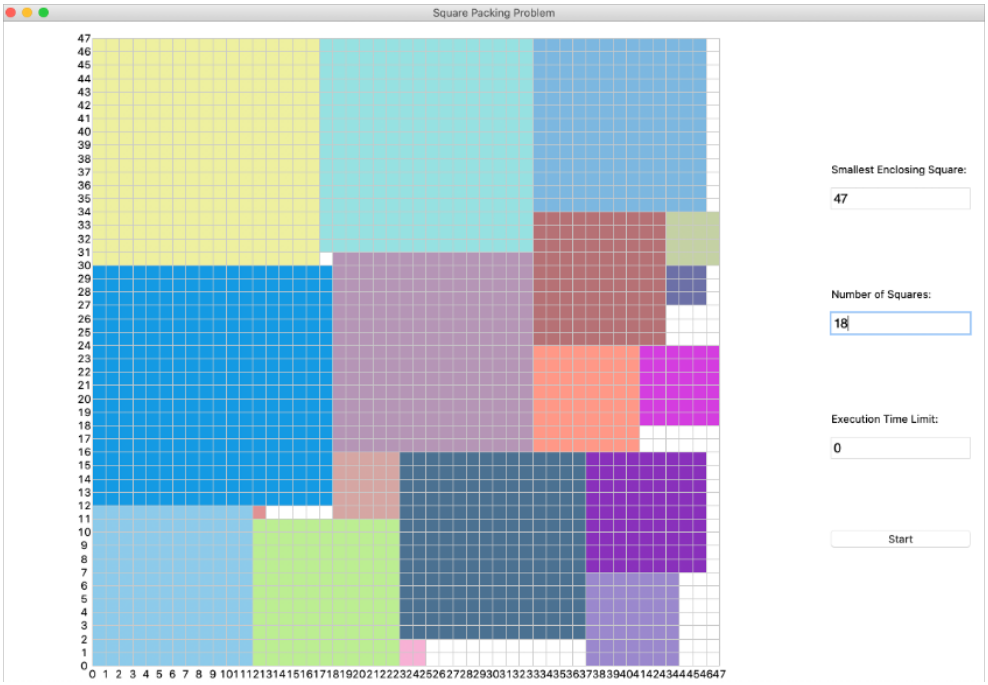


Figure 3.1: Solution for $n = 18$, and $K = 47$, using Python interface for the Square Packing Problem, and CPLEX as a solver.

In this thesis, I also review some related packing problems, where an arbitrary set of squares with a total area of at most 1 should be packed into a smallest possible square or into a rectangle of minimum area [12].

In this line of thinking it was proved in [101] that any rectangle of area 2 suffices, provided that its side lengths are not smaller than the largest square to be packed. A stronger upper bound can be derived from [81], stating that a covering rectangle with an area of at most $\sqrt{8/3} < 1.633$ can be given for any input of this type. This result was further improved to 1.53 in [103]. A further improvement to this was given in [72], where it was proved that a rectangle with area smaller than 1.4 can be used to cover all squares.

Moreover, several papers deal with higher dimension; see e.g. [10, 79, 104, 110]. Gap-free arrangements, called perfect packings, of the infinite series of squares and boxes of side lengths $\{n^t\}_{n\geq 1}$ for a certain range of t have also been considered; see [29, 75, 77, 78, 110].

A similar question, where the container is a square was answered in [25]. Here the authors proved that any set of squares can be packed into the unit square if (i) the total area of the squares is at most 5/9 and (ii) none of the squares has edge length

larger than 1/2. They also proved that 5/9 is the best possible such value. It is worth mentioning here that in their packing, the patterns used fulfill the guillotine pattern properties. Their packing is similar to the First Fit Decreasing Height (FFDH) heuristic of [31] designed for the two-dimensional strip packing problem, but they used it both horizontally and vertically, resulted FFDW and FFDL (i.e. for width and length).

I give the first asymptotic results for the packing problem [12]. Namely, it is shown that there exists a constant $c < 1$ such that the square of size $N + cn$ admits a guillotine-type packing of the squares of sizes$[1, n]$, where $N := \sqrt{A_n} = \sqrt{\frac{n(n+1)(2n+1)}{6}}$.

3.1.1 The covering version

Question: *Can a square of size $K \times K$ be completely covered by using the consecutive squares $[1, n]$, i.e., the set $[1, n]$ of small squares? In other words, what is the largest K such that these small squares can completely cover $K \times K$ when allocated carefully?*

In this formulation, overlapping between small squares is allowed. To the best of the author's knowledge, this question has not been previously investigated in the literature.

Observation. For the special case $n = 24$, the small squares $[1, 24]$ cannot be packed without overlap into a 70×70 accommodation square, although their total area is exactly 70×70. It follows that in the corresponding *covering* problem, the same set of small squares cannot completely cover the 70×70 square either. Hence, for $n = 24$, the largest admissible value of K in the covering problem must be strictly smaller than 70.

Thus, the two problems are naturally connected:

1. determining the smallest square that accommodates $[1, n]$ without overlap.
2. determining the largest square that can be completely covered by $[1, n]$, allowing overlap.

Results for the covering problem:

- For small values of n the covering problem is easy, and the optimal solutions are given. These can be calculated by hand or by a mathematical programming solver. There are several ways to formulate the problem using mathematical programming.
- For moderately bigger values of n, such as $n = 23$, the problem is already hard for mathematical programming solvers, so a heuristic algorithm was created to find near-optimal solutions also for large values of n.
- An expansion algorithm is provided that, from a given good cover of $K \times K$ for some n, can generate a cover for some $K' > K$ using small squares up to and including $n + 1$.

- It is also shown that a simple covering policy can generate an asymptotically optimal covering.

3.2 Square Packing Problem

3.2.1 Small cases

Question: *what is the smallest accommodation square where the small squares can be packed into it without overlap?*

This problem originated from Martin Gardner [58, 59]. For this packing problem, Gardner [59] listed the tight results up to $n = 17$, based on earlier works.

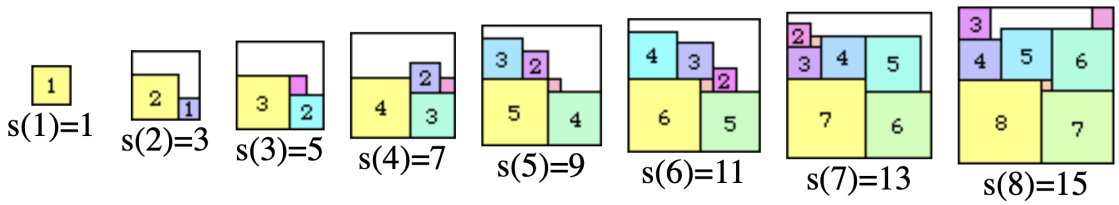


Figure 3.2: The best known feasible packing solutions of the first cases until $n = 8$ (Friedman [57]).

Note, that the case for $n = 1$ is trivial, also for the case of $n = 2$. If $n = 3$, the two biggest squares must fit next to each other, so the accommodating *big* square must have size at least $2 + 3$. This property holds for any bigger instances too, such that I get a valid lower bound for a packing.

Claim 5. *The following value is a lower bound for the size of the accommodation square:*

$$LB_1 = n + (n - 1).$$

The lower bound applies not only to consecutive instances, but to any set of squares: the sum of the sizes of the two largest squares constitutes a valid lower bound.

It can be observed that LB_1 is a *tight* bound for any $1 \leq n \leq 8$. Consider the case $n = 8$. In this case, $K = 15$ is a lower bound for the size of the accommodating square. A feasible packing into this size can be obtained without the need for a sophisticated construction method, since valid packings are relatively easy to identify. Moreover, there exist multiple valid packings of $[1, \ldots, 8]$ into $K = 15$; for example, the square of size 3 may be shifted slightly to the right in Figure 3.2.

It has been established that LB_1 is a *tight* bound for any $1 \leq n \leq 8$, but it is not tight for $n = 9$. For example, when $n = 8$, the squares of $[1, 8]$ fit into an accommodating square of size $8 + 7 = 15$. In contrast, for $n = 9$ the size $9 + 8 = 17$ is insufficient, since the squares of $[1, 9]$ cannot be packed into a square of this size.

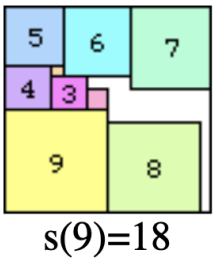


Figure 3.3: The best known feasible packing solution for the case $n = 9$ (Friedman [57]).

The reason for this discrepancy is not immediately clear. The total area of the squares up to $n = 9$ is $\sum_{k=1}^{9} k^2 = 285$, while the area of a square of size 17 is $17^2 = 289$. Thus, when considering only the total area of the small squares, there is no obvious explanation for why $[1, 9]$ cannot fit into a square of size 17. This shows that LB_0 is not tight for $n = 9$. However, there exist cases where LB_0 is tight; for example, when $n = 17$.

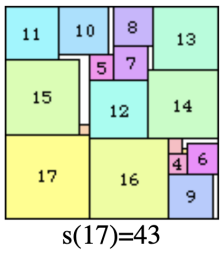


Figure 3.4: The best known feasible packing solution for the case $n = 17$ (Friedman [57]).

For $n = 17$, the sum of areas of the small squares is $\sum_{k=1}^{17} k^2 = 1785$, then the square root is $\sqrt{1785} \approx 42.249$, after rounding up the result is $LB_0 = 43$. Figure 3.4 shows that this size is sufficient. Since $43^2 - 1785 = 64$ cells remain empty, constructing a valid packing is relatively straightforward. Tightness (i.e., LB_0 is tight) implies that at least $K = 43$ is required as the size of the accommodating square. If a valid packing can be constructed within a square of this size, then $K = 43$ constitutes the tight bound. The detailed construction of such a packing is not addressed here. In some cases, valid packings are relatively easy to identify, whereas in other cases they require computer-based search. Since such packings are already available in the literature and online repositories [113], it is sufficient for the present purpose to note that a valid packing exists. This confirms that the lower bound LB_0 is tight, for example when $n = 17$.

It should be noted that LB_0 is tight for many values of n. For example, [113] reports several cases where LB_0 is tight. Considering $n \leq 32$, the tight cases (as listed in [113]) are highlighted in bold in the table below.

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
tight	3	5	7	9	11	13	15	18	21	24	27	30	33	36	39
LB_0	**3**	4	6	8	10	12	**15**	17	20	23	26	29	32	**36**	**39**

Table 3.1: The tight values and the lower bounds from $n = 2$ to $n = 16$.

n	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
tight	43	47	50	54	58	62	66	71	75	80	84	89	93	98	103	108
LB_0	**43**	46	**50**	**54**	**58**	**62**	**66**	70	**75**	79	**84**	88	**93**	**98**	**103**	107

Table 3.2: The tight values and the lower bounds from $n = 17$ to $n = 32$.

Now, I will introduce another, and more tricky lower bound. For this, I need some preparation.

Definition 6. *Square A is said to be to the left of square B if there exists a vertical line l that separates them, such that A lies to the left of l and B lies to the right. Similarly, square A is said to be above square B if there exists a horizontal line l that separates them, such that A lies above l and B lies below.*

Note, that if A is to the left from B then B is to the right from A. Similarly, if A is above B, then B is below A. Note also that the border of the squares are allowed to be on the separator line, but no interior point of the squares are allowed to.

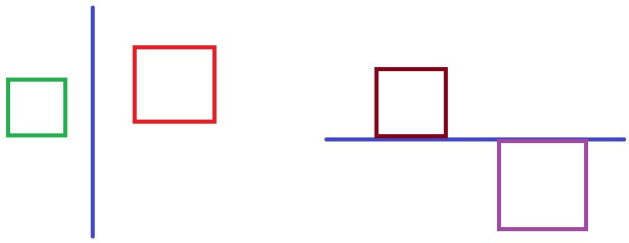


Figure 3.5: The vertical separator, and the horizontally separated squares.

Definition 7. *Squares A, B, and C are said to be collinear if either A is to the left of B and B is to the left of C, or A is above B and B is above C.*

Now my statement is the following:

Lemma 8. *Taking any five squares, there are at least three squares among them which are collinear.*

Proof. Let us suppose that I have five squares in a valid position (parallel to the sides of the accommodating big square and in not overlapping positions). Let us

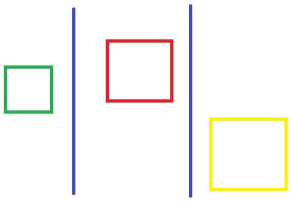
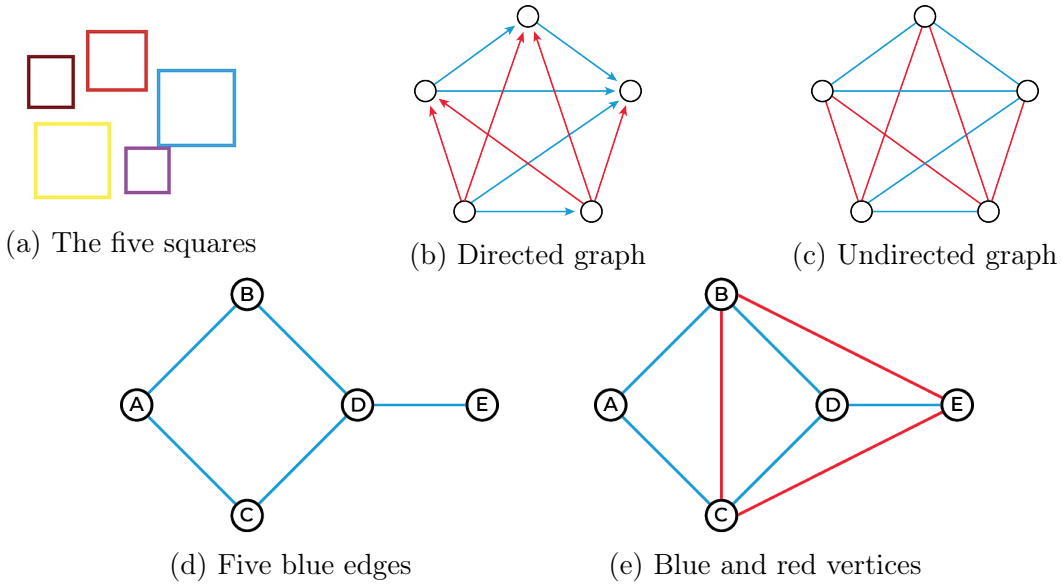


Figure 3.6: Collinear squares

construct a graph. I take a vertex for any square. If two squares are separated by a horizontal line, then I connect the corresponding vertices by a red edge. The edge is directed so that it departs from the vertex that belongs to the square below and ends in the vertex that belongs to the square up.

Similarly, if two squares are separated by a vertical line, then I connect the corresponding vertices by a blue edge. The edge is directed so that it departs from the vertex that belongs to the left square and ends in the right vertex.

In this way, I get a colored directed complete graph, as any two vertices are connected. If e.g. A is to the left from B, and also, A is below B, from the two edges I keep any of them and delete the other. So there is exactly one directed edge between any two vertices. A visualization can be found in Figure 3.7.



(a) The five squares (b) Directed graph (c) Undirected graph

(d) Five blue edges (e) Blue and red vertices

Figure 3.7: The five squares and the corresponding colored graphs.

It suffices to prove that there exists an oriented P_3 in the graph, where the two edges have the same orientation. Specifically, there are three vertices, A, B, and C, such that there is a directed edge from A to B and another from B to C, with both edges sharing the same color.

For a moment, consider the graph obtained by deleting the orientations of the edges (see Figure 3.7c). The result is a colored complete graph on five vertices. Since

the graph has ten edges, at least one color must occur on at least five edges. Without loss of generality, assume that this color is blue. From these, select arbitrarily five blue edges.

Since I have 5 vertices and at least 5 edges in this blue graph, there must be a blue cycle in the graph (as any cycle-free graph on k vertices can have at most $k-1$ edges). It is easy to check, that taking a triangle from an undirected graph, giving any orientation to the edges, I will have an oriented P_3 in the graph. Similarly, if I have a C_5 in an undirected graph, giving any orientation to the edges, I will have an oriented P_3 in the graph, see Figure 3.8.

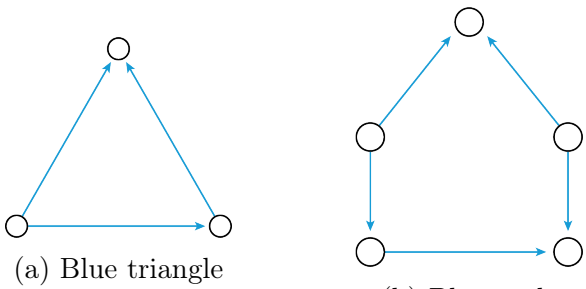
Thus, if there exists a blue triangle or a blue C_5, the argument is complete, since in that case the original graph (after restoring the edge orientations) necessarily contains an oriented P_3.

Let us suppose that there is no C_3, and no C_5 in the blue graph (of 5 vertices and 5 undirected edges), then there must be a C_4, and thus the graph looks as the following graph down left in Figure 3.7d.

Then (see Figure 3.7):

- If there is a blue edge in the original graph between B and E, then I have a blue triangle (with vertices B, D and E) and I will have an oriented P_3.
- If there is a blue edge in the original graph between C and E, then I have a blue triangle (with vertices C, D and E) and I will have an oriented P_3.
- If there is a blue edge in the original graph between B and C, then I have a blue triangle (with vertices B, C and D) and I will have an oriented P_3.

Hence, all of these edges in the original graph are red edges (see Figure 3.7e), and I have an oriented red P_3.



(a) Blue triangle

(b) Blue cycle

Figure 3.8: Oriented C_3 and C_5

□

Corollary 9. *The following value is a lower bound for the size of the accommodation square:*

$$LB_2 = (n-2) + (n-3) + (n-4) = 3n - 9.$$

Proof. Let us consider the five biggest squares, with sizes $n, n-1, n-2, n-3, n-4$. There are three squares among these five squares that are collinear. Thus the size of

the accommodation big square is at least the sum of the sizes of the three smallest squares among the five squares. □

It can be observed that LB_2 is a tight lower bound for $n = 9, \ldots, 16$, but not for larger values of n.

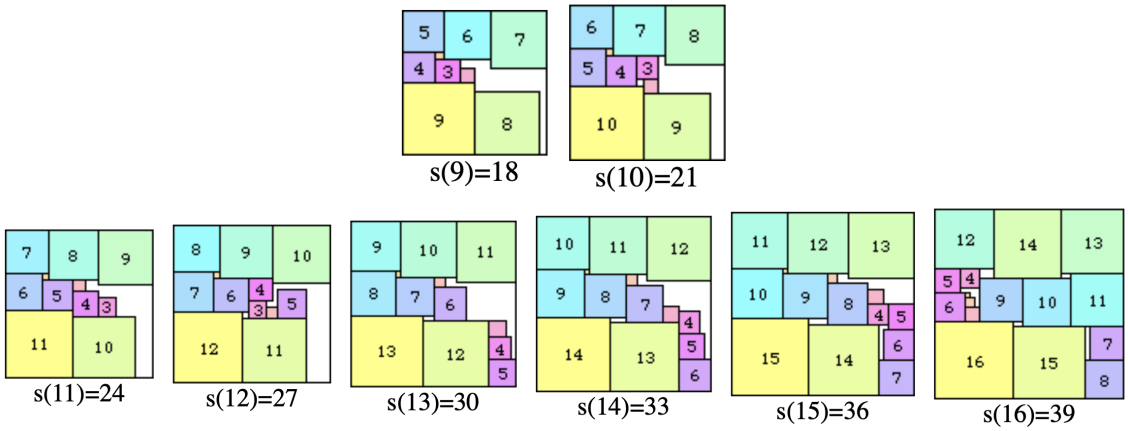


Figure 3.9: The best known feasible packing solution for the cases $n = 9$ to $n = 16$ (Friedman [57]).

3.2.1.1 All cases up to $n = 24$ and more

The following presents a summary of the results for $n = 2, ..., 24$. First, for $n = 2, ..., 8$. For all these cases, LB_1 is tight, and this lower bound is the only one that is tight (except that LB_0 is also tight but only for $n = 2$ and $n = 8$, and LB_2 is also tight for $n = 8$). I give the tight lower bound values by bold numbers.

n	2	3	4	5	6	7	8
tight	3	5	7	9	11	13	15
LB_0	**3**	4	6	8	10	12	**15**
LB_1	**3**	**5**	**7**	**9**	**11**	**13**	**15**
LB_2	–	–	–	6	9	12	**15**

Table 3.3: The tight values and the lower bounds for $2 \leq n \leq 8$.

Next, for $n = 9, ..., 16$. For all these cases, LB_2 is tight, and this lower bound is the only one tight, except that LB_0 is also tight but only for $n = 15$ and $n = 16$. Unfortunately, after $n = 16$, LB_2 is never tight any more.

For $n = 17, \ldots, 24$, the bound LB_1 is already far from the tight value and is therefore omitted from the table. Observe that when increasing n to $n + 1$, the value of LB_2 always increases by 3. In contrast, LB_0 increases by 3 initially, but then by 4 in the considered range of n values. Consequently, for $n > 16$, LB_2 cannot remain competitive.

Finally, I present the cases for $n = 25, ..., 56$. For these, both LB_1 and LB_2 are far away from the tight value, so I provide only the best known value of the

n	9	10	11	12	13	14	15	16
tight	18	21	24	27	30	33	36	39
LB_0	17	20	23	26	29	32	**36**	**39**
LB_1	17	19	21	23	25	27	29	31
LB_2	**18**	**21**	**24**	**27**	**30**	**33**	**36**	**39**

Table 3.4: The tight values and the lower bounds for $9 \leq n \leq 16$.

n	17	18	19	20	21	22	23	24
tight	43	47	50	54	58	62	66	71
LB_0	**43**	46	**50**	**54**	**58**	**62**	**66**	70
LB_2	42	45	48	51	54	57	60	63

Table 3.5: The tight values and the lower bounds for $17 \leq n \leq 24$.

accommodation square, denoted by $UB(n)$ (known from literature), and the value of LB_0, which is sometimes tight, sometimes not.

A review of the current best upper bounds for the packing problem is provided in [12], and the same information can also be found on the webpage [113], under the entry for the integer sequence A005842.

The best known values of $UB(n)$ are listed up to $n = 56$ (most of them tight), where $UB(n)$ denotes the upper bound for the packing problem with parameter n. Since no other lower bound close to the tight value is known for the range $25 \leq n \leq 56$, only LB_0 is reported. The tightness of $UB(n)$ has not been proved for $n = 38, 40, 42, 48, 52, 53$, and 55, whereas for all other cases with $1 \leq n \leq 56$, the values of $UB(n)$ reported in [113] have been proved to be tight.

n	25	26	27	28	29	30	31	32	33	34	35	36
$UB(n)$	75	80	84	89	93	98	103	108	113	118	123	128
LB_0	**75**	79	**84**	88	**93**	**98**	**103**	107	112	117	**123**	**128**

n	37	38	39	40	41	42	43	44	45	46	47	48
$UB(n)$	133	139	144	150	155	161	166	172	178	184	190	196
LB_0	**133**	138	**144**	149	**155**	160	**166**	**172**	**178**	**184**	189	195

n	49	50	51	52	53	54	55	56
$UB(n)$	202	208	214	221	227	233	240	246
LB_0	**202**	**208**	**214**	220	226	**233**	239	**246**

Table 3.6: The upper bounds and the lower bounds for $n = 25$ to $n = 56$.

The following lemma also holds:

Lemma 10. *Taking any $k^2 + 1$ squares, there are at least $k + 1$ squares among them which are collinear.*

The proof is not provided here, but it can be derived by some more deep, graph theoretic fact. I used this lemma for $k = 2$ to provide the lower bound LB_2. Unfortunately, this kind of lower bound will not give any tight lower bound for $k > 2$, since the sizes of the squares in sequence $[1, \ldots, n]$ decrease very fast. For example, if I take $k = 3$ and $n = 18$, then I take the 10 biggest squares, which are $9, 10, ..., 18$, and take the four smallest squares among them. I get that the accommodating square must have size at least $9 + 10 + 11 + 12 = 42$, very far from the tight value.

At present, no additional lower bounds are known to the author. LB_1 and LB_0 are trivial and from the literature, while LB_2 is provided first in [13, 46]. As it can be seen in the tables above, LB_0 is sometimes tight, sometimes not, but it can be observed the following facts:

- if LB_0 is not tight, still its difference from the tight value is only 1,
- Considering the range of n values with $2 \leq n \leq 56$, it can be observed that LB_0 ceases to be tight after a certain point.

In this sense, the two following questions arise:

Question 1. What is the following limit superior: $\limsup_{n\to\infty} (tight(n) - LB_0(n))$. Does it remain 1 or it grows above 1. If the latter case is the truth, then what is the value of this limit superior.

Question 2. What is the following limit inferior: $\liminf_{n\to\infty} (tight(n) - LB_0(n))$. Is it 0 (i.e. for an arbitrarily big K, there exists such $n > K$, for which that the difference is zero)?

In the above formulas, $tight(n)$ denotes the tight value for a given n, and $LB_0(n)$ denotes the value of LB_0 for the corresponding n. Further investigation is required to answer the open questions. For larger values of n, the tight values are not known; therefore, when computing $LB_0(n)$, there is no reference value available for comparison.

Regarding the **special** case of $n = 24$ is presented in Appendix C due to space limitations.

3.2.2 An asymptotically optimal algorithm by guillotine cutting

To the best of the author's knowledge, no asymptotic-type upper bounds have been established for the problem of packing consecutive squares into a square. This study addresses this gap in the literature by presenting the first results of this kind, considering both guillotine-type packings and unrestricted axis-aligned packings. So I will fill this gap of the literature by delivering the first results of this nature, considering both guillotine-type packings and unrestricted axis-parallel packings.

My results are stated in terms of packing squares into rectangles, from which packing into squares will be a corollary. To simplify formulas, I introduce the nota-

tion $A_n := \frac{n(n+1)(2n+1)}{6}$ and $N := \sqrt{A_n}$. Hence $LB_0 = \lceil N \rceil$, and no smaller square can contain a packing of the squares of sizes (= side lengths) $1, 2, \ldots, n$. This result demonstrates that a slightly larger square can accommodate a valid packing.

Theorem 11. *There exists a constant $c < 1$ such that the square of size $N + cn$ admits a guillotine-type packing of the squares of sizes $1, 2, \ldots, n$.*

From the proof of the more general Theorem 14, it follows immediately that any $c > 7/8$ is a suitable choice for $n \geq n_0(c)$, where $n_0(c)$ denotes a threshold value that may depend on c. In the concluding section, I outline potential improvements aimed at reducing the constant. Even better bounds can be proved if the structure of cuts is not restricted, as shown in following result.

Theorem 12. *For every $\epsilon > 0$ there exists a packing of the squares of sizes $1, 2, \ldots, n$ into a square of size $N + (\frac{1}{2} + \epsilon)\, n + O(\sqrt{n})$, if n is sufficiently large with respect to ϵ.*

In Section 3.2.2.1, I prove this result in the following more general form.

Theorem 13. *Let c be a fixed real number, $0 < c < 1$. For every $\epsilon > 0$ there exists a constant $C' = C'(\epsilon, c)$ such that, if $N_1 N_2 \geq A_n$ and $cN_2 \leq N_1 \leq N_2$, then the squares of sizes $1, 2, \ldots, n$ can be packed into an $(N_1 + n) \times (N_2 + \epsilon n + C'\sqrt{n})$ rectangle.*

Certainly the $O(\sqrt{n})$ term becomes superfluous for sufficiently large $n \geq n_0(\epsilon, c)$.

Theorem 11 is proved here in the following, more general form.

Theorem 14. *If N_1 and N_2 are real numbers such that $N_1 N_2 \geq A_n$ and $N_1 \leq N_2$, then an $(N_1 + n) \times (N_2 + cn)$ rectangle admits a packing of the squares $1, 2, \ldots, n$ with guillotine cutting, where c is a suitable constant smaller than 1.*

The purpose of the argument below is not to optimize the value of c; rather, it will be demonstrated that $c = 3/4$ suffices. Then, putting $N_1 = N - n/8 + r(n)$ and $N_2 = N + n/8 + r(n)$, it follows that the condition $N_1 N_2 \geq A_n = N^2$ can be satisfied with a suitably chosen function $r(n) = O(\sqrt{n})$, hence implying Theorem 11 for any $c > 7/8$ and large enough n.

The packing will be arranged in the decreasing order of the sizes of the squares, i.e. in the order n, $n-1$, $n-2, \ldots, 2, 1$. The packing is done in several phases; the rough structure of Phase 1 and Phase 2 in cases where N_1 and N_2 are not very far from each other—say, $N_1 > N_2/8$ or so—is shown in Figure 3.10.

In Phase 1, I deal with the squares of size i in the range $n/2 - x < i \leq n$, for a relatively small x, as will be explained later. I consider a rectangle whose width is $(N_1 + n)$ and height is $(N_2 + cn)$, and take horizontal edge-to-edge cuts resulting in new rectangles $R_1, R_2, \ldots, R_m$, into which squares are placed side by side.

Each region R_j is filled to a width between N_1 and $N_1 + n$. The height of R_j, denoted by w_j, is defined as the size of the largest square contained in R_j. The starting height is $w_1 = n$. I pack the squares $n, n-1, \ldots, w_2 + 1$ into R_1 to reach width N_1 or a little more; that is, $\sum_{i=w_2+2}^{n} i < N_1$ and $\sum_{i=w_2+1}^{n} i \geq N_1$.

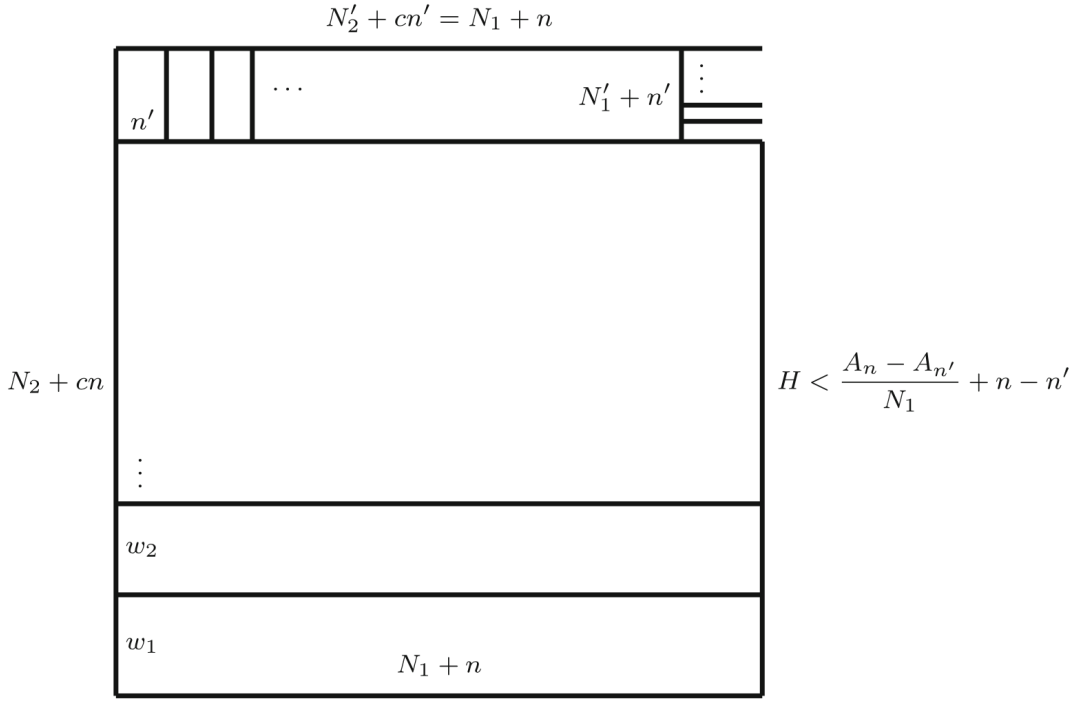


Figure 3.10: Scheme of recursion

The phase continues analogously: If R_{j-1} has been constructed and packed, let w_j be the size of the largest square which has not yet been packed, and pack the squares $w_j, w_j - 1, w_j - 2, \ldots, w_{j+1} + 1$ into R_j—which is an $(N_1 + n) \times w_j$ rectangle—to reach width just N_1 or slightly beyond. This procedure terminates when also the rectangle R_m containing the square $\lfloor n/2 \rfloor + 1$ is packed up to at least N_1. This condition fixes the value of x, defined as a non-negative integer if n is even, or $1/2$ plus a non-negative integer otherwise, such that the inequalities $\sum_{i=n/2-x+2}^{w_m} i < N_1 \leq \sum_{i=n/2-x+1}^{w_m} i$ hold. Hence, the number of squares packed in Phase 1 is $n/2 + x$, and the largest unpacked square has size $n' := n/2 - x$.

The total height of this partial packing is $H := w_1 + \ldots + w_m$. Phase 2 deals with the rectangles $n', n' - 1, \ldots, n'/2 - x'$ with some small x'. If $N_1 \leq N_2 - H$ holds, then Phase 2 continues the procedure with horizontal edge-to-edge cuts exactly in the same way as I did in Phase 1, defining e.g. $N_1' := N_1$ and $N_2' := N_2 - H + cn - cn'$. This leaves much space (namely, with width n) beyond N_1' horizontally and cn' space beyond N_2' vertically, keeping $N_2' \geq N_1'$. Otherwise, if $N_1 > N_2 - H$, the roles of the two parameters are switched, and new values are defined as follows: $N_1' := N_1$ and $N_2' := N_2 - H + cn - cn'$. In that case, Phase 2 creates vertical edge-to-edge cuts, placing the next squares on the top of each other and not leaving an empty vertical space more than n'. In either case, I provide at least a blank rectangle $(N_1' + n') \times (N_2' + cn')$ for the packing of the squares of sizes at most n'. Hence, if I prove that $N_1' N_2' \geq A_{n'} = \sum_{i=1}^{n'} i^2$ holds, it will imply that all squares are properly packed after at most $\lceil \log_2 n \rceil$ phases, because there is always at least a halving of the number of items to pack.

Let us recall that in Phase 1 every R_j $(1 \leq j \leq m)$ is filled to width N_1 or

beyond, but this property does not remain valid if I remove any one square from the contents of R_j. The latter fact will be applied to the largest square in each rectangle, in order to derive an upper bound on the total unused area within the union $R_1 \cup \cdots \cup R_m$, restricted to width N_1. Choose any one j, let $w = w_j$, and assume that exactly $k+1$ squares are packed in R_j. It is known that

$$N_1 > \sum_{i=1}^{k}(w-i) = kw - \frac{k(k+1)}{2} = k\Big(w - \frac{k+1}{2}\Big).$$

There is no blank space above the square of size w, and for $i = 1, \ldots, k$ the blank area above the square of size $w-i$ is $i \cdot (w-i)$. Hence, using that the sum of squares of the first k positive integers is A_k, the total blank area in this rectangle is

$$\begin{aligned}\sum_{i=1}^{k} i\,(w-i) &= \binom{k+1}{2} w - \frac{k(k+1)(2k+1)}{6} = \frac{k+1}{2}\,k\Big(w - \frac{2k+1}{3}\Big) \\ &= \frac{k+1}{2}\,k\Big(w - \frac{k+1}{2}\Big) - \frac{k+1}{2}\,k\,\frac{k-1}{6} < \frac{k+1}{2}\,k\Big(w - \frac{k+1}{2}\Big) \\ &< \frac{k+1}{2} N_1\,.\end{aligned}$$

Summing for $j = 1, \ldots, m$, it can be seen that the total blank area in $R_1 \cup \cdots \cup R_m$ containing the $n - n'$ largest squares is smaller than $\frac{1}{2}(n-n')N_1$. The total area of squares packed in Phase 1 is $A_n - A_{n'} = \sum_{i=n/2-x+1}^{n} i^2$. Then the height packed so far is

$$H = \sum_{i=1}^{m} w_i < \frac{A_n - A_{n'} + \frac{1}{2}(n-n')N_1}{N_1} \le N_2 - \frac{A_{n'}}{N_1} + \frac{n}{2} - \frac{n'}{2}\,. \tag{3.1}$$

Hence $N_2 - H > A_{n'}/N_1 - \frac{1}{2}n + \frac{1}{2}n'$. What remains is the following:

- squares of sizes $1, 2, \ldots, n'$, having total area $A_{n'}$, to be packed;
- a rectangle for the packing, with side lengths at least $N_1' + n'$ and $N_2' + cn'$.

Depending on whether the cutting direction has been switched from horizontal to vertical, the following needs to be verified for a suitable value of c:

(a) If $N_1 \le N_2 - H$, then $N_1'N_2' = N_1(N_2 - H + cn - cn') \ge A_{n'}$.
(b) If $N_1 > N_2 - H$, then $N_1'N_2' = (N_2 - H + cn - n')(N_1 + n - cn') \ge A_{n'}$.

Case (a) is implied by Eq. (3.1) immediately for any $c \ge 1/2$:

$$N_1'N_2' = N_1(N_2 - H + cn - cn') > N_1\Big(\frac{A_{n'}}{N_1} + \frac{(2c-1)(n-n')}{2}\Big) \ge A_{n'}.$$

Case (b) is not hard either. First apply Eq. (3.1) also here, to obtain

$$\begin{aligned}N_1'N_2' &= (N_2 - H + cn - n')(N_1 + n - cn') \\ &> \Big(\frac{A_{n'}}{N_1} + \frac{(2c-1)n}{2} - \frac{n'}{2}\Big)\big(N_1 + n - cn'\big) \\ &= A_{n'} + \frac{(n-cn')A_{n'}}{N_1} + \frac{(2c-1)n - n'}{2}\Big(N_1 + n - cn'\Big).\end{aligned}$$

The second term in the last line is positive. Taking into account that $n' \leq n/2$, it is also clear that $(2c-1)n \geq n'$ holds whenever $c \geq 3/4$. Thus, the sum is at least $A_{n'}$ for this choice of c.

It is clear from the procedure that the structure is a guillotine cutting. This completes the proof.

3.2.2.1 Unrestricted packing

Theorem 13 is established here, showing that all squares of sizes between 1 and n can be packed into a rectangle of dimensions $(N_1 + n) \times (N_2 + \epsilon n + O(\sqrt{n}))$. The proof follows a similar line of argument as in Theorem 11, but employs a denser packing within the subrectangles. Throughout the proof I assume that $\epsilon > 0$ is a parameter independent of the input size, and also without loss of generality that N_1, N_2 are shrunk to satisfy $N_1 N_2 = A_n$. Furthermore, $c_1, c_2, \ldots$ denote absolute constants which are also independent of ϵ, while ϵ_1 and ϵ_2 will be very near to ϵ. The $O(\sqrt{n})$ term, whose hidden constant depends on the parameters ϵ and c of Theorem 13 but on nothing else, will be adjusted at the end.

This procedure applies a two-phase recursive strategy. In the first phase, all squares of size at least ϵn are considered, while in the second phase, squares of size i satisfying $\epsilon^2 n \lesssim i \lesssim \epsilon n$ are handled. Intuitively, the rectangles from the previous construction are grouped into pairs (R_{2i-1}, R_{2i}) for $i = 1, 2, \ldots$. In each pair, the rectangle with the odd index is rotated by 180°, allowing the gap between them to be eliminated. This idea is illustrated in Figure 3.11.

Note that in some cases, the smallest square at the top may extend beyond the base of the largest square at the bottom.

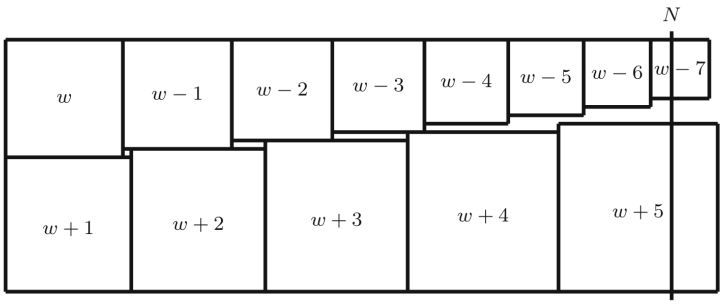


Figure 3.11: Example of a slice in which $5 + 8$ squares fit;$w = w_{2i}$ (for some value i)

Originally the height of $R_{2i-1} \cup R_{2i}$ is $w_{2i-1} + w_{2i}$, but the size of the last square in R_{2i-1} is only $w_{2i} + 1$. Hence the two rows of squares can be pushed towards each other by $w_{2i-1} - (w_{2i} + 1)$ because the decrease of square sizes in R_{2i} is faster than their increase in R_{2i-1}. This operation reduces the blank area considerably. Let S_i denote the slice of height $2w_{2i} + 1$, obtained from the union $R_{2i-1} \cup R_{2i}$ through the described construction. For complete slices an even number of rectangles R_j is needed; hence I fill horizontal rectangles $R_1, R_2, \ldots, R_m$ up to N_1 or slightly beyond (as in the case of guillotine cuts) by packing the squares of sizes $n, n-1, \ldots, n'+1$ under the following conditions: The rectangle R_m is packed to at least N_1, m is

even, $n' \le \epsilon n$, and $A_{n'} \le \epsilon^3 A_n$. I take the largest possible such n'; this determines the halting condition for Phase 1.

In the computations below, the $O(\,.\,)$ terms will always mean functions where the "hidden constants" may only be related to c, never to ϵ. First of all let us note that the assumptions $N_1 N_2 = \Theta(n^3)$ and $cN_2 \le N_1 \le N_2$ imply that both N_1 and N_2 have growth order $\Theta(n\sqrt{n})$, as $N\sqrt{c} \le N_1 \le N_2 \le N/\sqrt{c}$.

Defining $\epsilon_1 := \sqrt[3]{A_{n'}/A_n}$, it can be seen that all squares of size at least $\epsilon_1 n$ are packed in Phase 1. This follows from the fact that $xn(xn+1)(2xn+1) > x^3 n(n+1)(2n+1)$ holds for all $0 < x < 1$. Also, $\epsilon_1 \le \epsilon$, but $\epsilon - \epsilon_1$ is rather small, because $\epsilon n - n' = O(\frac{1}{\epsilon}\sqrt{n})$. Indeed, I have $N_1 = \Theta(n\sqrt{n})$, and the sizes of squares packed in $R_{m-1} \cup R_m$ are $\Theta(\epsilon n)$, therefore the number of squares in $R_{m-1} \cup R_m$ is $\Theta(\frac{1}{\epsilon}\sqrt{n})$. Moreover, $A_{\epsilon n} \le \epsilon^3 A_n + O(n^2)$ is valid, hence the requirement $A_{n'} \le \epsilon^3 A_n$ yields that the difference between ϵn and the size of the largest square in R_{m-1}, if the latter is smaller, has an upper bound proportional to $1/\epsilon^2$. Otherwise I would have stopped with R_{m-2} rather than R_m.

After the construction of horizontal slices in Phase 1, the procedure is continued in Phase 2 with vertical slices placed on the top of the horizontal slices. Before giving its numerical conditions, let us state and prove several properties. The first claim is width-independent.

1° *If the number of squares packed in R_{2i} is $k+1$, then the blank area in S_i is at most $\sum_{j=1}^{k} j^2$.*

Proof. Denote the height of slice S_i by $2w+1$; by construction, it holds that $w = w_{2i}$. As one can observe, there is a blank unit square surrounded by the four squares $w-1$, w, $w+1$, $w+2$; a blank rectangle of height 1 and width 4 surrounded by the squares $w-2$, $w-1$, $w+2$, $w+3$; and so on. Generally, for any $\ell \ge 0$, the width of the unit-high blank rectangle surrounded by the squares $w-\ell-1$, $w-\ell$, $w+\ell+1$, $w+\ell+2$ is equal to

$$\sum_{j=1}^{\ell}(w+j) - \sum_{j=0}^{\ell-1}(w-j) = \binom{\ell+1}{2} + \binom{\ell}{2} = \ell^2\,.$$

More precisely, this is the distance between the bottom-left corner of the square $w-\ell-1$ and the top-left corner of the square $w+\ell+2$, in the case that the latter also appears in the slice. If the latter is not present, I can still use this formula as an upper bound. This implies the validity of 1°. □

2° *The number of squares packed in S_i is $O(\frac{1}{\epsilon}\sqrt{n})$ for every i, and the number of slices is $O(\sqrt{n})$.*

Proof. Recall that both N_1 and N_2 have growth order $\Theta(n\sqrt{n})$. Moreover, since every packed square has size $\gtrsim \epsilon n$, the number of squares in any S_i is between $c_1\sqrt{n}$ and $\frac{c_2}{\epsilon}\sqrt{n}$. Since the number of squares is less than n, the number of slices is at most $c_3\sqrt{n}$. □

3° *The largest vertical gap between squares of R_{2i-1} and R_{2i} in S_i is $O(\frac{1}{\epsilon^3})$.*

Proof. Let a_i and b_i be the size of the smallest square in R_{2i} and the largest square in R_{2i-1}, respectively. Recall that $a_i \gtrsim \epsilon n$ and $b_i = a_i + O(\frac{1}{\epsilon}\sqrt{n})$. There are at most $N_1/a_i + 1$ squares in R_{2i} and at least N_1/b_i squares in R_{2i-1}. Hence $w_{2i} - a_i \le N_1/a_i$ and $b_i - (w_{2i}+1) \ge N_1/b_i - 1$. Since $N_1 = \Theta(n\sqrt{n})$, the height of the largest gap can be estimated as

$$2w_{2i} + 1 - (a_i + b_i) \le \frac{N_1}{a_i} - \frac{N_1}{b_i} + 1 = \frac{N_1(b_i - a_i)}{a_i b_i} + 1 = \frac{\Theta(n\sqrt{n})\, O(\frac{1}{\epsilon}\sqrt{n})}{\epsilon^2 n^2} = O\Big(\frac{1}{\epsilon^3}\Big).$$

The blank area in S_i is $O(\frac{1}{\epsilon^3} n\sqrt{n})$, in agreement with 1° as $k = O(\frac{1}{\epsilon}\sqrt{n})$. □

4° *The total height of the slices is at most* $(1-\epsilon_1^3)N_2 + O(\frac{1}{\epsilon^3}\sqrt{n})$.

Proof. The total area of packed squares is $A_n - A_{n'} = (1-\epsilon_1^3)N_1 N_2$, moreover by 3° the blank area is $O(\frac{1}{\epsilon^3} N_1 \sqrt{n})$. Since the totally packed width is at least N_1, the assertion follows. □

The term $O\big(\frac{1}{\epsilon^3}\sqrt{n}\big)$ is interpreted as an additional slice of that height, packed on top of an $(N_1 + n) \times (N_2 + \epsilon n)$ rectangle. Then in Phase 2 the slices of vertical position have a blank rectangle $(\epsilon_1^3 N_2 + \epsilon n) \times (N_1 + n)$, into which only the squares $n', n'-1, \ldots, n''+1$ have to be packed, for some n'' (and a blank rectangle has to be left empty for $n'', n''-1, \ldots, 1$). Hence the slices can be completely packed up to at least $\epsilon_1^3 N_2$, based again on the principle shown in Figure 3.11. Here I require that $n'' \le \epsilon n'$ and $A_{n''} \le \epsilon^3 A_{n'}$. I take the largest possible such n''; this determines the halting condition for Phase 2. Define $\epsilon_2 := \sqrt[3]{A_{n''}/A_{n'}}$, i.e. $A_{n''} = \epsilon_2^3 A_{n'} = (\epsilon_1\epsilon_2)^3 A_n$. Note that $n'' \le \epsilon_2 n'$, hence $n'' \le \epsilon_1\epsilon_2 n$, but still $\epsilon^2 - \epsilon_1\epsilon_2$ is rather small, and all squares larger than $\epsilon_1\epsilon_2 n$ are already packed.

In this phase 1° remains valid, and the following analogues of 2°–3°–4° can be observed.

5° *The number of squares packed in any vertical slice is* $O(\epsilon\sqrt{n})$*, and the number of vertical slices is* $O(\frac{1}{\epsilon}\sqrt{n})$.

Proof. A slice is packed to $\epsilon_1^3 N_2$ or slightly beyond, and all squares are larger than $n'' \approx \epsilon_1\epsilon_2 n$, while the largest square is $n' \le \epsilon_1 n$. Therefore the number of squares in a slice is least $\approx 2\epsilon^2 N_2/n$ and at most $\approx 2\epsilon N_2/n$. Fewer than ϵn squares are packed in this phase, hence the number of slices does not exceed $\approx \frac{1}{2\epsilon}\frac{n^2}{N_2} = O(\frac{1}{\epsilon}\sqrt{n})$. □

6° *The largest horizontal gap inside a vertical slice is* $O(1)$.

Proof. Let again a_i and b_i denote the size of the smallest and the largest square in an arbitrarily chosen vertical slice. Now, I have $a_i \gtrsim \epsilon^2 n$ and $b_i = a_i + O(\epsilon\sqrt{n})$. Since the slice is packed to $\epsilon_1^3 N_2$, the gap can be estimated from above with

$$\frac{\epsilon_1^3 N_2}{a_i} - \frac{\epsilon_1^3 N_2}{b_i} = \epsilon_1^3 N_2 \frac{b_i - a_i}{a_i b_i} \approx \epsilon_1^3 \frac{\Theta(n\sqrt{n})\, O(\epsilon\sqrt{n})}{\epsilon^4 n^2} = O(1).$$

This upper bound does not depend on ϵ. Consequently the blank area inside each vertical slice is $O(\epsilon^3 N_2)$. □

7° *The total width of the vertical slices is at most* $(1-\epsilon_2^3)N_1 + O(\frac{1}{\epsilon}\sqrt{n})$.

Proof. The area of squares packed during Phase 2 is $A_{n'} - A_{n''} = \epsilon_1^3(1-\epsilon_2^3)N_1N_2$, moreover by 5° I have $O(\frac{1}{\epsilon}\sqrt{n})$ slices with $O(\epsilon^3 N_2)$ blank area in each, hence the total blank area is $O(\epsilon^2 N_2\sqrt{n})$. Since the totally packed height is at least $\epsilon_1^3 N_2$, the assertion follows. □

If ϵ is very small and n is not too large, it may be technically complicated to separate the $O(\frac{1}{\epsilon}\sqrt{n})$ term from n. This situation can be handled by packing no vertical slices beyond $O(\frac{1}{\epsilon}\sqrt{n})$, turning the rest to horizontal position and placing them on the top of the additional slice of Phase 1. The term $O(\frac{1}{\epsilon^3}\sqrt{n})$ then swallows the current $O(\frac{1}{\epsilon}\sqrt{n})$.

After these two phases an $(\epsilon_2^3 N_1 + n) \times (\epsilon_1^3 N_2 + \epsilon n)$ rectangle remains blank. Apart from an additional lower-order height, it is more than enough for the packing of the squares not larger than $\epsilon_1\epsilon_2 n$, by a repeated application of such alternating horizontal/vertical phases whose number is at most $\lceil \log_{1/\epsilon} n \rceil$. For the necessary extra height, say $h(n)$, I obtain the following recursion:

$$h(n) \le h(\epsilon^2 n) + O\Big(\frac{1}{\epsilon^3}\sqrt{n}\Big).$$

This yields $h(n) \le C'(\epsilon, c)\sqrt{n}$ for a suitable constant $C'(\epsilon, c)$.

3.2.2.2 Concluding remarks

It has been proved that the squares of sizes $1, 2, \ldots, n$ can be packed using guillotine-type cuts into a square of size $\sqrt{1^2 + 2^2 + \ldots + n^2} + cn$, where $c < 1$ is a constant. The calculation yielding $c \le 7/8$ offers the possibility of obtaining slightly better bounds, for example by applying the following ideas:

- In Case (b), i.e. $N_1 > N_2 - H$ in the proof of Theorem 14, the coefficient of the major term nN_1 is positive not only for $c \ge 3/4$ but also for all $c > 21/32$, and for large n it dominates the lower-order negative term of $\Theta(n^2)$. This improves the upper bound for guillotine-cuts.
- Instead of $N_1 + n$ on the horizontal side one may take $N_1 + cn$ where c is not very small but smaller than 1. Although this does not guarantee complete packing to N_1 or beyond in the first few rectangles, the rectangles containing squares not larger than $\lfloor cn \rfloor$ can be overpacked and so compensate for the loss in rectangles of small indices. This improves the bounds in both types of packings.

However, to keep this note concise and not overly technical, the upper bounds given in the theorems are not reduced to the limits of these methods.

3.3 The covering problem

Let us recall that, given n squares—one of size 1, one of size 2, and so on, up to one of size n—the problem is to determine the largest square, of size $K \times K$, that can be completely covered by this set of smaller squares.

3.3.1 Small cases for covering

In this section, small cases up to $n \leq 7$ are discussed. Let $\mathcal{Q}[n]$ denote the squares with sizes $1, 2, \ldots, n$, exactly one copy from each. Note that

$$UB_n := \left\lfloor \sqrt{\sum_{k=1}^{n} k^2} \right\rfloor = \left\lfloor \sqrt{n(n+1)(2n+1)/6} \right\rfloor = \left\lfloor \sqrt{A_n} \right\rfloor \tag{3.2}$$

is a trivial upper bound for the largest size K_n of a square that can be completely covered by $\mathcal{Q}[n]$. The increment of K from $n-1$ to n is denoted by Δ_K, that is, $\Delta_K = K_n - K_{n-1}$. If no confusion arises, the index n will be omitted, and K will be used to denote K_n. Moreover, let L denote a lower bound for K for any fixed value of n. The values for the cases $n \leq 7$ are shown in Table 3.7. Their validity can be checked by hand.

n	1	2	3	4	5	6	7
UB_n	1	2	3	5	7	9	11
K	1	2	3	4	6	9	11
Δ_K	1	1	1	1	2	3	2

Table 3.7: The cases $n \leq 7$

The cases $n \leq 3$ are trivial. For $n = 4$ the square 5×5 cannot be covered by $1 \times 1, \ldots, 4 \times 4$, since 4×4 leaves in 5×5 an empty L-shape of $5+4 = 9$ unit squares (or cells), but 3×3, 2×2, 1×1 can cover at most $5 + 2 + 1 = 8$ of those empty cells. The bigger cases are less immediate; they are discussed in the following two subsections.

3.3.1.1 The special case of $n = 5$, excluding $K = 7$

For $n = 5$, it is easy to construct a cover for $K = 6$ (see Figure 3.12). To show that $K = 6$ is indeed the optimal value, it remains to exclude $K = 7$. Suppose, for contradiction, that a complete cover exists for $K = 7$. Since it is impossible to cover a 7-long stripe using only the three smallest squares 1×1, 2×2, and 3×3, the union of the 5×5 and 4×4 squares necessarily forms a polygon of width 7 and height 7.

At this point two rectangles R' and R'' remain uncovered, of sizes 2×3 and 3×2, in a position where the 3×3 square can cover at most 6 units of area from $R' \cup R''$, while the two smallest squares together have a total area of only 5. Hence, the cover cannot be completed.

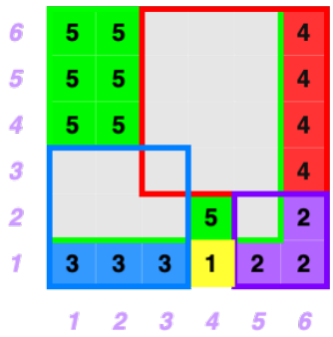


Figure 3.12: A value of $K = 7$ can be excluded, and it is known that a cover for $K = 6$ exists, as shown on the left. However, this solution is not trivial; it requires some manual construction to arrange the squares correctly. This case illustrates how overlapping allows for slightly larger values of K, but careful layout is still needed to reach the lower bound.

Another way to argue is that at least one horizontal side H and one vertical side V of the $K \times K$ square are disjoint from the largest square 5×5. The corner $C = V \cap H$ is contained in just one of the other four squares, because a second square at C would be superfluous (namely the smaller one). If C is covered with the square of size $x \leq 4$, then the side lengths must satisfy $1 + 2 + 3 + 4 + x \geq |V| + |H| = 14$, i.e. $x = 4$ holds and the remaining 3-length parts of V and H should be covered by the 3-square and by the union of the 1-square and 2-square, respectively. But then there is a 1×6 stripe inside $K \times K$ that touches the 1-square and is internally disjoint from all the four. It is impossible to cover this stripe with the 5-square.

3.3.1.2 The small cases $n = 6$ and $n = 7$

The largest possible covers for $n = 6$ and $n = 7$ are shown in Figure 3.13. Consider e.g. $n = 6$. Since a valid cover exists for $n = 6$ and $L = 9$, and the upper bound for $n = 6$ satisfies $\mathrm{UB}_6 = L$, it follows that $K_6 = 9$. Therefore, no further analysis is required for this value of n. This specific cover is constructed so that the two biggest squares are allocated in two opposite corners (red and green squares of sizes 6 and 5, Fig. 3.13, left), the next two biggest squares are placed in the other two corners, and the remained few cells are covered by the two smallest squares. The same technique works also for $n = 7$ since $UB_7 = 11$ (cf. Fig. 3.13, right); but let us note that this simple way of covering will not work after some value of n.

3.3.2 Medium cases for covering

For medium values of n, the problem cannot be solved easily by hand. The problem is considered straightforward when the trivial upper bound UB_n can be computed for a given n, and a complete cover can be constructed for $L = UB_n$; in this case it follows that the optimal solution has been obtained, i.e. $K = L = UB_n$. In other cases, however, K may be smaller than UB_n, or a covering of size UB_n cannot be

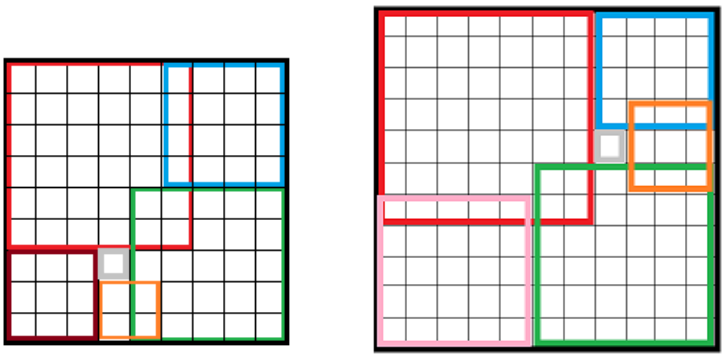


Figure 3.13: $n = 6$ (left) and $n = 7$ (right)

identified easily. For such instances, a different approach is adopted: a mathematical programming model is defined that takes the sizes of the squares as input and is solved with a commercial solver. If the solver confirms that no solution exists for a given L value, while one exists for $L - 1$, then the optimal solution is $K = L - 1$.

3.3.2.1 Mathematical programming formulation

The problem is related to the Board Packing Problem (BoPP) [2, 47].

Definition 15. *Consider a rectangular board with rows $M = \{1, \ldots, |M|\}$ and columns $N = \{1, \ldots, |N|\}$. Each position $(i, j) \in M \times N$ has an associated integer value $g_{i,j}$ representing the revenue obtained if the position is covered. A set R of rectangles is given, where each rectangle $r \in R$ has a specified height h_r, width w_r, and cost c_r.*

The objective is to select and place rectangles on the board so as to maximize the profit, defined as the total revenue of the covered positions minus the cost of the purchased rectangles. All revenues, heights, widths, and costs are integers. Rectangles must be placed with sides parallel to the sides of the board. Overlaps are permitted, but the revenue of a board position can be collected at most once.

By setting $|M| = |N| = K$, the revenue equal to 2 for every cell, and defining the set of rectangles R as the n squares of consecutive sizes, each with a cost of 1, then if the whole board can be covered, the optimal value is at least $2K^2 - n$. On the other hand, if the optimal value is less than $2K^2 - n$, I can conclude that no cover of $K \times K$ exists using the squares up to size n. In Section 3.3.3 I will adapt a heuristic developed for the board packing problem to handle big cases. However, for medium cases, I will apply a specialized mathematical programming formulation.

In the specialized model, I place all the available objects, and I need to cover all cells. I can utilize some symmetry, or even try to solve the model with certain squares in fixed positions. Let K be the size of the square that needs to be covered, and let $n \times n$ be the size of the largest consecutive square. Let T_r be the set of coordinates such that square $r \times r$ can be placed with its upper left corner at such coordinates. E.g., $T_r = \{(i, j) : i = 1, \ldots, K - r + 1, j = 1, \ldots, K - r + 1\}$. Let B_{ijr} be defined as the set of positions (u, v) for the top left corner of the square of size $r \times r$ that will result in position (i, j) being covered by that square. Define the

binary variable x_{ijr} to be 1 if and only if the square of size $r \times r$ is located with the upper left coordinate at (i, j). Let continuous variable y be 0 if all cells are covered, and at least 1 otherwise. The model can be written as:

$$\min \ y \tag{3.3}$$

$$\sum_{(i,j)\in T_r} x_{ijr} = 1, \qquad r \in \{1, \ldots, n\} \tag{3.4}$$

$$y + \sum_{r\in\{1,\ldots,n\}} \sum_{(u,v)\in B_{ijr}} x_{ijr} \geq 1, \qquad u = 1, \ldots, K, \ v = 1, \ldots, K \tag{3.5}$$

$$x_{ijr} \in \{0, 1\}, \qquad r \in \{1, \ldots, n\}, \ (i, j) \in T_r \tag{3.6}$$

$$y \geq 0. \tag{3.7}$$

Equation (3.3) defines the objective of minimizing y. Here, $y = 0$ indicates that all cells are covered, while $y = 1$ indicates that at least one cell is not covered (note that y cannot take values larger than 1).

Constraint (3.4) ensures that each square r is placed exactly once in one of its feasible top-left positions $(i, j) \in T_r$. Constraint (3.5) considers each cell (u, v) of the board. If the cell is covered, the value of the double sum is at least 1, since the sum counts the number of rectangles covering the cell. If the sum equals 0 (i.e., the cell is not covered by any rectangle), then the inequality requires that y takes the value 1. Consequently, in equation (3.3), if the minimum is 0, all cells are covered.

The binary domain constraint (3.6) specifies that x_{ijr} is equal to 1 if square r is placed at position (i, j), and 0 otherwise. Finally, constraint (3.7) ensures that the board size variable y is nonnegative.

The problem has some symmetry that can be exploited. For example, one may require that the largest square should be placed to the left and above the middle of the covered square. For example, the instance T_n can, without loss of generality, be reduced as follows. Let $D_n = \lceil (K - n + 1)/2 \rceil$. The set T_n is then defined as $T_n = \{(i, j) : i = 1, \ldots, D_n; \ j = 1, \ldots, D_n\}$.

3.3.2.2 Results using the mathematical models

The models are solved using the commercial mixed-integer programming solver CPLEX. When using the models (i.e., the general model of BoPP and the specialized model) for values in the range $n = 8, \ldots, 21$, their performances do not differ significantly. However, the second, specialized model, typically has slightly lower running times when attempting to prove optimality. In Table 3.8 I provide the optimal results for $8 \leq n \leq 21$. Any case with $UB_n = K$ is simple, or at least seems simple. It is enough to find a good cover for the value of UB_n. But, note that this good cover cannot be easily found for a relatively big n, like $n = 21$. These are the cases $n \in \{9, 13, 14, 16, 18, 19, 20, 21\}$.

For the other cases, i.e. for $n \in \{8, 10, 11, 12, 15, 17\}$, the optimal value of L can be obtained so that CPLEX returns the non-existence of a complete cover for $L+1$.

The running time of CPLEX was about 7 seconds for $n = 11$ (to conclude that $K = 22$ is not possible), but for $K = 21$ CPLEX found an optimal solution soon. For $n = 12$, excluding $K = 25$ needed about 12 seconds. For $n = 15$, excluding $K = 35$ needed about 214 seconds. For $n = 17$, CPLEX needed *more than 10 hours* of running time to conclude that $K = 42$ cannot be covered, thus proving that $K = 41$ is optimal. Thus, for larger values of n, solving models using CPLEX becomes too hard.

n	8	**9**	10	11	12	**13**	**14**	15	**16**	17	**18**	**19**	**20**	**21**
UB_n	14	**16**	19	22	25	**28**	**31**	35	**38**	42	**45**	**49**	**53**	**57**
K	13	**16**	18	21	24	**28**	**31**	34	**38**	41	**45**	**49**	**53**	**57**
Δ_K	2	**3**	2	3	3	**4**	**3**	3	**4**	3	**4**	**4**	**4**	**4**

Table 3.8: The cases $8 \leq n \leq 21$

The experience is that finding a good cover is easy if it exists, but excluding some K value is harder and harder as n grows. The case of $n = 21$ with a cover for $K = 57$ is detailed in Table 3.9, and is illustrated in Figure 3.14. In the table I give the top-left coordinates of each small square $r \times r$ in the covering, $r \leq n$, where "-" means that the square in question is not needed for the covering.

r	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Top	-	-	43	42	41	30	35	35	34	21	21	46	45	31	43	1	41	17	22	1	1
Left	-	-	27	23	18	38	20	27	35	48	37	18	45	44	31	22	1	20	1	38	1

Table 3.9: Coordinates for $n = 21$, $K = 57$, a complete covering

n	22	23	24	25
UB_n	61	65	70	74
L	60	64	68	72

Table 3.10: The cases $22 \leq n \leq 25$

Some further results can be found in Table 3.10 for $22 \leq n \leq 25$. Here CPLEX cannot provide an optimal result in reasonable time. For all cases covered in Table 3.10, the exact value of K is between L and UB_n, and has yet to be determined. For example, in case of $n = 22$, I have $L = 60 \leq K \leq UB_{22} = 61$.

3.3.3 Big cases for covering

In this context, *big cases* are defined as instances for which CPLEX is unable to prove optimality within reasonable time. While CPLEX may still compute lower bounds for such instances, this process becomes increasingly time-consuming, particularly when the square to be covered exceeds dimensions of 70×70. As a result, alternative methods are explored to generate lower bounds, including a heuristic algorithm and an expansion-based approach.

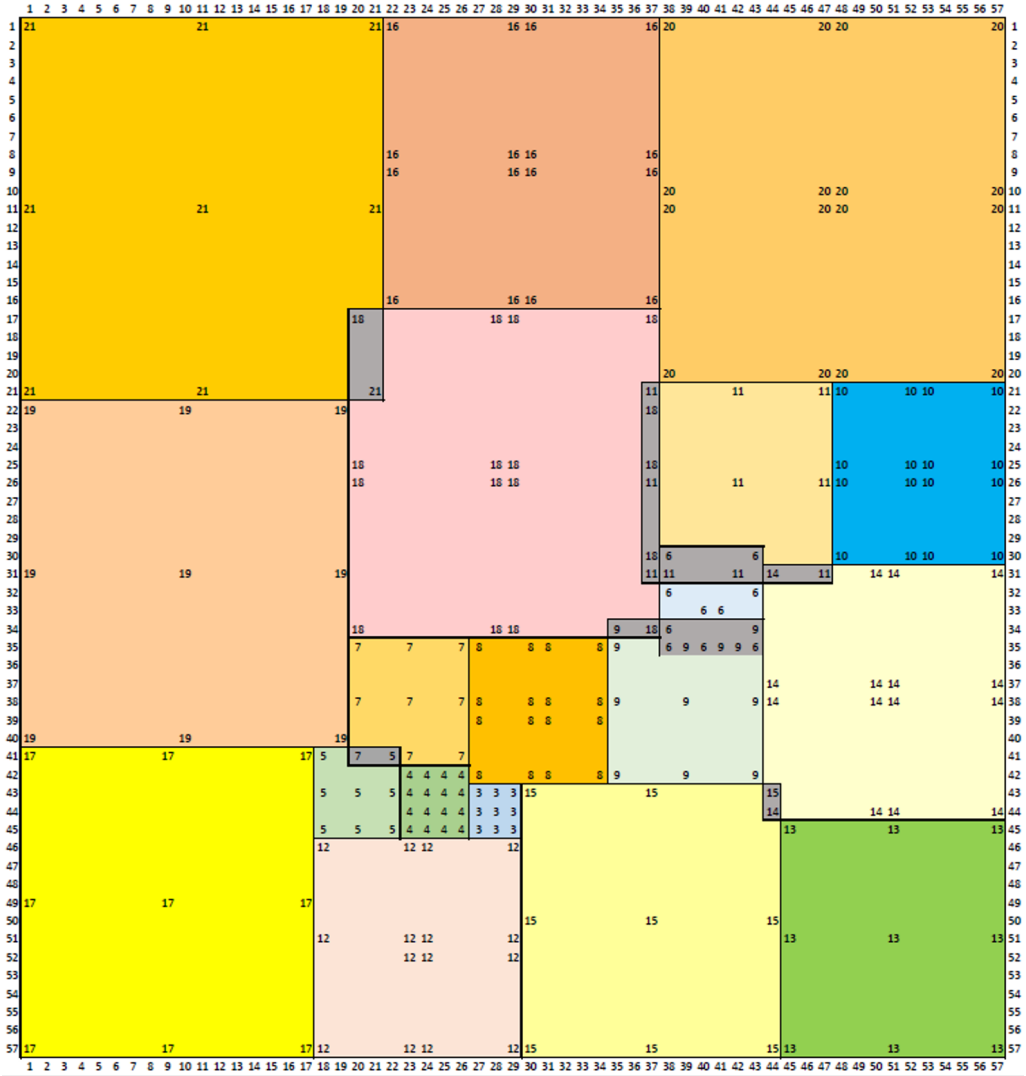


Figure 3.14: Cover of $K = 57$ using squares up to size $n = 21$.

3.3.3.1 Heuristic search

After a certain point, the solver alone is not sufficient. In such cases, a heuristic can be applied to obtain valid lower bounds on K for a given value of n. That is, the board packing problem [2] presented in Section 3.3.2.1 can be generated for given values of K and n, and a computer search can attempt to find a feasible solution with an objective function value that implies that a cover has been found.

The heuristic that I use is an evolutionary algorithm, modified from the algorithm proposed in [2] to solve general instances of the board packing problem. Figure 3.15 shows the outline of the method, which is based on the leading metaheuristic for the highly competitive combinatorial task of solving vehicle routing problems [116], the hybrid genetic search.

First, a set of initial solutions is generated. Some of these solutions are generated by a construction heuristic, where the squares are sorted in decreasing size, and then inserted in a location that maximizes the number of covered cells. The

construction heuristic is deterministic, but can be repeated several times by moving the first square to the last position to modify the insertion sequence. Other initial solutions are simply generated by randomly placing all the squares onto the grid. As a refinement of this, I force the four largest squares to be initially placed in a corner. The method is allowed to restart. In that case, the single best solution found is kept, and the other initial solutions are generated at random, without forcing the largest squares to be placed in the corners.

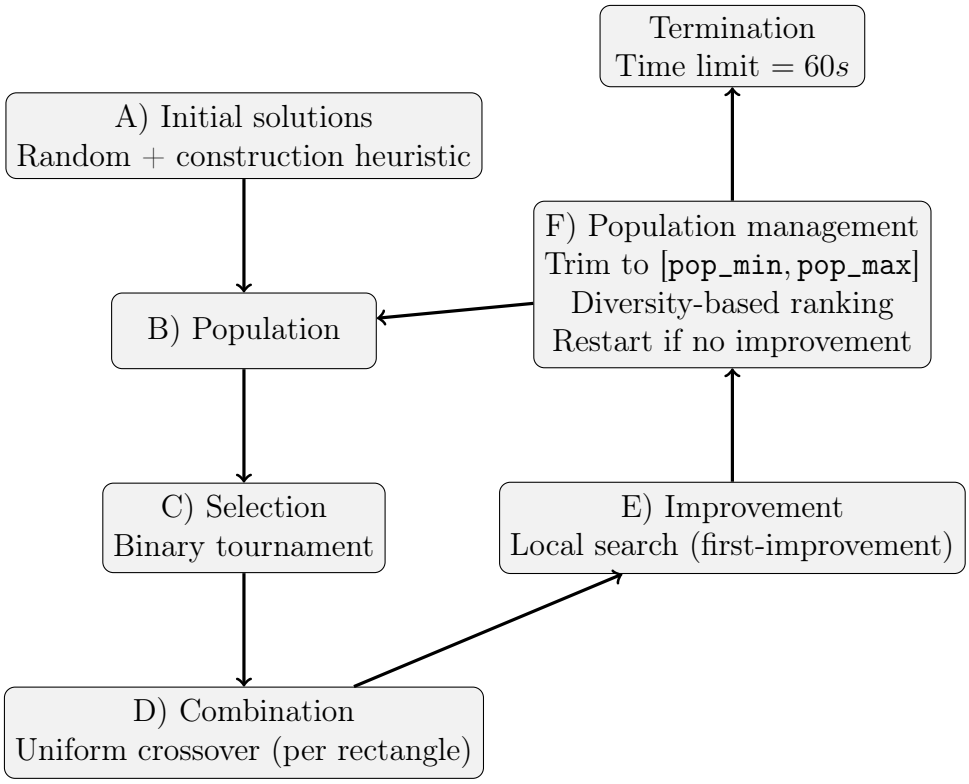


Figure 3.15: Detailed schematic of the implemented evolutionary algorithm for the BoPP, showing the initialization strategy, selection operator, crossover type, local improvement, population management, and termination criterion.

Figure 3.15 provides a detailed schematic of the evolutionary algorithm developed for the BoPP. The process starts with the generation of initial solutions, obtained through a combination of random initialization and a construction heuristic. The population is then evolved iteratively. In each iteration, two solutions are selected using a binary tournament, and combined through a uniform crossover operator applied independently to each rectangle. The resulting offspring are subsequently improved using a first-improvement local search procedure, which acts as a replacement for traditional mutation operators.

After improvement, population management is performed: the population is trimmed to remain within the interval [`pop_min`, `pop_max`], and a diversity-based ranking is applied to prevent premature convergence. If no improvement has been observed after a specified number of reductions, the algorithm is restarted. The search terminates when the maximum runtime limit of 60 seconds is reached. This design ensures a balance between exploration and exploitation, while maintaining

both quality and diversity in the population throughout the search process.
In a regular iteration of the heuristic, two solutions are selected from the current population using two binary tournaments. These are then combined into two new solutions using a type of uniform crossover, where each square and its placement is assigned at random from one parent to one child. The new solutions are then subjected to a local search improvement. Here, each square can be moved one step in either Cartesian direction, or if this does not lead to an improvement after considering all the squares, one square is selected to be temporarily removed and then inserted into the best possible position (covering as many cells as possible).

The initial population is created with a fixed number of solutions. New solutions are admitted into the population only if their objective function values differ from those already present, in order to promote diversity. Although structurally different layouts may yield identical objective values, such solutions are filtered out under this rule. This approach reduces the risk of filling the population with near-duplicates that provide little additional information while still triggering costly local search procedures. The trade-off is that some structural diversity may be lost. A more refined mechanism could instead eliminate only exact duplicates, or rely on geometric similarity measures, thereby allowing multiple solutions with the same profit to coexist if their layouts are sufficiently different. Such extensions are left as potential directions for future work, aiming to balance efficiency and diversity more effectively.

When the population exceeds its maximum size, it is trimmed back to the size of the initial population. The reduction procedure follows [116], combining measures of solution quality and solution diversity. If this reduction is repeated several times without improving the best-found solution, the search is restarted.

New offspring solutions are inserted into the current population if they are admissible, so the population size gradually increases. When the number of solutions exceeds the maximum threshold, the population is reduced to the target size by applying the reduction procedure of [116], which combines measures of solution quality and solution diversity. This trimming ensures that the population remains within a bounded range, avoiding uncontrolled growth. If the population has been reduced several times without improving the best-found solution, the entire search is restarted.

3.3.3.2 Expansion algorithm

My next idea to obtain lower bounds, is an *expansion algorithm*. Given an original cover of a square of dimensions $K \times K$, using consecutive squares up to size n, I can apply an expansion algorithm to find a covering of a larger square $K' = K + s$ using consecutive squares of sizes from 2 to $n + 1$. The 1×1 square can then be added arbitrarily to this cover. The size of s and thus K' depends on the minimum number of squares to appear in any given row or column in the original cover of the $K \times K$ square.

The idea of the algorithm is easier to express using rectangles. Let us assume that I have a rectangle of size $K^I \times K^J$ that is covered by a set of n rectangles where rectangle r has the size $M_r \times N_r$. I will then compute the number of rectangles in each row that are actively used to cover the bigger rectangle. Let s be the minimum

number of such rectangles across all rows. Then, I will find a cover of a rectangle of size $K^I \times (K^J + s)$ by using n rectangles of sizes $M_r \times (N_r + 1)$. This will be achieved by increasing the size of each individual rectangle by one column and shifting them horizontally to cover the enlarged area.

By applying this procedure to a $K \times K$ square covered with n consecutive squares, the squares can first be expanded horizontally. Then, by transposing the result and applying the procedure again, I can obtain a figure of size $(K + s) \times (K + s')$ covered by consecutive squares of sizes from 2 to $n + 1$. In the case that $s \neq s'$, I may then have to redo the process while ensuring that the square is extended in equal length $\min\{s, s'\}$ in both directions.

Algorithm 1 Pseudo-code for expansion algorithm

Input: Original cover of a $K^M \times K^N$ rectangle using n rectangles of sizes $M_r \times N_r$
Output: Updated positions and dimensions of the rectangles
for each rectangle r appearing in the first column **do**
    $h(r) \leftarrow 0$ ▷ Cannot move these rectangles horizontally
for $j = 2, 3, \ldots, K$ **do** ▷ Iterate over columns
    for each rectangle r in column j but not in column $j - 1$ **do**
        Let Q be the set of rectangles in column $j - 1$ overlapping rows of r
        $h(r) \leftarrow \min_{q \in Q} h(q) + 1$ ▷ Move r one step more than neighbors
Let H be the set of rectangles in column K
$s \leftarrow \min_{r \in H} h(r) + 1$ ▷ New size is $K^M \times (K^N + s)$
for $r = 1, 2, \ldots, n$ **do** ▷ Adjust each rectangle
    Move rectangle r to the right by $\min\{h(r), s - 1\}$ units
    Resize rectangle r by increasing its width by 1 unit

Proposition 16. *The expansion algorithm works, i.e. provides a covering for each n and s.*

However, this provides only a cover, and it is not known whether the solution is optimal.

Remark 17. *It is essential in executing Algorithm 1 horizontally for s and vertically for s' that s' be computed only after the determination of s and construction of the $K^M \times (K^N + s)$ cover. Computing both s and s' from the initial configuration and executing expansion in both directions simultaneously may or may not work. Among the covers constructed for $\mathcal{Q}[n]$ I have not found a counterexample so far. However in Fig. 3.16 I show an arrangement of 20 squares that cover an area of 4 rows and 8 columns, but its expansion leaves an uncovered cell (crossed in Fig. 3.17) if s' is calculated from the original cover rather than from the s-expanded one.*

3.3.3.3 Results using the heuristic and the expansion-algorithm

For small values of K, the heuristic finds lower bounds as good as those determined by using the commercial MIP solver. However, since the heuristic does not provide

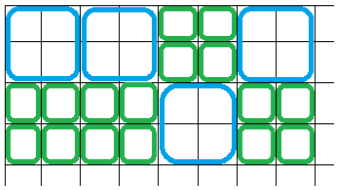


Figure 3.16: Initial 4×8 cover

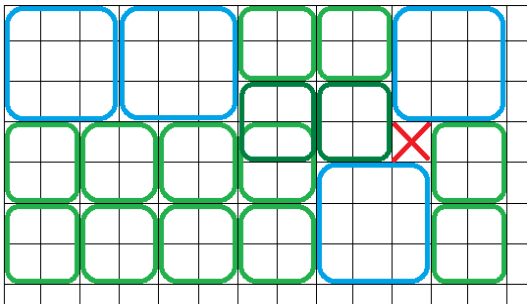


Figure 3.17: Simultaneous computation of s and s' leaves a cell uncovered

any upper bound, the commercial MIP solver is preferred up to values of K around 65. For larger K, the MIP solver is unable to provide conclusive results, and the heuristic is faster at finding lower bounds.

Table 3.11 shows results for $25 \leq n \leq 30$. The time taken to find a cover for a given combination of L and n by the heuristic for this range is typically less than one hour. However, when the heuristic fails to find a cover, I simply do not know whether the method just failed to find the cover or whether a cover did not exist. For $n = 25$, which was the largest case considered by CPLEX, the heuristic is able to find the same lower bound. The table also shows lower bounds produced by applying the expansion algorithm. To generate these, I consider the cover found by the heuristic for n and create a new cover using $n + 1$ small squares. This is able to find the same lower bounds as the heuristic for $n = 26$ and $n = 28$ by using the output from the heuristic for $n = 25$ and $n = 27$, respectively.

n	25	26	27	28	29	30
UB_n	74	78	83	87	92	97
L	72	76	81	85	90	95
L'	NA	76	80	85	89	94

Table 3.11: The cases $25 \leq n \leq 30$ and results using the heuristic (L) and the expansion algorithm (L').

To stress test the heuristic, I chose $n = 100$, and attempted to find the largest value of L such that there can be proven a feasible cover using the consecutive squares from 1 to $n = 100$. The heuristic succeeded at finding a solution for $L = 559 \leq UB_{100} = 581$. The run that succeeded terminated after $3,166$ seconds, but before

this, two other runs with different random seeds had failed after running for four days each. Thus, even finding lower bounds is challenging for such large values of n.

In the cover for $n = 100$ and $L = 559$ that was found by the heuristic, there are at least $s = 6$ squares in each row and each column (after performing the extension in one direction). Thus, by the expansion algorithm I also get a cover for $L' = 565$ using $n = 101$; for $L'' = 571$ using $n = 102$, and so on.

3.3.4 The asymptotic case

Recall that $N = \sqrt{A_n}$ with $A_n = \frac{n(n+1)(2n+1)}{6}$ is an upper bound on the size of a square coverable by the squares 1×1, 2×2, $\ldots$, $n \times n$. In this section I prove that this bound is asymptotically tight, i.e., a cover can be constructed for a square of area at least $(1 - o(1))A_n$ as $n \to \infty$. More explicitly, the square of side length $N - 2n$ admits a cover with those smaller squares. I prove this fact in the following more general form.

Theorem 18. *If $p \cdot q \le A_n$ and $|p - q| < 3n$, then a $(p - 2n) \times (q - 2n)$ rectangle can be covered with the squares of side lengths $1, 2, \ldots, n$.*

Proof. I apply induction on n, assuming that for all $n' < n$, every $(p'-2n') \times (q'-2n')$ rectangle can be covered with the squares of side lengths $1, 2, \ldots, n'$ provided that $p' \cdot q' \le A_{n'}$ and $|p' - q'| < 3n'$.

Let R denote a $p \times q$ rectangle (p wide and q high), and assume $p \le q$. The task is to cover the $(p - 2n) \times (q - 2n)$ rectangle obtained from R by trimming a $2n$-wide rectangle from its right end and a $2n$-high rectangle from the top. If $p \le 2n$, then the task is void, which is necessarily the case when $2n(2n + 1) > A_n$ (that means $n < 11$). Beyond that, the validity of the theorem can very easily be checked for a wider range of not too large n; I demonstrate this for $n = 24$. I then have $A_{24} = 4900$, $p \le \sqrt{A_{24}} = 70$, and assuming $p \ge 2n + 1 = 49$, $q \le \sqrt{A_{24}}/49 = 100$ holds. Hence $p - 2n \le 22$ and $q - 2n \le 52$. On the other hand, even the rectangle 22×69 is obviously coverable with just the three largest squares $24, 23, 22$.

For general n I take the $s+t$ largest squares, i.e. of sizes $n, n-1, \ldots, n-s-t+1$, determined as the smallest natural numbers s and t satisfying the inequalities

$$p - 2n \;\le\; \sum_{i=n-s+1}^{n} i \quad \text{and} \quad p - 2n \;\le\; \sum_{i=n-s-t+1}^{n-s} i\,.$$

From these squares I create a "slice" that completely covers a rectangle of width $p - 2n$ and height $2n - s - t + 1$, as sketched in Figure 3.18; or height $2n - s - t$ if the case explained in the next paragraph occurs. The unified notation $2n - s - t + \varepsilon$ will be used, with the understanding that ε can mean either of 0 and 1. I arrange the s largest squares in increasing order, with their bottom sides aligned, and the next t largest squares in decreasing order, with their top sides aligned, both from left to right, in the way that square $n - s - t + 1$ touches square n from above, and the vertical left sides of the squares $n - s + 1$ and $n - s$ touch the left side of R, the lower left corner of square $n - s + 1$ being the same as that of R.

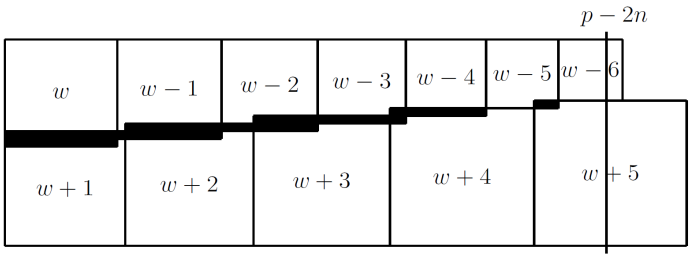


Figure 3.18: Example of a slice in which $5+7$ squares cover total width at least $p-2n$; i.e., $s=5$, $t=7$ (cf. text) and w denotes $n-5$. Dark areas are covered twice.

It may occur that the bottom-left corner of square $n-s-t+1$ is located to the left of the top-left corner of square n. In that case an uncovered area would be surrounded by the four squares n, $n-1$, $n-s-t+2$, $n-s-t+1$. This situation can be handled by aligning the top row lower by 1; that is, square $n-s-t+1$ should then touch square $n-1$, rather than n. This yields a slice of height $2n-s-t$, as one of the options in the notation $2n-s-t+\varepsilon$. For instance, concerning the example of Figure 3.18, if $\sum_{i=1}^{4}(w+i) < p-2n \leq \sum_{j=0}^{5}(w-j)$ holds instead of $p-2n > \sum_{j=0}^{5}(w-j)$ then $w-6$ does not belong to the slice.

As the bottom squares are larger than the top ones, it follows that $t \geq s$. Moreover, the height of the slice is $2n-s-t+\varepsilon$, hence the largest vertical overlap between top and bottom squares is $t-s+(1-\varepsilon) \leq t-s+1$. A key point in the computation will be that I have

$$t-s \leq 2.$$

To prove this, I first observe that for every n the following chain of inequalities is valid:

$$\begin{aligned} p-2n \;\leq\; N-2n \;&=\; \sqrt{\frac{n(n+1)(2n+1)}{6}} - 2n \\ &<\; \frac{3}{5}(n+1)\sqrt{n} - 2n \;<\; (n-\sqrt{n}+2)\sqrt{n} - 2n \\ &=\; (n-2\sqrt{n})(\sqrt{n}-1)\,. \end{aligned}$$

By the choice of s, the sum of the $s-1$ largest squares does not reach $p-2n$, and the smallest of them is bigger than $n-s$ (hence all are), thus

$$(n-s)(s-1) < p-2n < (n-2\sqrt{n})(\sqrt{n}-1)\,,$$

implying $s < \sqrt{n}$. Similarly, by the choice of t, the sum of the $t-1$ squares between $n-s$ and $n-s-t+2$ does not reach $p-2n$, and the smallest of them is bigger than $n-s-t > n-\sqrt{n}-t$, thus

$$(n-\sqrt{n}-t)(t-1) < p-2n < (n-2\sqrt{n})(\sqrt{n}-1)\,,$$

implying $t < \sqrt{n}$. Moreover, it follows that

$$s \geq \left\lceil \frac{p-2n}{n} \right\rceil \geq \frac{p-2n}{n}$$

and from $s \le t < \sqrt{n}$ it follows that the size $n - s - t + 1$ of the smallest square in the top row is larger than $n - 2\sqrt{n}$, thus

$$t \le \left\lceil \frac{p-2n}{n-2\sqrt{n}} \right\rceil < \frac{p-2n}{n-2\sqrt{n}} + 1.$$

As a consequence,

$$t - s < 1 + \frac{(p-2n)2\sqrt{n}}{n^2 - 2n\sqrt{n}} \le 1 + \frac{2(N-2n)\sqrt{n}}{(n^{3/2}-2n)\sqrt{n}} < 3.$$

Here the last step is equivalent to $N < n^{3/2}$, what clearly is a valid inequality because $6N^2/n = (n+1)(2n+1) < 6n^2$ holds for all $n > 1$. Of course, $t - s < 3$ means $t - s \le 2$, as claimed.

By the choice of s and t, the two rightmost squares, namely n and $n - s - t + 1$, reach $p-2n$ but do not reach $p-n$. Hence a rectangle at least $(n+1)\times(2n-s-t+\varepsilon)$, with area bigger than $2n^2 - 2n\sqrt{n} + 2n - 2\sqrt{n}$, remains uncovered in R up to height $2n - s - t + \varepsilon$, within which the width $p - 2n$ is entirely covered. On the other hand, as the largest vertical overlap is at most $t - s + 1 \le 3$, the area covered twice within width $p - n$ by the squares used so far is not larger than $3p - 3n$. It is smaller than $\frac{9}{5}n\sqrt{n} - 3n + \frac{9}{5}\sqrt{n}$, using the upper bound $p < \frac{3}{5}(n+1)\sqrt{n}$. Thus, the blank area exceeds the double-covered one by more than

$$n^{1/2} \cdot (2n^{3/2} - \tfrac{19}{5}n + 5n^{1/2} - \tfrac{19}{5}) .$$

Here, it can be seen that for all $n \ge 1$ both factors are increasing functions of $\sqrt{n}$ (and also of n), thus the product is positive already for $n = 4$ (hence for all $n \ge 4$).

Consequently, cutting off the $p \times (2n - s - t + \varepsilon)$ rectangle from the bottom of R, I obtain a $p \times q'$ rectangle with $q' = q + s + t - 2n - \varepsilon$, that should be covered with the squares from 1 to $n' := n - s - t$. The above calculations show that

$$p(q - q') = A_n - A_{n'} - |\text{double}| + |\text{blank}| > A_n - A_{n'} ;$$

thus, as $pq < A_n$ holds by assumption, I obtain $pq' < A_{n'}$. Consequently, the theorem follows by induction, once I verify that $|p - q'| < 3n'$ also holds.

Since $0 \le q - p < 3n$ has been assumed, I now have $-2n + s + t - \varepsilon \le q' - p < n + s + t - \varepsilon$. In this way, if n is not very small, I obtain

$$\begin{aligned} 0 \;&<\; n + s + t - \varepsilon \;<\; n + 2\sqrt{n} \;\le\; 2n - 2\sqrt{n} \;<\; |-2n + s + t - \varepsilon| \\ &\le\; 2n + 1 - s - t \;<\; 2n + 1 - 3(s+t) + 4\sqrt{n} \;<\; 3(n - s - t) \;=\; 3n' ; \end{aligned}$$

e.g., $n \ge 18$ is sufficient.

This inductive proof also provides a linear-time algorithm that generates a square cover of any given rectangle with the parameters p, q, n satisfying the conditions of the theorem. (In many steps of the procedure, consecutive slices will alternate between horizontal and vertical position.) □

3.3.5 Further questions

The following question is studied: Given the set of consecutive squares $[1, n]$, what is the largest square size $K \times K$ that can be completely covered by this set?

Up to $n \leq 6$ I can solve the problem easily by hand. Already for $n = 7$ the case is not trivial. For n between 8 and 21, the optimal solutions can be determined using a commercial mathematical programming solver, and for some larger values of n, close lower and upper bounds on K can still be obtained.

A heuristic algorithm is also introduced, capable of providing lower bounds for relatively large values of n, such as $n = 100$. It is not yet clear how far the obtained results are from the optimal values. More specialized heuristic algorithms may be able to produce improved lower bounds.

An expansion-like algorithm is introduced, capable of constructing a cover for any $n' > n$ if a cover is known for n. Here there are at least two interesting questions. One is that I do not know how good this algorithm is in the sense that, e.g., if I have an optimal solution for n, how far will the constructed cover be for $n + 1$ (in the worst case) from the optimal solution. Another related observation is the following: if a cover of a square of size L contains at least s squares aligned in the horizontal (or vertical) direction, then these squares can be expanded in the same direction to obtain a cover for a larger square of size $L + s$. A small counterexample was also provided against performing horizontal and vertical expansions in parallel, illustrating that the expansion procedure should be carried out in two phases. First, s is counted in one direction and the expansion is applied; then s is counted again in the other direction, followed by the corresponding expansion. If s is counted simultaneously for both directions before applying the expansions, a hole may be created, leading to an invalid cover. The counterexample involves several squares of identical size, instead of exactly one of each consecutive square in $[1, n]$. An open question is whether the same phenomenon (i.e., the creation of a hole) can also occur for the set of consecutive squares $[1, n]$, when determining s prior to the expansions in both directions.

During my investigation, the trivial upper bound has been used (for the size of K of the biggest covered square) as $K \leq UB_n = \left\lfloor \sqrt{\sum_{k=1}^{n} k^2} \right\rfloor$. Naturally, if I do have a covering of a square of size L where $L = UB_n$, I know for sure that this is the optimal value. For some small values of n, $K = UB_n$ really is the optimal value, but unfortunately, from some bigger values of n, this is not the case. Thus, it would be useful to know some other, more effective upper bound. At moment I have only this upper bound UB_n, which I know by example is not always tight, so this is a question whether one can find some tighter bound.

It is also an interesting question about the nature of the "gap" between UB_n, the trivial upper bound, and K, the optimal value. It is clear (following from my asymptotic result) that $\frac{UB_n - K}{UB_n} \to 0$ as $n \to \infty$. It would be nice to give a more exact characterization about the gap.

Chapter 4

New scientific results

This thesis contributes new scientific results in the field of combinatorial optimization through the introduction of the Board Packing Problem (BoPP), a novel hybrid formulation combining principles from facility location and two-dimensional bin packing.

While traditional models treat these domains separately, BoPP integrates spatial gain functions with variable rectangle selection under cost constraints. Additionally, new lower and upper bounds for square packing and covering are introduced, improving upon existing results for small and medium instances. The developed models, algorithms, and benchmark sets provide both theoretical insight and practical tools for geometric facility deployment problems.

For an overview of the author's academic publications and related bibliographic data, see the MTMT profile:
`https://m2.mtmt.hu/gui2/?type=authors&mode=browse&sel=10071311`

Thesis 1. *A new optimization problem, called the Board Packing Problem (BoPP), has been defined on a board with m rows and n columns, where each cell holds an integer revenue. A set R of rectangles is available, each defined by integer height, width, and purchase cost. The goal is to select and place rectangles on the board, aligned with the board's sides, to maximize profit, calculated as the total revenue from covered cells minus the total rectangle cost. Rectangles may overlap, but revenue from any cell can be collected only once. Its investigation has been initiated.*

1.1. It is proved in different ways that the BoPP is $\mathcal{NP}$-hard in different settings.

1.2. Benchmark classes were constructed, each comprising multiple generated instances. To facilitate result interpretation, a graphical interface was developed. Furthermore, the problem was formally modeled as a mixed-integer linear program (MILP).

1.3. A specialized hybrid evolutionary algorithm was developed for the BoPP. It uses a constructive heuristic to generate the initial population, providing near-optimal solutions already at the start of the evolutionary process.

1.4. The instances were solved using the commercial solver CPLEX and a custom evolutionary algorithm. It was conducted a comprehensive evaluation of the results across six different benchmark instance sets.

Thesis 2. *I addressed the following problem: given n squares with side lengths $[1, n]$, the task is to pack them, without rotation or overlap, into the smallest possible enclosing square. This problem is listed as sequence A005842 in the On-Line Encyclopedia of Integer Sequences (OEIS). I proposed new lower and upper bounds for the minimal enclosing square, and developed a simple algorithm suitable for large values of n.*

2.1. A new lower bound, LB_2, was introduced. For small and medium values of n (up to $n = 56$), the lower bounds were compared with known upper bounds from the literature. For larger values of n, a simple greedy-type algorithm was defined, and shown to be asymptotically optimal.

Thesis 3. *The following new problem was introduced: given n squares with side lengths $[1, n]$, the objective is to cover completely a square of size K using these items completely the largest square of size K. An evolutionary algorithm was developed, along with an expansion-based approach. These two algorithms, together with results obtained from CPLEX, were used to establish new lower-bound estimates for the largest possible value of K for various values of n.*

3.1. For small values of n, the optimal value of K was determined using a combination of simple observations, dedicated algorithms, and the commercial solver CPLEX.

3.2. For moderate values of n, lower and upper bounds for K were established through the use of CPLEX, an evolutionary heuristic, and an expansion algorithm.

3.3. For large values of n, a simple covering algorithm was developed that is proven to be asymptotically optimal.

Chapter 5

Conclusions

In this thesis, I address various problems related to packing and covering. These can be found in the most diverse areas of science and industry, such as manufacturing, storage and warehousing, transportation, finance and construction. If solved optimally, it may lead to cost savings and better use of resources.

In Chapter 2, I deal with the Board Packing Problem (BoPP). BoPP consists on covering the certain set of cells in a board. To cover cells, a finite set of rectangles is given: each of these rectangles can be purchased for a cost and placed on the board to cover cells. The rectangles to be purchased have their own height, width, and cost, and when placed on the board the rectangles are allowed to overlap. The goal is to obtain the highest possible profit, which is the sum of all gain values of the covered cells, minus the cost of purchased rectangles

The most simple instance of the BoPP is already NP-hard, when gains are binary and all rectangles are squares of height 2. Since it is likely that the problem cannot be solved in polynomial time, I proposed an evolutionary algorithm to find near optimal solutions. The performance of the evolutionary algorithm is compared against the mixed integer programming model, solved in CPLEX.

For certain instances, while the exact model (MIP) is unable to find any feasible solution within the time limit (600 seconds), the heuristic can find solutions with up to a small gap. The heuristic method has proven to be highly robust, with the performance of the solutions remaining stable even as instance size increases. It was also shown that the factors most influencing instance difficulty are the size of the board, the number and variety of rectangles, and the board's topography. Some questions and investigations that remained are for the future:

- For the BoPP for large instances, CPLEX is not able to determine an optimal or even a feasible solution. As mentioned, I applied an evolutionary algorithm. It remains for future research to apply another metaheuristic algorithm, like Ant Colony Optimization (ACO), or Variable Neighbourhood Search (VNS). It can be that some of these algorithms can be more efficient.
- Another option is to continue using the applied metaheuristic algorithm, while attempting to improve it. Such algorithms always offer possibilities for refinement. For example, the algorithm could be combined with different constructive heuristics to generate higher-quality solutions in the initial pool.
- I have defined several benchmark instances on which I examined my algorithms

for the BoPP. Of course, several other types of benchmark instances are also possible to be defined. One such interesting benchmark would be, where all gain values are either zero or one (note that the considered instance in the very early publication of Masek [95] is also of this type).

- In this thesis, I gave a very detailed investigation of the BoPP when overlap of rectangles is allowed. But, the other case when overlap is not possible is also worth to investigate. I already made some initial work in [47], but the detailed investigation remains for future research.
- What will happen if the items are not rectangles, but they have other geometrical shapes, e.g. circles? Such kind of investigations are interesting, but it will be much more difficult, because of handling these objects are harder.
- In Chapter 3, both the square packing and square covering problems are clearly special cases of my BoPP. An intriguing question is whether other specific cases might also be explored using the techniques I applied in the investigation of BoPP.

More than half a century ago, Martin Gardner popularized a question that led to the benchmark problem of determining the minimum side length of a square required to pack squares of sizes $[1, n]$ without overlap. Constructions are known for a certain range of n, and summing up the areas yields that a packing in a square of size smaller than $N := \sqrt{n(n+1)(2n+1)/6}$ is not possible. I proved that an asymptotically minimal packing exists in a square of size $N + cn + O(\sqrt{n})$ with $c < 1$, and such a packing is achievable with guillotine-cuts. An improved construction is also given for the case where the constraint of guillotine cutting is dropped.

Further, I have addressed the following square covering problem: What is the biggest square of size $K \times K$ that can be *covered* completely by this given set of "small" squares? It is assumed that the small squares must stand parallel to the sides of the big $K \times K$ square, and overlap is allowed.

In contrast to the *packing* version of the problem (asking for the smallest square that can accommodate all small squares without overlap) which has been studied in several papers since the 1960's, the covering version of the problem is new.

Optimal coverings have been constructed for small values of n. For moderately bigger n values I solve the problem optimally by CPLEX, and for even bigger n values I give a heuristic algorithm that can find near optimal solutions.

An expansion algorithm is provided, that from a given good cover of $K \times K$ for some n, can generate a cover for some $K' > K$ using small squares up to and including $n + 1$. Finally, I proved that a simple covering policy can generate an asymptotically optimal covering. It is easy to see that an upper bound to the problem is $\sqrt{\sum_{k=1}^{n} k^2}$. The lower bound is calculated by a heuristic. One of the most intriguing open questions is whether a new, non-trivial upper bound can be derived for this problem, given that only the trivial upper bound is currently known.

An interesting variation arises when packing and covering are considered together. For example, given a board:

- Some cells must be covered because their gain value is a huge positive value.
- Some cells must not be covered because their gain value is negative with a large absolute value.

In this scenario, it may happen that packing must remain within a specific area, but within this area, certain parts must be covered.

Many issues remain open for future research. In the investigated packing and covering problem, one copy of each square size between 1 and n is considered. It is natural to define other types of instances, for example with multiple copies of each size. A specific question of interest is whether it is possible to solve optimally any packing or covering problem when the squares are restricted to only two distinct sizes, but several copies of each size are available.

Naturally, many questions remain unanswered, and only a few have been highlighted here. The intention has been to contribute to the rich field of two-dimensional packing and covering problems by providing new perspectives and insights.

Appendices

Appendix A

A.1 Different board configurations for BoPP

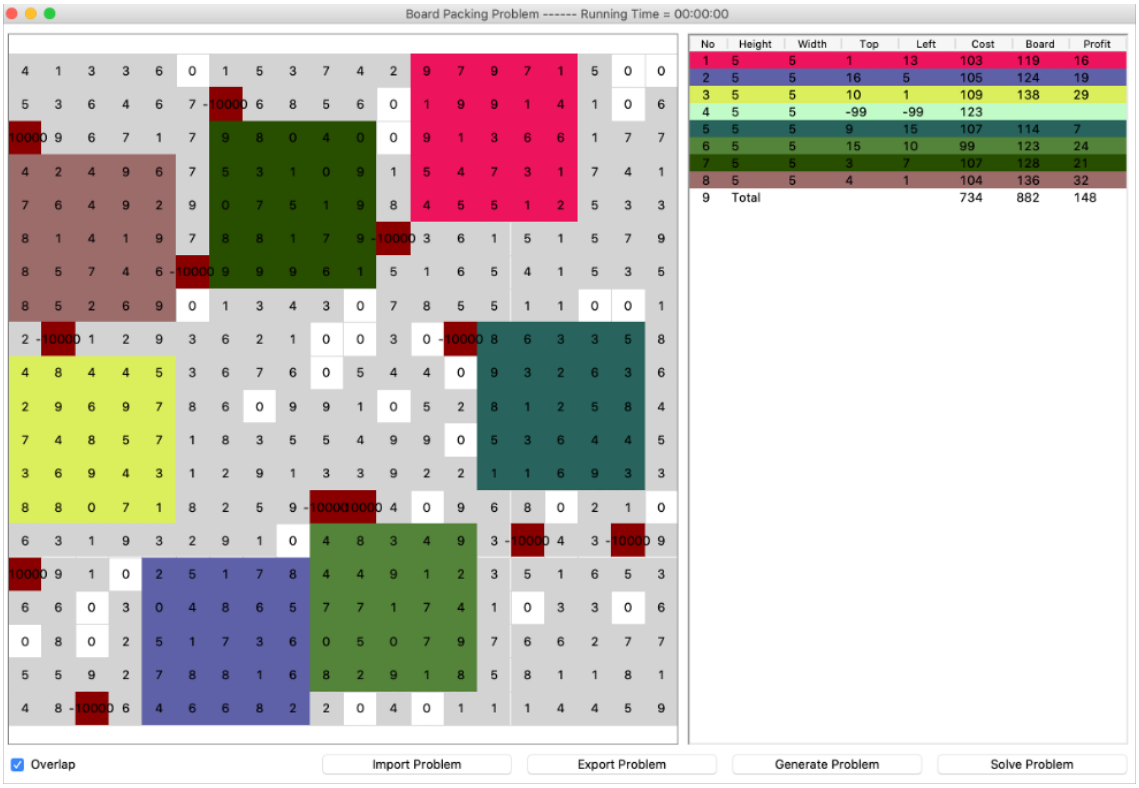


Figure A.1: The board´s input class is changed with gains from $[0, 9]$, and -10000 (a penalty value). Solution for 7 squares is found to give some positive profit. Overlapping is allowed.

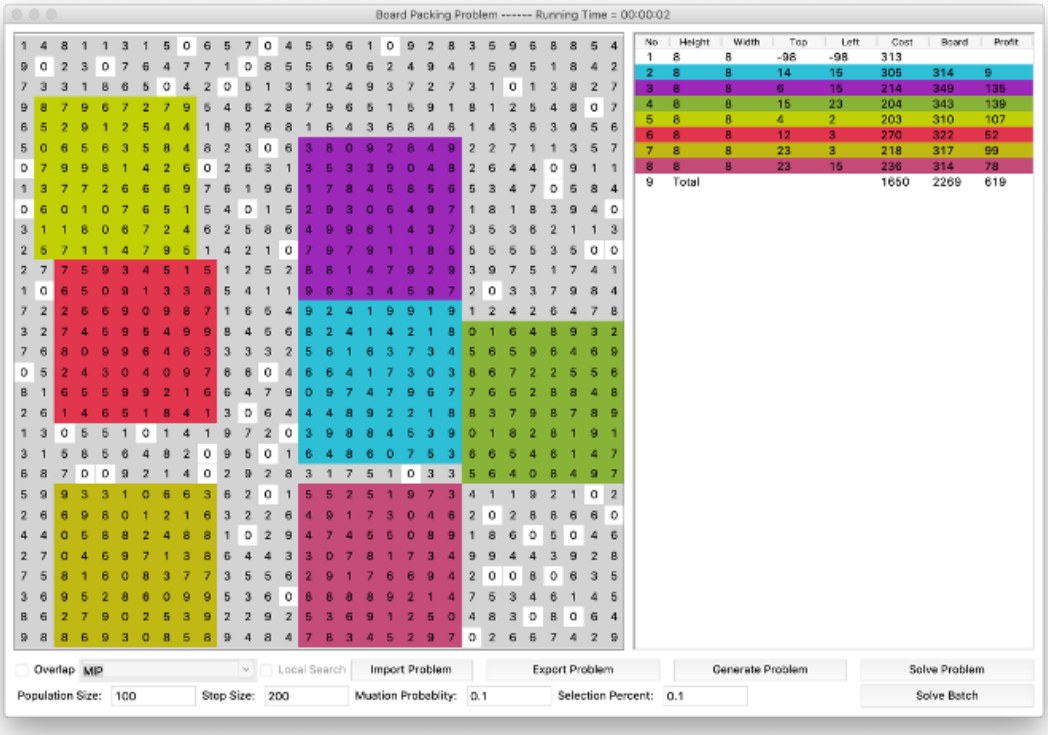


Figure A.2: Input Class (IC), Board (B), Rectangles to be packed (R), square Sizes (S): $IC = [0, 9], B = 30 * 30, R = 8, S = 8$

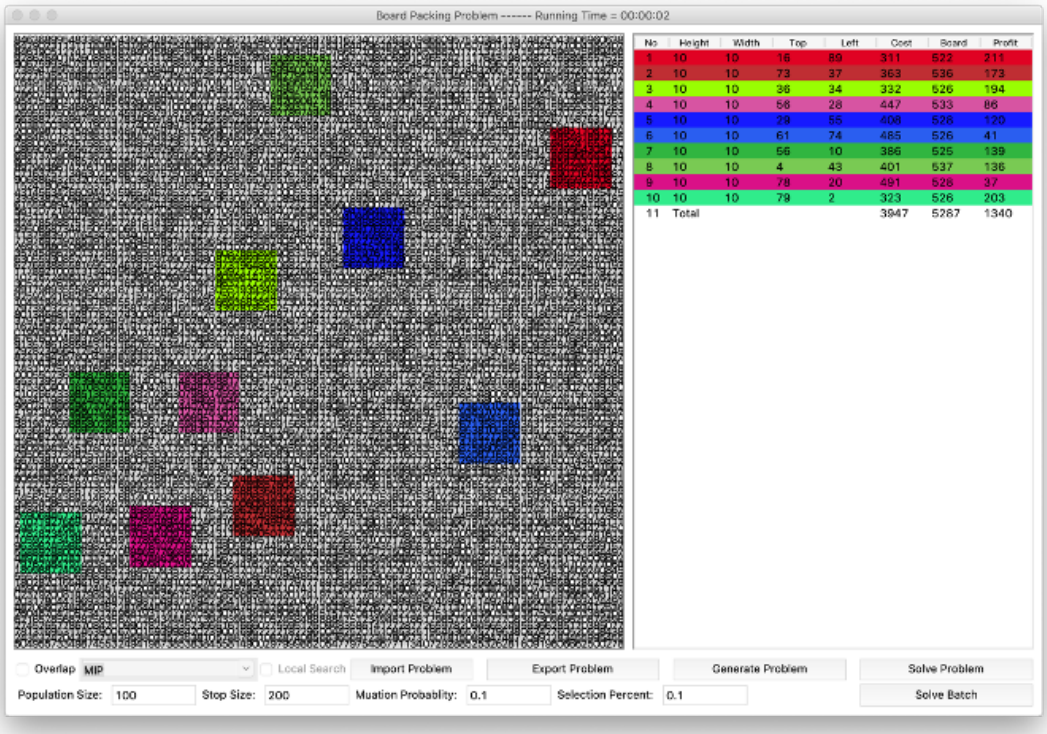


Figure A.3: Input Class (IC), Board (B), Rectangles to be packed (R), square Sizes (S): $IC = [0, 9], B = 100 * 100, R = 10, S = 10$

Appendix B

B.1 Reduction from some NP-hard SAT problems

Proof of (i)**.**
In this part of the proof, the term "box" refers to an 3×3 submatrix. A reduction is applied from the Boolean satisfiability problem 3-SAT.

Let $\Phi = c_1 \wedge \cdots \wedge c_p$ be any instance of 3-SAT over q variables $x_1, \ldots, x_q$; i.e., each c_j is the disjunction of exactly three distinct literals, each literal being either x_i or $\neg x_i$ for some $1 \leq i \leq q$, with $1 \leq j \leq p$.

It is assumed that x_i and $\neg x_i$ do not occur together in any clause. A 0–1 matrix $M(\Phi)$ of size $M \times N$ is constructed, where $M = O(p)$ and $N = O(q)$.

Almost all entries of $M(\Phi)$ will be 0, except for a relatively sparse set of entries 1, as specified below and indicated with $\bullet$ (and sometimes with $\otimes$ or $\copyright$) in the illustrations. In the text, their positions will be referred to as *1-cells*. Two 1-cells are *related* if they can be covered with a box; i.e., if the indices of their rows, as well as of their columns, differ by at most 2.

A *trajectory* is defined as a cyclic sequence of distinct 1-cells that satisfies the following properties:

- it consists of an even number of 1-cells;
- any two consecutive 1-cells in the cyclic order are related;
- if two 1-cells are not consecutive in the cyclic order, then they are not related.

Along a trajectory, the *distance* between any two 1-cells is defined in the natural way. Related 1-cells are said to be at distance 1; the two 1-cells related to a 1-cell are at distance 2; and so on.

Each variable x_i is represented with a trajectory t_i. Those trajectories will express the inclusion relation between clauses and variables. A schematic illustration of a possible arrangement of trajectories for a formula with three clauses and five variables is given in Figure B.1. Figure B.2 shows the detailed trajectory for the first of the five variables.

By definition, no t_i is allowed to repeat any 1-cell in the cyclic sequence. On the other hand, two trajectories are allowed to cross. It is required that such crosses occur in a specific way, as indicated in Figure B.3, thereby providing four possibilities for a box to cover exactly two 1-cells simultaneously from both trajectories.

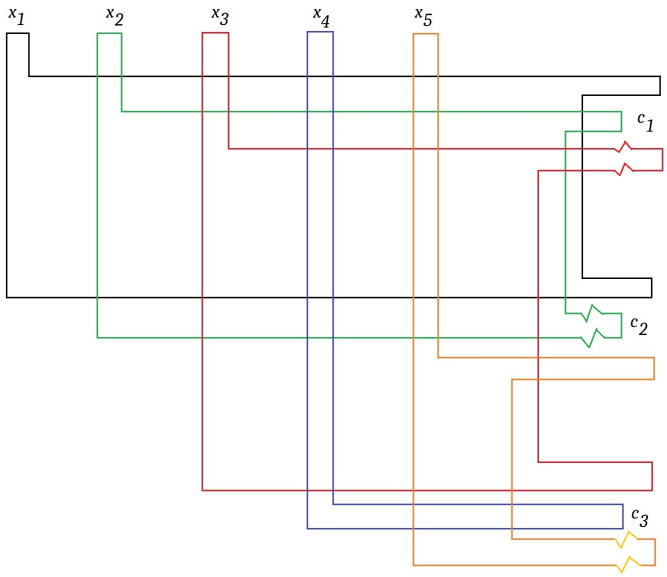


Figure B.1: Schematic arrangement of trajectories for Boolean formula $\Phi = c_1 \wedge c_2 \wedge c_3$ with three clauses $c_1 = x_1 \vee x_2 \vee \neg x_3$, $c_2 = x_1 \vee \neg x_2 \vee x_5$, $c_3 = x_3 \vee x_4 \vee \neg x_5$ over five variables x_1, x_2, x_3, x_4, x_5.

Apart from this very restricted situation, 1-cells of distinct trajectories are not allowed to be related to each other. In particular, trajectories cannot share a single 1-cell like a touching point of two planar curves. If consecutive pairs of a trajectory are connected by straight line segments, a simple polygon is obtained that partitions the plane into an internal (finite) region and an external (infinite) region. It follows that a trajectory either is disjoint from all the other trajectories or it contains an even number of crossing 1-cells.

On segments of non-crossing 1-cells, the following requirement is imposed: for each t_i that meets at least one other t_j, between any two crossing 1-cells there must be an odd number of intermediate 1-cells; hence the distance between any two crossing 1-cells is even.

Also recall that, by definition, if t_i is disjoint from all the other trajectories, then it consists of an even number of 1-cells.

As the number of 1-cells in a trajectory is even in either case, each t_i admits precisely two types of "perfect covers" with half that many boxes, as illustrated in Figure B.4. One of them is associated with the truth assignment $x_i = \mathsf{T}$ and the other with $x_i = \mathsf{F}$, referred to as the *true cover* and the *false cover*, respectively.

Furthermore, each clause c is represented with a 1-cell, drawn as © in subsequent figures. Exactly three trajectories are arranged close to ©. Namely, if an x_i is a literal in c, then there is a consecutive pair of 1-cells in t_i such that a true cover of t_i can cover © together with this pair, but a false cover cannot; and if $\neg x_i$ is a literal in c, then a false cover can do so, but a true cover cannot. All three literals of c are required to satisfy this property. Figure B.5 shows that this can be done, indeed,

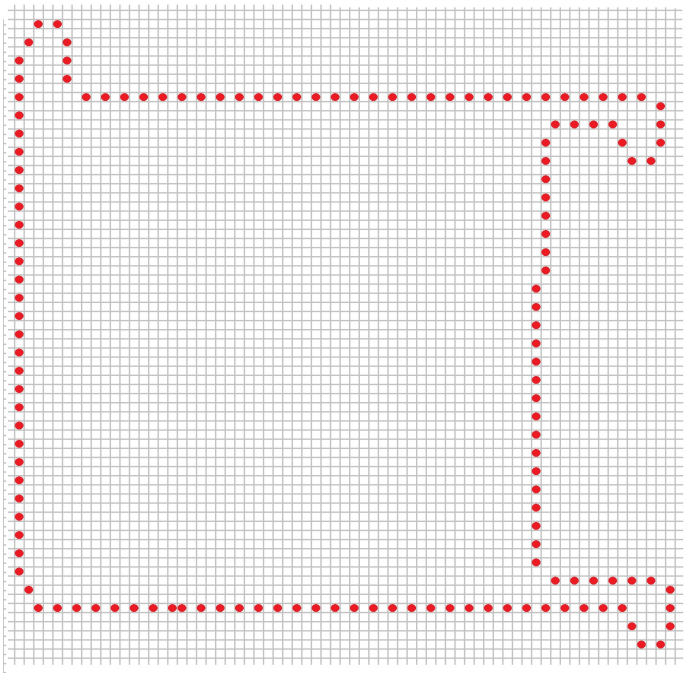


Figure B.2: Detailed trajectory for x_1 in Figure B.1, with red dots corresponding to a 1 in the matrix and other cells corresponding to a 0.

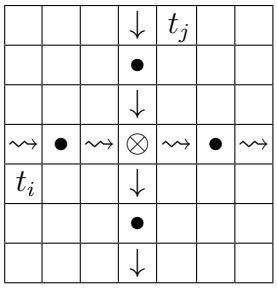


Figure B.3: Crossing 1-cell $\otimes$ of trajectories t_i and t_j.

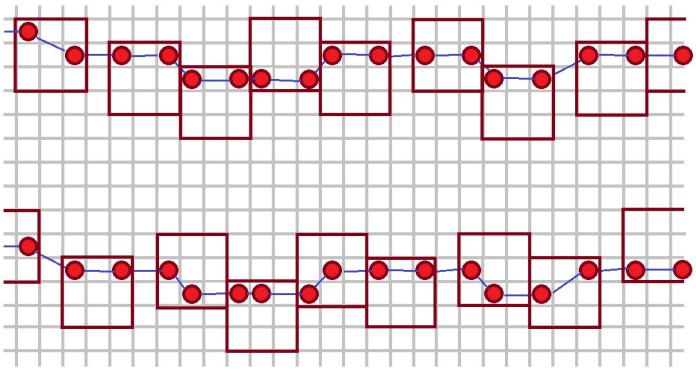


Figure B.4: The two perfect covers are shifted by one 1-cell.

keeping the three trajectories around © unrelated.

Moreover, the set of all crossing 1-cells together with all clause 1-cells must not contain any related pair. This condition implies that every box contains at most three 1-cells.

It is possible to arrange the trajectories within a rectangle proportional to $p \times q$ in a way that respects the requirements on all clause 1-cells. Namely, the condition that defines "related pairs" in the trajectories means that if two cells are at distance d in the cyclic sequence then their row/column indices differ by at most $2d$; but within this bound there is a high degree of flexibility concerning local shape and density, as illustrated in Figure B.6. On the other hand, to keep distinct trajectories unrelated except at crossings, large gaps between horizontal or vertical lines are not required in the schematic structure shown in Figure B.1, since a difference of 3 in row/column indices already ensures unrelatedness. These observations imply that for any Φ a matrix of size $O(p) \times O(q)$ will suffice. Once $M(\Phi)$ has been constructed for Φ, the following notation is introduced:

- T — the set of all 1-cells in $t_1 \cup \cdots \cup t_q$;
- K — the set of all crossing 1-cells, that is the union of the intersections $t_i \cap t_j$ for all $1 \leq i < j \leq q$;
- C — the set of all clause 1-cells.

Moreover, set the input parameters as follows:

- problem input: $M(\Phi)$;
- each box costs 1, hence its profit is the number of 1-cells in it, minus 1;
- ask whether a profit of $|C| + \frac{1}{2}(|T| + |K|)$ is achievable.

It will be proved that the answer is yes if and only if Φ is satisfiable. Since 3-SAT is an NP-complete problem, the same holds true also for the Board Packing problem.

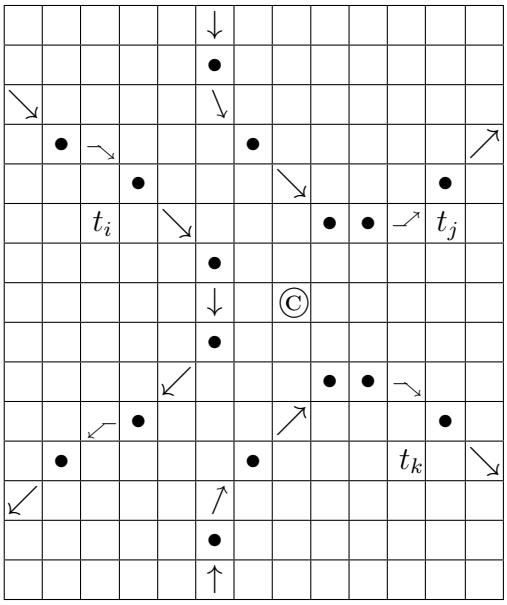


Figure B.5: Example of a clause 1-cell © and its neighborhood, for a clause $c = \ell_i \vee \ell_j \vee \ell_k$ with $\ell_i \in \{x_i, \neg x_i\}$, $\ell_j \in \{x_j, \neg x_j\}$, $\ell_k \in \{x_k, \neg x_k\}$.

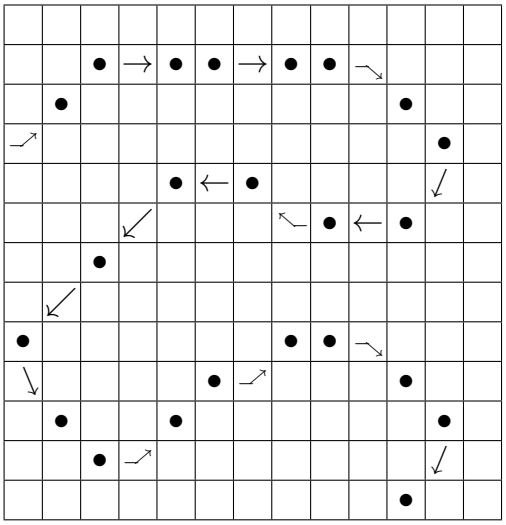


Figure B.6: A possible segment of a trajectory.

Suppose first that Φ is satisfiable. A total of $\frac{1}{2}(|T|-|K|)$ boxes are to be purchased in order to cover all $|T|+|C|$ 1-cells, thereby achieving the required profit.

Let $f : \{x_1, \ldots, x_q\} \to \{\mathsf{T}, \mathsf{F}\}$ (meaning $\mathsf{T} = \textsc{true}$ and $\mathsf{F} = \textsc{false}$) be a satisfying truth assignment of Φ. Each trajectory will be covered with a true or false cover, in accordance with f. Observe that if $t_i \cap t_j \neq \varnothing$, then at each crossing 1-cell for each of the four possibilities of $f(x_i), f(x_j)$ there exists a box usable in the required true/false covers of t_i and t_j that also contains $t_i \cap t_j$. In this way each selected box covers precisely two 1-cells from $T \setminus K$, and none of those 1-cells is covered twice, thus the entire $T \setminus K$ is covered; and as just has been said, properly chosen boxes also cover the entire K without any extra cost. Moreover, since every clause c is satisfied, the corresponding cover of t_i can be taken in the neighborhood of © in such a way that the clause 1-cell representing c is also covered. These facts ensure the profit of $|C| + \frac{1}{2}(|T| + |K|)$.

Suppose now that a profit of at least $|C| + \frac{1}{2}(|T| + |K|)$ has been achieved, and that k boxes have been purchased to obtain it. Let $R_1, \ldots, R_k$ denote these boxes. Without loss of generality, it may be assumed that each R_ℓ covers exactly two 1-cells outside $C \cup K$. It may happen that some 1-cell is contained in more than one R_ℓ; but its gain $g_{i,j} = 1$ can be obtained only once. To make the counting transparent, one may consider a mapping $h : (i, j) \mapsto \ell$ to express that the gain of $g_{i,j}$ is counted for R_ℓ (but for no other box covering the corresponding cell). Under this assignment, $h^{-1}(\ell) \in \{0, 1, 2, 3\}$ holds. In fact, if $h^{-1}(\ell) = 0$, then R_ℓ increases the cost, and omitting it from the covering increases the profit.

1. If $h^{-1}(\ell) = 1$, then R_ℓ costs 1 and gets gain 1. In this case, R_ℓ is omitted from the cover without changing the profit.

2. If $h^{-1}(\ell) = 2$, then one of the following situations occurs:

   (a) R_ℓ covers two 1-cells outside of $C \cup K$, both of them are assigned to it, but no cell from $C \cup K$ is assigned to R_ℓ.

   (b) R_ℓ covers a cell from $C \cup K$, which is assigned to it, but one of the other two 1-cells in R_ℓ is assigned to another $R_{\ell'}$.

3. If $h^{-1}(\ell) = 3$, then R_ℓ covers a cell in $C \cup K$ and two 1-cells outside of $C \cup K$, and all the three are assigned to it.

Situation 2(b) can be eliminated by modifying h. Indeed, if the 1-cell of $g_{i,j}$ is covered by R_ℓ but $h : (i, j) \mapsto \ell'$, then the mapping is redefined as $h : (i, j) \mapsto \ell$. After this modification, $h^{-1}(\ell') = 1$, and then $R_{\ell'}$ is omitted by case 1. At the same time R_ℓ becomes an item of case 3. After these adjustments, only cases 2(a) and 3 remain. Cells in the former and the latter make profit 1 and 2, respectively. The number of the latter is at most $|C| + |K|$, and the total number of boxes is k. Consequently the total profit is at most $k + |C| + |K|$; but at the same time it is at least $|C| + \frac{1}{2}(|T| + |K|)$ by assumption. Hence

$$k \geq \frac{1}{2}(|T| - |K|)\,.$$

On the other hand, the total number of 1-cells is $|C| + |T|$, and k boxes have been purchased; hence the profit cannot exceed $|C| + |T| - k$. If this is at least $|C| + \frac{1}{2}(|T| + |K|)$, then

$$k \leq \frac{1}{2}(|T| - |K|)$$

also holds, and thus equality follows. Note further that in the cases of 2(a) and 3 each box R_ℓ has its two private 1-cells outside of $C \cup K$. As a consequence, if $k = \frac{1}{2}(|T| - |K|)$ holds and the profit is $|C| + \frac{1}{2}(|T| + |K|)$, then the boxes $R_1, \ldots, R_k$ partition $T \setminus K$ and entirely cover K and also C. (If R_ℓ covers a crossing cell, then its other two 1-cells belong to two distinct trajectories, while the boxes disjoint from K contain two 1-cells from the same trajectory.)

It is proved that such a cover defines either a true or a false cover on each trajectory and, in this way, corresponds to a truth assignment f on Φ. Once this will be shown, the only possibility to cover C at the same time is that f satisfies Φ.

The claimed property is obvious for any trajectory t_i which is disjoint from all the others, because then there can be only two ways for $R_1, \ldots, R_k$ to partition t_i. The situation is different when $t_i \cap t_j \neq \varnothing$, because in principle it might happen that a crossing 1-cell is covered with two antipodal boxes. It is shown that this is impossible under the restrictions imposed on the trajectories.

Consider an intermediate segment of t_i with its two neighbor crossing 1-cells, say $w_0, v_1, v_2, \ldots, v_{2s-1}, w_{2s}$, where w_0, w_{2s} are crossing 1-cells and $v_1, v_2, \ldots, v_{2s-1}$ are non-crossing 1-cells. Assume first that $s \geq 2$. Then, since $T \setminus K$ is partitioned by the covering boxes, either there is a box R_j containing the pair v_1, v_2 or else there is a box R_j containing the pair v_{2s-2}, v_{2s-1}. Assume that the latter holds, the former case being similar. The continuation $w_{2s}, v_{2s+1}, v_{2s+2}, \ldots, v_{2t-1}, w_{2t}$ of t_i is then considered, where w_{2t} is the next crossing 1-cell. It may occur that $w_{2t} = w_0$, but the argument works for this particular situation, too. Here a box $R_{j'}$ containing w_{2s} cannot contain v_{2s-1} because then this 1-cell of $T \setminus K$ would be covered twice. Thus $R_{j'}$ must contain v_{2s+1}, and then a next box must contain the pair v_{2s+2}, v_{2s+3}, and so on. Also the pair v_{2t-2}, v_{2t-1} is contained in a selected box. Hence, the situation at the end of this second segment is the same as it was at the end of the first segment. In this way, the property propagates through the entire t_i, resulting in either a true cover or a false cover. The situation is similar if $s = 1$. Then a box covering v_1 has to contain either w_0 or w_2, causing a transfer to the next segment in the same way.
In conclusion, it has been shown that a gain of $|C| + \frac{1}{2}(|T| + |K|)$ is achievable if and only if Φ is satisfiable. This completes the proof of part (i).

Proof of (ii)**.**
The cases of $(2, 3)$ and $(3, 2)$ are analogous and can be obtained from each other by reflection on a diagonal (i.e., by row $\leftrightarrow$ column transformation). Thus, it is sufficient to restrict our discussion to boxes that have two rows and three columns. Recall that the entries are now taken from the set $\{-1, 0, +1\}$.

The proof for this case is similar to the previous one, with a reduction applied from the 3-SAT problem. However, in the neighborhood of a clause 1-cell less space is available, making it impossible to arrange three entirely unrelated trajectories

unless some restrictions are relaxed. To address this difficulty, cells with value -1 are used, as indicated with N (abbreviating the word "negative") in Figure B.7, thereby allowing certain boxes to contain three non-crossing 1-cells of a trajectory.

Actually this -1 is strictly necessary only for t_k, since t_j might be modified, placing its first two 1-cells in the column next to © on its left.

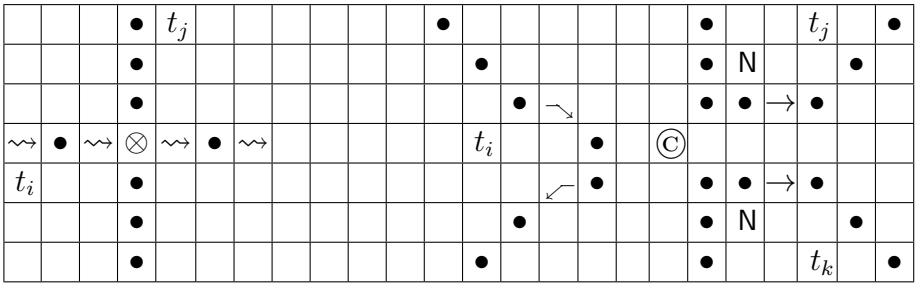


Figure B.7: Crossing 1-cell ⊗ and the neighborhood of a clause 1-cell © for 2×3 boxes; nonzero cells are $\bullet = 1$ and $\mathsf{N} = -1$.

Note that true covers and false covers can be ensured properly according to the clauses, because 2×3 boxes still offer flexibility for arranging a trajectory. This fact is illustrated in Figure B.8.

Now there exist boxes containing more than two non-crossing 1-cells of the same trajectory, but they necessarily contain a negative cell also. As one can see, the gain of such a box is exactly the same as that of a box not containing the (-1)-cell and the 1-cell above or under it. Hence, concerning profit, a cover using at least one box with more than two non-crossing 1-cells is equivalent to a cover without (-1)-cells such that not all trajectories are completely covered. Due to the computation above, such a cover can never achieve a profit of $|C| + \frac{1}{2}(|T| + |K|)$, thus it cannot be optimal when Φ is satisfiable. It has also been shown that for a non-satisfiable formula the profit $|C| + \frac{1}{2}(|T| + |K|)$ cannot be achieved either. These facts prove (ii), hence the theorem for 2×3 boxes in boards whose entries are from $\{-1, 0, +1\}$. This completes the proof of part (ii).

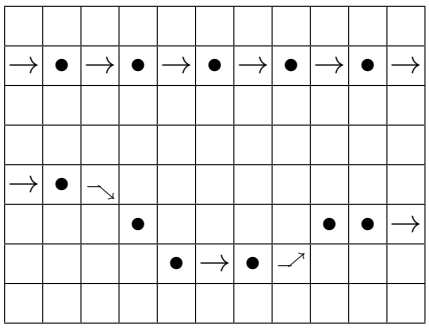


Figure B.8: Horizontal or vertical flexibility of trajectories for 2×3 boxes.

Proof of (iii)**.**
In this last part of the proof "box" means a 2×2 rectangle. Recall that entries from $\{0, 1\}$ will be considered. The case of 2×2 boxes is substantially different from the previous ones, because three mutually unrelated trajectories around a clause 1-cell are not at all possible anymore, no matter what entries the board may contain. In this situation, a reduction is applied from another variant of satisfiability, namely the Max-2-SAT problem. Its input is a Boolean formula in conjunctive normal form such that every clause has at most two variables, and also an integer k is specified in the input. The question is whether there exists a truth assignment that satisfies at least k clauses.

Garey, Johnson and Stockmeyer [60] proved that the Max-2-SAT problem is NP-complete with the value $k = 7p/10$, where p is the number of clauses in the input formula. This fact alone is not sufficient for the proof of the theorem; a closer examination of the construction in [60, Theorem 1.1], which is a reduction from 3-SAT, is also required. In addition, a special geometric arrangement of the trajectories with respect to the clause 1-cells is needed.

Consider an instance Φ of 3-SAT, written in the form

$$\Phi = (a_1 \vee b_1 \vee c_1) \wedge \cdots \wedge (a_s \vee b_s \vee c_s)$$

where each of a_i, b_i, c_i is a positive or negative literal for $i = 1, \ldots, s$. From every clause $C_i = (a_i \vee b_i \vee c_i)$, an expression E_i of ten clauses is created, each clause with at most two literals, using an auxiliary variable d_i, too:

$$E_i = (a_i) \wedge (b_i) \wedge (c_i) \wedge (d_i) \wedge (\neg a_i \vee \neg b_i) \wedge (\neg a_i \vee \neg c_i) \wedge (\neg b_i \vee \neg c_i)$$

$$\wedge (a_i \vee \neg d_i) \wedge (b_i \vee \neg d_i) \wedge (c_i \vee \neg d_i)\,.$$

Then the input of Max-2-SAT is defined as the formula

$$\Phi^* = E_1 \wedge \cdots \wedge E_s$$

which has $p = 10s$ clauses; and one puts $k = 7s$. A key observation, which is very easy to check, is that in any truth assignment on Φ,

- if $(a_i \vee b_i \vee c_i)$ is T then there is a proper choice of $d_i \in \{\mathsf{T}, \mathsf{F}\}$ that makes seven[1] clauses of E_i satisfied simultaneously, but more than seven is impossible independently of the truth values of a_i, b_i, c_i;
- if $(a_i \vee b_i \vee c_i)$ is F then only six[2] clauses of E_i can be satisfied simultaneously.

As a consequence, Φ^* admits a truth assignment satisfying $7s$ clauses if and only if the input Φ of 3-SAT is satisfiable.

The basic principles of creating trajectories for variables is the same as in the previous arguments. Each trajectory has to admit a "true cover" and a "false cover", and it has to approach a clause according to whether setting the variable to T or F

[1] Put $d_i = \mathsf{F}$ unless $a_i = b_i = c_i = \mathsf{T}$.
[2] Six can always be attained by setting $d_i = \mathsf{F}$.

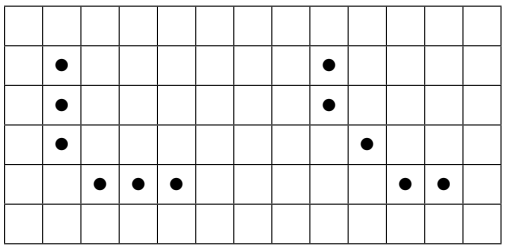


Figure B.9: A parity-switching local transformation.

satisfies the clause in question. This requirement, as well as the condition that the distance between any two crossing 1-cells is even, requires a way to adjust parities properly. This can be done at any turning part of a trajectory, as shown in Figure B.9. A possible situation of crossing 1-cells is indicated in Figure B.10.

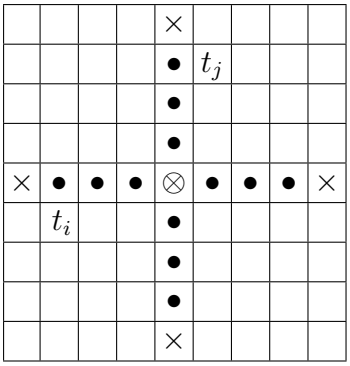


Figure B.10: Crossing 1-cell $\otimes$ of trajectories t_i and t_j, with an indication $\times$ of nearest allowed positions of other crossing 1-cells.

Furthermore, the trajectories in the neighborhood of a clause 1-cell can be arranged as shown in Figure B.11, to ensure that the trajectories are unrelated. If a box covers © together with two 1-cells of a trajectory, then shifting it by one unit horizontally (to the left for t_i or to the right for t_j) yields a box that still covers the same two 1-cells of the trajectory but no longer contains ©. Consequently, without loss of generality the analysis can be restricted to packings in which each clause 1-cell is covered at most once.

The notation T, K, C—as in the proof for the 3×3 case—is used for the set of 1-cells in the union of all trajectories, crossing 1-cells, and clause 1-cells, respectively.

For technical reasons, a lower bound is imposed on the number of 1-cells between any two crossing 1-cells on each trajectory (a value of 3 is already sufficient). Similarly, it is assumed that the two 1-cells on a trajectory that can be covered together with a clause 1-cell by a box are not located too close to the next crossing 1-cell. These conditions can be satisfied without difficulty.

In contrast to the case of 3-SAT, in the partial satisfaction problem Max-2-SAT it does not automatically follow that every optimal solution defines a true cover or a false cover on each trajectory. To preserve this property for Max-2-SAT, it is noted

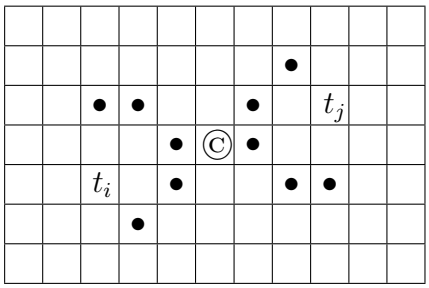


Figure B.11: The neighborhood of a clause 1-cell © for 2×2 boxes.

that each of a_i, b_i, and c_i appears in E_i exactly twice as a positive literal and twice as a negative literal. Consequently, every original variable of Φ occurs in Φ^* an even number of times, half of them positive and half negative. Based on this observation, it is required that each trajectory—except those corresponding to the variables d_i, which occur once as positive and three times as negative—alternates between clauses in which its variable appears positively and clauses in which it appears negatively. Prior to this, no restriction was imposed on the order in which a trajectory visits the clause 1-cells of its variable; therefore, this order can be chosen freely.

The simplest way of ensuring positive-negative alternation is to relax the definition of trajectories, allowing that a trajectory may cross itself. The reader can check that the proof below works under this relaxation, too. On the other hand it can be proved that self-crossing is avoidable by a careful design of trajectories with respect to the order of clause 1-cells. To avoid unnecessary technical details, this fact is illustrated with an example only, although a rigorous proof could also be provided, since crossings may be permitted in the proof of the theorem.

For instance, if variable x_1 occurs in the two clauses $c_1 = (x_1 \vee x_2 \vee x_4)$ and $c_2 = (\neg x_1 \vee x_3 \vee x_5)$, then among the 20 clauses of $E_1 \wedge E_2$ there are four containing x_1 as positive and four as negative:

$$x_1\ ,\ \ \neg x_1 \vee \neg x_2\ ,\ \ \neg x_1 \vee \neg x_4\ ,\ \ x_1 \vee \neg d_1\ ;\ \ \neg x_1\ ,\ \ x_1 \vee \neg x_3\ ,\ \ x_1 \vee \neg x_5\ ,\ \ \neg x_1 \vee \neg d_2\ .$$

The trajectory t_1 is not allowed to approach the corresponding clause vertices in this order, because it is not alternating. Among the 8! possible orders only $2 * (4!)^2$ alternate. Once the positions of clause cells are decided, many of the $2 * (4!)^2$ possible orders cannot be realized with a trajectory which does not cross itself, due to topological reasons. Perhaps the simplest way to overcome this difficulty is to relax the definition of trajectories, by allowing that a trajectory crosses itself. Keeping the condition that any two crossing 1-cells are separated by an odd number of non-crossing 1-cells, the argument below works for the proof of (*iii*). On the other hand, no matter in which order the clause 1-cells are placed, it can be rigorously proved that they admit a non-self-crossing trajectory that approaches those clause 1-cells in an alternating order as regards the sign of the corresponding variable. Rather than presenting a general formal proof, the procedure is described only for the current example. Starting from the order given above, two neighboring clauses in which x_1 occurs with opposite signs (e.g., $x_1 \vee \neg d_1$ and $\neg x_1$) are selected and

placed at the end of the order to be constructed. There remain the six clauses

$$x_1 \ , \quad \neg x_1 \vee \neg x_2 \ , \quad \neg x_1 \vee \neg x_4 \ , \quad x_1 \vee \neg x_3 \ , \quad x_1 \vee \neg x_5 \ , \quad \neg x_1 \vee \neg d_2 \ .$$

This subsequence also contains consecutive opposite occurrences of x_1, e.g. $\neg x_1 \vee \neg x_4$ and $x_1 \vee \neg x_3$. These clauses are placed before the previously identified pair, ensuring that the alternation for x_1 is preserved. Analogously, removing the latter two clauses, in the remaining subseuence x_i occurs with opposite signs in the two consecutive clauses $\neg x_1 \vee \neg x_2$ and $x_1 \vee \neg x_5$. These two clauses will precede the previously found four others in the sequence under construction. Finally x_1 and $\neg x_1 \vee \neg d_2$ remain, which will be placed at the beginning.

After all, the following sequence is obtained:

$$x_1 \ , \quad \neg x_1 \vee \neg d_2 \ ; \quad x_1 \vee \neg x_5 \ , \quad \neg x_1 \vee \neg x_2 \ ; \quad x_1 \vee \neg x_3 \ , \quad \neg x_1 \vee \neg x_4 \ ; \quad x_1 \vee \neg d_1 \ , \quad \neg x_1 \ .$$

Figure B.12 shows that there exists a trajectory without self-crossing, that visits the clause 1-cells in this order and also respects the original order of the placement of those cells.

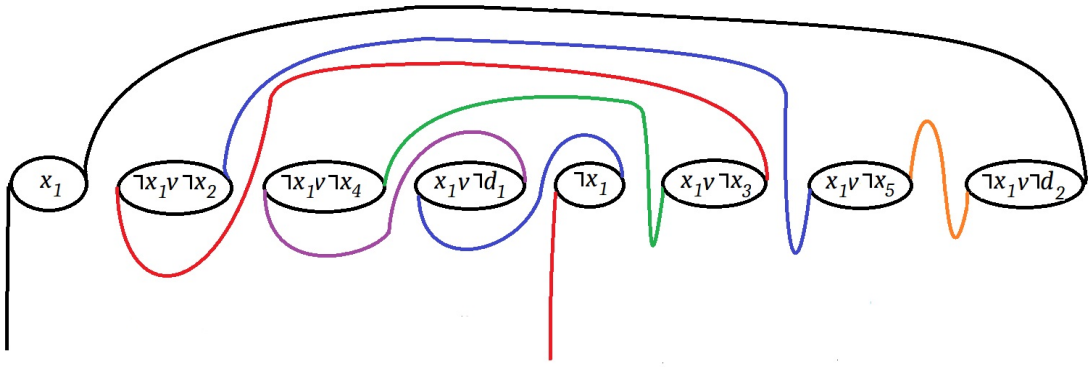


Figure B.12: Trajectory t_1 that respects a pre-defined placement order of clause vertices and does not cross itself, while alternates between positive and negative occurrences of variable x_1.

Next, the benefit obtained from alternation is examined. Consider any packing $\mathcal{P}$ of boxes that constitutes an optimal solution of the board packing problem. Without loss of generality, it can be assumed that among all optimal packings, $\mathcal{P}$ contains the minimum possible number of boxes. In this case, the boxes are mutually disjoint in $T \cup C$, and each of them has its two private 1-cells in $T \setminus K$. At this point, the assumptions are applied that any two K-cells are separated by a distance of at least 3, no C-cell is covered more than once, and C-cells and K-cells are not located too close to each other. If a box $R \in \mathcal{P}$ meets K, it covers two pairs of consecutive 1-cells along the trajectories. (Those two pairs belong to distinct trajectories, unless the trajectory in question crosses itself at that 1-cell of K.) Hence there is a natural way to associate a set $\mathcal{P}_2$ of pairs to $\mathcal{P}$, namely the set of those pairs of consecutive

1-cells in the trajectories which are contained in the members of $\mathcal{P}$. A box meeting K contains two pairs from $\mathcal{P}_2$.

Recall that there exists a well-defined partition of any trajectory, referred to as a true cover, and another referred to as a false cover. Each means a partition into pairs. This classifies the members of $\mathcal{P}_2$ as "true pairs" and "false pairs". It is immediately observed that a sequence of true pairs is separated from a sequence of false pairs by one (or more) 1-cell which does not belong to any $R \in \mathcal{P}$. Without loss of generality, it may be assumed that none of the uncovered cells are crossing 1-cells. Suppose, for the sake of contradiction, that $k \in K$ is an uncovered cell. By inserting a box R' into $\mathcal{P}$ that covers k, and removing the members of $\mathcal{P}$ that intersect R', one or two boxes of profit 1 are eliminated while one box of profit 2 is added. In the first case, where only one box is removed, $\mathcal{P}$ cannot have been optimal. In the second case, where two boxes are removed, the cardinality $|\mathcal{P}|$ is not minimal, contradicting the initial assumptions.

Suppose that on each trajectory, $\mathcal{P}_2$ determines either a true cover or a false cover. Then $\mathcal{P}$ corresponds to a truth assignment, say f, of Φ^*. The number of clause 1-cells covered by $\mathcal{P}$ is at most the number of satisfied clauses under f in Φ^*, and equality can be achieved for any f by properly choosing the members of $\mathcal{P}$. In particular, if the instance Φ of 3-SAT is satisfiable, then Max-2-SAT admits a truth assignment for the variables $d_1, \ldots, d_s$ in such a way that the derived packing $\mathcal{P}$ covers $7|C|/10$ clause 1-cells and the entire T is covered at the cost of $\frac{1}{2}(|T| - |K|)$, so that a total profit of $7|C|/10 + \frac{1}{2}(|T| + |K|)$ is achieved. Also conversely, under the assumption that $\mathcal{P}_2$ determines either a true cover or a false cover on each trajectory, the cost of the packing is equal to $\frac{1}{2}(|T| - |K|)$, therefore a profit of $7|C|/10 + \frac{1}{2}(|T| + |K|)$ is achievable only if Φ is satisfiable.

It remains to prove that all the other kinds of $\mathcal{P}$—i.e., if there are trajectories on which $\mathcal{P}$ is neither a true cover nor a false cover—make smaller profit. In proving this, it can be assumed that no trajectory contains two consecutive uncovered 1-cells. If such a pair existed, a box covering those two cells would yield a gain of 2 while incurring a cost of only 1, thereby increasing the profit. So, from now on let $\mathcal{P}$ be an optimal packing in which one or more trajectories consist of sequences of true pairs and sequences of false pairs, separated by uncovered 1-cells, while the other trajectories are completely covered.

As an auxiliary tool, let f_0 be a truth assignment that maximizes the satisfied clauses of Φ^*. The packing $\mathcal{P}$ is interpreted as a modification of f_0. Say, along a certain number u of segments on a trajectory the pairs are switched from “true” to “false” (or from “false” to “true”, depending on the truth value of f_0 on the corresponding variable). This change yields $2u$ uncovered 1-cells, which means a loss with respect to f_0. This loss may possibly be partially compensated by clauses that become satisfied when the truth value of the corresponding variable is switched.

Consider first any variable of Φ. By assumption, along its trajectory the positive and negative literals in the corresponding clauses alternate. Hence, if switching on a segment satisfies a number say v of previously unsatisfied clauses Φ^*, then by alternation it also makes $v - 1$ clauses unsatisfied, which were satisfied under f_0. It implies that with the loss of $2u$ it is possible to make an extra gain no more than u. In this way the total profit decreases by at least u.

Consider now the variables d_i. Each of them occurs only four times, one positive and three negative. Since f_0 maximizes the number of satisfied clauses, at most one of the four clause 1-cells d_i, $(a_i \vee \neg d_i)$, $(b_i \vee \neg d_i)$, $(c_i \vee \neg d_i)$ remains uncovered; e.g., $f_0(d_i) = \mathsf{F}$ can be applied. Compared to this, making a switch in the trajectory of d_i would yield at most one extra gain, while it would require two uncovered 1-cells and hence lose gain 2, decreasing the profit.

It follows that the best packings are those originating from truth assignments maximizing the satisfied clauses in Φ^*. This completes the proof of the theorem.

Appendix C

C.1 Small cases for the Square Packing Problem

The special case of $n = 24$

An interesting special case of the problem is $n = 24$, which was unsolved until 2004. As proved already in [118], $n = 24$ is the unique non-trivial case where $A_n := \frac{n(n+1)(2n+1)}{6}$ is the square of an integer. However, while $LB_0 = 70$, the optimum is 71 as proved by Korf [83]. Staying at the very special $n = 24$ case, since the small squares of $[1, 24]$ cannot be packed without overlap into the 70×70 accommodating big square, and their total size is just 70×70, it follows also that those small squares cannot cover completely the 70×70 big square.

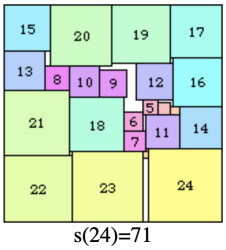


Figure C.1: The best known feasible packing solution for the case $n = 24$ (Friedman [57]).

The fact that for $n = 24$ the squares do not fit into a $70 * 70$ board was proved by computer search, as claimed first in [21] without giving details and later in [83, 84]. However, no proof without use of computer was given. The first complete combinatorial proof is given in the very recent paper [111].

Some initial steps towards a combinatorial proof were given in [86]. Using an extensive case analysis, it is shown there that squares of size up to 5 cannot be placed on the edges of the 70×70 board; in addition some further combinations of squares on edges, typically involving 6, 7, and/or 8, are also excluded.

This problem belongs to a wide class of Mrs. Perkins's Quilt problems, that in general ask for "squaring the square", which means a tiling of a given square with smaller squares. See [7] for more details of this area. Thus, this problem is one special case.

The case of $n = 18$

It can be observed that $n = 18$ and $n = 24$ are the only two cases for which no lower bound is tight. For $n = 24$, there exists a purely theoretical proof that the squares cannot fit into a 70×70 square. Thus, within the range up to $n = 24$, only the case $n = 18$ remains where the size of the smallest accommodating square is known, but the underlying reason is not understood. Some considerations on this case are provided in [14].

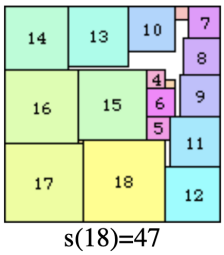


Figure C.2: The best known feasible packing solution for the case $n = 18$ (Friedman [57]).

For $n = 18$, I have $LB_0 = 46$. Here $\sum_{k=1}^{18} k^2 = 2109$ while $46^2 = 2116$. Let us suppose that the small squares fit into the $46*46$ board. It means that $2116-2109 = 7$ small $1*1$ cells remain empty. Let us add 7 such small squares to the set of $[1, 18]$, then these $18 + 7$ squares completely fill the $46 * 46$ board. Let us cut all squares into slices of unit width. In this way I get $\sum_{k=1}^{18} k + 7 = 178$ slices.

Then, there must be such row of the completely filled board, where there are at most 3 slices (as $4 \cdot 46 > 178$). As two slices are not enough to completely fill a row (as its length is 46 and $18 + 17 < 46$), it means that there exists such row where there are exactly 3 slices. The same holds for the columns. Let us consider one such row and one such column. In the intersection the square is the same. The following observation can therefore be stated:

Observation: There exist 5 squares, denoted by A, B, C and E, F, G, so that A, B and C have a common row and they completely fit that row, and E, F and G have a common column and they completely fill that column. Moreover, one square out of $\{A, B, C\}$ and one square out of $\{E, F, G\}$ is the same.

By symmetry, there are 3 cases about what square can be the common square: These cases are as follows:

- Case 1: The common square is in the middle in both triplets.
- Case 2. The common square is in side in both triplets.
- Case 2. The common square is in the middle in one of the triplets and in the side in the other triplet.

What triplets are possible? There are not too many options to have a triplet of the squares so that the sum of their sizes is just 46. These are the options:

$$18 + 17 + 11$$
$$18 + 16 + 12$$
$$18 + 15 + 13$$
$$17 + 16 + 13$$
$$17 + 15 + 14$$

Now all combinations for both triplets can be counted as well, noting that one square must be the same:

$$\mathbf{18} + 17 + 11 \text{ and } \mathbf{18} + 16 + 12$$
$$\mathbf{18} + 17 + 11 \text{ and } \mathbf{18} + 15 + 13$$
$$18 + \mathbf{17} + 11 \text{ and } \mathbf{17} + 16 + 13$$
$$18 + \mathbf{17} + 11 \text{ and } \mathbf{17} + 15 + 14$$
$$\mathbf{18} + 16 + 12 \text{ and } \mathbf{18} + 15 + 13$$
$$18 + \mathbf{16} + 12 \text{ and } 17 + \mathbf{16} + 13$$
$$18 + 15 + \mathbf{13} \text{ and } 17 + 16 + \mathbf{13}$$
$$18 + \mathbf{15} + 13 \text{ and } 17 + \mathbf{15} + 14$$
$$\mathbf{17} + 16 + 13 \text{ and } \mathbf{17} + 15 + 14$$

The common squares are denoted by bold numbers. Now let us focus only one of the possible situations, namely the last one in the list. Let us suppose that the square of size 17 is in the middle (see Figure C.3).

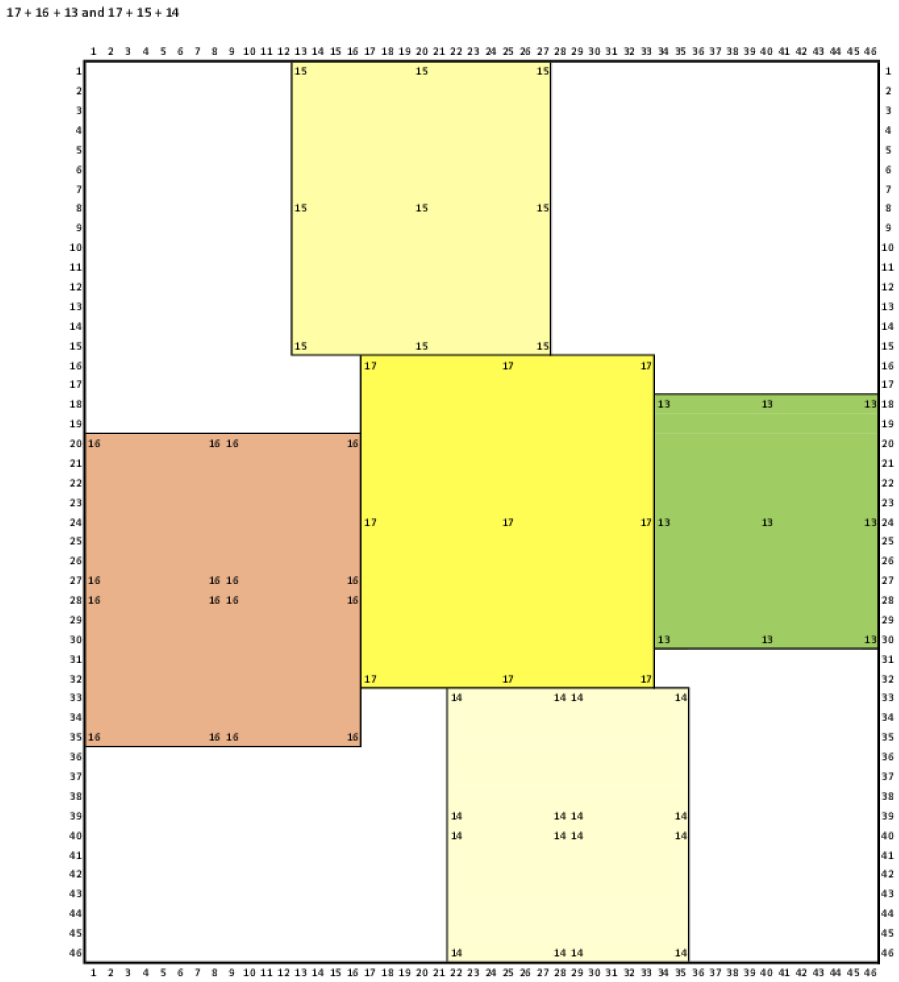


Figure C.3: The special case for $n = 18$.

Here the position of the square in the middle is tightly determined, it cannot be shifted into no direction. It follows that the biggest square of size 18, does not fit into any place, which means that this situation is already excluded.

The remaining cases are substantially more difficult to verify. A purely combinatorial proof that the 46×46 board cannot accommodate the required set of squares would require the exclusion of all possible configurations; the arguments given above constitute only preliminary steps. Moreover, the structural analysis developed for the $n = 24$ case does not transfer directly. Consequently, a fully computer-free proof remains open at present.

Bibliography

[1] Abed, I. A., Ali, M. M., Kadhim, A. A. A.: Using particle swarm optimization to solve test functions problems. *Bulletin of Electrical Engineering and Informatics*, 10(6), pages 3422–3431, 2021.

[2] Abraham Gy., Dosa Gy., Hvattum L. M., **Olaj, T. A.**, Tuza Zs.: The board packing problem. *European Journal of Operational Research*, Volume 308, Issue 3, pages 1056–1073, `https://doi.org/10.1016/j.ejor.2023.01.030`, **D1 journal, IF=6.365**, 2023.

[3] Abraham, Gy., Dósa, Gy., Hvattum, L. M., **Olaj, T. A.**, Tuza, Zs.: Egy téglalap pakolási feladat, és megoldása különféle módszerekkel. *XXXV. Magyar Operációkutatási Konferencia*, Budapesti Corvinus Egyetem, Budapest, június 21.-23, 2023.

[4] Akaria, I., Epstein, L.: An optimal online algorithm for scheduling with general machine cost functions. *J Sched 23*, pages 155–162, 2020.

[5] Allender E., Hellerstein L., McCabe P., Pitassi T., Saks M.: Minimizing DNF formulas and *AC* circuits given a truth table. *Electronic Colloquium on Computational Complexity*, Report No. 126, 22 pages, 2005.

[6] Amaldi E., Capone A., Malucelli F.: Planning UMTS base station location: optimization models with power control and algorithms. *IEEE Transactions on Wireless Communications*, vol. 2, no. 5, pages 939–952, doi: 10.1109/TWC.2003.817438, 2003.

[7] Anderson, S.: Mrs Perkins's quilt. 2020. `http://www.squaring.net/quilts/mrs-perkins-quilts.html`. Accessed: April 10, 2024.

[8] Assmann, J. S. F., Johnson D. S., Kleitman, D. J., Leung, J. Y.-T.: On a dual version of the one-dimensional bin packing problem. *Journal of Algorithms*, Vol. 5, No. 4, 12.1984, pages 502–525, 1984.

[9] Aupperle, L. J., Conn, H. E., Keil, J. M., and O'Rourke, J.: Covering orthogonal polygons with squares. *In 26th Annual Conference on Communication, Control and Computing*, pages 97–106, 1988.

[10] Bálint, V., Adamko, P.: Universal asymptotical results on packing of cubes. Studies of the University of Žilina. *Mathematical Series*, 28, pages 1–4, 2016.

[11] Baeldung: P, NP, NP-Complete and NP-Hard Problems in Computer Science. https://www.baeldung.com/cs/p-np-np-complete-np-hard. Accessed: November 10, 2024.

[12] Balogh, J., Dosa, Gy., Hvattum, L.M., **Olaj, T. A.**, Tuza, Zs.: Guillotine cutting is asymptotically optimal for packing consecutive squares. *Optimization Letters*, 16, pages 2775–2785, doi: 10.1007/s11590-022-01858-w, **Q1 journal, IF=1.502**, 2022.

[13] Balogh, J., Dosa Gy., Hvattum, L. M., **Olaj, T. A.**, Tuza, Zs.: Covering a square with consecutive squares. *MATCOS 2022 conference*, Koper, Slovenia, 2022.

[14] Dosa, Gy., Packing the 1,2,...,18 squares into the smallest square. *Manuscript*, 2024.

[15] Balogh, J., Dósa, Gy., Hvattum, L. M., **Olaj, T. A.**, Szalkai, I., Tuza, Zs.: Covering a square with consecutive squares. *Annals of Operational Research, Springer*, https://doi.org/10.1007/s10479-025-06633-5, **Q1 journal, IF=4.82**, 2025.

[16] Barnhart, C., Johnson, E. L, Nemhauser, G. L., Savelsbergh, M. W. P., Vance, P. H.: Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research, INFORMS*, vol. 46(3), pages 316–329, https://ideas.repec.org/a/inm/oropre/v46y1998i3p316-329.html, June, 1998.

[17] Bereg, S., Cabello, S., Diaz-Banez, J. M., Pérez-Lantero, P., Seara, C., Ventura, I.: The class cover problem with boxes. *Computational Geometry*, 45(7), pages 294–304, 2012.

[18] Berman, O., Drezner, Z., Wesolowsky, G. O.: The expropriation location problem. *Journal of the Operational Research Society*, 54(7), pages 769–776, 2003.

[19] Berman, O., Krass, D.: The generalized maximal covering location problem. *Computers & Operations Research*, 29, pages 563–581, 2002.

[20] Bentsen, H., Hvattum, L. M.: Variable neighborhood search for binary integer programming problems. *International Journal of Metaheuristics (IJMHeur)*, 8, pages 1–26, 2022.

[21] Bitner, J. R., Reingold, E. M.: Backtrack programming techniques. *Commun. 542 ACM*, 18(11), pages 651–656, doi:10.1145/361219.361224, 1975.

[22] Blanquero, R., Carrizosa, E., G.-Tóth, B., Nogales-Gómez, A.: p-facility Huff location problem on networks. *European Journal of Operational Research*, 255(1), pages 34–42, 2016.

[23] Blanquero R., Carrizosa E., G.-Tóth B.: Maximal Covering Location Problems on networks with regional demand. *Omega*, 64(C), pages 77-85, 2016.

[24] Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3), pages 268–308, 2003.

[25] Buchwald, T., Scheithauer, G.: A 5/9 theorem on packing squares into a square. Preprint *MATH-NM-04-2016*, TU Dresden, `http://www.math.tu-dresden.de/~scheith/ABSTRACTS/PREPRINTS/16-5-9-Theorem.pdf`, 2016.

[26] Çakir, O., Weselowsky, G. O.: Planar expropriation problem with non-rigid rectangular facilities. *Computers and Operations Research*, 38, pages 75–89, 2011.

[27] Caprara, A., Monaci, M.: On the two-dimensional Knapsack Problem. *Operations Research Letters*, Volume 32, Issue 1, pages 5–14, 2004.

[28] Chaiken S., Kleitman D. J., Saks M., Shearer J.: Covering regions by rectangles. *SIAM J. Algebraic Discrete Methods*, 2, pages 394–410, 1981.

[29] Chalcraft, A.: Perfect square packings. *Journal of Combinatorial Theory*, Series A 92(2), pages 158–172, 2000.

[30] Church, R., ReVelle, C.: The maximal covering location problem. *Papers of Regional Science Association*, 32, pages 101–18, 1974.

[31] Coffman, Jr., E. G., Garey, M. R., Johnson, D. S., Tarjan, R. E.: Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9, pages 808–826, 1980.

[32] Coffmann, Jr., E. G., Garey, M. R., Johnson D.S.: Approximation algorithms for bin packing: A survey. In: D. Hochbaum (Ed.), *Approximation algorithms for NP-hard problems*, PWS Publishing, Boston, 1997.

[33] Coffman, Jr., E. G., Csirik, J., Galambos, G., Martello, S., Vigo, D.: Bin Packing Approximation Algorithms: Survey and Classification. *Handbook of Combinatorial Optimization*, Reference Work Entry, pages 455-531, Springer New York, 2013.

[34] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C.: Introduction to Algorithms (3rd ed.). *MIT Press*, 2009.

[35] J. Csirik, J. B. G. Frenk, M. Labbé, S. Zhang: Two simple algorithms for bin covering. *Acta Cybernetica*, vol. 14, no. 1, pages 13–25, 1999.

[36] J. Csirik, D. S. Johnson, C. Kenyon. Better approximation algorithms for bin covering. *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '01)*, Washington, D.C., USA: Society for Industrial and Applied Mathematics, pages 557–566, 2001.

[37] Culberson, J. C., Reckhow, R. A.: Covering polygons is hard. *SFCS '88 Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 601–611, October 24–26, 1988.

[38] Czort, S.: The complexity of minimizing disjunctive normal form formulas. *Master's Thesis*, 1999.

[39] Dantzig, G. B.: Maximization of a Linear Function of Variables Subject to Linear Inequalities. In: Koopmans, T.C., Ed., *Activity Analysis of Production and Allocation, Wiley & Chapman-Hall*, New York, London, pages 339–347, 1947.

[40] Dantzig, G. B.: Linear Programming and Extensions. *Princeton Univ. Press*, Princeton, New Jersey, 1963.

[41] Demiröz, B.E., Altınel, K., Akarun, L.: Rectangle blanket problem: Binary integer linear programming formulation and solution algorithms. *European Journal of Operational Research*, 277, pages 62–83, 2019.

[42] Dorigo, M., Gambardella, M.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1), pages 53-–66, 1997.

[43] Dosa, Gy., He, Y.: Scheduling with machine cost and rejection. *Journal of Combinatorial Optimization, Springer*, vol. 12(4), pages 337–350, 2006.

[44] Dosa, Gy., Sgall, J.: First Fit bin packing: A tight analysis. In Proc. of the 30th Ann. Symp. on Theor. *Aspects of Comput. Sci. (STACS2013), LIPIcs*, 3, pages 538–549. Schloss Dagstuhl, 2013.

[45] Dósa, Gy.: Tight results for some classical bin packing algorithms. *University of Pannonia*, Veszprém, 2018. Also in Hungarian: Dósa György, Klasszikus ládapakolási algoritmusokra vonatkozó éles eredmények. *Habilitációs tézisek, Pannon Egyetem*, Veszprém, 2018.

[46] Dosa, Gy., **Olaj, T. A.**: On Square Packing. *Veszprem Optimization Workshop (VOW)*, Veszprem, 2019.

[47] Dosa, Gy., Hvattum, L. M., **Olaj, T. A.**, Tuza, Zs.: The board packing problem: Packing rectangles into a board to maximize profit. In I. Vassányi, editor, *Proceedings of the Pannonian Conference on Advances in Information Technology (PCIT 2020)*, pages 10–16. University of Pannonia, Veszprém, Hungary, 2020.

[48] Dósa, Gy., Kellerer, H., Tuza, Zs., **Olaj, T. A.**: Online scheduling with estimates on the total size. *MATCOS 2019 Middle European Conference on Applied Theoretical Computer Science, conference booklet*, pages 15–15, 2020.

[49] Dósa, Gy., Kellerer, H., **Olaj, T. A.**, Tuza, Zs.: An improved parametric algorithm on two-machine scheduling with given lower and upper bounds for the total processing time. *Theoretical Computer Science*, volume 880, pages 69–81, `https://doi.org/10.1016/j.tcs.2021.06.002`, **Q1 journal, IF=0.827**, 2021.

[50] Drezner, Z.: On the rectangular p-center problem. *Naval Research Logistics*, 34, pages 229–234, 1987.

[51] Dupont, L., Lauras, M., Yugma, C.: Generalized covering location problem with multiple-coverage: Exact and heuristic method. *7th IFAC Conference on Manufacturing Modelling, Management and Control. International Federation of Automatic Control*, June 19–21, pages 442–447, Saint Petersburg, Russia, 2013.

[52] Erdös, P., Graham, R. L.: On packing squares with equal squares. *Journal of Combinatorial Theory*, Series A, Volume 19, Issue 1, pages 119–123, 1975.

[53] Falkenauer, E.: A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics, Kluwer Academic Publishers*, 2(1), pages 5–30, 1996.

[54] Feo , A., Resende, M.: Greedy randomized adaptive search procedures. *Journal. of Global Optimization*, 6, pages 109–133, 1995.

[55] Fowler, R. J., Paterson, M. S., Tanimoto, S. L.: Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12(3), pages 133–137, 1981.

[56] Franzblau, D. S., Kleitman, D. J.: An algorithm for covering polygons with rectangles. *Information and Control*, 63, pages 164–189, 1984.

[57] Friedman, E.: Problem of the Month. 1999, `https://erich-friedman.github.io/mathmagic/1099.html`, Accessed: October 10, 2010.

[58] Gardner, M.: Mathematical games: The problem of Mrs. Perkins' quilt, and answers to last month's puzzles. *Scientific American*, 215(3), pages 264–272, 1966.

[59] Gardner, M.: Mrs. Perkins' quilt and other square-packing problems. *In Mathematical Carnival, New York: Alfred A. Knopf*, pages 139–149, 1975.

[60] Garey, M. R., Johnson, D. S., Stockmeyer, L.: Some simplified *NP*-complete graph problems. *Theoretical Computer Science*, 1(3), pages 237–267, 1976.

[61] Garey, M. R., Johnson, D. S.: Computers and Intractability: A Guide to the Theory of *NP*-Completeness. *W. H. Freeman and Company*, New York, 1979.

[62] Ghaffarian, S., Kerle, N.: Towards post-disaster debris identification for precise damage and recovery assessments from UAV and satellite images. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W13, pages 297–302, 2019.

[63] Gilmore, P. C., & Gomory, R. E.: A linear programming approach to the cutting stock problem. *Operations Research*, 9(6), pages 849–859, `https://doi.org/10.1287/opre.9.6.849`, 1961.

[64] F. Glover, Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5), pages 533–549. doi:10.1016/0305-0548(86)90048-1, 1986.

[65] Gomory, R. E.: Outline of an Algorithm for Integer Solutions to Linear Programs. *Bulletin of the American Mathematical Society*, 64(5), pages 275–278. `https://doi.org/10.1090/S0002-9904-1958-10237-4`, 1958.

[66] Guo, D., Li, X., Lou, W.: Detection and analysis of road information based on optical satellite image. *2021 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*, pages 94–96, 2021.

[67] Györi, E.: A minimax theorem on intervals. *J. Combinatorial Theory Ser.*, B 37(1), pages 1-9, 1984.

[68] Hales, T. C.: A proof of the Kepler conjecture. *Annals of Mathematics*, Second Series, 162 (3), pages 1065–1185, arXiv:math/9811078, doi:10.4007/annals.2005.162.1065, ISSN 0003-486X, MR 2179728, 2005.

[69] Hochbaum, J., Dorit, S., Maass, W.: Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM 32*, pages 130–136, `https://doi.org/10.1145/2455.214106`, 1985.

[70] Holland, J.: Adaptation in Natural and Artificial Systems. *MA : MIT Press*, Cambridge, `https://doi.org/10.7551/mitpress/1090.001.0001`, 1992.

[71] Hvattum, L., Løkketangen, A., Glover F.: Comparisons of commercial MIP solvers and an adaptive memory (Tabu search) procedure for a class of 0-1 integer programming problems. *Algorithmic Operations Research*, Vol. 7, No. 1, pages 13–21, 2012.

[72] Hougardy, S.: On packing squares into a rectangle. *Computational Geometry*, 44(8), pages 456–463, 2011.

[73] Hougardy, S.: A scale invariant algorithm for packing rectangles perfectly. *Proceedings of the fourth International Workshop on Bin Packing and Placement Constraints (BPPC'12)*, `http://www.or.uni-bonn.de/~hougardy/paper/PerfectPacking.pdf`, 2012.

[74] Jang, J., Choi, J., Bae, H.-J., Choi, I.-C.: Image collection planning for KOrea Multi-Purpose SATellite-2. *European Journal of Operational Research*, 230, pages 190–199, 2013.

[75] Januszewski, J., Zielonka, L.: A note on perfect packing of d-dimensional cubes. *Sibirskie Èlektronnye Matematicheskie Izvestiya – Siberian Electronic Mathematical Reports*, 17, pages 1009–1012, 2020.

[76] Johnson, D. S.: Near-optimal bin packing algorithms. *PhD thesis*, MIT, Cambridge, MA, 1973.

[77] Joós, A.: Perfect packing of cubes. *Acta Mathematica Hungarica*, 156(2), pages 375–384, 2018.

[78] Joós, A.: Perfect packing of d-cubes. *Sibirskie Èlektronnye Matematicheskie Izvestiya – Siberian Electronic Mathematical Reports*, 17, pages 853–864, 2020.

[79] Kazanskiy, N., Kuznetsov, M.: The necessary bound of rectangle's square for packing into this any system of five and more than five finite quantity squares with total area 1. *Procedia Engineering*, 201, pages 801–805, 2017.

[80] Kirkpatrick, S., Gelatt Jr, C. D., Vecchi, M. P.: Optimization by Simulated Annealing. *Science*, 220 (4598) : pages 671–680, 1983.

[81] Kleitman, D. J., Krieger, M. M.: An optimal bound for two dimensional bin packing. *In Proceedings of the 16th Annual Symposium on Foundations of Computer Science, IEEE*, Long Beach, pages 163–168, 1975.

[82] Korf, R.E.: Optimal rectangle packing: Initial results. *In Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS 2003)*, pages 287–295, 2003.

[83] Korf, R.E.: Optimal rectangle packing: New results. *In Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 142–149, 2004.

[84] Korf, R. E., Moffitt, M. D., Pollack, M. E.: Optimal rectangle packing. *Ann. Oper. Res.*, 179(1), pages 261–295, `https://doi.org/10.1007/s10479-008-0463-6`, doi:10.1007/S10479-008-0463-6, 2010.

[85] Korte, B., Vygen J.: *J.: Combinatorial Optimization, Theory and Algorithms, Textbook*, Springer Berlin, Heidelberg, 2018.

[86] Laverty, B., Murphy T.: Optimal rectangle packing for the 70 square. *Recreational Mathematics Magazine*, 5(9), pages 5–47, 2018.

[87] Lemke, C. E., & Howson, J. T.: Equilibrium Points of Bimatrix Games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2), pages 413–423, 1964.

[88] Leung, J. Y.-T., Tam, T. W., Wong, C. S., Young, G. H., Chin, F. Y. L.: Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3), pages 271–275, 1990.

[89] Lodi, A., Martello, S., Monaci, M.: Two-dimensional packing problems: A survey. *European Journal of Operational Research*, Volume 141, Issue 2, pages 241–252, 2002.

[90] Luenberger, D. G., Ye, Y.: Linear and Nonlinear Programming. *International Series in Operations Research & Management Science, Springer US*, `https://books.google.no/books?id=-pD62uvi9lgC`, ISBN: 9780387745022, 2008.

[91] Mahapatra, P. R. S., Goswami, P. P., Das, S.: Covering Points by Isothetic Unit Squares. *Proceedings of the 19th Annual Canadian Conference on Computational Geometry*, CCCG 2007, August 20–22, 2007, Carleton University, Ottawa, Canada, 4 pages, 2007.

[92] Maier, H. R., et al.: Introductory overview: Optimization using evolutionary algorithms and other metaheuristics. *Environmental modelling and software*, 114, pages 195–213, 2019.

[93] Maros, I.: Computational Techniques of the Simplex Method. *International Series in Operations Research & Management Science*, Springer New York, NY, `https://doi.org/10.1007/978-1-4615-0257-9,ISBN: 978-1-4020-7332-8`, 2003.

[94] Martinez-Sykora, A., Alvarez-Valdes, R., Bennell, J. A., Ruiz, R., Tamarit, J. M.: Matheuristics for the irregular bin packing problem with free rotations. *European Journal of Operational Research*, 258(2), pages 440–455, 2017.

[95] Masek, W. J.: Some *NP*-complete set covering problems. *Unpublished manuscript*, 1978.

[96] Megiddo, N., Supowit, K. J.: On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13(1), pages 182–196, 1984.

[97] Megiddo, N., Zemels, E., Hakimi, L.: The maximum coverage location problem. *Siam J. Alg. Disc. Math*, 4(2), pages 253–261, 1983.

[98] Mladenovic, N., Hansen, P.: Variable neighborhood Search. *Computers and Operations Research*, 24(11), pages 1097–1100, 1997.

[99] Moffitt, M. D., Pollack, M. E.: Optimal rectangle packing: a meta-CSP approach. *In D. Long, S. F. Smith, D. Borrajo, and L. McCluskey, editors, Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pages 93–102, 2006.

[100] Mondino, E. B., Perotti, L., Piras, M.: High resolution satellite images for archeological applications: the Karima case study (Nubia region, Sudan). *European Journal of Remote Sensing*, 45(1), pages 243–259, 2012.

[101] Moon, J. W., Moser, L.: Some packing and covering theorems. *Colloquium Mathematicae*, 17(1), pages 103–110, 1967.

[102] Mukherjee, M., Chakraborty, K.: A polynomial-time optimization algorithm for a rectilinear pratitioning problem with application in VLSI design automation. *Information Processing Letters*, 83, pages 41–48, 2002.

[103] Novotný, P.: On packing of squares into a rectangle. *Archivum Mathematicum*, (Brno) 32, pages 75–83, 1996.

[104] Novotný, P.: On packing of four and five squares into a rectangle. *Note di Matematica*, 19, pages 199–206, 1999.

[105] Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, 31(12), pages 1985–2002, 2004.

[106] ReVelle, C.S., Eiselt H.A., Daskin M.S.: A bibliography for some fundamental problem categories in discrete location science. *European Journal of Operational Research*, 184, pages 817–848, 2008.

[107] Resende, M. G.: Computing Approximate Solutions of the Maximum Covering Problem with GRASP. *Journal of Heuristics*, 4, pages 161–177, 1998.

[108] Roos, C., Terlaky, T., Vial, J.-Ph.: Theory and algorithms for linear optimization - an interior point approach. *Wiley-Interscience series in discrete mathematics and optimization*, 1998.

[109] Scheithauer, G.: Introduction to Cutting and Packing Optimization: Problems, Modeling Approaches. *Solution Methods (International Series in Operations Research & Management Science*, 263, Springer, 2018.

[110] Sedliačková, Z., Adamko, P.: Packing three cubes in d-dimensional space. *Mathematics*, 9, 2046, 9 pages, doi:10.3390/math9172046, 2021.

[111] Sgall J., Balogh J., Békési J., Dósa Gy., Hvattum L. M., Tuza Zs.: No Tiling of the 70*70 Square with Consecutive Squares. *In 12th International Conference on Fun with Algorithms (FUN 2024). Leibniz International Proceedings in Informatics (LIPIcs)*, volume 291, pages 28:1–28:16, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, `https://doi.org/10.4230/LIPIcs.FUN.2024.28`, 2024.

[112] Simonis, H., O'Sullivan, B.: Search strategies for rectangle packing. *In P.J. Stuckey, editor, Constraint Programming, 2008, volume 5202 of Lecture Notes in Computer Science*, pages 52–66,2008.

[113] The on-line encyclopedia of integer sequences, sequence a005842, `http://oeis.org`, Accessed: March 26, 2021.

[114] Trevor Hale's Location Science References, `http://gator.uhd.edu/~halet/`, Accessed: February 10, 2020.

[115] Ullman, J. D.: The performance of a memory allocation algorithm. *Technical Report 100*, Princeton Univ., Princeton, NJ, 1971.

[116] Vidal, T.: Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research*, 140, pages 105643-105643. https://doi.org/10.1016/j.cor.2021.105643, 2022.

[117] Vidal, T., Crainic, T. G., Gendreau, M., Prins, C.: A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234, pages 658–673, 2014.

[118] Watson, G.: The problem of the square pyramid. *Messenger of Mathematics, New Series*, 48, pages 1–22, 1918.

[119] Wäscher, G., Haußner, H., Schumann, H.: An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3), pages 1109–1130, 2007.

[120] Y. Xiong, J. Gan, B. An, C. Miao, A. L. C. Bazzan: Optimal Electric Vehicle Charging Station Placement. *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*, AAAI Press, pages 2662–2668, `https://www.ijcai.org/Proceedings/15/Papers/377.pdf`, 2015.

[121] Zhou, Z., Das, S., Gupta, H.: Connected k-coverage problem in sensor networks. *Proceedings. 13th International Conference on Computer Communications and Networks (IEEE Cat. No.04EX969)*, Chicago, IL, USA, 2004, pages 373–378, doi: 10.1109/ICCCN.2004.1401672, 2004.

[122] Zong, C.: Packing, covering and tiling in two-dimensional spaces. *Expositiones Mathematicae*, Volume 32, Issue 4, pages 297–364, ISSN 0723-0869, `https://doi.org/10.1016/j.exmath.2013.12.002`, 2014.