SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Kakurasu is NP-complete

av

Morris Lundberg Allerholm

2025 - No K9

Kakurasu is NP-complete

Morris Lundberg Allerholm

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Per Alexandersson

2025

Abstract

English

NP-complete puzzle games demonstrate the intriguing complexity that can arise from the combination of simple rules, and allow for non-mathematicians to interface with one of the largest unsolved questions in mathematics: P versus NP. In this thesis, we introduce the concept of NP-completeness in the context of other, similar types of problems. We then show that the puzzle game *Kakurasu* is NP-complete.

Svenska

NP-kompletta pusselspel visar på den fängslande komplexitet som kan uppstå genom kombinationen av enkla regler, och tillåter på så vis icke-matematiker att interagera med en av de största olösta frågorna inom matematiken: P kontra NP. I den här uppsatsen introducerar vi konceptet NP-kompletthet i kontext av andra, liknande problem. Vi visar sedan att pusselspelet *Kakurasu* är NP-komplett.

Contents

Acknowledgement

Heartfelt thanks to Per Alexandersson for your guidance. You have made the process of writing this thesis feel easier than expected. Thanks to your pertinent recommendations the right papers have always appeared at the right time. This thesis, and proof, would not have been completed without your help.

The large language model *ChatGPT* has been used for assistance with LaTeX, and the creation of images using the package *Tikz*.

1 Introduction and description of *Kakurasu*

1.1 Introduction

Puzzle games are widely popular, and for good reason; they let the player experience the stimulating struggle of figuring out, and later on the satisfying release of tension when completing the game. For a puzzle game to give this experience it must be sufficiently hard, but not too hard. Many puzzle games that fit this description, such as *Sudoku*, *Minesweeper* and *Kakuro*, are recognized as belonging to a certain group of problems: *NP-complete* problems (NP is short for nondeterministic polynomial time) [YS03, Kay00, Tak01].

It seems reasonable then that the mathematical definition of this kind of problem places them between types of problems traditionally considered easy and types of problems traditionally considered hard. As a matter of fact it is still unknown if NP-complete problems fall into the category of easier problems, such as sorting and simple mathematical calculation, or if NP-complete problems should rather be considered hard in the same way that chess is. The uncertainty preserves a kind of mystique around these puzzle games: they seem to be both easy and hard at the same time.

Take a challenging instance of *Sudoku*, for example. It is usually not too difficult to find a few numbers, but at a certain stage multiple possibilities reveal themselves, and it is then no longer certain which number that should go where. At this crossroad the player must, in their mind, complete the different possibilities of the puzzle to eliminate the contradictory dead ends, and thus find the way forward. This is no easy task, and yet the rules are simple.

In this thesis we give an informal, intuitive description of NP-completeness in the context of famous puzzle games and problems. We also describe a typical method of proving NP-completeness: *reduction*. The method relies on reducing an already known NP-complete problem to the problem that is to be shown to be NP-complete. Thus many NP-complete problems are linked to each other. To list the problems mentioned in this thesis, and show how they are related by proofs via reduction, we include the following illustration in Figure 1.

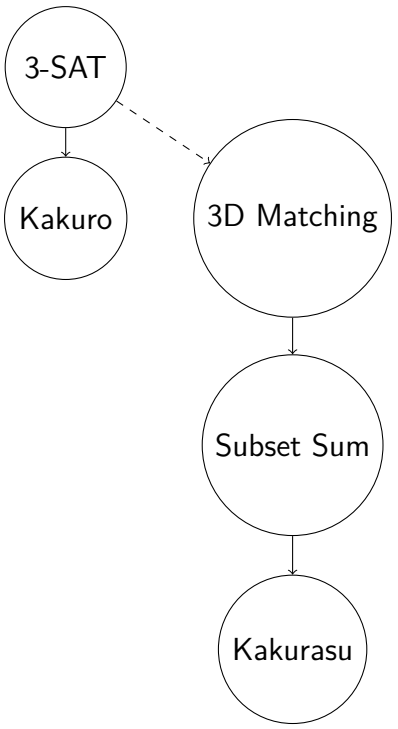


Figure 1: An illustration of the problems mentioned in this thesis and of how they are related via proofs by reduction. The solid lines represent connections we describe, while the dotted line represents a connection that is not mentioned in detail [Kar75].

As seen in Figure 1 we make a reduction to the puzzle game *Kakurasu*, and show that it is NP-complete. To the best of our knowledge, this is a new result.

We also include two additional short descriptions of reduction proofs in the appendix, in Section 4. Both are from 3-SAT to puzzle games: *Minesweeper* and *LaserTank*.

1.2 Description of the game *Kakurasu*

Kakurasu is played on a square $n \times n$ grid. The grid thus consists of n^2 entries, each indexed according to its row and column. A 7×7 example of this can be seen in Figure 2, with the indexing outlined along the left and top edges.

The player interacts by deciding which tiles in the grid are to be filled in, and which are to be left blank. If a tile is filled in, it contributes to the sum along its row by the index of its column, and similarly to the sum along its column by the index of

its row. To solve the puzzle is to fill in all the necessary tiles such that the sums along all rows and columns agree with the indicated numbers, marked $r_1, ..., r_7$ and $c_1, ..., c_7$ in Figure 2.

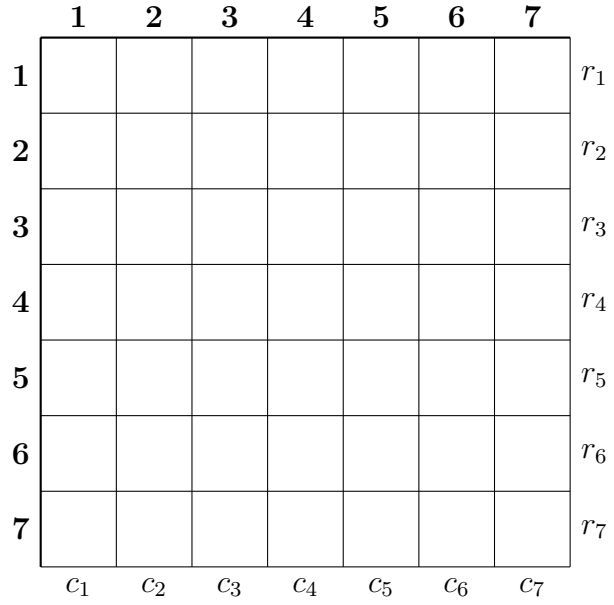


Figure 2: A 7×7 *Kakurasu* grid with row- and column sums marked with $r_1, ..., r_7$ and $c_1, ..., c_7$.

To further illustrate how the puzzle is solved, a short example is provided. Figure 3 displays an empty 4×4 grid with given row- and column sums.

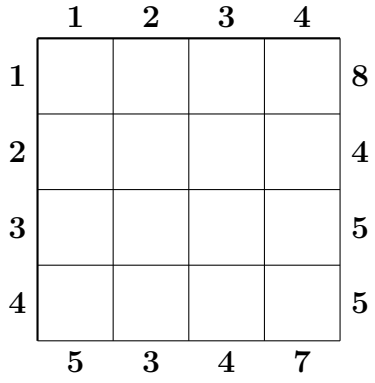
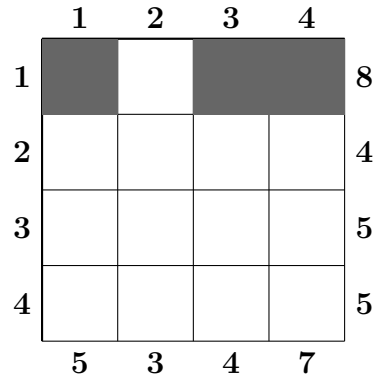


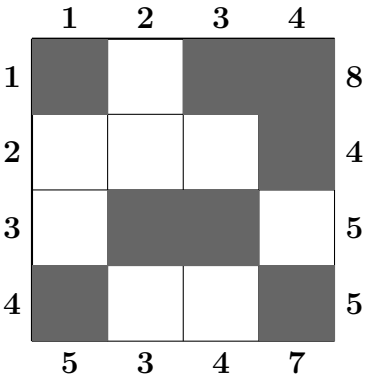
Figure 3: A 4×4 *Kakurasu* grid with given sums for rows and columns.

To solve the puzzle, we note that all row- and column sums can be expressed in multiple ways, except for the first row, which has 8 as its designated sum. With the

numbers 1 to 4 in a sum, 8 is uniquely represented as $1 + 3 + 4 = 8$. So we fill in these squares, as seen in Figure 4a.



(a) The first step in the solution.



(b) The finished puzzle.

Figure 4: An example of how to solve a simple Kakurasu puzzle in two steps.

With the first row completed, all the column sums become uniquely determined, so we fill these in as well. This results in the finished puzzle, which can be seen in Figure 4b. To check the solution, all the sums are verified.

2 Theory and earlier results

2.1 NP-completeness

This section gives a short, intuitive introduction to the concept of NP-completeness, and how the class of NP-complete problems relates to similar classes of problems. The entirety of this section, and of Section 2.2, is based on *Introduction to Algorithms* by Cormen et al. [CLRS22] (specifically Chapter 34). Note that we do not describe a formal, theoretical foundation of NP-completeness, which is beyond the scope of this thesis, and not necessary to understand the proof.

When discussing NP-completeness we are only interested in a specific form of problem, called *decision problem*. Decision problems are solved by correctly answering "yes" or "no". Most problems are not of this form originally, but can be transformed by asking the right questions. When considering *Kakurasu* in this context, the interesting question is not "Which tiles are to be filled in?" Instead it is "Is this instance of *Kakurasu* solvable?"

Another concept necessary to understand NP-completeness is *polynomial time*. In this context polynomial time refers to the maximum amount of time it can take for an algorithm to complete its task, if that time scales proportionally to some polynomial function of the number of inputs.

The number of inputs required to specify a problem instance is called the size of the problem. This is the amount of information that differs between instances of the problem. In the case of *Kakurasu* the only things that vary are the row and column restraints, so for a $m \times m$ grid the number of inputs, or size, is $2m$.

So if we call the number of inputs n, a problem is solvable in polynomial time if there exists an algorithm whose maximum solution time scales with a polynomial function of n with degree k, for some constant k. Usually the class of problems which are solvable in polynomial time is denoted P. In general, these problems are regarded as being easy to solve computationally.

The class NP, on the other hand, denotes the set of problems whose *solutions* are verifiable in polynomial time. This means that any proposed solution can be verified

in polynomial time. As a consequence of this definition, P is a subset of NP, which is illustrated in Figure 5. It is important to note that the solution which is verified is not a solution to the decision problem, i.e. an answer of the form "yes" or "no", but a specific solution to the puzzle or problem at hand, for example a proposed tiling of a *Kakurasu* puzzle.

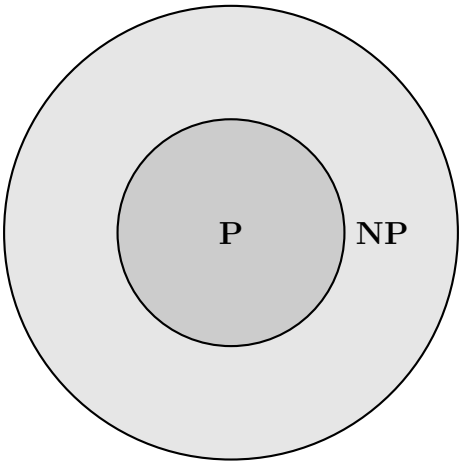


Figure 5: An illustration of the two sets of problems: P contained in NP.

It is still unknown if P is a proper subset of NP, which would mean that there are problems that are in NP but not in P, or if the two sets are equal. In the latter case the lightly shaded region in Figure 5 would not exist.

Due to the uncertainty regarding the equality of P and NP, an interesting subset of NP is the set of *NP-complete problems*, denoted NPC. This set consists of all problems in NP which are as hard as any other problem in NP.

Because these problems are at least as hard as any problem in this class (which is also true when comparing two NP-complete problems; they are at least as hard as each other), if there were to be a polynomial time solution algorithm for a NP-complete problem, then there must be a polynomial time solution algorithm for all problems in NP. Otherwise there would be some problem harder than the NP-complete problem in NP, contradicting the definition of NP-completeness. Finding such a polynomial time solution algorithm thus constitutes a proof of P and NP being the same class of problems.

On the other hand, if one could prove the impossibility of such a polynomial time

solution algorithm for a specific NP-complete problem, then it would follow that P is a proper subset of NP and that the two classes are not equal.

The difference between problems which are in NPC and problems which are not in NPC is often minute. An example of this is the difference between two types of problems: 2-SAT and 3-SAT. The general form of the problem is described by Biere et al. [BHvM09].

A 2-SAT problem considers a logical formula consisting of however many disjunctions with 2 logical variables. Each variable may or may not be negated. So that each disjunction is of the form: $([\neg]x_i \vee [\neg]x_j)$ (where "$[\neg]$" represents a possible negation). These disjunctions are combined with conjunctions, creating a larger formula:

$$([\neg]x_i \vee [\neg]x_j) \wedge ([\neg]x_k \vee [\neg]x_l) \wedge ([\neg]x_m \vee [\neg]x_n) \wedge ...$$

It is important to note that one logical variable may occur multiple times.

The general problem of deciding if there is an assignment of truth values to the logical variables such that the whole formula is true is called 2-SAT, and is in P. That is, the decision problem of whether or not such a formula is satisfiable is solvable in polynomial time.

The following formula is an example of a 2-SAT problem:

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_4 \vee x_5) \wedge (x_1 \vee x_6).$$

On the other hand 3-SAT is almost the exact same sort problem, but with disjunctions that contain 3 logical variables. Its formulas look like this:

$$([\neg]x_i \vee [\neg]x_j \vee [\neg]x_k) \wedge ([\neg]x_l \vee [\neg]x_m \vee [\neg]x_n) \wedge ...$$

An example would thus be

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_5 \vee x_6 \vee \neg x_7).$$

3-SAT is an NP-complete problem, as shown by A. Cook [Coo23], and thus at least as hard as any other problem in NP. This illustrates how subtle the difference can be between problems in the different classes.

2.2 Method of proof: reduction

The fact that 3-SAT is NP-complete is key for many of the proofs that demonstrate that other specific problems are NP-complete. This is because of a method called "reduction" which is used to show that *Kakurasu* is NP-complete in this project.

A *reduction* involves making a sort of translation from one problem to another. If we are able, for any instance of a problem which is known to be NP-complete, call it A_{NPC}, to translate it to an instance of another problem, B, such that the answer to the translation is "yes" if and only if the answer to the original instance of A_{NPC} is "yes", then we have demonstrated that the complexity of A_{NPC} is in some way inherent in B.

There are restrictions imposed on this translation: it must be performable in polynomial time. Would the translation require greater than polynomial time there would be no way of guaranteeing that it actually transports the property of NP-completeness. So by virtue of this polynomial time restriction, the method ensures that if there is no polynomial time solution for A_{NPC}, then we can guarantee that there is no polynomial time solution algorithm for B. Conversely, we can guarantee that there is a polynomial time solution algorithm for A_{NPC} if there is a polynomial time solution algorithm for B.

This step demonstrates that the other problem, which we called B, is also NP-hard: as hard as any other problem in NP. What remains then is to show that B is a member of NP, by checking that any proposed solution for B can be verified in polynomial time.

In summary, such proofs consist of two steps: a polynomial time reduction, and showing that the problem is in NP.

The reason why 3-SAT is especially interesting is because it serves as the problem we denoted by A_{NPC} in many such proofs. The proof in this project relies instead

on a problem called subset-sum.

As an illustration of how reductions may look, we provide a description of a proof that demonstrates the NP-completeness of a puzzle game similar to *Kakurasu*, namely *Kakuro*.

2.3 Proof of *Kakuro´*s NP-completeness

Kakurasu is in many ways similar to the puzzle game *Kakuro* (or *Cross Sum*). They both involve summing numbers along rows and columns, the difference lies in *Kakurasu* being on an $n \times n$ grid, with the orders of the numbers being predetermined, and the player choosing which of the numbers that need to be present for the sums to be correct. *Kakuro*, on the other hand, exists on a irregular grid, created by interconnecting rows and columns of different sizes haphazardly, and allows the placing of numbers from 1 to 9 on whichever square, as long as that number does not already exist on the same row or column. To solve the puzzle, the numbers on each row and column must add up to the given number at the end of the row or column. All boxes must be filled in *Kakuro*. In *Kakurasu*, on the other hand, the game is about determining which boxes are to be filled.

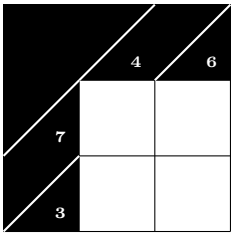


Figure 6: A simple instance of the puzzle *Kakuro*.

To illustrate how the game *Kakuro* works we display a simple example of it in Figure 6. The goal is, as mentioned, to fill in all the squares with numbers from 1 to 9 such that the sums along both rows and columns add up to the predetermined numbers, and such that all digits along each row and column are unique.

The first step is to note that 4 can only be represented by the sum $1 + 3 = 4$ (because 2 would require another 2, breaking the duplicate rule), and that the 3 must be placed on the above of the two available squares in the 4-column, since

placing it on the bottom row would make it impossible to fill the other tile on that row. Once the first column is completed we can fill in the other based on the row sums, and check that the solution is correct by verifying all sums. The solved puzzle can be seen in Figure 7

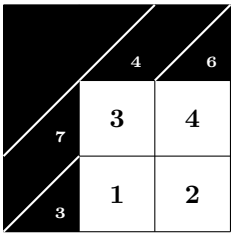


Figure 7: The puzzle from Figure 6, solved.

Kakuro was first shown to be NP-complete by Takahiro [Tak01]. This original proof made a reduction in two steps, from 3-SAT to an intermediate problem, and then from this intermediate problem to *Kakuro*.

A later proof of *Kakuro´*s NP-completeness, by Ruepp et al. [RH10], accomplishes this reduction in one step, from planar 3-SAT to *Kakuro*. Planar 3-SAT is similar to 3-SAT; it describes 3-SAT as a sort of graph. Because of this later proof´s relative simplicity we choose to describe it instead of the original proof.

The proof by Ruepp et al. relies on so called *gadgets* to describe planar 3-SAT in *Kakuro*. Creating gadgets is a common strategy when making a reduction from a NP-complete logical problem to the problem which NP-completeness is to be proved. The gadgets, which are constructed in the latter problem (*Kakuro* in this case) function like the operators and logical variables in the logical problem. By having a gadget for each operator and variable it is possible to formulate any instance of the logical problem in the latter problem, that is to be shown to be NP-complete. This constitutes a reduction.

Most of the gadgets were created manually, but for difficult gadgets the authors had the creative idea of using a *Kakuro* solver. The solver was created by transforming instances of *Kakuro* to instances of SAT-problems, then solved by a program that solves SAT-problems. The computer program that is used is Minisat [ES03]. This *Kakuro* solver is in turn used to test which constructions have the properties that they are looking for.

The main idea of the gadgets in this proof is that they transport the values 1 and 2 (and in rare cases other values) and that they can have two results, true or false, depending on the input(s) of 1 or 2. One might compare this to electronic circuitry.

The gadgets created can thus be combined to formulate any planar 3-SAT problem directly in the *Kakuro* environment, such that answering the question if there is a solution amounts to solving a planar 3-SAT-problem.

For the interested reader we have included two other short descriptions of proofs of NP-completeness, that also rely on gadgets. They are included in the appendix, in Section 4.

2.4 3-dimensional matching

As a prerequisite for the discussion regarding subset-sum being NP-complete we give a description of a known NP-complete problem called 3-dimensional matching, described by Garey et al. [GJ02]. Given three sets, X, Y, Z, with $|X| = |Y| = |Z| = n$, and a subset $T \subseteq X \times Y \times Z$ with $|T| = m$. For example, if $x \in X$, $y \in Y$ and $z \in Z$ then $(x, y, z) \in X \times Y \times Z$, and T consists of such ordered triples, but not necessarily every such ordered triple.

The question regards a type of subsets of T, which are called 3-dimensional matchings. We denote such a subset with M. The requirements for $M \subseteq T$ to be a 3-dimensional matching are that for every pair of ordered triples, (x_1, y_1, z_1) and (x_2, y_2, z_2) in M, we have that the one element from each respective set do not equal the other element from the same set: $x_1 \neq x_2$, $y_1 \neq y_2$ and $z_1 \neq z_2$.

Nowadays the decision problem usually asks if it is possible to find M such that $|M| \geq r$, for a given integer r. We are going to use the original version (in the context of NP-completeness), as formulated by Karp [Kar75], which requires $|M| \geq n$, where $n = |X| = |Y| = |Z|$. The rest of the criterion for the elements in M enforce $|M| \leq n$, i.e. we require $|M| = n$. This version of the problem is called perfect 3-dimensional matching [Wik24], and requires the 3-dimensional matching to cover every element in X, Y and Z.

2.5 Subset-sum problem with natural numbers

Subset-sum is one of the first problems to be shown to be NP-complete. This was done by Karp [Kar75], approximately a year after the concept of NP-completeness was first introduced by Cook [Coo23]. Originally the problem was called the knapsack problem. Today the knapsack problem generally refers to a broader type of problem, which the subset-sum problem is a specific version of.

The original formulation of the subset-sum problem describes a set, $I = \{z_1, z_2, ..., z_n\}$, with n members, $z_1, z_2, ..., z_n \in \mathbb{Z}$, and a target value, S. The question that creates the associated decision problem is: "Is there a subset $I' \subseteq I$ such that $\sum_{i \in I'} i = S$?" or more informally "Is there a subset of the set of integers, I, such that the sum of all the members in that subset is equal to S?"

Because of the fact that *Kakurasu* only contains natural numbers we only focus on a different version of the subset-sum problem, in which the set of numbers contains only natural numbers. The version of subset-sum with only natural numbers is also NP-complete, which is shown in *Algorithm Design* by Kleinberg and Tardos [KT06]. We give a brief summary of their proof, which is a direct reduction from the NP-complete problem of 3-dimensional matching to subset-sum.

Consider a set of natural numbers $\{w_1, ..., w_n\}$ and a value W which a subset is supposed to sum to. There is a solution to the problem if there exists such a subset: $\{w_{i_1}, ..., w_{i_k}\}$. The decision problem is thus created by asking the question: "Is there a subset of $\{w_1, ..., w_n\}$ such that the sum of its members is equal to W?"

Theorem: Subset-sum with natural numbers is NP-complete.

Proof: (A summary of the proof by Kleinberg and Tardos [KT06])
To see that this problem is in NP we note that the verification only requires the numbers in the suggested subset to be summed, which can be done in polynomial time.

For the reduction we take an instance of 3-dimensional matching, as described in Section 2.4, with X, Y, Z all with cardinality n, a set T which is a subset of $X \times Y \times Z$, with cardinality m.

The first step is the construction of a special type of number that can represent any ordered triple $t = (x_i, y_j, z_k) \in X \times Y \times Z$. The number, denoted by w_t, consists of $3n$ digits, the first n digits in it each correspond to one unique element in X, the middle digits, in positions numbering $n+1$ to $2n$, each correspond to one element in Y, and the final digits, with positions numbering $2n+1$ to $3n$, each correspond to one element in Z. Then we can represent t by such a number, w_t, with 0 in each position except for a 1 in the positions corresponding to the elements in the ordered triple.

So for any ordered triple (x_i, y_j, z_k) we can represent it as $w_t = d^i + d^{n+j} + d^{2n+k}$, with the base $d = m + 1$. This is to make the operations addition and union correspond to each other, so that if we sum multiple w_t there is no risk of too many occurrences of a certain element causing carry over to the next digit.

Now, given an instance of the perfect 3-dimensional matching problem, we can create a corresponding subset-sum problem.

The first step is to create a corresponding w_t, as was detailed above, for every $t = (x_i, y_j, z_k) \in T$. We also define a new number $W = \sum_{i=0}^{3n-1}(m+1)^i$, which, because of us working in the base $m+1$, is the number with $3n$ digits, all of which are 1. The corresponding subset-sum problem is then to find a subset of $\{w_{t_1}, ..., w_{t_m}\}$ such that the sum of that subset is equal to W.

To see that this problem has a solution if and only if the corresponding perfect 3-dimensional matching problem has a solution, note that by summing different w_{t_i} the sum can only have a 1 in a certain position if that 1 occurs uniquely, in exactly one w_{t_i}, due to the base being $m+1$ and $|T| = m$. So for some subset to sum to W is equivalent to there being a solution for the perfect 3-dimensional matching problem. Thus we have that the subset-sum problem with natural numbers is NP-complete. □

With this completed we are ready for the result of this thesis.

3 *Kakurasu* is NP-complete

In this section we describe the main result of this thesis.

Before we describe the theorem and proof of *Kakurasu* being NP-complete, we define a useful notational trick for *Kakurasu*. The rules of *Kakurasu* allow for the row and column limits to be set to 0. Consider for example the following *Kakurasu* puzzle:

Since a row or column limit of 0 directly inhibits the filling of a tile in that row or column we might as well consider the puzzle without these rows and columns. So the *Kakurasu* puzzle to the left in Figure 8 can, without loss of information, be represented as the right puzzle in Figure 8.

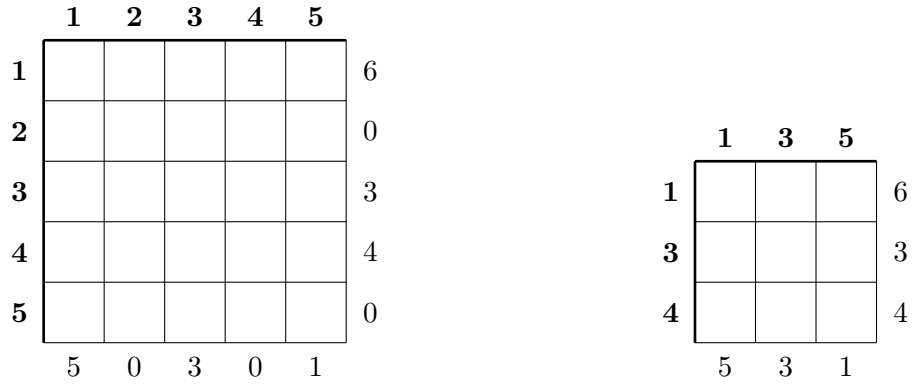


Figure 8: **Left**: a 5×5 *Kakurasu* puzzle with some row- and column sums set to 0. **Right**: The reduced puzzle.

The setting of row- or column sums to 0 may be useful, because it prohibits the use of a specific number in all columns or all rows respectively. In turn this makes it possible to create a *Kakurasu* puzzle that only has a specific set of positive integers as the tiles' values, as opposed to the standard tile values of $1, 2, 3, ...$

So this notational trick is useful because it allows for more concise representation of such restricted versions of *Kakurasu*. Note that this also allows for rectangular grids, since a different number of row- and column sums may be set to 0.

3.1 Result

Theorem: *Kakurasu* is NP-complete.

Proof:
The first step is to confirm that *Kakurasu* is in NP. For this to be the case the verification of a proposed solution for a *Kakurasu* puzzle must be possible to perform in polynomial time. All that is required for the verification is to sum along all rows and columns. This can be achieved in polynomial time, and thus *Kakurasu* is in NP.

Regarding the size of *Kakurasu*, for a $k \times k$ grid the Kakurasu puzzle is of size $O(k)$. We note that it is comparable to the size of the subset-sum problem, which is $|\{n_1, ..., n_k\}| + 1 = k + 1$ for the set $\{n_1, ..., n_k\}$ of positive integers.

The reduction is made from the subset-sum problem with positive integers to *Kakurasu*. The subset-sum problem with positive integers consists of deciding if there is a subset of a set of positive integers, S, such that the sum of that subset equals a certain value, T.

So for any $S = \{n_1, ..., n_k\}$ and T, where $n_1, ..., n_k, T \in \mathbb{Z}^+$, we need a polynomial-time algorithm that translates the subset-sum problem to a *Kakurasu* problem, with the property that it has a solution if and only if the subset-sum problem has one.

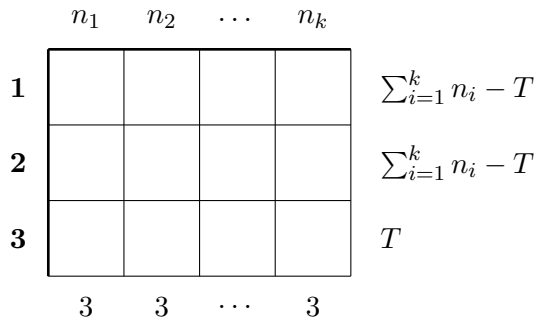


Figure 9: A representation of the general *Kakurasu* translation of a subset-sum problem

Consider the $3 \times k$ *Kakurasu* puzzle, as seen in Figure 9. It is reduced from a $n_k \times n_k$ grid with all row sum limits, except for the first three ones, set to 0, and all column limits for columns with indices $i \notin S$ set to 0. All remaining column sums are set to three. The row sums for the first and second row are the same, namely $\sum_{i=1}^{k} n_i - T$,

while the row sum of the third row is T.

This setup is a reduction of subset-sum to *Kakurasu*, and the reduction is easily done in polynomial time.

To understand why this reduction works, note that for each column either the first two tiles are filled in, or the third tile is filled in. So to solve the puzzle is to decide if each n_i, for $i = 1, ..., k$, should contribute to T or $\sum_{i=1}^{k} n_i - T$, which is the same as deciding if it should be included in the subset, which is supposed to sum to T, or if it is to be excluded, and thus included in the other, implied subset that sums to the remainder of $\sum_{i=1}^{k} n_i - T$.

It follows directly that this puzzle is solvable if and only if there is a subset of S whose integers sum to T, if there is no such subset then it is impossible to fill out the third row such that it satisfies the sum requirement, and by extension it is then also impossible to satisfy the requirements on the first and second row. □

To further illustrate the mechanics of the proof we translate two similar subset-sum problems, one which is solvable, and another which is unsolvable. Take the set $\{3, 5, 10, 12, 13, 17\}$ and the target sum 21. This subset-sum problem is solvable because $3 + 5 + 13 = 21$.

We use the translation method from the proof, which only requires the calculation $3 + 5 + 10 + 12 + 13 + 17 - 21 = 39$, as the sum limit for row 1 and 2. This allows us to reduce a 17×17 grid into a 3×6 grid with the appropriate limits, see the left grid in Figure 10. To the right, in Figure 10, we include the solution to the puzzle.

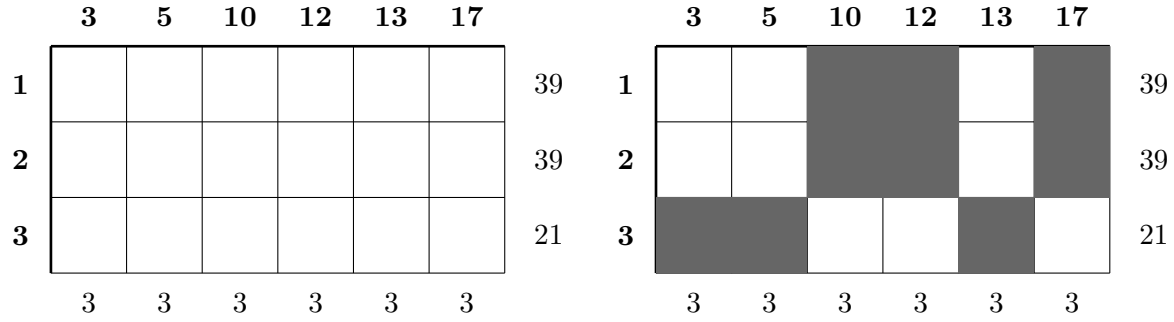


Figure 10: **Left**: the subset-sum problem translated to a *Kakurasu* puzzle. **Right**: its solution.

Consider the similar subset-sum problem with the set $\{3, 5, 10, 12, 14, 17\}$ and the target sum of 21. This problem has no solution. Again we use the translation method, calculating $3 + 5 + 10 + 12 + 14 + 17 - 21 = 40$, and rendering the puzzle, as seen in Figure 11.

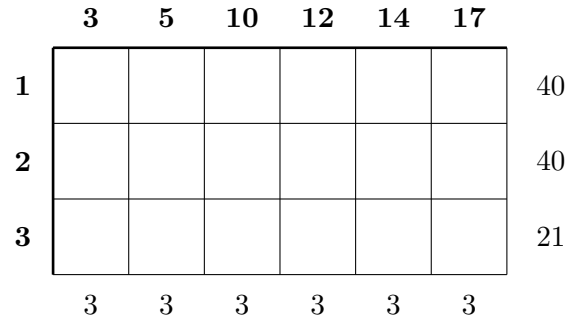


Figure 11: Another subset-sum problem translated to a *Kakurasu* puzzle.

This puzzle contains the aforementioned subset-sum problem in the third row. As anyone who attempts to solve this puzzle will therefore come to realize, it is as impossible as the subset-sum problem that generated it.

4 Appendix

4.1 Minsesweeper is NP-complete

The classic computer game *Minesweeper* is shown to be NP-complete by Kaye [Kay00]. His proof is a reduction from 3-SAT, which he represents with logical circuitry to closer imitate the planar grid of *Minesweeper*.

Before the proof he includes an instructive discussion on the topic of NP-completeness. We recommend this to the reader who has further interest in this topic; it is readable without much prior knowledge and yet it manages to divulge ideas related to the theoretical framework of the algorithms and computers that are presupposed in this thesis.

Minesweeper is played by revealing squares on a grid. If the player reveals a square that hides a mine it looses. All revealed, non-mine squares display the number of mines in the 8 adjacent squares. This is the information that is used to unravel the puzzle square by square without revealing a mine. As an assisting function the player can mark unrevealed squares which are sure to contain mines.

The decision problem he describes supposes a given a state of minesweeper, which is partially completed – meaning that some cells have already been revealed and that some mines may be marked. The yes/no-question arises by asking whether some structure of mines in the unrevealed cells can satisfy the given situation, i.e. the numbering of adjacent mines in the revealed squares. If it is the case that such a structure can exist the answer is yes. On the other hand, if there is no possible placing of mines in the unrevealed cells that could satisfy the given revealed cells and their information, then the answer is no.

Like the proof of *Kakuro* being NP-complete, in Section 2.3, this proof relies on the construction of gadgets. An interesting aspect of the proof is that it only creates gadgets for conjunction and negation, and then use the logical properties of the two operators to create the other gadgets.

We have that $P \vee Q \equiv \neg(\neg P \wedge \neg Q)$. So that it suffices to have those two gadgets to create the third.

The combination of the gadgets poses a problem for Kaye. Since he represents the logic in circuitry and circuitry has the ability to cross itself, while the gadgets cannot lay on top of each other, he comes up with a solution for combining gadgets in a way that is equivalent to circuits crossing each other. With this completed he has demonstrated that it is possible to create gadgets with all the necessary properties to recreate any instance of 3-SAT in minesweeper. The proof is thus complete.

4.2 LaserTank is NP-complete

Alexandersson, who is the supervisor of this thesis, and a collaborator [AR19], have written another proof of NP-completeness. This regards the puzzle game *LaserTank*, which involves moving a tank on a grid and shooting in a straight line, which may be manipulated by obstacles, to hit a a target.

An interesting aspect of this proof is that it originally considers a restricted version of the game, and then shows that the proof can be extended to the non-restricted version. Other than that the proof works in a similar way to the *Kakuro* proof we have discussed in Section 2.3, by constructing gadgets that represent literals and logic connections in the environment. This is done in such a way that the 3-SAT problem can be formulated in *LaserTank*, and thus shows that *LaserTank* is NP-complete.

References

[AR19] Per Alexandersson and Petter Restadh. Lasertank is np-complete. In *International Conference on Mathematical Aspects of Computer and Information Sciences*, pages 333–338. Springer, 2019.

[BHvM09] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

[CLRS22] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[Coo23] Stephen A Cook. The complexity of theorem-proving procedures. In *Logic, automata, and computational complexity: The works of Stephen A. Cook*, pages 143–152. 2023.

[ES03] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

[GJ02] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[Kar75] Richard M Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975.

[Kay00] Richard Kaye. Minesweeper is np-complete. *The Mathematical Intelligencer*, 22(2):9–15, 2000.

[KT06] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.

[RH10] Oliver Ruepp and Markus Holzer. The computational complexity of the kakuro puzzle, revisited. In *International Conference on Fun with Algorithms*, pages 319–330. Springer, 2010.

[Tak01] S Takahiro. The complexity of puzzles, cross sum and their another solution problems (asp). *Ph. D. Dissertation*, 2001.

[Wik24] Wikipedia contributors. 3-dimensional matching — Wikipedia, the free encyclopedia, 2024. [Online; accessed 6-May-2025]. URL: https://en.wikipedia.org/w/index.php?title=3-dimensional_matching&oldid=1261233801.

[YS03] Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 86(5):1052–1060, 2003.