SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics,

Verfication of selected NP-hard Problems

Zixuan Fan

SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics,

Verfication of selected NP-hard Problems

Verifikation der ausgewählten NP-schweren Probleme

Author:	Zixuan Fan
Supervisor:	Tobias Nipkow PhD.
Advisor:	Katharina Kreuzer
Submission Date:	Submission date

I confirm that this bachelor's thesis in informatics, is my own work and I have documented all sources and material used.

Munich, Submission date   Zixuan Fan

Acknowledgments

Abstract

NP-hardness is a widely discussed topic in the theoretical computer science. Since the publishing of the Cook-Levin-theorem in the early 1970s, the researchers have proved the NP-hardness of many problems. Among many NP-hard problems, the most famous are the 21 NP-complete problems given by Karp in the paper *Reducibility of Combinatorical Problems*, for it covers a considerable amount of NP-hard problems from various mathematical disciplines. The proofs of the NP-hardness, however, were limited on-paper. With the existence of the interactive theorem provers, it is possible to reproduce and verify the proofs with the aid of computers. In order to demonstrate the capability of interactive theorem provers in verifying the NP-hardness, we formalized and verified a few NP-hard problems using the well-known interactive theorem prover, Isabelle.

Contents

1 Introduction

1.1 Motivations

We may encounter many real-life problems that require a decision process to find a solution. When shopping at a supermarket, for example, we always want to choose the shortest queue. Another example is board games like go and chess. These problems are formally defined as decision problems. One of the most famous decision problem classes is the **NP-Hard**. **NP-Hard** problems have been a fundamental research topic in theoretical computer science since the 1970s, when Cook and Levin showed that the **Satisfiability** problem is **NP-Complete** and Karp gave a list of 21 **NP-Hard** problems. In the next few decades, many attempts were made to show that $\mathbf{P} = \mathbf{NP}$ and to develop algorithms that computes **NP** problems efficiently. Among many fields related to **NP-Hardness**, we focus on the polynomial reductions, which show the **NP-Hardness** of decision problems.

All the existing proof of **NP-Hardness** were on-paper proofs, which lack the automated verification by a computer. With the existence of powerful interactive theorem provers, it is meaningful to formalize and verify the classical results of **NP-Hardness** in a computer, contributing to the theoretical basis of many existing formalisation results, e.g. cryptography, approximation algorithms etc. There has been an attempt to formalise **NP-Hard** problems that were given in Karp's paper in 1972. Our work benefits from this attempt and continues to formalise the rest of the 21 **NP-Hard** problems in the interactive theorem prover Isabelle.

1.2 Contributions

Our work contains 2 categories of problems.

1. Set Covering problems: **Exact Cover**, **Exact Hitting Set**
2. Weighted sum problems: **Subset Sum**, **Number Partition**, **Integer Programming**, **Knapsack**

For each listed problem, we present a polynomial reduction either from **Satisfiability** or from another problem that is listed above. Thus, a reduction trace from **Satisfiability** is witnessed. Furthermore, a proof for the soundness, completeness, and the polynomial complexity of each polynomial reduction is also presented.

1.3 Outline

In Chapter 2, we introduce the Isabelle dependencies and mathematical backgrounds of our work.

Chapter 3 and Chapter 4 follow with the formalisation and verification of the listed problems. For each decision problem, we present a definition of the problem and the reduction. Then, we sketch the proof of the correctness of the reduction and the polynomial bound. Finally, we present a few concrete implementation details. In Chapter 3, we discuss the polynomial reduction of the set cover problems, while Chapter 4 consists of that of the weighted sum problems.

To finish, we conclude the current status of the project of formalisation of Karp's 21 **NP-Hard** problems and present a few possibilities for the verification of the rest of the problems in Chapter 5.

2 Preliminaries

2.1 Isabelle and Dependencies

Isabelle/HOL

Isabelle is a generic interactive theorem prover. HOL is the Isabelle's formalization of Higher-Order Logic, a logical system with inductive sets, types , well-founded recursion etc. Our implementation requires the introduction of new datatypes, formalisation of natural numbers and integers. Thus, this type system is necessary.

HOL-Real_Asymp and Laudau_Symbols

TODO

NREST

TODO

The Karp21 Project

The project aims to formalise all of the 21 **NP-Hard** problems in Karp's paper in 1972. Up till now, there are **TOCOUNT** problems of them finished, with a few other **NP-Hard** problems that are related but not in Karp's list. Our work also contributes to this project, formalising six of the remaining problems. Though dependent on this project, our work only reuses a few definitions by the predecessors, while the most formalisation and verification is original. An overview of the project is given in the following graph.

DigitsInBase

This entry of Archive of Formal Proofs shows the uniqueness of representation of natural numbers given an arbitrary base. In other words, it proves the well-definedness of the n-ary counting systems. Our implementation benefits from this repository in showing the correctness of the polynomial reduction from **Exact Cover** to **Number Partition**.

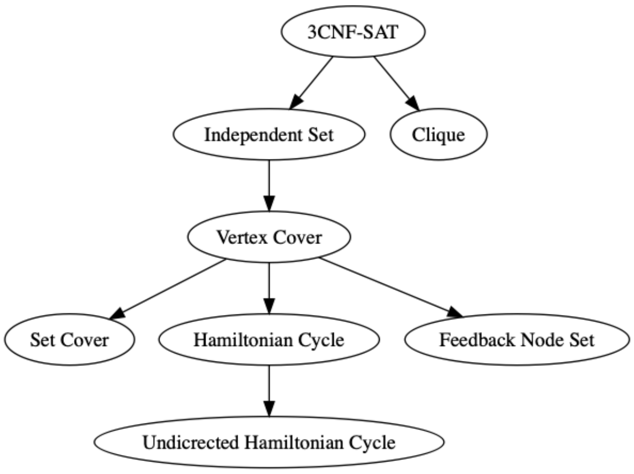

Figure 2.1: Name

2.2 NP-Hardness and polynomial reductions

2.2.1 Asymptotic Notation

Conventionally, the asymptotic notation is used for defining complexity classes and for performing the algorithm analysis. We follow this convention and choose the big $\mathcal{O}$ notation for algorithm analysis. To begin with, we present a brief introduction to the asymptotic notation.

Definition 1 *Let $f : \mathbb{R} \to \mathbb{R}$, $g : \mathbb{R} \to \mathbb{R}$ be two real valued functions with the same domain. $f(x)$ is big $\mathcal{O}$ of $g(x)$, which writes*

$$f(x) \in \mathcal{O}(g(x))$$

if there exists a real $M \geq 0$ and a real x_0 s.t.

$$|f(x)| \leq M|g(x)|, \forall x \geq x_0$$

Thus, f is above bounded by g. In other words, f is utmost as complex as g. Following this definition we can derive many complexity classes by g. A short list of commonly encountered complexity classes is given in table.

Name	Big $\mathcal{O}$ Notation	Algorithmic Examples
Constant	$\mathcal{O}(1)$	Parity check
Logarithmic	$\mathcal{O}(\log n)$	Binary search in a sorted array
Linear	$\mathcal{O}(n)$	Addition of integers
Quasilinear	$\mathcal{O}(n \log n)$	Merge-sort and heap-sort
Polynomial	$\mathcal{O}(n^c), c \in \mathbb{N}$	Matrix multiplication
Factorial	$\mathcal{O}(n!)$	Enumeration of partitions of a set

Table 2.1: List of commonly encountered complexity classes.

In most cases of this paper, we only consider the polynomial class, which contains the most classes listed above as subclasses except for the factorial class. For the purpose of simplicity, we did not formalise the theory of asymptotic classes and big $\mathcal{O}$ notation, but used the available Isabelle dependencies of **HOL-Real_Asymp** and **Landau_Symbols**.

2.2.2 Decision problems

Definition 2 *A decision problem is a yes-no question on an infinite set of fixed type of inputs.*

Generally, if we refer to a decision problem A, we are referring to the set of all inputs for which the answer to the yes-no question is yes. The handling of a decision problem usually involves two questions:

1. Is there an algorithm, which computes the solution to this problem, terminating on all inputs?
2. If the answer to the first question is yes, is this algorithm efficient?

If the answer to the first question is yes for a problem, it is a decidable problem, otherwise it is non-decidable. We do not expect a yes or no answer for the second question, but would like to find the optimal complexity for the algorithm. While some problems are possible to computed in an optimal upperbound by a deterministic algorithm, there are also a few problems, for which no deterministic polynomial algorithm is found. We define them formally as **NP**.

Definition 3 *If there is a non-deterministic algorithm that decides the solution to the problem in polynomial time, it is in the complexity class* ***NP****.*

Definition 4 *If a problem is at least as complex as the most complex problems in* ***NP****. It is in the complexity class* ***NP-Hard****.*

Although many attempts have benn made to prove or reject the existence of a non-deterministic algorithm for the **NP** problems, our work focuses on the NP-Hardness. We would like to formally prove that many classical decision problems are **NP-Hard**. For this reason, we have to introduce polynomial reduction.

2.2.3 Polynomial reductions

Given two decision problems A and B, a reduction is a function $f : A \to B$, which maps the inputs of the question of the first problem to that of the second problem. A reduction is polynomial if and only if the reduction function has a polynomial bound. For a polynomial reduction from A to B, we writes $A \leq_p B$.

Let M and N denotes the domains of A and B respectively. A function $g : M \to N$ is a polynomial reduction if and only if the following conditions are fulfilled.

$$x \in A \iff g(x) \in B \tag{2.1}$$

$$\exists k \in \mathbb{N}. f \in \mathcal{O}(n^k) \tag{2.2}$$

```
datatype 'a lit = Pos 'a | Neg 'a

type_synonym 'a three_sat = "'a lit set list"

definition lift :: "('a ⇒ bool) ⇒ 'a lit ⇒ bool" ("_↑" 60) where
  "lift σ ≡ λl. case l of Pos x ⇒ σ x | Neg x ⇒ ¬ σ x"

definition models :: "('a ⇒ bool) ⇒ 'a three_sat ⇒ bool" (infixl "⊨" 55)
where
  "σ ⊨ F ≡ ∀cls ∈ set F. ∃l ∈ cls. (σ↑) l"

definition sat :: "'a three_sat ⇒ bool" where
  "sat F ≡ ∃σ. σ ⊨ F"

definition"cnf_sat ≡ {F. sat F ∧ (∀cls ∈ set F. finite cls)}"
```

Figure 2.2: Definition of satisfiability problem in conjunctive normal form

For the convenience reason, we usually separate (2.1) into the soundness and completeness of the reduction.

$$\textit{soundness}: \quad x \in A \Longrightarrow g(x) \in B \tag{2.3}$$
$$\textit{completeness}: \quad g(x) \in B \Longrightarrow x \in A \tag{2.4}$$

2.2.4 NP-Hardness and Satisfiability

To show a decision problem B is **NP-Hard**, we have to find a **NP-Hard** problem and polynomial reduction s.t. $A \leq_p B$. A first proven **NP-Hard** problem is **Satisfiability**, which was independently proven by Cook in 1971 and Levin in 1973. The **Satisfiability** problem is denoted by

Definition 5 ***Satisfiability***
Input: *A propositional logical formula in conjunctive normal form.*
Output: *Is there a valid assignment for this formula?*

In the previous implementation of the project, the **Satisfiability** is defined by a list of clauses, with the clauses as the sets of variables. There have been many attempts to solve **Satisfiability** problem in a polynomial bound, as well as many approaches to solve **Satisfiability** problem efficiently in certain scenarios. Thus, **Satisfiability** is one of the most studies **NP-Hard** problems, from which there are also many **NP-Hard** problems reduced. Our first reduction also stems from **Satisfiability**, while all the other reductions are constructed upon novel introduced problems. More details on the reduction and implementation are given in Chapter 3 and Chapter 4.

Application of NREST and paradigms

The NREST package offers an approach for approximating the complexity of non-deterministic processes. This is especially useful when iterating a set, a collection or any other unordered data structures. Thus, we use this package throughout this work. In our complexity analysis, the following commands are used.

- *RETURNT* **res**. A command that returns the result **res**. It costs exactly one time unit.
- *SPECT* **[cond** $\rightarrow$ **cost]**. A command used for checking a condition. Checking the condition **cond** take **cost** time units.
- *SPEC P Q*. A command used for assignment. Should $P\ x$ hold for an object x, it is a valid object after the assignment, which takes $Q\ x$ time units.

To apply the NREST approach in the complexity analysis, we convert the algorithm into the NREST commands. Let f denote a polynomial reduction from A to B, while f_alg denotes the NREST version of the reduction. Furthermore, we define sizing functions s_A and s_B as metrics for the asymptotic classes. To show that the reduction is polynomial, we show that the reduction is polynomially bounded in terms of space and time.

$$\begin{aligned} s_B(f(A)) &\leq \mathcal{O}(s_A(A)^k) \\ f\_algA &\leq SPEC\ (\lambda.B = f(A))\ (\lambda\_.\mathcal{O}(s_A(A)^k)) \end{aligned}$$

1. Proving that the reduction is correct.
2. Implementing the reduction in NREST commands.
3. Proving that the reduction costs polynomial time.
4. Proving that the algorithm costs polynomial space.

3 Set Covering Problems

In this chapter, we discuss about the **NP-Hardness** of a few set covering problems. Covering problems ask whether a certain combinatorical structure A covers another structure B, or how large A has to be to cover B. We focus on a subclass of covering problems, the exact covering problem. In this subclass, A covers B exactly, i.e. no element in B is covered twice in A. In Karp's paper in 1972, the following covering problems were included: Exact Cover, Exact Hitting Set, 3-Dimensional Matching, Steiner Tree, and Max Cut. In our implementation, we reduced **Satisfiability** to **Exact Cover**, and then reduced **Exact Cover** to **Exact Hitting Set**.

3.1 Exact Cover

The **Exact Cover**problem is a special of the set cover problem. Besides the covering property, it also requires the uniqueness of the elements. Being one of the most studied set cover problems, there are many other **NP-Hard** problems reduced from the **Exact Cover**. Among them, we will present the reduction to the SS and **Exact Hitting Set**.

Definition 6 *Exact Cover*
Input: *A set X and a collection S of subsets of X.*
Output: *Is there a disjoint subset S' of S s.t. each element in X is contained in one of the elements of S'?*

```
definition "exact_cover ≡ {(X, S). finite X ∧ ⋃S ⊆ X ∧ (∃S' ⊆ S. ⋃S' = X
∧ disjoint S')}"
```

3.1.1 Choice of reduction

Since **Exact Cover** is a fundamental **NP-Hard** problem, there are many different reductions available. Karp's reduction is from the Chromatic Number problem. Although the Chromatic Number problem was formalized in Karp's 21 project, we chose not to reduce from this problem because of the complexity of the graph traversal and the slight difference between Karp's definition and the available Isabelle definition. On the other hand, there is an easy reduction from **Satisfiability** to **Exact Cover**. This reduction does not involve graph traversal. The only technical barrier is the untyped set. While Isabelle only supportes typed sets, we resolve this problem by creating a container type. More details follow in the **ADD HYPERREF** chapter.

3.1.2 Reduction Details

Reduction

Given a propositional logical formula F, we index the variables and the clauses and use the following notations.

1. x_i denotes the i-th variable in the formula with $x_i \in vars\ F$
2. c_i denotes the i-th clause in the formula with $c_i \in F$
3. p_{ij} denotes the j-th position/literal in the i-th clause with $p_{ij} \in c_i$

Then we construct a set X and which contains all 3 different kinds objects.

$$X = vars\ F \cup F \cup \bigcup_{c_i \in F} c_i$$

Furthermore, we construct S, a collection of subsets of X. We determine the following subsets

1. $\{p_{ij}\}$. The unary set of positions
2. $\{c_i, p_{ij}\}$. The binary set of a clause and a position in it.
3. $pos(x_i) := \{x_i\} \cup \{p_{ab} | p_{ab} = x_i\}$. The set of its positive occurrences as positions.
4. $neg(x_i) := \{x_i\} \cup \{p_{ab} | p_{ab} = \neg x_i\}$. The set of a variable with its negative occurrences as positions.

S contains all of the four types of subsets.

$$\begin{aligned} S =& \{p_{ij} | p_{ij} \in c_i, c_i \in F\} \cup \{\{c_i, p_{ij}\} | p_{ij} \in c_i, c_i \in F\} \\ & \cup \{\{x_i\} \cup \{p_{ij} | p_{ij} \in c_i, c_i \in F\} | x_i \in vars\ F, x_i = p_{ij}\} \\ & \cup \{\{x_i\} \cup \{p_{ij} | p_{ij} \in c_i, c_i \in F\} | x_i \in vars\ F, \neg x_i = p_{ij}\} \end{aligned}$$

The pair of (X, S) is the input for the **Exact Cover** problem.

Correctness

Let F be a satisfiable propositional formula, with $\sigma \models F$ as a valid assignment. We construct an exact cover $S' \subseteq S$ of X in the following steps.

1. For each $x_i \in vars\ F$, $pos(x)$ is included in S' if $\sigma(pos(x)) \equiv \top$. Otherwise we insert $neg(x)$ into S'.
2. For each $c_i \in F$, we choose the minimal j with $\sigma(p_{ij}) \equiv \top$, and insert $\{c_i, p_{ij}\}$ into S'
3. For each $p_{ij} \in c_i$, if $\sigma(p_{ij}) \equiv \top$ and $\{c_i, p_{ij}\}$ is not in S', the unary set $\{p_{ij}\}$ is included.

Obviously, each position in included $pos(x)$ and $neg(x)$ will have the false value under the assignment σ, while the positions in the other sets all have truth value. By design, the positions in the second and the third steps never duplicate. Thus, the positions never occur in two different sets in the collection S'. Furthermore, the clauses and variables occurs in exactly one set in S'. Hence the constructed collection is disjoint.

From the sole occurrence of the clauses and variables, we can also conclude that they are covered in this collection. Now we only have to prove that all the positions are covered. If a position p_{ij} has the false value under σ, it is covered in the first step. Otherwise it is either covered in the second step or the third step. With the disjointness, we may conclude that S' covers X exactly and that the reduction is sound.

Given an exact cover pair (X, S), it is easy to reconstruct the model σ with the same approach as in the proof of the soundness, which also shows that F is satisfiable. Thus, the completeness of the reduction is proven.

Polynomial Complexity

In the reduction, we have to iterate all of the variables, the clauses and the positions. Thus, we have to find a polynomial bound with regards to three metrics. Let n, m and k denotes the number of variables, clauses and postitions respectively. To derive the three metrics, we have to iterate all of the clauses, resulting the complexity of $\mathcal{O}(m)$. Unexpectedly, this results in the set X, indicating a linear complexity for the construction.

Now the interesting part is the collection S. For each type of subsets, we give a polynomial bound

1. $\{p_{ij}\}$. It is sufficient to merely iterate the positions, which requires the complexity of k.
2. $\{c_i, p_{ij}\}$. Each clause is iterated for $|c_i|$ times. Since $|c_i|$ is a constant, there is a $c \in \mathbb{N}$ s.t. $|c_i| \leq c$. Obviously, the complexity is above bounded by $c \cdot m$.
3. $pos(x)$ and $neg(x)$. For each variable x, it is required to iterate all of the positions, which produces the complexity of $2 \cdot nk$ in total.

Thus, the construction costs the polynomial complexity of $k + cm + nk \in \mathcal{O}(nk + m)$. With the linear complexity of the construction of X, we conclude that the reduction has the polynomial complexity.

3.1.3 Implementation Details

3.2 Exact Hitting Set

The hitting set problems are variants of the set covering problems. Essentially, they are two different ways of viewing the same problem. Just as hitting set is a variant of set covering, exact hitting set is a variant of exact cover.

Definition 7 ***Exact Hitting Set***
Input: *A collection of sets S*
Output: *Is there a finite set W s.t. the intersection of W and each element $s \in S$ contains exactly one element?*

Note that the exact hitting set problem was referred to as the hitting set problem in Karp's work, whereas it is generalized to be another problem nowadays.

3.2.1 Reduction Details

Reduction

Given an exact cover pair (X, S_0), the hitting set input S_1 is constructed by

$$S_1 = \{\{s | u \in s, s \in S_0\} | u \in X\}$$

Thus, S' is the set of sub-collections, which all contains one specific element.

Correctness

The soundness is directly proven with the existence of S' that covers X exactly in the exact cover problem. Let $W = S'$ and $c_u = \{s | u \in s, s \in S_0\}$ be an arbitrary element in S_1. Since S' covers X exactly, there is exactly one $s \in S'$, which contains u. Moreover, this s is also included in the S_0. Hence it holds that $W \cap c_u = s$ and hence $|W \cap c_u| = 1$.

The proof of the completeness shares a similar construction of S' from W. The only difference is that W is not necessarily a subset of S_0. Nevertheless, there exists a subset $W' \subseteq W$ s.t. it is also a subset of S_0, fulfills the same property as W. The completeness is then proven analogous as the soundness.

Polynomial Complexity

To construct S_1 as in the reduction, it is necessary to iterate the set X and the collection S in a nested loop. With the cardinality $|X|$ and $|S|$ as the metrics, it is obvious that the reduction costs the complexity of $\mathcal{O}(|X||S|)$.

3.2.2 Implementation Details

Reduction and Correctness

Since the exact cover problem is defined over a finite set X and a finite collection S_0, we have to check if the X and S_0 are finite and if S_0 only contains the subsets of X. Thus, a condition statement, which checks this requirement, is added to the implementation. Furthermore, the proof of the correctness is implemented as described above. Following is a snippet of implemented definitions and lemmas.

```
definition "exact_hitting_set ≡ {S. ∃W. finite W ∧ (∀s ∈ S. card (W ∩ s)
= 1)}"

definition "xc_to_ehs ≡ λ(X, S). (if finite X ∧ ⋃S ⊆ X then {{s. u ∈ s ∧
s ∈ S} | u. u ∈ X} else {{}})"

lemma xc_to_ehs_sound: "(X, S) ∈ exact_cover ⟹ xc_to_ehs (X, S) ∈
exact_hitting_set"

lemma xc_to_ehs_complete: "xc_to_ehs (X, S) ∈ exact_hitting_set ⟹ (X,
S) ∈ exact_cover"

theorem is_reduction_xc_to_ehs:
  "is_reduction xc_to_ehs exact_cover exact_hitting_set"
```

Figure 3.1: Definitions and lemmas about the reduction and correctness

Polynomial Complexity

We determine the size of the exact hitting set entry S_1 as $|S_1|$, the cardinality of S_1.

A General Addenda

If there are several additions you want to add, but they do not fit into the thesis itself, they belong here.

A.1 Detailed Addition

Even sections are possible, but usually only used for several elements in, e.g. tables, images, etc.

B Figures

B.1 Example 1

✓

B.2 Example 2

✗

List of Figures

List of Tables