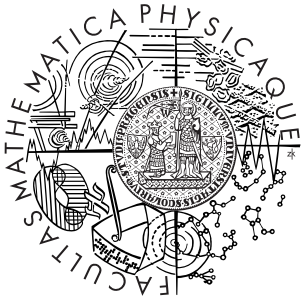
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Veronika Stankovianska

NP-úplné problémy

Katedra algebry

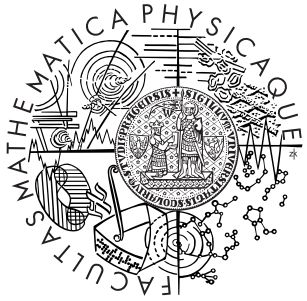
Vedoucí bakalářské práce: Prof. RNDr. Jan Krajíček, DrSc.,

Studijní program: Matematika, Obecná matematika

2009

Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Veronika Stankovianska

NP-complete Problems

Department of Algebra

Supervisor: Prof. RNDr. Jan Krajíček, DrSc.

Study Program: Mathematics, General Mathematics

2009

Acknowledgements

I would like to thank my supervisor, Professor Jan Krajíček, for his time, suggestions, enthusiasm and support.

I declare that the following BA thesis is my own work for which I used only the sources and literature mentioned. I have no objections to the BA thesis being borrowed and used for study purposes.

Prague 06/08/2009 Veronika Stankovianska

Contents

Title: NP-complete Problems
Author: Veronika Stankovianska
Department: Department of Algebra
Supervisor: Prof. RNDr. Jan Krajíček, DrSc.
Supervisor's e-mail address: krajicek@math.cas.cz

Abstract: The present paper is concerned with $\mathcal{NP}$-complete problems from various areas of mathematics and their subsequent reductions. First, computation of deterministic and non-deterministic Turing Machine is introduced to phrase decision problems by means of formal languages as a "yes-no" question. Definition of time complexity formalises polynomial hierarchy and leads to inquiry about the relationship of the classes $\mathcal{P}$ and $\mathcal{NP}$, conjectured to $\mathcal{P} \neq \mathcal{NP}$. Polynomial time reducibility and the notion of $\mathcal{NP}$-completeness, arising from further research in $\mathcal{P}$ vs. $\mathcal{NP}$, are central to the rest of the work. Second, Satisfiability of Boolean formulae (SAT) is presented noting its variants. Third, classical $\mathcal{NP}$ problems, 3-Satisfiability, Vertex Cover and Hamiltonian Circuit are reduced to each other. Fourth, departing from already proved results, others are to be shown $\mathcal{NP}$-complete, with intent to illustrate the relationship of problems from seemingly disconnected mathematical disciplines as well as importance of the notion of $\mathcal{NP}$-completeness.

Key: $\mathcal{NP}$-complete Problems, Polynomial Time Reductions.

Název práce: NP-úplné problémy
Autor: Veronika Stankovianska
Katedra (ústav): Katedra algebry
Vedoucí bakalářské práce: Prof. RNDr. Jan Krajíček, DrSc.
e-mail vedoucího: krajicek@math.cas.cz

Abstrakt: Předkládaná práce se zabývá $\mathcal{NP}$-úplnými problémy z různých oblastí matematiky a jejich následnými redukcemi. Pro formulaci rozhodovacího problému za pomoci formálních jazyků jako zjišťovací otázky je prvně zavedeno počítání deterministického a nedeterministického Turingova stroje. Definice časové složitosti pak umožňuje formalizaci polynomiální hierarchie a vede ke zkoumáni vztahu tříd $\mathcal{P}$ a $\mathcal{NP}$, odhadovaného na $\mathcal{P} \neq \mathcal{NP}$. Polynomiální převoditelnost a $\mathcal{NP}$-úplnost, vznikající z dalšího výzkumu v oblasti $\mathcal{P}$ vs. $\mathcal{NP}$, jsou ústředními pojmy zbytku práce. Dále je prezen-

tován problém splnitelnosti Booleovských formulí (SAT) a jeho varianty. Následně jsou na sebe převedeny klasické $\mathcal{NP}$-úplné problémy: splnitelnost formulí v 3-CNF, vrcholové pokrytí a Hamiltonovské kružnice. Konečně na základe již dokázaných tvrzení bude ukázána $\mathcal{NP}$-úplnost dalších problémů se záměrem ilustrovat vztah mezi problémy ze zdánlivě nepropojených matematických disciplin a také význam $\mathcal{NP}$-úplnosti.

Klíčová slova: $\mathcal{NP}$-úplné problémy, redukce v polynomiálním čase.

Chapter 1

Introduction

1.1 Motivation

$\mathcal{NP}$-completeness was conceptualised in 1970s as a result of vast investigations in the field of mathematics from late 19^{th} to early 20^{th} century and Entscheidungsproblem originating in the 17^{th} century. The concept of a decision problem inspired by Gottfried Leibniz was formalised by David Hilbert in 1928 and defined as a "yes-no" question phrased in a formal system. Embracing all intractable decision problems, qualitatively characterised as hard to compute and easy to verify in terms of their solution, $\mathcal{NP}$-complete problems play a central role in complexity theory. The importance of $\mathcal{NP}$-complete problems lies in both their dispersion over different areas of mathematics due to their reductions of one to another, providing a better insight into the problematic, and direct relation to one of the most famous and beautiful open problems; $\mathcal{P}$ versus $\mathcal{NP}$.

To study $\mathcal{NP}$-completeness and its properties, formalisation of the notion of computation is needed. Therefore, algorithms, Turing machines and relevant classes of formal languages are defined. The binary encoding of finite objects is assumed to be used.

1.2 Notation

An algorithm is a method operating on the basis of a finite set of instructions primarily designed to provide a solution to a problem. A more precise approach to the notional concept of algorithms is based on viewing algorithms

as Turing machines. In brief, Turing machine abstractly models computation using an infinite tape, a finite state control and a read-write head.

The program limited by instructions, reads the input written in the form of a string of symbols using the head. Starting from a given square q_s, the head scans the corresponding symbol and proceeds according to the instructions. A step-by-step computation either ends by reaching a halt state, providing an answer to the problem, or defines further proceedings of Turing machine by means of a transition function. Then, the head rubs the contents of the square out and writes a new symbol in its place. With respect to the direction imposed by the transition function, the read-write head and the finite state control move to the next square of the tape. The described procedure is fully illustrative of a computation step of a deterministic Turing machine as defined below.

Definition 1 (Deterministic Turing Machine - DTM)**.** *Deterministic Turing machine is a 8-tuple M where $M = \{\Gamma, \Sigma, b, Q, q_s, q_Y, q_N, T\}$, Γ, a non-empty finite set of at least two symbols, denotes the input symbols and a finite set Σ; $\Gamma \subset \Sigma$, denotes the tape symbols, b; $b \in \Sigma - \Gamma$, represents a blank symbol, Q, a non-empty finite set of all state symbols of which q_s is the start symbol, q_Y, q_N, two halt states, and T, is a transition function defined as follows:*

$$T : \Sigma \times (Q - \{q_Y, q_N\}) \longrightarrow \Sigma \times Q \times \{\leftarrow, \rightarrow\}$$

where the arrows $\leftarrow, \rightarrow$ indicate the sense in which the head's shift is executed.

On the condition that the computation of a deterministic Turing machine M ends, it is possible to analyse its tape to extract results to define function f_M corresponding to the computation of M on an input x.

Gradually, a large variety of Turing machines came into existence. Ranging from multiple-tape models to versions restrained to one-way infinite tapes with heads moving either right or left, all TM are computationally equivalent and can be successfully simulated using DTM. More complex nondeterministic Turing machines are also the case.

Definition 2 (Non-Deterministic Turing Machine - NDTM)**.** *Non-deterministic Turing machine is a 6-tuple M where $M = \{\Gamma, \Sigma, b, Q, q_s, T\}$; Γ, Σ, b, q_s, T as in DTM, Q contains non-deterministic states and the computation on given string $x \in \Gamma$ simulates DTM computation until a non-deterministic state q_{ND} is reached. Then, the next state is chosen arbitrarily.*

An NDTM computation is said to be accepting if it halts in q_Y, otherwise it is classified as non-accepting. In other words, an input is accepted if there is a sequence of non-deterministic choices resulting in answering "yes".

It is important to note that unlike a single computation path in DTMs, non-deterministic Turing machines produce a computation tree where any branch halting with an "accept" condition is sufficient for an NDTM to accept the input. However, if an input x is not accepted, it does not imply that x does not belong to a language L, a formal system to be defined. It means that there was at least one faulty decision in non-deterministic states in the process of NDTM computation. Consequently, the computation is to be started afresh.

Formalisation of the computation model from algorithms to one-tape Turing machines necessitates further development of formal systems to enable formulation of decision problems. The process of solving a decision problem is then equivalent to the acceptance or recognition of formal languages described subsequently as sets of natural numbers computable by a specific type of recursive functions.

Definition 3 (Partially Recursive Function)**.** *Let A be the domain of a function f. Function $f; f : A \subseteq \mathbb{N} \longrightarrow \mathbb{N}$ is a partially recursive function $\Leftrightarrow \exists M$, an algorithm:*

- $A = dom(f_M)$
- $\forall x \in A; f(x) = f_M(x);$

where the function f_M is given by (the computational output provided by) the algorithm M.

The class of partially recursive functions is denoted PRF.

Definition 4 (Recursive Function)**.** *Function $f; f : \mathbb{N} \longrightarrow \mathbb{N}$ is a recursive function $\Leftrightarrow$*

- $\exists f : f \in PFR$
- $dom(f) = \mathbb{N}.$

The class of recursive functions is denoted RF.

Definition 5 (Recursively Enumerable Language)**.** *Language* $L; L \subseteq \mathbb{N}$ *is a recursively enumerable language* $\Leftrightarrow \exists f;\ f \in PRF;\ dom(f) = L$.

The class of recursively enumerable languages is denoted RE.

Definition 6 (Recursive Language)**.** *Language* $L; L \subseteq \mathbb{N}$ *is a recursive language* $\Leftrightarrow \exists f \in RF; f = \chi_L$, *a characteristic function:*

$$\chi_L(x) = \begin{cases} 1, & x \in L \\ 0, & x \notin L. \end{cases}$$

The class of recursive languages is denoted R.

Recursively enumerable and recursive languages are also referred to as partially decidable and decidable languages since one of the most important problems put on algorithms and Turing machines is to decide the language by determining whether all given words are in the target language. The outcome of such a task is to reach a halt state, answering either "yes" or "no."

Nevertheless, it is possible that a state solving the problem unambiguously on an input is never attained, classifying the problem as formally undecidable. Halting problem, formalising the notion of unconditional response provided by algorithms, is undecidable.

Definition 7 (Halting Problem)**.** *Let M denote an algorithm, x ranges over possible inputs to M. Halting problem is defined as follows:*

$$HALT := \{(M, x) \mid M \text{ halts on } x\}.$$

To grasp the structure and the complexity of decision problems, a function assessing the time spent on computing the answer to the problem is needed. Thus, time complexity and its variant classes, depending on the model of the Turing machine used in computation, will be defined.

Definition 8 (DTM Time Complexity)**.** *Time complexity is the function* $t_M : \mathbb{N} \longrightarrow \mathbb{N} \cup \{\infty\}$; $t_M := max\{time_M(x) | x : |x| \leqslant n\}$, *where* $time_M(x)$ *denotes the number of steps needed for M, a DTM, to halt on x.*

Definition 9 (Polynomial Time DTM)**.** *Deterministic Turing machine M is a polynomial time DTM if* $\forall n$:

$$t_M(n) \leqslant n^{O(1)}.$$ [1]

[1] According to Bachmann-Landau notation, $\exists c > 0\ \forall n : t_M(n) \leqslant n^c$.

It is important to note that a polynomial time DTM halts on all inputs.

The time a non-deterministic Turing machine requires to accept the input string $x \in \Gamma^*$ of length n is defined as the minimal time over all accepting computations of NDTM on x on the condition that such a computation exists. Otherwise, $time_M := \infty$.

A polynomial bound in the following definition is restricted only to the accepting computations of the NDTM in question.

Definition 10 (Polynomial Time NDTM)**.** *Non-deterministic Turing machine M is a polynomial time NDTM if* $\forall x;\ |x| = n$ *and* $time_M(x) \neq \infty \Rightarrow$

$$time_M(n) \leqslant n^{O(1)}.$$

Polynomial time deterministic and non-deterministic Turing machines can be seen as specific computational resources for solvability of computational problems from complexity classes $\mathcal{P}$ and $\mathcal{NP}$.

Definition 11 ($\mathcal{P}$)**.** $\mathcal{P} = \{L \subseteq \mathbb{N} \mid \exists$ *polynomial time DTM* $M : f_M = \chi_L\}$.

Analogously, $\mathcal{NP}$ is the class of languages accepted in polynomial time using non-deterministic Turing machine. There is an alternative equivalent definition based on polynomially decidable relation as a verifier of the problem whose decidability is checkable in a polynomial time.

Definition 12 ($\mathcal{NP}$)**.** $L \in \mathcal{NP}$*:* $\exists c > 0 \wedge$ *binary relation* $R(x, y)$*:*

- $R(x, y)$ *is decidable in polynomial time;*
- $\forall x;\ x \in L \Leftrightarrow \exists y : |y| \leqslant |x|^c \wedge R(x, y)$.

The importance of both the classes $\mathcal{P}$ and $\mathcal{NP}$ lies in incapacity of proving that either $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \neq \mathcal{NP}$ rendering the construction of substantial class of $\mathcal{NP}$-complete problems a promising approach for further consideration of theoretical question of $\mathcal{P}$ (easy to solve) versus $\mathcal{NP}$ (easy to verify).

The following concept of $\mathcal{NP}$-completeness is infinitely more complex and significant if $\mathcal{P}$ is distinct from $\mathcal{NP}$ since it provides a whole range of conditional results. In spite of being nearly as intractable as the unconditional ones, the results based on conjecture that $\mathcal{P} \neq \mathcal{NP}$ tend to be proved straightforwardly using, among other techniques, the influential notion of polynomial

time reducibility first presented in Richard Karp's 1972 paper "Reducibility Among Combinatorial Problems"[8].[2]

Definition 13 (Polynomial Time Function)**.** *Function* $f; f : \mathbb{N} \longrightarrow \mathbb{N}$ *is a polynomial time function* $\Leftrightarrow$ $\exists$ *DTM running in polynomial time computing the function* f*.*

Definition 14 (Polynomial Time Reducibility)**.** *Let* $L_1, L_2 \subseteq \mathbb{N}$*. Polynomial time reducibility is a polynomial time function* f*;* $f : \mathbb{N} \longrightarrow \mathbb{N}$*,* $\forall x \in \mathbb{N}$ *:*

$$x \in L_1 \Leftrightarrow f(x) \in L_2.$$

Polynomial reducibility of L_1 *to* L_2 *is denoted* $L_1 \leqslant_p L_2$*.*

Polynomial time reducibility is transitive.

Lemma 1. *Let* $L_1,\ L_2,\ L_3 \subseteq \mathbb{N}$*. If* $(L_1 \leqslant_p L_2) \wedge (L_2 \leqslant_p L_3) \Rightarrow (L_1 \leqslant_p L_3)$*.*

Eventually, to specify the characteristics of $\mathcal{NP}$-completeness, a problem equivalent to the hardest problem found in $\mathcal{NP}$ is going to be formally presented in terms of polynomial reductions.

Definition 15 ($\mathcal{NP}$-Hard)**.** *Language* $L \subseteq \mathbb{N}$ *is* $\mathcal{NP}$*-hard if* $\forall L' \in \mathcal{NP}$ *:*

$$L' \leqslant_p L.$$

Definition 16 ($\mathcal{NP}$-Complete)**.** *Language* $L \subseteq \mathbb{N}$ *is* $\mathcal{NP}$*-complete* $\Leftrightarrow$

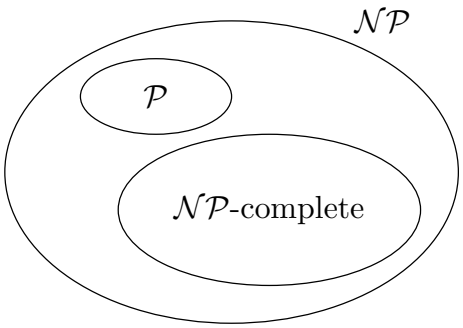
- $L \in \mathcal{NP}$;
- L *is* $\mathcal{NP}$*-hard.*

The class of $\mathcal{NP}$-complete problems was destined to play an important role in complexity theory as it allows a deeper understanding of the intractability of $\mathcal{P}$ versus $\mathcal{NP}$ problem.

It is widely believed that $\mathcal{P} \neq \mathcal{NP}$ whereas the majority of failed attempts intended to prove that $\mathcal{P} = \mathcal{NP}$. Main motivation for proving an apparently less probable conjecture is due to a prospective proof strategy. While there is no unambiguous means of proceeding to prove $\mathcal{P} \neq \mathcal{NP}$, to show that $\mathcal{P} = \mathcal{NP}$, it is sufficient to prove that an $\mathcal{NP}$-complete problem is in $\mathcal{P}$.

[2] Twenty-one problems were proved $\mathcal{NP}$-complete within a range of 19 pages to foreshadow the potential of the technique of reducibility.

Thus, applying polynomial time reductions, the whole class of $\mathcal{NP}$-complete problems, being by definition $\mathcal{NP}$-hard and in $\mathcal{NP}$, is consequently proved to be included in $\mathcal{P}$. Effacement of the distinction between problems that are easy to compute and easy to verify then yields equality of both the classes, formally, $\mathcal{P} = \mathcal{NP}$, and lies entirely in the properties of the $\mathcal{NP}$-complete complexity class as defined.



$\mathcal{NP}$-completeness in terms of $\mathcal{P}$ and $\mathcal{NP}$ hierarchy if $\mathcal{P} \neq \mathcal{NP}$.

Chapter 2

First $\mathcal{NP}$-complete Problem

An $\mathcal{NP}$-complete problem can be alternatively referred to as a decision problem in $\mathcal{NP}$ to which all other problems included in $\mathcal{NP}$ are reducible. The privilege of being the first $\mathcal{NP}$-complete problem, hence, the hardest problem contained in $\mathcal{NP}$ was granted to the problem of the existence of a satisfiable evaluation of Boolean formulae. The problem of Satisfiability was first formulated and proved to be $\mathcal{NP}$-complete by Stephen A. Cook in his seminal 1971 paper "The Complexity of Theorem Proving Procedures"[4].

Since then, a significant number of $\mathcal{NP}$-complete problems emerged ranging from the fields of logic, automata, language, graph, number and set theory, algebra, mathematical programming, program optimisation and network design even to seemingly unrelated domains of games and puzzles.

2.1 SAT – Satisfiability

By design, Satisfiability has an undeniably central role in the process of proving intractable problems $\mathcal{NP}$-complete. By definition, Satisfiability signifies evaluation of Boolean formulae to true.

Let U; $U := \{u_1, u_2, \ldots, u_n\}$, be a finite set of Boolean variables, taking T, true, and F, false, for truth values. Boolean formula φ is constructed using arbitrarily many times the following rules, consisting of Boolean variables joined by Boolean connectives of $\wedge$, $\vee$ and $\neg$.

Definition 17 (Boolean Formula)**.** *The set of Boolean formulae built from atoms $U$is the smallest set such that:*

- *contains U;*

- $u_j \in U$, *a Boolean variable;*
- $(\neg\, \varphi_k)\,;\ k < j$, *the negation of previously defined Boolean expression;*
- $(\varphi_k\ \wedge\ \varphi_l)\,;\ k, l < i$, *the conjunction; or*
- $(\varphi_k\ \vee\ \varphi_l)\,;\ k, l < i$, *the disjunction of Boolean expressions defined earlier.*

Expressions of the form of a Boolean variable u_j and its negation $\neg\ u_j$ are referred to as literals. Conjunctions of literals are equally denoted as terms.

Definition 18 (Truth Assignment)**.** *Truth assignment for* φ*, a finite set of Boolean variables* $U := \{u_1, u_2, \ldots, u_n\}$*, is a function* $t : U \longrightarrow \{\mathrm{T}, \mathrm{F}\}$*.*

Notation:

- if $t(u) = \mathrm{T} \Rightarrow u$ is true under t
- if $t(u) = \mathrm{F} \Rightarrow u$ is false under t.

Conjunctive normal form (CNF) is a conjunction of clauses where clauses are disjunctions of literals. Disjunctive normal form (DNF) can be defined in a dual way as a disjunction of one or more terms. In other words, every Boolean expression can be represented by an equivalent formula in both conjunctive and disjunctive normal forms. In spite of disjunctive normal forms being equally representative of Boolean expressions, conjunctive normal forms are chosen as a more elegant approach towards evaluation of formulae.

Definition 19 (Satisfiability – SAT)**.** $SAT := \{\varphi \in \mathrm{CNF} \mid \varphi$ *is satisfiable*$\}$.

The definition of the class of all satisfiable CNF Boolean formulae facilitates the formulation of an answer to the question of existence of a satisfying truth assignment for C, a collection of clauses over the variables of a finite set U. As early as in 1971, the response given by Stephen A. Cook classifies SAT as the first $\mathcal{NP}$-complete problem and lays the foundations for proving a wide range of intractable problems $\mathcal{NP}$-complete by their reduction to SAT, stated $\mathcal{NP}$-complete by Cook's Theorem.

2.2 Cook's Theorem

Theorem 1 (Cook's Theorem)**.** *Satisfiability is $\mathcal{NP}$-complete.*

Outline of the Proof.

First, SAT $\in \mathcal{NP}$ since a truth assignment for all the variables in a given Boolean formula can be guessed by an NDTM and verified in a deterministic polynomial time.

Second, it is necessary to show that every language from $\mathcal{NP}$ is polynomial time reducible to the language encoding Satisfiability, in other words, that SAT is $\mathcal{NP}$-hard.

To begin, M, specified by Γ, Σ, b, Q, q_s, q_Y, q_N and T, denotes an arbitrary polynomial time NDTM program, accepting the language L. Let $f_L : \Sigma \longrightarrow$ SAT be the transformation of an instance of L to a collection of clauses C. The construction of f_L is to ensure that $\forall x \in \Sigma^* : x \in L \Leftrightarrow f_L(x) \in$ SAT. A set of clauses constructed from U, a set of Boolean variables, verifies whether an input x is accepted by M. The clauses, containing a limited number of variables, are satisfiable if and only if there exists a truth assignment forced by an accepting computation on x; $|x| = n$, where both the number of checking stages as well as the length of guessed string are bounded by $n^{O(1)}$.

If $x \in L$, there exists an accepting computation of M on x of length bounded by $n^{O(1)}$ imposing such a truth assignment that $C \in$ SAT.

In an opposite way, by construction of C, any satisfiable assignment yields correspondence to an accepting computation of M on x.

Finally, reduction f_L imposes several restrictions on parameters which result in polynomial time boundedness of f_L as $|f_L(x)| = |U| \cdot |C| \leqslant n^{O(1)}$.

Thus, SAT is $\mathcal{NP}$-complete.[1]

2.3 Variants of Satisfiability

Nearly every problem can be generalised and become whimsically intractable or considerably simplified when restricted to a special case. The problem of SAT is not an exception. The distribution of such instances thus varies from undecidable problems to those found in $\mathcal{P}$. As SAT presents a wide range of satisfiability problems, it is interesting to see the exact limits for transition of yet undecidable problems to $\mathcal{NP}$-complete, the hardest of decidable ones,

[1] The complete proof of Cook's Theorem can be found in [6].

and those easily solvable or found in $\mathcal{P}$. This task is approached by limiting the number of literals per clause.

Definition 20 (k-CNF)**.** *k-CNF denotes a CNF with k literals per clause; $k \in \mathbb{N}$.*

Definition 21 (k-Satisfiability – k-SAT)**.** *k-SAT* $:= \{\varphi \in k\text{-CNF} \mid \varphi$ *is satisfiable* $\}$.

A truth assignment of chosen u_is satisfying all the clauses is referred to as a solution to the problem of k-Satisfiability.

Usage of k-SAT problem, the restriction of Satisfiability problem to instances where k denotes the number of distinct literals in each clause, seems to be the most effective approach not only in the process of establishment of the upper and lower bounds for Satisfiability but also in recognising and proving newly phrased problems $\mathcal{NP}$-complete.[2]

The matter of k-SAT can be illustrated with greater precision if 2-SAT is considered. While 2-SAT is in $\mathcal{P}$, its transition of generalisation to 3-SAT has crossed the line of $\mathcal{NP}$-completeness, classifying 3-SAT as another $\mathcal{NP}$-complete problem.

[2] Further information on lower and upper bounds for Satisfiability and related $\mathcal{NP}$-completeness can be found in [13] and [11] respectively. Unsatisfiable k-CNF formulae are treated in [7].

Chapter 3

Basic $\mathcal{NP}$-complete Problems and Their Reductions

The main aim of this section is to show the most influential strategies in proving $\mathcal{NP}$-completeness by presenting the basic $\mathcal{NP}$-complete problems as foundations for further complexity theory results obtained by polynomial reductions.

Consequently, the problems of Satisfiability of the 3-CNF formulae, Vertex Cover and Hamiltonian Circuit are to be specified and proved $\mathcal{NP}$-complete by reducing one to another in order of mention. The present proofs, based on those introduced by Michael R. Garey and David S. Johnson[1] in [6], are written to be uniformly structured in the following manner:

- Argument for $L_2 \in \mathcal{NP}$, where L_2 denotes a targeted problem.
- Definition of the transformation of L_1 to L_2 by means of a reduction function f, where L_1 stands for an already proved $\mathcal{NP}$-complete problem and L_2 is as specified above.
- Argument for polynomial time reducibility of L_1 to L_2 ($L_1 \leqslant_p L_2$).
- Statement: $\forall x : \ x \in L_1 \Rightarrow f(x) \in L_2$, where x represents an instance of a decision problem.
- Statement: $\forall x : \ x \notin L_1 \Rightarrow f(x) \notin L_2$.

[1] Garey and Johnson's transformations differ from the original Karp's [8] in terms of different reduction path and modifications as well as replacements of some reductions. Richard Karp's approach towards proving the above succession of reductions can be seen in [8], page 96.

3.1 3-SAT – 3-CNF Satisfiability

By definition of k-SAT, the problem of 3-SAT is a restriction of SAT to 3-CNF formulae having exactly three literals per clause. The reduction of SAT to 3-SAT lies in rewriting a given clause with more than three literals as an equivalent set (in the sense of satisfiability) of three-literal clauses. Following its rather non-complex construction, 3-SAT is one of the largely used initial problems to depart from with intent to show $\mathcal{NP}$-completeness.

Theorem 2. *3-Satisfiability is $\mathcal{NP}$-complete.*

Proof.

The problem of 3-SAT is in $\mathcal{NP}$ as a truth assignment for the variables guessed by an NDTM algorithm is easily checked (in polynomial time) to satisfy all the given three-literal clauses.

The transformation f: SAT $\longrightarrow$ 3-SAT is based upon previously described representation of arbitrary instances of SAT formed by variables in $U = \{u_1, u_2, \ldots, u_n\}$ and clauses in $C = \{c_1, c_2, \ldots, c_m\}$.
Let the reduction be defined by construction of C' a collection of three-literal clauses constructed on U', a set of variables such that the satisfiability of C' implies the satisfiability of C and vice versa. The sets U' and C' are then formally represented by:

$$U' = U \cup \left(\bigcup_{j=1}^{m} U'_j \right) \quad \text{and} \quad C' = \bigcup_{j=1}^{m} C'_j,$$

where U'_j denotes additional auxiliary variables, restrained only to C'_j, a collection of three-literal clauses which just replace all the individual clauses $c_j \in C$.

Polynomial time boundedness follows directly from the outline of the proof of Cook's Theorem and the argument based on the observation that the number of 3-CNF clauses found in C' is bounded by a polynomial in $|U| \, . \, |C| = nm$, thus $|C'| \leqslant (nm)^{O(1)}$.

To prove that $C \in \text{SAT} \Rightarrow f(C) = C' \in \text{3-SAT}$, it is sufficient to show that the collection of clauses denoted as C'_j can be constructed on the basis of $c_j \in C$.
Following the construction of an arbitrary instance of k-SAT (normalisation of Boolean formulae to formulae in k-CNF), $c_j = \{z_1, z_2, \ldots, z_k\}$, where z_i; $1 \leqslant i \leqslant k$, represents a literal ($u_h$ or its negation $\neg u_h$; $h \leqslant n$) obtained

from U. The reduction of a clause in C depends on the number of literals k present in C'_j and branches accordingly:

$k = 1$.
$U'_j = \{y_j^1, y_j^2\}$, where y_j^* denotes auxiliary variables,
$C'_j = \{\{z_1, y_j^1, y_j^2\}, \{z_1, \neg y_j^1, y_j^2\}, \{z_1, y_j^1, \neg y_j^2\}, \{z_1, \neg y_j^1, \neg y_j^2\}\}$,

$k = 2$.
$U'_j = \{y_j^1\}$,
$C'_j = \{\{z_1, z_2, y_j^1\}, \{z_1, z_2, \neg y_j^1\}\}$,

$k = 3$.
$U'_j = \emptyset$,
$C'_j = \{\{c_j\}\}$,

$k \geqslant 4$.
$U'_j = \{y_j^i : 1 \leqslant i \leqslant k-3\}$,
$C'_j = \{\{z_1, z_2, y_j^1\}\} \cup \{\{\neg y_j^i, z_{i+2}, y_j^{i+1}\} : 1 \leqslant i \leqslant k-4\} \cup \{\{\neg y_j^{k-3}, z_{k-1}, z_k\}\}$.[2]

Having defined the transformation f: SAT $\longrightarrow$ 3-SAT and setting a truth assignment $t : U \longrightarrow \{\mathrm{T}, \mathrm{F}\}$ satisfiable to the collection C, the task reduces to showing that there exists a truth assignment $t' : U' \longrightarrow \{\mathrm{T}, \mathrm{F}\}$, an extension of t, satisfying C'. By design of U', it is only necessary to verify whether the assignment t can be extended to the variables in $U'_j = U' - U$, in other

[2] More explicitly, the instances of U'_j and C'_j for $k \geqslant 4$ are represented respectively by:

$k = 4$.
$U'_j = \{y_j^1\}$,
$C'_j = \{\{z_1, z_2, y_j^1\}, \{\neg y_j^1, z_3, z_4\}\}$,

$k = 5$.
$U'_j = \{y_j^1, y_j^2\}$,
$C'_j = \{\{z_1, z_2, y_j^1\}, \{\neg y_j^1, z_3, y_j^2\}, \{\neg y_j^2, z_4, z_5\}\}$,

$k = 6$.
$U'_j = \{y_j^1, y_j^2, y_j^3\}$,
$C'_j = \{\{z_1, z_2, y_j^1\}, \{\neg y_j^1, z_3, y_j^2\}, \{\neg y_j^2, z_4, y_j^3\}, \{\neg y_j^3, z_5, z_6\}\}$, etc..

It is clear that the satisfiability of C'_j is attributable to the satisfiability of the clause $\{z_1, z_2, \ldots, z_k\} \in C$; $k \geqslant 4$.

words, whether it holds for C'_j, the sole clauses with auxiliary variables. Therefore, each class of C'_js defined by the value of k is reviewed.

In the case of $k = 1, 2$, the clauses can be transformed into equivalent 3-literal clauses by copying their literal once or twice. Hence, corresponding C'_js are satisfied by t. As a result, t is extended either to $t' : y \mapsto \mathrm{T}$ or $t' : y \mapsto \mathrm{F}$ for $\forall y \in U'_j$.

If $k = 3$, the only clause $c'_j \in C'_j$ is satisfied by the original assignment t since it contains no auxiliary variables for further extension of t.

For $k \geqslant 4$ and t, a satisfying truth assignment to C, there exist a minimal l such that $z_l \in \{z_1, z_2, \ldots, z_k\} : t(z_l) = \mathrm{T}$. The construction of C'_j yields following satisfiable truth assignments:

$$
\begin{array}{ll}
l = 1, 2. & \\
t'(y^i_j) = \mathrm{F}; & 1 \leqslant i \leqslant k-3, \\
 & \\
l = 3, \ldots, k-2. & \\
t'(y^i_j) = \mathrm{T}; & 1 \leqslant i \leqslant l-2, \\
t'(y^i_j) = \mathrm{F}; & l-1 \leqslant i \leqslant k-3, \\
 & \\
l = k-1, k. & \\
t'(y^i_j) = \mathrm{T}; & 1 \leqslant i \leqslant k-3.
\end{array}
$$

If already one of the first two literals is assigned true, then c^1_j is satisfied and so are the clauses c^i_j; $2 \leqslant i \leqslant k-2$. Similar arguments apply for $l = k-1, k$. Otherwise, all the clauses preceding the one containing the first literal set true under t are satisfied due to evaluation of variables y^*_j to true. The successive clauses are equally satisfied as they are constructed from negations of variables assigned to false. Consequently, all clauses in C'_j are satisfied and C' is an instance of 3-SAT.

In an opposite way, if $C' \in$ 3-SAT then t, a restriction of t' to variables in U, is a satisfiable truth assignment to C. It follows immediately that if C is not satisfiable, nor f(C) is. Thus, satisfying collection of clauses C is necessary and sufficient condition for satisfiability of 3-CNF formulae derived thereafter. □

The problem of 3-Satisfiability itself has many other variants developed under restrictions put on the number of desirably assigned literals in every clause. For instance, Not-All-Equal 3-SAT is alternatively phrased to contain a minimum of one true and one false literal and One-in-Three 3-SAT

requires exactly one true literal per clause. The property of $\mathcal{NP}$-completeness of 3-SAT is granted even when the 3-literal clauses are constructed strictly from variables which are all negated or not. Previously defined decision problem is also referred to as of Monotone 3-SAT.

Large applicability of the structure of 3-SAT encourages further expansion of the notion of $\mathcal{NP}$-completeness from the domain of logic into different areas of research in both computer science and mathematics. Regarding the nature of studied problems, graph theory certainly classifies as one of the first intuitive choices for the employment of the concept of $\mathcal{NP}$-completeness. As a result, the problem of 3-Satisfiability is to be reduced to Vertex Cover, considered central to graph theory.

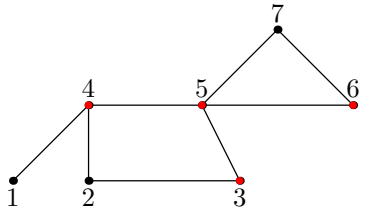
3.2 VC – Vertex Cover

A graph G is formalised as an ordered pair of sets (V, E), where V denotes an arbitrary finite non-empty set of vertices and $E, E \subseteq \binom{V}{2}$, comprises all the edges joining different vertices in V. A vertex cover for G is defined on the basis of the endpoints in every edge.

Definition 22 (Vertex Cover)**.** *A set* V'; $V' \subseteq V$, *is a vertex cover for* G $\Leftrightarrow \ \forall\ e;\ e = \{u, v\} \in E :\ (u \in V') \vee (v \in V')\,.$

Identically named decision problem (VC) asks if there exists a collection of vertices V' bounded by a positive integer k, a vertex cover of size k or less.

Example 1 (Instance of Vertex Cover)**.** *The character of the problem for a graph* $G = (V, E)$, *presented below, given by sets* $V = \{1, 2, 3, 4, 5, 6, 7\}$ *and* $E = \{\{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 5\}, \{4, 5\}, \{5, 6\}, \{5, 7\}, \{6, 7\}\}$, *can be illustrated by a vertex cover set* $V' = \{3, 4, 5, 6\}$ *marked in red.*



The vertex cover V' *stands for a minimal vertex cover for* G, *thus, by* $|V'| \leqslant k \leqslant |V|\,,\ k \in [4, 7]\,.$[3]

[3] By definition, Example 1 provides an instance of a vertex cover for any VC of size 4 to 7 involving graph G. Minimal Vertex Cover is a related $\mathcal{NP}$-hard optimisation problem.

Theorem 3. *Vertex Cover is $\mathcal{NP}$-complete.*

Proof.

To begin, by NDTM's guess of a subset $V' \subseteq V$ and polynomial time verification of V' as a vertex cover of a required size, it is straightforward that VC $\in \mathcal{NP}$.

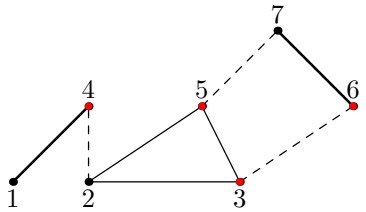
Let f: 3-SAT $\longrightarrow$ VC denote a reduction function. The transformation of a 3-SAT instance, given by $U = \{u_1, u_2, \ldots, u_n\}$ and $C = \{c_1, c_2, \ldots, c_m\}$, to a graph $G = (V, E)$ having a vertex cover $V' \subseteq V; |V'| \leqslant k;\ 0 < k \leqslant |V|$, exists exactly in the case of satisfiable C. The construction of the transformation comprises the following components:

Truth evaluation components T.
$T = \bigcup_{i=1}^{n} T_i;\ T_i = (V_i, E_i);$
$V_i = \{u_i, \neg u_i\},\ E_i = \{\{u_i, \neg u_i\}\}.$

Satisfaction testing components S.
$S = \bigcup_{j=1}^{m} S_j;\ S_j = (V'_j, E'_j);$
$V'_j = \{a_1[j], a_2[j], a_3[j]\},\ E'_j = \{\{a_1[j], a_2[j]\}, \{a_1[j], a_3[j]\}, \{a_2[j], a_3[j]\}\}.$

Collection of communication edges R.
$R = \bigcup_{j=1}^{m} E''_j;$
$E''_j = \{\{a_1[j], x_j\}, \{a_2[j], y_j\}, \{a_3[j], z_j\}\};\ \{x_j, y_j, z_j\} = c_j \in C.$[4]

[4] By applying previously defined construction on sets $U = \{u_1, u_2\}$ and $C = \{c_1\}$; $c_1 = \{x_1, y_1, z_1\}$, it is possible to obtain a graph defined on the set of vertices of the graph G presented in Example 1. Associating $u_1 := 1,\ \neg u_1 := 4,\ u_2 := 6,\ \neg u_2 := 7$ and $x_1 := 4,\ x_2 := 6,\ x_3 := 7$, the components E_i, E'_j and R are constructed and represented in bold, thin and dashed lines respectively.



An instance of vertex cover of size 4 for G is marked in red. Further conclusions also set the value of k to $k = n + 2m = 4$. This case presents $C \in$ 3-SAT irrespective of the truth assignment since $c_1 = \{\neg u_1, u_2, \neg u_2\}$.

Further construction considers at least one of the literals $u_i, \neg u_i$ forming the edges in E_i and at least two of the vertices $a_1[j], a_2[j], a_3[j]$ of triangle edges E_j being in a vertex cover V', $\forall i \in \{1, 2, \ldots, n\}$, $\forall j \in \{1, 2, \ldots, m\}$. Hence, the bound k is set to $k = n + 2m$ and $G = (V, E)$, where

$$V = \left(\bigcup_{i=1}^{n} V_i \right) \cup \left(\bigcup_{j=1}^{m} V'_j \right) \text{ and } E = \left(\bigcup_{i=1}^{n} E_i \right) \cup \left(\bigcup_{j=1}^{m} E'_j \right) \cup R.$$

Polynomial time boundedness follows easily from the construction of Vertex Cover instances built from restricted number of clauses and variables.

A mapping t; $t : U \longrightarrow \{\mathrm{T}, \mathrm{F}\}$, is assumed to be a satisfying truth assignment for a collection of clauses $C = \{c_1, c_2, \ldots, c_m\}$. If $t(u_i) = \mathrm{T}$, the vertex $u_i \in V'$. Dually, if $t(u_i) = \mathrm{F}$, the vertex $\neg u_i \in V'$. By satisfying each clause c_j, t is a satisfiable truth assignment to at least one its literal implying that there exists at least one edge in E''_j covered by V'. The endpoints of the edge non-incident to the vertex from related communication edge complement the previously established set V'. The construction is repeated for $\forall j \in \{1, 2, \ldots, m\}$.

Proving that $C \notin$ 3-SAT $\Rightarrow G \notin$ VC is equivalent to showing that $G \in$ VC $\Rightarrow C \in$ 3-SAT. Let $V' \subseteq V$; $|V'| \leqslant k$ be a vertex cover for G. By earlier observations, V' contains at least one vertex from V_i equivalent to T_i for $\forall i \in \{1, 2, \ldots, n\}$ and at least two vertices from V'_j equivalent to S_j for $\forall j \in \{1, 2, \ldots, m\}$. Thus, the minimal size of $|V'| = n + 2m$.
A truth assignment t evaluates u_i to true when $u_i \in V'$, otherwise, u_i is evaluated to false. A clause $c_j = \{x_j, y_j, z_j\}$ is satisfied on the condition that at least one of its literals is set true. Considering the triangle formed by edges from E''_j, only two of those can be covered by vertices from V', the remaining one is to be satisfied by the truth assignment of variables from U, u_i or its negation $\neg u_i$. Once being in V', the literal is assigned true by t. Since the argument is valid for all clauses c_j, C is satisfied. □

The problem of vertex cover can be approached in a number of alternative ways. It is rather simple to reduce Vertex Cover to Independent Set, where the vertex cover vertices do not form any edges, or Clique, where every two vertices are mutually adjacent. Vertex Cover, Independent Set and Clique tend to be all regarded either as variants or equivalent formulations of the same decision problem. Having proved Vertex Cover $\mathcal{NP}$-complete, $\mathcal{NP}$-completeness of the other two then follows easily.

Variants of Vertex Cover and their reformulations lead to arising decision problems concerned with vertex ordering. Vertex Cover is to be reduced to one of the initial problems in Hamiltonian graphs, offering the possibility of travelling from an arbitrary vertex to another on the condition that each vertex is only visited once. The problem of Hamiltonian Circuit is described and proved $\mathcal{NP}$-complete by reduction from Vertex Cover in the succeeding section.

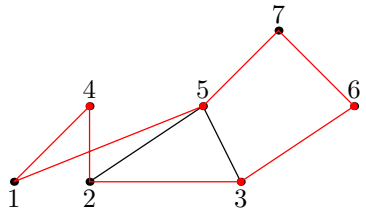
3.3 HC – Hamiltonian Circuit

The term "Hamiltonian" is chosen in reference to Sir William Rowan Hamilton, whose interest in cycles containing every graph vertex is demonstrated by a 1859 puzzle called "The Traveller's Dodecahedron." The game, based on Hamilton's observations, lies in creating a closed path using all 20 vertices labeled as cities.[5] Additionally, it facilitates the formulation of a definition of the circuits obtained alongside.

Definition 23 (Hamiltonian Circuit)**.** *Let $G = (V, E)$ denote an undirected graph. A Hamiltonian circuit is defined as a sequence $\langle v_1, v_2, \ldots, v_n \rangle$ of vertices such that its edges $\{\{\{v_i, v_{i+1}\} = e_i : \ 1 \leqslant i < n\} \cup \{v_n, v_1\}\} \subseteq E$ and $|V| = n$.*

The related decision problem (HC) asks if a given graph G is Hamiltonian, in other words, whether G contains a Hamiltonian circuit.

Example 2 (Instance of Hamiltonian Circuit)**.** *The figure below presents a Hamiltonian Circuit, drawn red, on previously introduced vertices.*



Formally, the sequence defining the Hamiltonian circuit can be written as $\langle 1, 4, 2, 3, 6, 7, 5 \rangle$. The subset $\{3, 4, 5, 6\}$ is still a valid vertex cover instance for the given graph.

[5] A concise introduction to history of Hamiltonian Circuit and Travelling Salesperson problem is given in [2], Chapter 1.

To complete the series of reductions presented at the beginning of the chapter, Vertex Cover, a covering problem, is to be polynomially reduced to Hamiltonian Circuit, the problem of vertex ordering. $\mathcal{NP}$-completeness of HC then follows from $\mathcal{NP}$-completeness of VC.

Theorem 4. *Hamiltonian Circuit is $\mathcal{NP}$-complete.*

Proof.

For HC $\in \mathcal{NP}$, it is sufficient for an NDTM to guess a sequence of vertices which is afterwards verified (in polynomial time) to be a Hamiltonian circuit in the given graph.

The reduction function f: VC $\longrightarrow$ HC characterises the transformation of a graph $G = (V, E)$ with vertex cover of size k; $0 < k \leqslant |V|$, or less to an arbitrary instance of HC represented by a Hamiltonian graph $G' = (V', E')$. Paralleling the previous proof, the procedure of construction of the transformation branches into three sets of components:

Choice making vertices A.
$A = \{a_i : \ 1 \leqslant i \leqslant k, \ a_i \in V\}$, where A arbitrarily selects k out of n vertices from V.

Cover testing components G'_E.
$G'_E = \bigcup_{e \in E} G'_e$; $G'_e = (V'_e, E'_e)$ and $e = \{u, v\}$;
$V'_e = \{(u, e, i), (v, e, i) : \ 1 \leqslant i \leqslant 6\}$,
$E'_e = \{\{(u, e, i), (v, e, i+1)\}, \{(v, e, i), (v, e, i+1)\} : \ 1 \leqslant i \leqslant 5\}$
$\cup \ \{\{(u, e, 3), (v, e, 1)\}, \{(v, e, 3), (u, e, 1)\}\}$
$\cup \ \{\{(u, e, 6), (v, e, 4)\}, \{(v, e, 6), (u, e, 4)\}\}$.

Collection of connecting edges E'_V and E''.
$E'_V = \bigcup_{v \in V} E'_v$; $E'_v = \{\{(v, e_{v[i]}, 6), (v, e_{v[i+1]}, 1)\} : \ 1 \leqslant i < \deg_G(v)\}$;
$\deg_G(v) := |\{e \in E : \ v \in e\}|$, the degree of a vertex v in G and $v[i]$ denotes an arbitrary mapping labelling the edges incident with v,
$E'' = \{\{a_i, (v, e_{v[1]}, 1)\}, \{a_i, (v, e_{v[\deg_G(v)]}, 6)\} : \ 1 \leqslant i \leqslant k, \ v \in V\}$.

The total number k of drawn vertices from A is verified to be a vertex cover for G by cover testing components from G'_E. Finally constructed Hamiltonian graph G' will only contain the cover testing component vertices of the form $(w, e, 1)$ and $(w, e, 6)$; $w = u$ or $w = v$; $\{u, v\} = e \in E$. By design of vertex

cover, the passage of a Hamiltonian circuit is reduced to one of the three cases giving locally unclosed cycles also referred to as paths:

$e = \{u, v\};\ u \in A,\ v \notin A.$
A single path: $(u, e, 1)$, $(u, e, 2)$, $(u, e, 3)$, $(v, e, 1)$, $(v, e, 2)$, $(v, e, 3)$, $(v, e, 4)$, $(v, e, 5)$, $(v, e, 6)$, $(u, e, 4)$, $(u, e, 5)$, $(u, e, 6)$.

$e = \{u, v\};\ u \in A,\ v \in A.$
Two distinct paths: $(u, e, 1)$, $(u, e, 2)$, $(u, e, 3)$, $(u, e, 4)$, $(u, e, 5)$, $(u, e, 6)$;
$(v, e, 1)$, $(v, e, 2)$, $(v, e, 3)$, $(v, e, 4)$, $(v, e, 5)$, $(v, e, 6)$.

$e = \{u, v\};\ u \notin A,\ v \in A.$
A single path: $(v, e, 1)$, $(v, e, 2)$, $(v, e, 3)$, $(u, e, 1)$, $(u, e, 2)$, $(u, e, 3)$, $(u, e, 4)$, $(u, e, 5)$, $(u, e, 6)$, $(v, e, 4)$, $(v, e, 5)$, $(v, e, 6)$.

It is worth noting that the connecting edges from E'_v then create a sole path for every vertex $v \in E$ in a newly constructed G'. The construction is completed once the first and last vertices from E'_V are joined by E''. Hence $G' = (V', E')$, where

$$V' = A \cup \left(\bigcup_{e \in E} V'_e \right) \quad \text{and} \quad E' = \left(\bigcup_{e \in E} E'_e \right) \cup E'_V \cup E''.$$

Polynomial time boundedness follows from the construction of Hamiltonian graph G' on the basis of G, a graph with a vertex cover of restricted size k and a total number n of vertices in V.

If a set V_G denotes a vertex cover of a graph G, bounded by k, no addition of edges on already defined vertices would necessitate an expansion of the original set in order to cover graph vertices. Labelling vertex cover elements as $v_1, v_2, \ldots, v_k$, the choice of what edges to include in HC is made in accordance with the rules introduced in the cover testing components as V_G is a vertex cover of G and the endpoints create components of Hamiltonian circuit. Therefore, the needed edges are selected from E'_{v_i} in the following manner:

Hamiltonian circuit edges:
$E_{HC} = \{\{a_i, (v_i, e_{v_i[1]}, 1)\} : 1 \leqslant i \leqslant k\}$
$\cup\ \{\{a_{i+1}, (v_i, e_{v_i[\deg_G(v_i)]}, 6)\} : 1 \leqslant i < k\} \cup \{a_1, (v_k, e_{v_k[\deg_G(v_k)]}, 6)\}.$

For completion of the proof it is necessary to verify whether the constructed set of edges E_{HC} generates a sequence defining a Hamiltonian circuit in the given graph. The edges in E_{HC} are chosen to enter and depart from every vertex in VC exactly once. Since at least one endpoint of every edge is in VC, it is impossible to avoid a vertex from V'. In such a way, a path between the succeeding vertices forming a complete cycle is created. Hence, E_{HC} corresponds to a Hamiltonian circuit in G'.[6]

Implication $G \notin \text{VC} \Rightarrow G' \notin \text{HC}$ is equivalently phrased and proved as $G' \in \text{HC} \Rightarrow G \in \text{VC}$. Let $\langle v_1, v_2, \ldots, v_n \rangle$ denote a Hamiltonian circuit in G', thus $|V'| = n$. By construction of vertex cover testing components, any path which starts and ends with a vertex from $a_1, a_2, \ldots, a_k$, its only vertices from A, is divided into k different paths corresponding to distinct vertices $v_i \in V$. Since Hamiltonian circuit is to comprise all vertices, it is inevitable that it also contain the cover testing component vertices, implying a path associated with an edge endpoint is a part of Hamiltonian circuit. Therefore, at least one of the endpoints forming the relevant edge is included in vertex cover set of size k. □

The decision problem of Hamiltonian Circuit is closely tied to the problem of Hamiltonian path, the sequence of vertices $\langle v_1, v_2, \ldots, v_n \rangle$, where $\{\{v_i, v_{i+1}\} = e_i : \ 1 \leqslant i < n\} \subseteq E, \ |V| = n$. $\mathcal{NP}$-completeness of the latter then follows from a slightly modified construction used in the problem of Hamiltonian circuit. A detailed account of alternations is described in [6].

[6] Having restrained the transformation f to Example 2 with a vertex cover of size 4, the obtained set of Hamiltonian circuit edges E_{HC} is the following:

$$E_{HC} = \{a_1, (v_1, e_{v_1[1]}, 1)\}, \ \{a_2, (v_2, e_{v_2[1]}, 1)\}, \ \{a_3, (v_3, e_{v_3[1]}, 1)\}, \ \{a_4, (v_4, e_{v_4[1]}, 1)\},$$
$$\{a_2, (v_1, e_{v_1[\deg_G(v_1)]}, 6)\}, \ \{a_3, (v_2, e_{v_2[\deg_G(v_2)]}, 6)\}, \ \{a_4, (v_3, e_{v_3[\deg_G(v_3)]}, 6)\},$$
$$\{a_1, (v_4, e_{v_4[\deg_G(v_4)]}, 6)\}.$$

More specifically, by association of vertex cover vertices $v_1 := 3$, $v_2 := 4$, $v_3 := 5$, $v_4 := 6$ with their respective degrees $\deg_G(3) = 3$, $\deg_G(4) = 2$, $\deg_G(5) = 4$, $\deg_G(6) = 2$,

$$E_{HC} = \{a_1, (3, \{2, 3\}, 1)\}, \ \{a_2, (4, \{1, 4\}, 1)\}, \ \{a_3, (5, \{1, 5\}, 1)\}, \ \{a_4, (6, \{3, 6\}, 1)\},$$
$$\{a_2, (3, \{3, 6\}, 6)\}, \ \{a_3, (4, \{2, 4\}, 6)\}, \ \{a_4, (5, \{5, 7\}, 6)\},$$
$$\{a_1, (6, \{6, 7\}, 6)\}.$$

Thus, the construction verifies that the guessed sequence $\langle 1, 4, 2, 3, 6, 7, 5 \rangle$ forms a Hamiltonian circuit in G. By design of the graph, the presented instance of Hamiltonian Circuit is unique up to isomorphism.

Considering a directed graph, every original edge $e = \{u, v\} \in E$ decomposes into two oriented ones, (u, v) and (v, u), to provide a variety of problems connected with Hamiltonian Circuit. In fact, all undirected variants can be reduced to their directed counterparts and vice versa.

This chapter introduced and proved $\mathcal{NP}$-completeness of several decision problems, ranging from the domain of logic to graph theory to border the area of optimisation, in the given order:

$$\text{HC} \leqslant_p [7] \text{ SAT} \leqslant_p \text{3-SAT} \leqslant_p \text{VC} \leqslant_p \text{HC}.$$

By Lemma 1, transitivity of polynomial time reducibility denoted $\leqslant_p$, it is evident that any decision problem from the provided set can be reduced to another. Their membership in $\mathcal{NP}$ then yields $\mathcal{NP}$-completeness of the whole set. Equally, by reduction from k-SAT, the property of solvability in polynomial time for $k \leqslant 2$ and $\mathcal{NP}$-complete for $k \geqslant 3$ was conserved for generalised Vertex Cover and Hamiltonian Circuit, where k denotes the size of maximum vertex cover for a given graph.

The proof strategies illustrated in the present reduction path included Local Replacement (SAT $\leqslant_p$ 3-SAT), based on rewriting a given instance so that its principal structure is replaced by a different but equivalent collection of instances, and Component Design (3-SAT $\leqslant_p$ VC, VC $\leqslant_p$ HC), lying in defining a number of components either designed to form a newly structured instance or exchange information about the satisfiability of a certain condition put on the structure forming components.

Departing from the basic $\mathcal{NP}$-complete decision problems, the following chapter aims at proving $\mathcal{NP}$-completeness of other problems with intent to relate problems from seemingly disconnected mathematical disciplines.

[7] The statement follows from Cook's Theorem.

Chapter 4

Other $\mathcal{NP}$-complete Problems

Previous chapter presented several of classical $\mathcal{NP}$-complete problems while demonstrating proof strategies of Local Replacement and Component Design. This section is to derive new $\mathcal{NP}$-complete results from the existing ones.

Reduction by Restriction is another promising approach in terms of problem transformations. Much simpler in nature, the technique proves that a known $\mathcal{NP}$-complete problem is a special case of a target decision problem. A preferable reduction function, a rather obvious one-to-one mapping, ensures close correspondence of instances found in both problems.

A proof using the technique of Restriction, structured as announced in Chapter 3, will be presented in the process of proving $\mathcal{NP}$-completeness of a widely known optimisation problem of Travelling Salesperson.

4.1 TS – Travelling Salesperson

The foundations of Travelling salesperson problem were implicitly introduced by Hamilton's puzzle "The Traveller's Dodecahedron," marking the vertices as cities, asking for a closed path visiting each city once, corresponding to Hamiltonian Circuit. Furthermore, the edges are assigned a non-negative integer value representing either distance or cost of the travel. A better insight into the problematic is provided by formalisation of the Travelling Salesperson round.

Definition 24 (Travelling Salesperson Round)**.** *Let* $C = \{c_1, c_2, \ldots, c_m\}$ *denote a finite set of vertices with a "length function"*[1] $l; l : C \times C \longrightarrow \mathbb{N}$,

[1] Function l need not to be a metric as subadditivity of l is not required.

where $\mathbb{N} = \{0, 1, 2, \dots\}$ *is the set of natural numbers. Travelling Salesperson round is defined as an ordering* $\left\langle c_{\pi(1)}, c_{\pi(2)}, \dots, c_{\pi(m)} \right\rangle$; π *is a permutation on* $\{1, 2, \dots, m\}$. *The length* L *of Travelling Salesperson round is given by:*

$$L = \left(\sum_{i=1}^{m-1} l(c_{\pi(i)}, c_{\pi(i+1)}) \right) + l(c_{\pi(m)}, c_{\pi(1)}).$$

The problem of Travelling Salesperson (TS) then asks if there exists a round $\left\langle c_{\pi(1)}, c_{\pi(2)}, \dots, c_{\pi(m)} \right\rangle$ such that a total length $L \leqslant b; b \in \mathbb{N}$.

Theorem 5. *Travelling Salesperson is* $\mathcal{NP}$*-complete.*

Proof.

Travelling Salesperson is in $\mathcal{NP}$ since a non-deterministic TM can provide an ordering of cities such that the length of the tour L is bounded by a non-negative integer b and the guessed instance is verifiable in a deterministic polynomial time.

Let f : HC $\longrightarrow$ TS denote a reduction function. An instance of Hamiltonian Circuit, E_{HC}, is defined by $G = (V, E)$; $E_{HC} \subseteq E$, $|V| = m$. A TS instance is characterised by a set of cities C corresponding to V with "length function" l set to $l(v_i, v_j) = 1$ for $\{v_i, v_j\} \in E$ and $l(v_i, v_j) = 2$ for $\{v_i, v_j\} \notin E$; $v_i, v_j \in C$. The length L is bounded by $b = m$.

Reduction in polynomial time follows from a total number of tested combinations of $v_i, v_j \in C$ being $\frac{m(m-1)}{2}$. The evaluation of every edge necessitates examination of at maximum $\frac{m(m-1)}{2}$ edges formed on m vertices in the given graph G.

By design, a Hamiltonian circuit represented by $\langle v_1, v_2, \dots, v_m \rangle$, is a TS round of length $L = \left(\sum_{i=1}^{m-1} l(v_i, v_{i+1}) \right) + l(v_m, v_1) = m$.

In an opposite way, if an ordering $\langle v_1, v_2, \dots, v_m \rangle$, is not an instance of HC for G, at least one of the edges formed thereafter is, by definition of l, evaluated to 2. Necessarily, $L > m$ which contradicts the fact that L is bounded by $b = m$. Hence, $f(G) \notin$ TS. □

By restriction of TS to HC, the reduction path of equivalently hard problems extends to:

$$\text{SAT} \leqslant_p \text{3-SAT} \leqslant_p \text{VC} \leqslant_p \text{HC} \leqslant_p \text{TS}.$$

Travelling Salesperson has a number of variants operating on the basis of limitations put on the distances between the cities and bound b, Bottleneck

TS, or further specification of the "length function" such as Euclidean metric based Geometric TS. Additional $\mathcal{NP}$-complete problems arise when directed graphs are considered.

Finally, replacement of a bound b by a minimum possible tour length leads to $\mathcal{NP}$-hard combinatorial optimisation problem as already noted for Vertex Cover. The theory of approximation and relevant algorithms are discussed in detail in [3].

4.2 EC – Ensemble Computation

The problem of Ensemble Computation is of different nature when compared to previously illustrated decision problems. An expression constructed from elements and subsets of elements of a given finite set, joined by union operations, lays the basis for formulation of another decision problem from the domain of code generation and program optimisation.

Definition 25 (Ensemble Computation Sequence)**.** *An Ensemble Computation sequence S of length j is a sequence $\langle z_1, z_2, \ldots, z_j \rangle$ of subsets of a finite set A so that each z_i can be decomposed into $x_i \cup y_i$, where $x_i \cap y_i = \emptyset$ and each x_i, y_i is either $\{a\}$ for $a \in A$ or one of z_ks for $k < i$.*

The problem of Ensemble Computation (EC) then asks if there is an ensemble computation sequence S of length bounded by a positive integer l such that for $\forall c \in C$; a collection of subsets of A, there is an identical subset $z_i \in S$. In other words, EC decides minimisation of the number of needed operations on given conditions on input (A, C, l), where A denotes a finite set, C a collection of its subsets and l stands for a bound on desired number of operations.

Theorem 6. *Ensemble Computation is $\mathcal{NP}$-complete.*

The proof, based on [6], follows the already presented uniform structure.

Proof.

The problem of EC $\in \mathcal{NP}$ as an NDTM's guess can be verified in deterministic polynomial time.

Let f : VC $\longrightarrow$ EC denote a reduction function. An instance of Vertex Cover, given by $G = (V, E)$ and $k \leqslant |V|$, is to be locally replaced. The transformation defines a new element a_0; $a_0 \notin V$, such that $\{u, v\} \longmapsto \{a_0, u, v\}$ for $\forall e = \{u, v\} \in E$; $u, v \in V$. Thus, an instance of Ensemble Computation,

specified by A, non-empty finite set of elementary units, C, a collection of its subsets and l, an integer bound for cardinality of C, can be written in the form:

$$A = V \cup \{a_0\},$$

$$C = \{\{a_0, u, v\} : \{u, v\} \in E\},$$

$$l = k + |E|\,.$$

Polynomial time boundedness of construction of an arbitrary EC instance is ensured by at maximum $\frac{|V|(|V|-1)}{2}$ edges on $|V|$ vertices that are to be verified to belong to E.

If V' is a vertex cover for G of size k or less, it can be assumed that $|V'| = k$ since V' is vertex addition invariant. The sets V' and and E; $|E| = m$ are then represented by $v_1, v_2, \ldots, v_k$ and $e_1, e_2, \ldots, e_m$ respectively. Each edge e_j comprises at least one vertex from V' and can be rewritten as $e_j = \{u_j, v_{r[j]}\}$; $r : \mathbb{N} \longrightarrow \mathbb{N}$; $r[j] \leqslant k$. Thus, a sequence S of length $k + m = k + |E| = l$, corresponding to an arbitrary EC instance is obtained.

$$\begin{aligned} S &= \left\langle \{a_0\} \cup \{v_1\}, \ldots, \{a_0\} \cup \{v_k\}, \{u_1\} \cup z_{r[1]}, \ldots, \{u_m\} \cup z_{r[m]} \right\rangle \\ &= \left\langle z_1, \ldots, z_k, z_{k+1}, \ldots, z_l \right\rangle . \end{aligned}$$

Therefore, $G = (V, E) \in \mathrm{VC} \Rightarrow (A, C, l) \in \mathrm{EC}$.

In an opposite way, showing that $G = (V, E) \notin \mathrm{VC} \Rightarrow (A, C, l) \notin \mathrm{EC}$ is equivalent to departing from an instance of Ensemble Computation and reducing it to a corresponding VC instance. For that matter, let S be a minimum ensemble computation sequence optimised in such a way that occurrence of forms constructed consecutively is rare. If there is such z_i in S, certainly $z_i \notin C$, the only possibility is that $\{u, v\} \in E$. Hence, $\{u, v\}$ is to appear in $\{a_0, u, v\} = \{a_0\} \cup z_i$ or $z_i \cup \{a_0\}$,[2] by design of C and will not reoccur in another operation since the sequence S is minimal. That yields possibility of reducing the number of operations in $z_i = \{u\} \cup \{v\}$ to $z_i = \{a_0\} \cup \{u\}$ and $\{a_0, u, v\} = \{a_0\} \cup z_i$ to $\{a_0, u, v\} = \{v\} \cup z_i$ without augmenting the length l contradicting the minimality of the sequence S. Consequently, S is to contain only the above optimised forms. Since $|C| = |E|$ and $|c| = 3\ \forall c \in C$, there are $|E|$ operations of the form $\{v\} \cup z_i$ and $j - |E| \leqslant l - |E| = k$

[2] The order of operands is further disregarded.

operations of the form $\{a_0\} \cup \{u\}$ in S. A vertex cover is then phrased as a set $V' = \{\, z_i = \{a_0\} \cup \{u\} : \; u \in V, \; z_i \in S \,\}$. By previous observation, cardinality of V' is at maximum $l - |E| = k$ and V' comprises at least one endpoint from every edge $\{u, v\} \in E$. □

The placement of the newly obtained $\mathcal{NP}$-complete result in the chain of previous reductions, relating decision problems from the area of code generation to logic, graph theory and optimisation, is illustrated by:

$$\begin{aligned} \text{SAT} \leqslant_p \text{3-SAT} \leqslant_p \text{VC} \leqslant_p \text{EC} \\ \leqslant_p \text{HC} \leqslant_p \text{TS}. \end{aligned}$$

According to [6], an $\mathcal{NP}$-complete modification of Ensemble Computation can be provided if each subset $c \in C$ has at maximum three members or the condition of $x_i \cap y_i = \emptyset$ does not hold under the same restriction.

4.3 QDE – Quadratic Diophantine Equations

Ever since David Hilbert's 1900 posting of 23 unsolved problems, the field of number theory had been challenged to find an algorithm to decide whether a given Diophantine equation, an algebraic indeterminate equation in several variables, with integral coefficients, is solvable in integers or not. A proof by Yuri V. Matiyasevich, built on Julia Robinson, Martin Davis and Hilary Putnam's results, classified Hilbert's Tenth Problem as formally undecidable in 1970. A wide range of decision problems arose from specifications of the degree of the equations or the number of unknowns. The problem of Quadratic Diophantine Equations restrains both numbers to two. To refer to a proof of $\mathcal{NP}$-completeness of the announced variant, formalisation of the equations in question is needed.

Definition 26 (Quadratic Diophantine Equation)**.** *A quadratic equation in two variables* $ax^2 + by - c = 0; \; a, b, c \in \mathbb{N}$ *is Diophantine precisely if* $\exists x, y \in \mathbb{N} : \; ax^2 + by = c$.

The problem of Quadratic Diophantine Equations (QDE) asks if a given quadratic equation $ax^2 + by - c = 0; \; a, b, c \in \mathbb{N}$, is Diophantine.

Theorem 7. *Quadratic Diophantine Equations problem is* $\mathcal{NP}$*-complete.*

Proving $\mathcal{NP}$-completeness in the field of number theory is rather complex since an instance in binary encoding scheme needs to be represented as a computational problem, in this case solvability of quadratic Diophantine equations. It may be preferable to attempt the proofs directly.[3]

The original proof, given in [1], nevertheless presents a possibility of reduction of a 3-SAT variant[4] to QDE. The following section briefly remarks on the properties of eventual reduction.

Transformation function f : 3-SAT $\longrightarrow$ QDE is defined algorithmically such that every satisfiable formula φ in 3-CNF forces natural number solutions to the quadratic equation $p(x, y) = 0$ it generates. Equally, solvability of a quadratic Diophantine equation corresponds to satisfiability of a Boolean formula given by Conversion Lemma. This is ensured by successive systems of definitions converting an arbitrary instance of 3-SAT to an arbitrary instance of QDE.

Polynomial time boundedness of the reduction follows from a limited number of variables obtained after deletion of all duplications and unconditionally satisfiable clauses from φ. Complexity of the needed operations – addition, multiplication and division as well as all their inputs are polynomially bounded.

The remaining condition of satisfiability of φ being equivalent to solvability of $p(x, y) = 0$ in natural numbers follows from the proof of correctness of the transformation algorithm.

The algorithm is proved correct by a succession of three number theory based lemmas yielding one-to-one correspondence of solutions to the systems of conditions provided by lemmas and satisfiable truth assignment to φ. A proof of "Lemma 1.,"the only omitted part, can be seen in Appendix.

Reduction of Boolean formulae satisfiability problem to number problem solvability not only relates logic to number theory but also provides a better insight into the nature of numerical problems and their intractability. The design of $\mathcal{NP}$-completeness and transitivity of polynomial reducibility clas-

[3] Major steps of the proof are discussed in [12].

[4] In the original proof, an arbitrary clause $c_j \in C$ has at maximum 3 literals. It is straightforward to reduce "At Maximum 3-SAT" to 3-SAT. It suffices that $\forall\ |c_j| < 3$ be replaced by $\left|c'_j\right| = 3$ where the latter is built by adding new atoms, unique for each clause, to the former one so that the truth evaluation is conserved. Limited number of used atoms and operations resulting thereafter yield "At Maximum 3-SAT" $\leqslant_p$ 3-SAT.

sifies the problem of Quadratic Diophantine Equations as equally hard as the problem of Travelling Salesperson, Ensemble Computation, SAT or any other decision problem given below:

$$\begin{aligned} \text{SAT} \leqslant_p \text{3-SAT} \leqslant_p \text{QDE} \\ \leqslant_p \text{VC} \leqslant_p \text{EC} \\ \leqslant_p \text{HC} \leqslant_p \text{TS}. \end{aligned}$$

The problem of Quadratic Diophantine Equations can be viewed in terms of decision problems concerning solvability of Diophantine equations either of lower or higher degree or in less or more variables. While the problems of Linear Diophantine Equations, regardless of the number of variables, and Diophantine Equations in One Unknown, both with integral solutions are in $\mathcal{P}$, two variables in Diophantine Equation of second degree with natural solutions yield $\mathcal{NP}$-completeness. The intractability of related problems even increases when facing a generally stated decision problem. Consisting in finding integer solutions, its undecidability for 13 unknowns was proved by Yuri V. Matiyasevich and Julia Robinson.

4.4 Concluding Remarks

Computational complexity theory was shown to be related to many branches of modern mathematics. The results of this section demonstrated $\mathcal{NP}$-complete problems distributed over optimisation, code generation and number theory, all tied to classical logic and graph theory. Newer results showed, by reduction of SAT to Minesweeper[5], that even the category of games and puzzles is involved. The notion of $\mathcal{NP}$-completeness thus relates a broad collection of decision problems distinct in formulation but equivalently hard to decide in nature. Therefore, a progress lying in providing an efficient algorithm operating in polynomial time for an arbitrary $\mathcal{NP}$-complete problem in any of the disciplines would mean a progress in all areas, including $\mathcal{P}$ vs. $\mathcal{NP}$.

[5] To be found in [9], [10] and [5].

Appendix

"Lemma 1."

Lemma 2 ("Lemma 1."). *Let* $\tau;\ 2 \nmid \tau,\ x \in \mathbb{Z},\ k \geqslant 3$. *Then,* $((\tau - x)(\tau + x) \equiv 0\ mod\ 2^{k+1}) \Leftrightarrow ((\tau - x) \equiv 0\ mod\ 2^k) \vee ((\tau + x) \equiv 0\ mod\ 2^k)$.

Proof.

"$\Rightarrow$" The expression $(\tau - x)(\tau + x) \equiv 0 \bmod 2^{k+1}$ is equivalent to $2^{k+1}|(\tau - x)(\tau + x)$. $2|2^{k+1} \Rightarrow 2|(\tau - x)(\tau + x) \Rightarrow 2|(\tau - x),\ 2|(\tau + x)\ (2 \nmid x)$. Without loss of generality, let $2^k|(\tau - x)$. Since $2|(\tau + x) \Rightarrow 2^{k+1}|(\tau - x)(\tau + x)$.

"$\Leftarrow$" If $2^{k+1}|(\tau - x)(\tau + x) \Rightarrow \exists m, n \in \mathbb{N} :\ m + n \geqslant k + 1 :\ 2^m|(\tau - x)$ and $2^n|(\tau + x)$. For $m \geqslant k$, the statement holds. Therefore, let $m \leqslant k - 1$. $2^m|(\tau - x) \Rightarrow 2^{2m}|(\tau - x)^2 \Rightarrow 2^{k+1}|(\tau - x)^2$ on the condition that $2m \geqslant k + 1 \Leftrightarrow 2(k - 1) \geqslant k + 1 \Leftrightarrow k \geqslant 3$. As $2^{k+1}|(\tau - x)(\tau + x)$ and $2^{k+1}|(\tau - x)^2$, $\Rightarrow 2^{k+1}|(\tau - x)(\tau + x) + (\tau - x)^2 = 2\tau^2 - 2\tau x = 2\tau(\tau - x);\ 2 \nmid \tau \Rightarrow 2^k|(\tau - x)$. □

Bibliography

[1] Adleman, L. M., Manders, K. L.: "NP-complete Decision Problems for Quadratic Polynomials", STOC '76: Proceedings of the Eighth Annual ACM Symposium on Theory of Computing, ACM, Hershey, 1976. 23–29.

[2] Applegate, D. L., Bixby, R. E., Chvátal, V., Cook, W. J.: *The Traveling Salesman Problem: A Computational Study (Princeton in Applied Mathematics)*, Princeton University Press, Princeton, 2007.

[3] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A. Protasi, M.: *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*, Springer, Berlin, 2002.

[4] Cook, S. A.: "The Complexity of Theorem Proving Procedures", STOC '71: Proceedings of the Third Annual ACM Symposium on Theory of Computing, ACM, New York, 1971. 151–158, March 2009, Available at: http://www.cs.toronto.edu/∼sacook/homepage/1971.pdf.gz.

[5] Fix, J. D., McPhail, B.: "Offline 1-Minesweeper is NP-complete", Unpublished Manuscript, May 2009, Available at: http://people.reed.edu/jimfix/papers/1MINESWEEPER.pdf.

[6] 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.

[7] Hoory, S., Szeider, S.: "Computing Unsatisfiable k-SAT Instances with Few Occurrences per Variable", Theoretical Computer Science, Vol. 337, No. 1-3, Elsevier Science Publishers Ltd., Essex, 2005. 347–359.

[8] Karp, R.: "Reducibility Among Combinatorial Problems", *Complexity of Computer Computations*, Miller, R. E., Thatcher, J.W., Eds., Plenum

Press, New York, 1972. 85–103, March 2009, Available at: http://www.cs.berkeley.edu/∼luca/cs172/karp.pdf.

[9] Kaye, R.: "Minesweeper is NP-complete", The Mathematical Intelligencer, Vol. 22, No. 2, June, Springer, New York, 2000. 9–15.

[10] Kaye, R.: "Some Minesweeper Configurations", Boletim Sociedade Portuguesea de Mathemática, (Número especial), Lisbon, 2007. 181–189, May 2009, Available at: http://for.mat.bham.ac.uk/R.W.Kaye/minesw/minesw.pdf.

[11] Kratochvíl, J., Savický, P., Tuza, Z.: "One More Occurrence of Variables Makes Satisfiability Jump from Trivial to NP-complete", SIAM J. Comput. Vol. 22, Is. 1, Society for Industrial and Applied Mathematics, Philadelphia, 1993. 203–210.

[12] Manders, K. L.: "Computational Complexity of Decision Problems in Elementary Number Theory", *Model Theory of Algebra and Arithmetic: Proceedings of the Conference on Applications of Logic to Algebra and Arithmetic Held at Karpacz, Poland, September 1 – 7*, 1979, Vol. 834/1980, Springer, Berlin, Heidelberg, 1980. 211–227.

[13] van Melkebeek, D.: *A Survey of Lower Bounds for Satisfiability and Related Problems*, Now Publishers Inc., Hanover, 2007.