Charles University in Prague

Faculty of Mathematics and Physics

NP Search Problems

Master Thesis

Author: Tomáš Jirotka

Supervisor: Jan Krajíček

Prague, May 2011.

ISSN 1433-8092

Abstract: The thesis summarizes known results in the field of NP search problems. We discuss the complexity of integer factoring in detail, and we propose new results which place the problem in known classes and aim to separate it from PLS in some sense. Furthermore, we define several new search problems.

Keywords: Computational complexity, TFNP, integer factorization.

Contents

I. Preface 4

II. Complexity Theory Preliminaries 6

II.1 Search Problems 8

II.2 Local Search 12

II.3 Parity Lemma Based Search Problems 15

II.4 Pigeonhole Principle 21

II.5 Separation of the Classes 22

III. An Upper Bound for Integer Factoring 25

IV. A Lower Bound for Integer Factoring 30

V. Further Related Problems 35

VI. Concluding Remarks and Open Problems 41

References 43

Chapter I

Preface

In this thesis we study a specific part of complexity theory which concerns with time consummation of the most optimal algorithms for solving problems.

It has always been a kind of challenge to compute fast and smartly. This effort has led people to create effective algorithms and tricky methods. Unfortunately, several problems have resisted thousands of mathematicians who have not found any recipe to solve them in feasible time. And these are precisely the question we study in the following chapters.

Chapter II is dedicated to introduction of the reader into complexity theory in general, and presents seminal results about our main topic, **NP**-search problems. We define a few subclasses of these problems, and show a way how they are connected.

Then we focus on an important question of modern computer science: *Is integer factorization really so hard as we expect?*

Chapters III and IV partially respond to that question, although they do not present any shocking new theorems. In the third chapter we show an upper bound on its complexity which would mean a negative answer, however the next chapter contains a new approach to estimating a lower bound of the complexity of integer factorization.

In Chapter V we introduce new related questions which can be a subject of further research as well as the list of open problems in Chapter VI.

Within the thesis we use standard notation. Natural numbers are denoted by $\mathbb{N}$, whereas the finite set of integers from 0 up to $n-1$ is referred as $[n]$. Generally, a set S has cardinality $\#S$, since we use $|x|$ for the bit length of the number x. We also utilize a symbol GF_q to denote the finite field of size q.

Without explanation we use the big-oh notation which is a standard in the branch. Shortly say that $O(1)$ stands for a positive constant, or $O(n)$ means a linear function,

$n^{O(1)}$ a polynomial etc. Likewise, the reader should be familiar with some basic facts on graph theory, algebra and number theory.

Chapter II

Complexity Theory Preliminaries

In computational complexity theory we investigate algorithms for solving problems which are in some sense optimal. There are usually two ways how to express this: the running time of the algorithm is minimal in comparison to any other algorithm which solves the problem, or it requires the least amount of memory. In many cases these two conditions are not both compatible. Fast algorithms frequently need large memory and vice versa.

However, we do not need to know the best algorithm for some problem. A typical example is a problem called ***Integer-Factoring*** when we are given a positive integer and we are asked to find its prime factors (or some non-trivial factor). There are a lot of algorithms which can solve this task, but we do not know if some of them is the best one.

In this section we consider only decision problems. These are questions of the form: is a word x member of a set L called language? Obviously, we have three possible answers: yes, no, and do not know. From the knowledge of running time or needed memory space of the optimal algorithm in the worst case we classify the problems in some realm. The most famous classes are **P** and **NP**.

The complexity class **P** contains all decision problems which are solvable in polynomial time (sometimes abbreviated p-time) with respect to the length of input.

First, we should explain what a deterministic Turing machine is. It is an abstract computational model. We can imagine it as a program working on an infinite tape divided in the squares where can be written letters by a read-write head. For a formal definition we need three non-empty finite sets: the set of symbols A called alphabet (typically we have only two – 0 and 1 – and an empty symbol b), a set of states Q, and a transition function

$$\delta : Q \times A \to Q \times A \times \{L, R\}$$

where the symbols R and L determine if the head should move to the right or to the left on the tape. In each step it can move by only one square. The total number of moves during the whole computation is the running time, and the number of visited squares on the tape can be viewed as the required memory space. The set of states, Q, also contains an initial state q_0 and two final states – one for accepting state, and one for rejecting state.

Now for the formal definition of **P** let us have a subset of all binary strings of arbitrary finite length, $L \subseteq \{0,1\}^*$. Then for some binary word x of length n, $x \in \{0,1\}^n$, we want to decide whether x is in L. If this task can be answered by deterministic Turing machine in time $t(n) = O(n^k)$ for all n and for all x of length n, and some fixed $k > 0$, then $L \in \mathbf{P}$.

In a similar way we may define also other classes by replacing the time function $t(n)$. Usual choices are $\exp(n^{O(1)})$, $\exp(\log^{O(1)}(n))$ etc.

A weaker concept than decision machine is an accepting machine. Its computation is nearly the same, but it terminates only if $x \in L$. In the opposite case we will not get any answer. This notion is closely connected to the nondeterministic computation. Nondeterministic Turing machine has in its set of states a nondeterministic state $q_?$ in which it has more options what to do next. Well, it is possible that its decision would be wrong, and so it would get "no". We can imagine the set of all computations as a tree. If the input x really lies in the set L, then some of the leaves are labeled by "yes". Due to some bad decisions we could miss them. So we say that the nondeterministic Turing machine accepts the input x if and only if there exists an accepting computation. And the length of the shortest path leading to an accepting state is the running time. If such computation does not exist, then the running time is set to be infinite, $t(x) = \infty$.

Now we are able to formulate the definition of the class **NP**. Let $L \subseteq \{0,1\}^*$ is a language. We say that $L \in \mathbf{NP}$ iff there exists a nondeterministic Turing machine N which accepts L and for all $x \in L$ of arbitrary length n the running time is $t(x) = = O(n^k)$ for some fixed $k > 0$.

The most important question in the field is whether $\mathbf{P} = \mathbf{NP}$? Obviously the class **P** is contained in **NP**, but the opposite implication is an open problem.

Now we shall introduce a way of reducing a problem to another one. Suppose we have two languages, i.e. two subsets L_1 and L_2 of $\{0,1\}^*$, and also two machines

M_1 and M_2 for deciding whether $x \in L_1$, or $x \in L_2$ respectively. We say that L_1 is polynomial time reducible to L_2 (and write $L_1 \leq_{\mathrm{p}} L_2$) if and only if there exists a p-time function f such that for all x it holds $x \in L_1 \Leftrightarrow f(x) \in L_2$.

In addition, if we have two p-time functions which can be used for two-way translation of inputs of the problems, then these problems are polynomial time equivalent.

We will see a generalisation of the p-time reducibility in the section about search problems. But the main idea is always the same.

When there is a problem in the class **NP** such that any other problem in **NP** is polynomial time reducible to the first one, we call it **NP**-complete. The set of **NP**-complete problems consists of hard problems which have a polynomial witness.

Complete problems have a special importance, because in some sense they mirror the whole class. Thus when a new class is introduced there is an effort to find some complete problem for the class.

For more information on the basics of complexity theory reader should consult, for example, Papadimitriou's book [P93] or Krajíček's one [K95].

II.1 Search Problems

In many situations the concept of decision problems is weak or unnatural. For example we may want to compute the sum of two numbers. In fact it is possible to ask for each bit of the result using "yes-no" questions (*Is the i-th bit equal to* 1*?*) but this is time-consuming. Let us ask for the result directly. Such computation needs more time than the previous one, but it suffices to do it only once.

In this case we have seen that the time complexity of one computation has increased. But that is not a rule.

Usually, when we proceed from a decision to a search problem, we receive more information in the solution. Consider the most famous problem ***SAT*** which is **NP**-complete [C71]. It is given a Boolean formula φ and we want to decide whether it is satisfiable. But in the search analogy we need to find some satisfying assignment or say that such assignment does not exist. Having solved the search problem we also have the answer to the decision one.

It should be mentioned that this reducibility is not automatic. Later, we will see several search problems which are not equivalent to any decision problem modulo some conditions.

The thesis is dedicated to the class of **NP**-search problems which is the most studied set of search problems. This class is frequently denoted by **FNP** where the letter F stands for functional.

Let us have a **P**-predicate $\psi(x, y)$. For any x of length $n = |x|$ we must be able to find y of length polynomial in n, i.e. $|y| = n^{O(1)}$, or say that no such y can exist. Moreover the predicate ψ must be decidable in polynomial time in the length of its parameters, and hence in n. Then finding y for an instance x is an **NP**-search problem.

In a similar way, the class **FP** of all polynomial-time solvable search problems is defined.

A simple example is: For given x find y such that $\psi(x, y) \equiv x = 2y$ holds. That is a linear equation and there should be no problem with finding convenient y if it exists.

But in the input x there can be encoded a Boolean formula and ψ can require y to be a satisfying assignment for that formula. It is obvious that a solution may not exist, and if we were able to solve this problem, then we could solve ***SAT*** too.

In some cases we know that the solution to the problem must exist, because it follows from an existential theorem. This is a very useful information, because it motivates us for solving the problem. These search problems are called "total" and in the case of **NP**-search problems we denote them intuitively **TFNP**.

This class is not very nice. There are a lot of different reasons for being a member of **TFNP** starting with combinatorial lemmas, followed by number theoretic results and ending with geometric or optimisation problems. Shortly, nearly any existential theorem with easily verifiable solution from all of mathematics can be translated into a search problem.

This was a motivation to define special "syntactic" subclasses of **TFNP**, where all problems can be presented in a fixed format.

Before defining some of these classes we generalize the definition of polynomial-time reducibility for search problems and introduce type 2 problems and oracles.

Definition. [BCEIP97] Type 2 search problem Q is a function that associates with each string function $\alpha : \{0,1\}^* \to \{0,1\}^*$ and each string x a set $Q(\alpha, x)$ of strings that are allowable answers to the problem on input α, x. Such a problem Q is in $\mathbf{FNP}^2$ if Q is p-time verifiable in the following sense: $y \in Q(\alpha, x)$ is a type 2 p-time computable predicate, and all elements of $Q(\alpha, x)$ are of length polynomially bounded by $|x|^{O(1)}$.

Previously defined search problems (without functional input) are also called type 1 problems.

By an oracle Ω we mean a set of strings. It can be helpful during the computation of a Turing machine M. The machine M may use sub-routines in its computation, but their running time must be counted in the total running time. When using oracles we omit their running time and calculate only the duration of creating questions and processing answers by M.

The oracle Ω can be viewed as a characteristic function:

$$\Omega(x) = \begin{cases} 1 & \text{if } x \in \Omega\,, \\ 0 & \text{if } x \notin \Omega\,. \end{cases}$$

Define $\mathbf{TFNP}^{\Omega}$ to be a set of all search problems $Q(\Omega, *)$ where Q is total type 2 search problem, $Q \in \mathbf{TFNP}^2$.

Definition. [BCEIP97] Suppose we have two type 2 problems, namely Q_1 and Q_2. (See the figure 1.) We say that Q_1 is many-one reducible to Q_2 ($Q_1 \leq_{\mathrm{m}} Q_2$), if there exist type 2 p-time functions f, g, and h, such that $h(\alpha, x, y)$ is a solution to Q_1 on input (α, x) for y, which is a solution to Q_2 on input $(g(\alpha, x), f(\alpha, x))$. The output of g must be again a function.

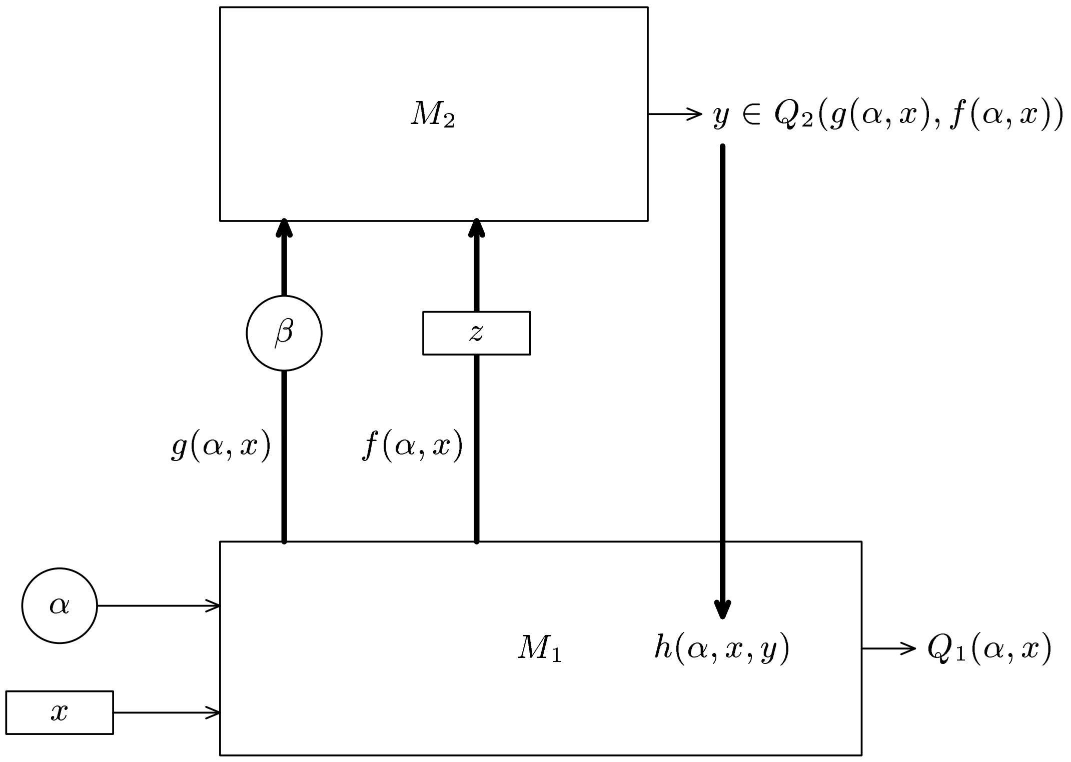

Fig. 1. Many-one reducibility

We can also apply this definition to the case of type 1 problems (or only Q_1 of type 1) by ignoring the functional part of the input.

If both problems are many-one reducible to each other, then we say that they are many-one equivalent.

Here we define the most important concept for the rest of our work. The relativized class $(\mathbf{C}Q)^\Omega$ is a subclass of $\mathbf{TFNP}^\Omega$ consisting of all problems $Q'(\Omega, *)$, where Q' is any problem in $\mathbf{TFNP}^2$ many-one-Ω reducible to Q. The suffix Ω means that the reduction is allowed to query the oracle. Formally, replace α by Ω in the scheme of the reduction. Notice that $(\mathbf{C}Q)^\Omega = \mathbf{C}Q$, if $\Omega \in \mathbf{P}$.

Theorem 1. [CIY97] *Let $Q_1, Q_2 \in \mathbf{TFNP}^2$. The following statements are equivalent:*

a) Q_1 is many-one reducible to Q_2;
b) for all oracles Ω, $(\mathbf{C}Q_1)^\Omega \subseteq (\mathbf{C}Q_2)^\Omega$;
c) there exists a generic oracle[1] Γ such that $(\mathbf{C}Q_1)^\Gamma \subseteq (\mathbf{C}Q_2)^\Gamma$.

The proof can be found in Cook, Impagliazzo and Yamakami [CIY97].

More general type of reduction between two total problems is the so called Turing reducibility. We say that Q_1 is polynomial-time Turing reducible to Q_2, if there exists a polynomial-time machine M that on input (α, x) and an oracle for solutions of Q_2-problems outputs some solution to Q_1, i.e. $y \in Q_1(\alpha, x)$. We also shortly say that Q_1 is reducible to Q_2 and write $Q_1 \leq_{\mathrm{T}} Q_2$.

Whenever M wants to make a query to the oracle for Q_2 it must prepare a pair (β, z), where β is a string function. In the case that M is a polynomial-time machine the function β must also be p-time computable and it can use the input parameters α and x, and all the questions which have been answered so far. The task of M is to produce a correct answer y for all possible computations and answers from Q_2.

A simple observation is that $Q_1 \leq_{\mathrm{m}} Q_2$ iff $Q_1 \leq_{\mathrm{T}} Q_2$ and M asks the oracle for Q_2 only once.

We enclose this section with an easy lemma, which gives us a basic connection between search and decision problems.

Lemma 2. $\mathbf{P} = \mathbf{NP}$ *if and only if* $\mathbf{FP} = \mathbf{FNP}$.

[1]) We will not need the notion of the generic oracle in this work; for its definition see [CIY97].

II.2 Local Search

An introductory paper to the complexity of local search was written by Johnson, Papadimitriou and Yannakakis in 1988 [JPY88]. We will expose their work in this chapter.

In optimisation there is an easy approach to finding solutions. We start in some initial position and seek for a better one. This is being done until we move to a place where no better neighbour exists.

From the idea one should deduce that an initial position and some neighbourhood structure will be needed. For example consider the famous ***Travelling-Salesperson-Problem***. Suppose we are in some state of finding optimal path (we have a tour which is not the best one). A classical neighbourhood is the one that assigns to this tour a set of tours which differ from it in just two edges (so called 2-change neighbourhood). Moreover, we need a fast algorithm which evaluates the states.

Definition. [JPY88] We define a class **PLS** of all polynomial-time local search problems. Such a problem L is specified as follows:

a) L has a set D_L of instances, which is a subset of all finite strings $\{0,1\}^*$.
b) For each instance x there is a finite set $F_L(x)$ of solutions. The terminology here is little confusing, better would be saying for example candidates. Without loss of generality all of them have the same polynomially bounded length $p(|x|)$, hence $F_L(x) \subseteq \{0,1\}^{p(|x|)}$.
c) For each candidate $s \in F_L(x)$ there is a nonnegative integer cost $c_L(x,s)$ and a subset $n(x,s)$ of $F_L(x)$ called the neighbourhood of s. The goal is to find some $y \in F_L(x)$ with locally minimal (or maximal) cost. No point from its neighbourhood can have smaller (or higher) cost.
d) There exists three polynomial-time algorithms I_L, C_L, and N_L. The first one given $x \in D_L$ produces an initial solution (start point) from $F_L(x)$. The algorithm C_L on input x and s computes the cost $c(x,s)$, if $s \in F_L(x)$. Finally, N_L has two types of output. If there is some solution $s' \in n(x,s)$ with better cost than s, it returns s'. Otherwise N_L returns s, and hence it is locally optimal.

There is a simple and straightforward algorithm for solving **PLS** problems. Just take the initial candidate s and repeat until locally optimal solution is found: Apply N_L to s and if it yields a better-cost neighbour s', then set $s = s'$. This algorithm will be called "standard". We know that the set of candidates is finite. Thus the algorithm must halt and at least one local optimum must exist. How long does the computation take? Since $F_L \subseteq \{0,1\}^{p(|x|)}$, enumeration of all the elements takes at most exponential amount of time, namely $2^{p(|x|)}$.

However, one might hope to obtain the result more quickly by other means. This evokes the following standard algorithm problem: Given x, find the local optimum s that would be output by the standard algorithm for L on input x.

Johnson et al. proved an easy, but interesting lemma.

Lemma 3. [JPY88] *There is a* **PLS** *problem L whose standard algorithm problem is* **NP***-hard.*

It should be said that if finding the specific local optimum by the standard algorithm is hard, it does not mean that finding some local optimum is hard as well. So the most important problem is to evaluate the complexity of finding some local optimum.

Since any problem in **FP** can be solved by a polynomial-time algorithm, we can use it as the initial algorithm I_L in the definition of **PLS**. This gives **FP** $\subseteq$ **PLS**

On the other hand, any **PLS** problem can be solved in a way that the local optimum is guessed, and then using the algorithm N_L the solution is validated in polynomial time. Thus **PLS** $\subseteq$ **FNP**.

Definition. [JPY88] A problem $P \in$ **PLS** is p-reducible to another problem $Q \in$ **PLS**, if there are polynomial-time computable functions φ and ψ such that

a) φ maps instances x of P to instances $\varphi(x)$ of Q,
b) ψ maps solutions of $\varphi(x)$ to solutions of x, and
c) for all instances x of P, if s is a locally optimal solution to the instance $\varphi(x)$ of Q, then $\psi(x, s)$ is a locally optimal solution to the instance x of P.

This is an intuitive generalization to the polynomial reducibility as it was defined in the preceding section.

As we mentioned in the introduction, a very important task is to show that there is a complete problem in the class. Johnson et al. in their paper proposed one problem which is **PLS**-complete.

It is a circuit computation problem: We have some Boolean circuit C with m inputs and n outputs. We need to look up for a string $a \in \{0,1\}^m$ such that the output of C on it has the minimal cost. The cost of a solution is simply the output $C(s)$ viewed as an integer. If $(y_1, \dots, y_n) = C(s)$, then $c(C, s) = \sum_{j=1}^{n} 2^j y_j$.

Formally, the set of candidates $F_L(C) = \{0,1\}^m$, neighbourhood of a string $s \in F_L(C)$ is any vector in m coordinates which differs from s in only one coordinate, i.e. they have Hamming distance one. To complete the definition, the initial algorithm always returns the vector of ones, and we want to find a solution with locally minimal (or maximal) cost.

Such a problem is called ***Flip***, since the moves from one candidate to another one in the neighbourhood structure evoke flipping.

Theorem 4. [JPY88] *__Flip__ is* **PLS***-complete.*

Corollary.

a) *The standard algorithm problem for* ***Flip*** *is* **NP***-hard.*
b) *There are instances of* ***Flip*** *for which the standard algorithm requires exponential time.*

The proofs of both these claims can be found in [JPY88].

It was also observed that the relationship of **PLS** to the traditional classes **P** and **NP** is very unclear and difficult to resolve. On one side, a problem in **PLS** cannot be **NP**-hard, unless **NP** = **coNP**. On the other side, if all problems in **PLS** were solvable in polynomial time, then showing this would require discovering of a general-purpose algorithm for finding locally optimal solutions that should be at least as sophisticated as the ellipsoid algorithm or Karmarkar's algorithm [JPY88].

Another remarkable problem in **PLS** is ***Max-Cut***. Suppose an undirected finite graph $G = (V, E)$ with weighted edges $w : E \to \mathbb{N}$. For such graph a cut is a partition of V into two disjoint sets V_1 and V_2. The weight of a cut (V_1, V_2) is the sum of the weights of the edges connecting nodes between V_1 and V_2. Computing the maximal cut is one of the most famous problem in theoretical computer science and it is also **NP**-complete on graphs of degree at most three [GJ79].

To use the problem in the world of **PLS**, we need to define a neighbourhood structure. Schäffer and Yannakakis proposed the simplest one: Two partitions are neighbours if one can be obtained from the other by swapping two vertices (they call it "swap neighbourhood"), and showed **PLS**-completeness of finding a local optimum for the ***Max-Cut*** problem with swap neighbourhood [SY91].

More precisely, we present the result by Elsässer and Tscheuschner [ET10].

Theorem 5. [ET10] *The problem of computing a local optimum of the* ***Max-Cut*** *problem on graphs with maximum degree five is* **PLS***-complete.*

In 2009 Pudlák and Thapen extended the definition of **PLS** to generalized polynomial search [PT09]. Their class is called $\mathbf{GPLS}_k$ and we can imagine it as k subsequent iterations of a **PLS** computation.

Definition. [PT09] A $\mathbf{GPLS}_k$ problem is defined by polynomial time functions v depending on $k+1$ variables and $h_1, \dots, h_k$ depending on $2, 3, \dots, k+1$ variables resp., where the first variable is a parameter. An instance of the problem is given by a number x (value of the parameter). The goal is to find numbers $b_1, c_2, b_3, c_4, \ldots < x$, such that

$$v(x, b_1, h_2(x, b_1, c_2), b_3, \ldots) \leq v(x, h_1(x, b_1), c_2, h_3(x, b_1, c_2, b_3), \ldots) .$$

The definition is inspired by a game in which two players A and B alternate in choosing values. After k steps the game ends, and A loses $v(x, b_1, \ldots)$, whereas B wins the same amount of money. Clearly, A tries to minimize the payoff, while B wants to enlarge it. The function v represents a value (or cost) function, and $h_1, \ldots, h_k$ are algorithms of both players (for the first one with even indices, for the second one with odd ones).

Particularly, if $k = 1$, then there is an obvious analogy to **PLS**: v is the cost function, h_1 the neighbourhood function, and the set of all $x_1 < x$ is the set of candidates. For given x we are asked to find a feasible solution b_1, such that the neighbourhood function h_1 cannot decrease the cost, i.e. the inequality $v(x, b_1) \leq v(x, h_1(x, b_1))$ holds.

II.3 Parity Lemma Based Search Problems

Whereas the inspiration for the definition of polynomial local search had come from mathematical logic, in this section we will present two combinatorial classes. They both were developed by Christos Papadimitriou in 1994. In this section we will frequently reference to his article "On the Complexity of the Parity Argument and Other Inefficient Proofs of Existence" [P94].

For this moment, the class **TFNP** will be the largest domain in the sense that all problems will be total. To prove their totality we need some existential theorems. These can be from any part of mathematics, namely combinatorics, algebra, number theory but also calculus or optimisation.

In his paper, Papadimitriou considers especially combinatorial problems which are motivated by basic graph properties. For example, if we have a finite graph, then it has an even number of odd-degree vertices. This is usually called the parity argument.

A more complicated claim based on the parity lemma, Smith's theorem, states that any graph with only odd degree nodes has an even number of Hamilton cycles through the edge xy for any vertices x and y. Thus when one has some Hamilton path going through the edge xy, then there must exist another (at least) one. So the existence of a solution is guaranteed by this theorem, and we can formulate a problem ***Smith***: Given an undirected finite graph $G = (V, E)$ with odd degrees, and a Hamilton cycle, find another one.

The input to the problem is a vector of $v = \#V$ different components, each of length at most $|v|$, coding the given Hamilton path. Hence the length of input is $O(v \log v)$, whereas the number of all possible Hamilton cycles might be up to

$$\frac{(v-1)!}{2} \approx \frac{\sqrt{2\pi(v-1)}}{2} \left(\frac{v-1}{\mathrm{e}}\right)^{v-1} .$$

Relations between nodes are given by a polynomial-time function

$$e(x, y) = \begin{cases} 1 & \text{if } (x, y) \in E\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

Since the function e is computable in polynomial time with respect to the lengths of x and y, we speak about polynomial parity argument.

When we find a different Hamilton cycle, we will be able to check correctness of this solution easily. But there are many possible candidates to be a solution. It is not known, whether the problem ***Smith*** is polynomial-time solvable, but as we have seen it is a total **NP**-search problem.

In Papadimitriou's paper it was a specimen for the class **PPA** (from polynomial parity argument mentioned two paragraphs before). Now we are going to define this class formally.

Definition. [P94] Suppose we have a deterministic polynomial-time Turing machine M. For any input x of length n, the configuration space $C(x) = \{0, 1\}^n$ serves as a set of graph vertices. It must hold:

(i) For $u \in C(x)$, $M(x, u)$ returns a list of neighbours of u as a tuple (v, w), (v), or $()$, where $v < w$, and $v, w \in C(x) \setminus \{u\}$.

(ii) For $u, v \in C(x)$, $v \in M(x, u)$ whenever $u \in M(x, v)$.

(iii) $0 \in C(x)$ has only one neighbour, $M(x, 0) = (a)$ for some $a \in C(x)$.

Since the node 0 has only one neighbour, we call it standard leaf and it provides us a "witness" for the parity lemma. It could be the given Hamilton cycle from ***Smith*** or simply a leaf in the graph G. Then the task is to find another leaf.

In the definition the machine M represents the function e from the text before, and in addition it is able to compute the opposite endpoint of an edge in the graph.

Papadimitriou also described a lot of other problems lying in the class **PPA**. Let G be an undirected graph and let $\bar{G}$ denotes its complement in some finite domain.

Let $H(G)$, and $H(\bar{G})$ is a number of Hamilton paths in G, and $\bar{G}$ respectively. Lovász has proved that $H(G) + H(\bar{G})$ is even under these conditions. We define ***Another-Hamilton-Path*** as the following problem: Given a finite undirected graph G and some Hamilton path in it, find another Hamilton path in G or in its complement $\bar{G}$.

Next example uses the parity lemma again. Let us have a function f with $\mathrm{Dom}(f) = \mathrm{Rng}(f)$ of even size, and consider a sentence

$$[f(0) = 0 \wedge (\forall x)x = f(f(x))] \Rightarrow [(\exists x)x \neq 0 \wedge x = f(x)] .$$

Here the function f should have been a bijection defining a pairing on its domain. The standard node 0 is lonely meaning that it is paired with itself. Due to the parity argument there must be one more lonely element x, such that $x = f(x)$, or some non-standard node has to be mapped to 0.

In the problem called ***Lonely*** the function f is polynomial-time computable, the domain consists of all zero-one strings of length n, and the task is to find a lonely node different from 0, or some x, such that $f(x) = 0$.

A similar problem is ***Leaf***. In a graph G, a leaf is a node of degree one. Like in the definition of ***Smith*** we have the function u defining edges in the finite graph G of degree at most two, and the standard leaf 0 having only one neighbour. The search problem ***Leaf*** is: Given the function u and an instance x coding the size of G, find a leaf in G other than the standard one.

Papadimitriou proposed also a problem inspired by number theory. Suppose a system of polynomial equations in n variables in the finite field GF_p for a prime p. Chévalley's theorem states that if the sum of the degrees of the polynomials is less then n, then the number of roots of the system is divisible by p. If we knew some solution to the system, then there should exist at least one more root, since the least prime number is 2.

The computational problem ***Chévalley-mod-p*** thus is: Given such a system and a root, find another. In a special case $p = 2$, Papadimitriou proved that ***Chévalley-mod-2*** is in **PPA**. However for $p > 2$ the problem fails to be in **PPA**. The class might be called **PPA-p** and the parity argument should be generalized to the form: If in a bipartite graph a node has degree not a multiple of p, then there is at least another such node.

Then ***Chévalley-mod-p*** is in **PPA-p** [P94].

In our expositions we have been considering only undirected graphs. Let us use the directed ones for a moment.

We define **PPAD** by modifying the previous version for undirected graphs. Suppose a finite directed graph $G = (V, E)$ on the words of length n ($V = \{0, 1\}^n$) which has in-degree and out-degree at most one. Since it is directed, the machine M should return an ordered pair on input $s \in V$, namely $M(x,s) = (s,s')$ where x is an instance code, $|x| = n$, and s' is the successor of s. In other words, there is an edge from s to s'. We use the standard node 0 as a witness anew. It has only one edge going out, but no one coming in. We are asking for another vertex whose in-degree plus out-degree equals to one. Such a node is called a sink, or a source respectively.

The class **PPAD** is the largest set of problems of the type described in the previous paragraph, which is closed under reductions.

An easy observation is formulated in the following theorem.

Theorem 6. [P94] *For the functional classes it holds*

$$\mathbf{FP} \subseteq \mathbf{PPAD} \subseteq \mathbf{PPA} \subseteq \mathbf{FNP}\,.$$

PPAD is under **PPA**, since we can forget the direction of the edges in the definition of a "directed problem" and we obtain a similar undirected version. As we have mentioned it is not know whether these inclusions are proper, or if there are equalities. Both are possible, but it is believed the first one is proper.

Sperner's lemma is a well-known claim speaking about colouring of a triangulation. In two dimensions it states that any admissible colouring of any triangulation of the unit triangle contains a trichromatic triangle. Suppose we have three colours, 0, 1, and 2, and divide the triangle 012 into approximately $n^2/2$ smaller triangles. Every vertex receives a colour. The colouring is admissible, if each vertex of the big triangle obtains its own name, and no vertex on the edge ij of the original triangle receives colour $3-i-j$. Then a trichromatic triangle other than the outer one can be found.

The outer trichromatic triangle will represent the standard source in the following computational problem ***2D-Sperner***: Given an integer n and an algorithm M assigning to each point $p = (i_1, i_2, i_3)$, with $i_1, i_2, i_3 \geq 0$ and $i_1 + i_2 + i_3 = n$ a colour $M(p) \in \{0,1,2\}$, such that $i_j = 0$ implies $M(p) \neq j$; find three points p_1, p_2 and p_3, such that their pairwise distances are one, and $\{M(p_1), M(p_2), M(p_3)\} = \{0,1,2\}$.

Although the generalization of the problem in higher dimensions does not seem obvious, the Sperner's lemma is valid in any dimension and the corresponding computational problem is in **PPAD**.

For example in three-dimensional space, the problem ***3D-Sperner*** asks as follows: Given an integer n an a polynomial-time algorithm computing for a point of the $n \times n \times n$ subdivision of the cube an admissible colour, find a tetrachromatic cubelet.

Theorem 7. [P94] *For any $k \geq 2$, k**D-Sperner** is in* **PPAD**.

We should also mention the computational problem inspired by the Brouwer's theorem: Any continuous function f from the unit simplex to itself has a fixpoint, i.e. there exists a point x such that $f(x) = x$. Since the proof is based on Sperner's lemma, the corresponding problem is in **PPAD** too.

But we need to represent a continuous function by a Turing machine. This is probably impossible, and so a simplification is used. For a given natural number n, and a point x in the unit cube with coordinates multiples of $1/n$ machine M returns in polynomial time a vector $\mu(x)$ such that $|\mu(x)| \leq 1/n^2$ and $f(x) = x + \mu(x)$ lies in the unit cube. Thus the function f can be extended to a piecewise linear map using the interpolation. In the problem ***Brouwer*** we are seeking for a point x satisfying $f(x) = x$.

The problem ***Brouwer*** is in **PPAD**.

Finally, we consider the Nash's theorem. He has found that there always exists an equilibrium in the following game. There are given two $m \times n$ matrices A and B, consisting of numbers a_{ij}, which is the payoff of player A when A plays strategy i and B plays strategy j; b_{ij} is the payoff of player B. The game is not zero sum, so $A + B \neq 0$. A Nash equilibrium is a pair of strategies i for A and j for B, such that neither A nor B have an incentive to change strategy (for all k it holds $a_{kj} \leq a_{ij}$, and $b_{il} \leq b_{ij}$ for all l).

For a convex space of strategies, Nash has shown that an equilibrium exists, but in our situation the space is discrete. Papadimitriou solved this problem using a probabilistic distribution over the rows and columns of the matrices. A row m-vector $x = (x_1, \ldots, x_m)$ is a mixed strategy of A, if for all i it holds $x_i \geq 0$, and $\sum x_i = 1$; and analogically for a column vector y for the player B. These two strategies are in equilibrium if $x'Ay \leq xAy$ for all mixed strategies x', and $xBy' \leq xBy$ for all y'. Such an equilibrium always exists. The problem ***Nash*** is defined in this way: Given two integer matrices A and B, find a mixed strategy equilibrium. It is not known, whether there is a p-time algorithm for this fundamental problem.

However, we know that ***Nash*** is in **PPAD**.

We have already explained the importance of complete problems for the classes. Papadimitriou in his seminal paper showed some **PPAD**-complete problems, but no one **PPA**-complete.

Theorem 8. [P94] ***3D-Sperner*** *is* **PPAD**-*complete.*

The proof is described in [P94]. It is based on construction of multicoloured tubes leading from the standard leaf to a solution throughout the cube.

Also ***Brouwer*** is **PPAD**-complete.

In 2001, M. Grigni generalized the result by Papadimitriou and showed for some generalization of ***Sperner*** problem to be **PPA**-complete [G01]. The most important difference is in considering non-orientable facets in the definition of the problem.

Grigni used a d-manifold, which is a topological space covered by open neighbourhoods homeomorphic to the Euclidean space $\mathbb{R}^d$. In an Euclidean space, a d-simplex is the convex closure of $d+1$ affinely independent points, and a face of a d-simplex is the convex closure of its corner points. A face with d corners is called a facet.

For a given d-manifold, a d-triangulation is a finite collection of d-simplexes covering the manifold, such that each pair of simplexes is either disjoint or intersecting on a common face. Each facet is shared by at most two d-simplexes. If it is only one, then the facet is situated in the boundary of the manifold. Having a d-triangulation, we may colour its points with the colours from the set $\{0, 1, \ldots, d\}$. A simplex, which contains all $d+1$ colours in its points is called full-colour; similarly for a facet.

Suppose it is given a d-triangulation, a colouring with no full-colour boundary facet, and a full-colour simplex. Sperner's lemma states that there exists another full-colour simplex.

Corresponding computational problem ***G-Sperner*** has an input x of length $n = |x|$, such that $2^{p(n)}$ is the number of triangulation points in one direction for a polynomial p, and uses a polynomial time Turing machine M describing a 4-colouring of the vertices of the triangulation: For each $i, j, k \in \{0, \ldots, N\}$ the colour of the point at coordinates $(i/N, j/N, k/N)$ is equal to $M(x, i, j, k) \in \{0, 1, 2, 3\}$ with the restriction that $S(x, 0, j, k) = S(x, N, N - j, k)$.

Theorem 9. [G01] ***G-Sperner*** *is* **PPA**-*complete.*

Grigni's proof is similar to Papadimitriou's one, but it employs more sophisticated notions from topology. Reader should consult [G01].

The class **PPADS** is a variant of **PPA**; in [P94] it was called **PSK**. A natural complete problem for **PPADS** is Positive Sperner's Lemma for dimensions three and above, which is exactly like Sperner's Lemma except that only a panchromatic simplex that is positively oriented is allowed to be a solution [BCEIP97].

The corresponding problem is called ***Sink***: For a given directed graph on $\{0, 1\}^n$ with in-degree and out-degree at most one in which 0 has in-degree zero and out-degree one (it is called a source), find a vertex with in-degree one and out-degree zero (sink).

II.4 Pigeonhole Principle

The pigeonhole principle is another combinatorial lemma which states that there must exist elements of some properties. Suppose $f : \{0, 1, 2, \dots, N\} \to \{1, 2, \dots, N\}$ is a polynomial time function. Such a function cannot be injective, since the size of its domain is strictly larger than the size of the range and both are finite.

The point is that we are given a big N which is represented by n bits, and it is claimed that the function is injective. Because that is impossible, there must exist a counterexample; i.e. an element $x \in \{0, \dots, N\}$ such that $f(x) \notin \{1, \dots, N\}$, or two different elements $x, y \in \{0, \dots, N\}$ which satisfy $f(x) = f(y)$. The computational problem ***Pigeon*** is: With a given number N and an access to a function f find a counterexample.

It is again a type 2 problem which is in **TFNP**, because having a solution we need only at most two queries to f (hence only constantly many) to verify the result.

All the problems which are polynomial time reducible to ***Pigeon*** create the class **PPP** (polynomial pigeonhole principle). Thus the problem ***Pigeon*** is a natural complete problem for **PPP**. The class was independently invented by Papadimitriou and Cook [P94].

There are many natural problems which can be solved easily using the ***Pigeon*** oracle. This means that we reduce a problem to an instance of ***Pigeon*** in polynomial time and ask the oracle for a solution. Having that solution to the ***Pigeon*** instance it is easy to reconstruct a solution to the given task (again in polynomial time).

For example the famous ***Discrete-Logarithm*** problem is described by two numbers p (prime number) and α (usually a generator of the multiplicative group $\mathbb{Z}_p^*$). Then for an instance $y \in \mathbb{Z}_p^*$ the question is: What is a value x for which the equality $\alpha^x \equiv y \bmod p$ holds?

We show a simple construction. For all $t \in \{0, \dots, p-1\}$ define a function

$$f(t) = \begin{cases} \alpha^t & \text{if } \alpha^t \neq y\,, \\ 0 & \text{if } \alpha^t = y\,, \\ y & \text{if } t = 0\,. \end{cases}$$

It is known that this function is polynomial time computable. Now use the ***Pigeon*** oracle. Since exponentiation on the invertible subset of a finite field is injective, the only collision with the pigeonhole principle is for t such that $\alpha^t = 0$. This is the solution.

If we could solve this instance of ***Pigeon*** effectively, then we would be able to compute the solution to any instance of ***Discrete-Logarithm*** problem.

In general, the ***Pigeon*** oracle is capable to invert any permutation. Let π be a polynomial-time computable permutation of a finite set S which does not contain 0. Suppose that we want to know a preimage of some $y \in S$ in the permutation π. It is sufficient to construct a new "permutation" of $S \cup \{0\}$. Define

$$\sigma(x) = \begin{cases} \pi(x) & \text{if } x \in S\,, \\ y & \text{if } x = 0\,. \end{cases}$$

It is easy to see that σ defines an injective map from $S \cup \{0\}$ to S with only one error which is in y.

Since all the encryption functions are polynomial-time pseudorandom permutations in fact, none of them is resistant against the "pigeon oracle attack" described above. The same holds for hash functions.

II.5 Separation of the Classes

Recall that a relativized problem is defined using an oracle whose computation time is considered to be a constant. For instance, let us in the last problem ***Pigeon*** replace the polynomial-time function f by an oracle. The obvious advantage is that we do not have to wait for its response, but on the other hand, we cannot verify its answer. In other words we have to believe to the oracle.

A simple example shows that relativized ***Pigeon*** is not solvable in polynomial time. We prove an easy lemma, because we want to explain the diagonal method which we will use in the following chapters.

Lemma 10. *There does not exist polynomial time oracle machine M^{Ω} that solves **Pigeon**$^{\Omega}$ for all oracles Ω.*

Proof. Suppose the contrary. Let x be an n-bit input defining the domain $[2^n] =$ $= \{0, \dots, 2^n - 1\}$ on which an injective mapping u into $[2^n - 1]$ is claimed to exist. The mapping u is computed by an oracle Ω. We have a polynomial time Turing machine M with an access to the oracle Ω which can find a contradiction with the injectivity of u. The oracle responds questions of the form "what is an image of t?" After at most n^k steps (for some $k > 0$ fixed) we have to stop the computation and give a result.

But during the run of the program the machine M has visited only at most n^k values of u, and for sufficiently large n it holds $n^k < 2^n$. Thus the oracle Ω

has an important advantage. It can choose values $u(t)$ independently with only two restrictions: Different questions must be answered differently, repeated questions must be answered always with the same value.

The machine M cannot find a contradiction in polynomial time, if the oracle behaves in the described manner. Since M must output a solution, it chooses a pair of non-visited points $(y, y') \in [2^n]$, but after asking oracle, the verifier will discover that $\Omega(y) \neq \Omega(y')$. ❑

This method of separating hard problems from some smaller class has come from mathematical logic and it was used, for example, by Beame, Cook, Edmonds, Impagliazzo and Pitassi [BCEIP97] in our context. In the following paragraphs we are going to list known results in this effort. These are done in a similar way as it was in our lemma, and since the classes are defined using a few "specimen" search problems we obtain separation of classes as corollaries.

Theorem 11. [BCEIP97] ***Lonely*** *is not reducible to* ***Pigeon****.*

Here mentioned problem ***Lonely*** is based on parity lemma and thus it is in **PPA**. The task is for a given pairing of even number of vertices and a standard lonely node of degree zero, locate another lonely node.

The proof of the theorem is by contradiction. It is supposed that we can solve any instance of ***Lonely*** using a ***Pigoen*** oracle. What is hard is to construct answers of the oracle in a way such that the machine solving ***Lonely*** is not able to uncover a lonely node.

Because ***Pigeon*** is a natural **PPP**-complete problem, any search problem in the class must be reducible to it.

Corollary. *There exists an oracle* Γ *such that* $\mathbf{PPA}^\Gamma \not\subseteq \mathbf{PPP}^\Gamma$.

It is not hard to see that ***Sink*** is many-one reducible to ***Pigoen***: Construct the input for ***Pigeon*** as a function f, which returns 0 on a sink, and for other vertices u, if there is an edge from u to v, $f(u) = v$.

Theorem 12. [BCEIP97] ***Sink*** *is not reducible to* ***Lonely****.*

Since we know that ***Sink*** $\in \mathbf{PPP} \cap \mathbf{PPADS}$, we can establish the following theorem.

Corollary. *There exist oracles* Γ *and* Δ *such that*
a) $\mathbf{PPADS}^\Gamma \not\subseteq \mathbf{PPA}^\Gamma$;

b) $\mathbf{PPP}^{\Delta} \not\subseteq \mathbf{PPA}^{\Delta}$.

Some more results are by T. Morioka [M01], who separated classes ***PPP*** and ***PPA*** from ***PLS***.

Theorem 13. [M01] *There exist oracles* Γ *and* Δ *such that*
a) $\mathbf{PPA}^{\Gamma} \not\subseteq \mathbf{PLS}^{\Gamma}$*;*
b) $\mathbf{PPP}^{\Delta} \not\subseteq \mathbf{PLS}^{\Delta}$.

And finally, in [BM04], J. Buresh-Oppenheim and T. Morioka partially answered the opposite.

Theorem 14. [BM04] *There exists an oracle* Γ *such that the separation* $\mathbf{PLS}^{\Gamma} \not\subseteq \mathbf{PPA}^{\Gamma}$ *holds.*

We have described a comprehensive list of separation results in the relativized world of **NP**-search problems. Whether $\mathbf{PLS}^{\Gamma} \subseteq \mathbf{PPP}^{\Gamma}$ or not, it is not known to us.

As we promised in the second part, some of the search problems are not equivalent to any decision problem.

Theorem 15. [BCEIP97] *None of the problems* ***Sink****,* ***Leaf****, or* ***Pigeon*** *is polynomial-time Turing equivalent to any decision problem.*

Chapter III

An Upper Bound for Integer Factoring

We have already mentioned the importance of the problem called ***Integer-Factoring***. It is believed that there are instances of it which are hard to solve, and this fact is utilised in many cryptographic protocols like RSA. But nobody can prove that, and hence the belief is based only on our experiences with different algorithms which solve the problem. These are numerous; for example naive division, Pollard's rho method, and quadratic or number theoretic sieves.

Let us formalize the problem ***Integer-Factoring***. Its input is a composite integer N of length n, and the task is to find a whole number d such that d divides N and $1 < d < N$.

In average case the problem is relatively easy to solve. Consider a random integer; with the probability $1/2$ it is divisible by two, with the probability $1/3$ it is divisible by three etc. If we had a list of all prime numbers less than 100, then the probability that a random number would have a divisor among these primes is safely more than 75 %.

That is why in cryptographic applications there are implemented sophisticated methods of finding secure pair of large prime numbers. These are then multiplied and the result is being used and considered impossible to be factorized back.

Hence we will study the case $N = pq$ for two different prime numbers p and q. Moreover, these primes are usually of roughly the same bit length, but this assumption is not very important in our case.

In this chapter we are going to present some results which create an upper bound for the complexity of the ***Integer-Factoring*** problem. On the other hand, in the next chapter we describe a method of estimating a lower bound for the same. Shortly, our aim is to insert the problem to a subclass of **TFNP**, but preferably far from polynomial-time solvable problems.

Let us define a special case of the ***Integer-Factoring*** problem. Suppose that $N \equiv 1 \bmod 4$ (i.e. the remainder after the division of N by 4 is equal to one), and -1 is not a square modulo N. The last condition says that there does not exist $x < N$ such that $x^2 \equiv -1 \bmod N$.[2] Numbers satisfying these two conditions will be called "good".

Intuitively, the ***Good-Integer-Factoring*** problem is: Given a good integer find its nontrivial divisor.

Proposition 16. *__Good-Integer-Factoring__ is in the class* **PPA**.

We have proved this independently with Joshua Buresh-Oppenheim, whose proof is a little different from our argument; see [B10].

Proof. Consider a good integer $N = pq$. Then $h = (N-1)/2$ is even. We construct a graph on $v = 2^{\lceil \log h \rceil}$ vertices from the set $\{1, \ldots, v\}$ in several steps (consult the figure 2 bellow).

a) Identify numbers $x < N$ with their opposites $-x$ modulo N, i.e. x and $N - x$ are both represented by only one node. (In the picture this is shown by the equivalence symbol $\sim$.)
b) Every vertex receives a unique name – an integer between 1 and v.
c) For each $i \in \{1, \ldots, h/2\}$ create an edge between $2i$ and $2i - 1$.
d) For each $i \in \{h+1, \ldots, v-1\}$ add an edge between i and $i+1$.
e) For each $i \in \{1, \ldots, h\}$ create an edge between i and $i^{-1} \bmod N$, if the inverse exists. If $i^{-1} > h$, use $-i^{-1} \bmod N$ instead.
f) Add an edge $\{h+1, v\}$.

We claim that this is a valid instance of ***Leaf*** with a standard leaf 1. Since $-1 \bmod N$ is not a quadratic residuum, the equation $x^{-1} = -x$ does not have a solution, and hence in the fifth step we always join two different nodes.

When the inverse element of i^{-1} is greater than h, the inverse of $-i^{-1} \bmod N$ must be smaller than h, and vice versa. Thus there cannot be three different neighbours of the element i or $-i$, and so there is no problem in our construction, step e.

[2] This really is possible: $8^2 = 64 \equiv -1 \bmod 65$ where $65 = 5 \cdot 13$, and $65 \equiv 1 \bmod 4$.

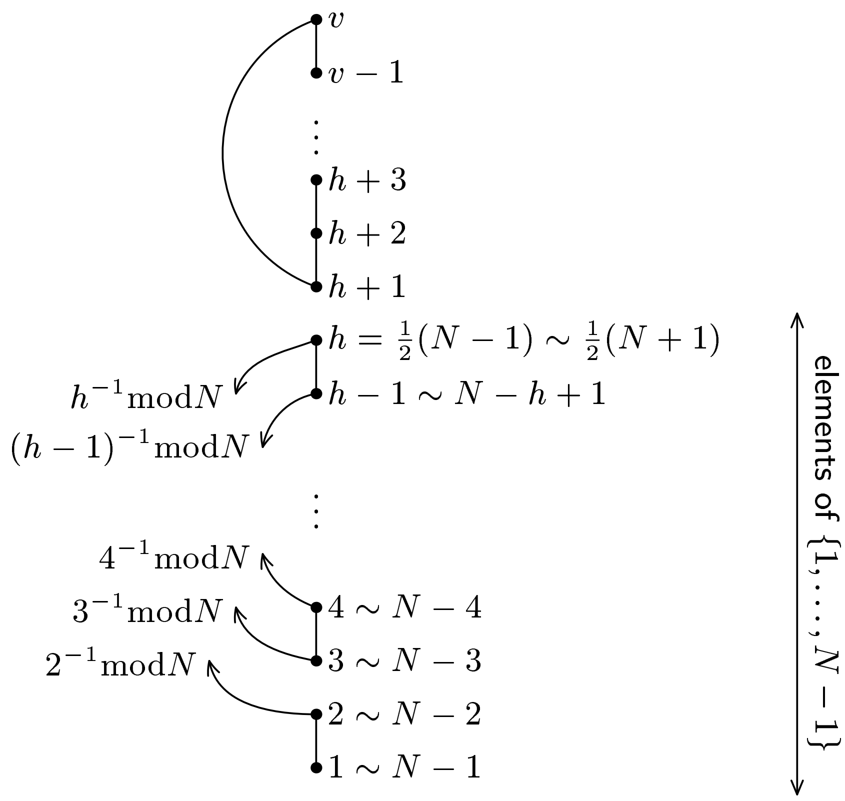

Fig. 2. Construction of the graph

The vertices which do not have two edges, are only the noninvertible integers modulo N, and the standard leaf, of course. So if we could solve the ***Leaf*** problem effectively, we would be able to find a non-trivial factor of a good integer, because the noninvertible elements of $\mathbb{Z}_N$ are precisely those numbers which do have a nontrivial greatest common divisor with N. But such an integer can be only a multiple of p or q. Note that the Euclid's algorithm may be used to quickly compute the GCD. ❑

Although the proposition holds only for good integers, it has a large impact. This is due to the cryptographic importance of the so-called Blum numbers. These are numbers of the form $N = pq$ where p and q are Gaussian prime numbers with no imaginary part. For us, the most significant fact is that all these numbers are equal to one modulo four. Also the second condition that -1 is not a square modulo N is sometimes useful in applications; see for example Feige-Fiat-Shamir Identification Scheme which is described in [K10].

An interesting thing is that the construction of an **PPA** instance is relatively easy for $N \equiv 1 \bmod 4$, but it seems unable to make it for the numbers equal to 3 modulo 4. We leave this problem open.

On the other hand, again J. Buresh-Oppenheim proved [B10] that the general case of the ***Integer-Factoring*** problem is a member of randomized **PPP** class. That means that there exists a reduction to a ***Pigeon*** instance which is able to find a factor in zero-error probabilistic polynomial time; that is a search variant of the famous **ZPP** class.

Proposition 17. [B10] *Integer-Factoring is in* $\mathbf{FZPP}^{\mathbf{PPP}}$.

Since this result is closely related to the main topic of this thesis, we include a sketch of the proof.

Proof. First, we test if the given N is not a multiple of 2 or a prime power. Otherwise, construct a random instance of ***Pigeon*** on strings of length $|N|$. Choose two random integers $a, b < N$. If one of them is noninvertible, then return its greatest common divisor with N. This happens with probability $1 - \alpha^2$ where α is a fraction of units in $\mathbb{Z}_N$.

Suppose that both these numbers are quadratic non-residues. This occurs with probability $\frac{3}{4} \cdot \frac{3}{4} = \frac{9}{16}$, since N is divisible by at least two odd primes, and there is at most a half of quadratic residues modulo a prime.

Construct a mapping on $\mathbb{Z}_N$:

$$0 \mapsto a\,,$$
$$x \mapsto \begin{cases} x^2 & \text{if } x \text{ is a unit and } x \leq \frac{1}{2}(N-1), \\ bx^2 & \text{if } x \text{ is a unit and } x \geq \frac{1}{2}(N+1), \\ 0 & \text{if } x \text{ is a non-unit and } x \neq 0. \end{cases}$$

Finally, map each string with value at least N to itself.

If the ***Pigeon*** oracle returns an element which is mapped to 0, then compute its greatest common divisor with N, and return it. If it returns two different elements x, y with the same non-zero image, then neither x nor y is 0. Because b was supposed to be a non-residuum and invertible, both these numbers are mapped in the same way, hence we have $x^2 \equiv y^2 \bmod N$, or $bx^2 \equiv by^2 \bmod N$. From these equations one can obtain a factorization $(x+y)(x-y) \equiv 0 \bmod N$.

Recall, that with probability $1 - \alpha^2$ either a or b is non-invertible. This leads to a solution. Otherwise, at most one quarter of elements of $[N-1]$ are quadratic residues. Both these numbers are non-residues with probability at least 9/16. Since every quadratic residue has two square roots which are greater than $N/2$, and only a half of them is of the form bx^2 for some $x > N/2$. The conditions on a and b are all satisfied in at minimum $9/16 - 1/4$ cases, and thus the algorithm succeeds with probability at least

$$1 - \alpha^2 + \alpha^2 \left(\frac{9}{16} - \frac{1}{4} \right) = 1 - \frac{11}{16}\alpha^2 \geq \frac{5}{16} .$$

□

It would be nice to derandomize this result. Though it follows from the Extended Riemann Hypothesis, which guarantees the existence of a non-residuum in the range $[1, O(\log^2 N)]$ as Eric Bach showed in 1990 [B90]. Even in 1975 Gary L. Miller proved some other interesting theorems based on the assumption that ERH is true.

Theorem 18. [M75] *Let $N = p_1^{v_1} \dots p_m^{v_m}$ is an integer. If Extended Riemann Hypothesis is valid, then the following functions are polynomial-time equivalent:*

a) prime factorization $N \mapsto ((p_1, v_1), \dots, (p_m, v_m))$;

b) Euler function

$$\varphi(N) = p_1^{v_1 - 1}(p_1 - 1) \dots p_m^{v_m - 1}(p_m - 1) ;$$

c) Carmichael λ-function

$$\lambda(N) = \operatorname{lcm}\left(p_1^{v_1 - 1}(p_1 - 1), \dots, p_m^{v_m - 1}(p_m - 1)\right) ;$$

d) $\lambda'(N) = \operatorname{lcm}(p_1 - 1, \dots, p_m - 1)$.

Nevertheless the Buresh-Oppenheim's result establishes the question of some relationship between probabilistic versions of **PPA** and **PPP**, or their connection to the class **FZPP**.

Also note, that there is a direct relation between some cryptographic primitives (e.g. hash functions or modular exponentiation) and the class **PPP** or even a class **WPPP** corresponding to the weak pigeonhole principle [CK98]. That is a similar statement to the pigeonhole principle with the only one difference: The size of the domain is twice larger than the cardinality of the range.

Chapter IV

A Lower Bound for Integer Factoring

In this chapter we are aiming to establish a lower bound of sorts to the complexity of the ***Integer-Factoring*** problem. We are going to construct a structure with a binary operation β. That model will represent the structure of whole numbers with multiplication. Adding more and more axioms we will be getting closer to the natural pattern. We want to show that even with a lot of axioms for multiplication supposed on β there is still not an oracle p-time machine M factoring successfully for all such β's.

Let N be a composite integer. The whole computation of an oracle p-time machine M takes a place on the domain of all strings of polynomial length with respect to $n = |N|$. Call it $D = \{0,1\}^{n^d}$ for a constant $d > 0$. Obviously, we have the binary representation of N in D. We want to find some "factor" of N with respect to an arbitrary binary operation β defined by an oracle.[3]

The oracle has the following advantage. Whenever it is asked for some result $\beta(x, y)$, $x, y \in D$, it can response any element from D.

When there are not any further conditions on oracle's answers it should be clear that we cannot guess the "factor" of N in polynomial time with respect to n. If we had one (call it r), the oracle would redefine β in a way that r would not "divide" N.

Now we are going to restrict the space of all possible β's on $D \times D$. This will be done by adding more axioms about β. For example, by requiring the commutativity

[3] Here we use the words factor and divide in quotas which means that we do not mean their proper meaning, but the modified one.

of β we reduce the number of queries to a half. Now we do not need to ask oracle for $\beta(x, y)$ and $\beta(y, x)$. It suffices to ask only for one of these values.

But there is still exponentially (in n) huge space of possible answers. Thus the oracle is able to define β in a way that in polynomial time in n it is impossible to find some "factor".

We can use this method of diagonalising also for the axioms of field. So requiring for example associativity of β does not help us. Details are described in the proof of the ***Field*** problem lemma in the following chapter.

The structure of natural numbers with multiplication has, in fact, many other properties. Let us fix some positive integer x. Then the function $f_x(y) = x \cdot y$ is monotone in its variable y. It is even linear, but we have only one binary function in our model, so the linearity would be hardly definable. For monotonicity we need only some ordering $<$ on the underlying set D.

The ordering is provided by another oracle, and thus we cannot compute the successor number for a given one etc. The only admissible type of query is to compare two elements of D, i.e. which one is smaller, or greater than the second one.

Now the task is to define oracle's behaviour when it is asked for values of monotone functions $\beta(x, \cdot)$ for all $x \in D$. We need to avoid leaking information about the "divisors" of N during the polynomial-time computation of Turing machine M.

Suppose that M's run takes n^k steps for a fixed $k > 0$. Since the space of all possible answers is of size 2^{n^d}, the oracle can construct its answers this way:

a) Let a is the minimal element in D with respect to $<$.
b) For any x define $\beta(a, x)$ small enough.
c) For arbitrary fixed x and any $c > b > a$ define $\beta(c, x) > \beta(b, x)$ such that the gap between these two values is large enough.

Now we are expected to make precise what the word "enough" in fact means. Recall that the machine M can ask for at most n^k function values, but there are 2^{n^d} possible answers. If the distribution were uniform, then there would be approximately $2^{n^d} n^{-k}$ different elements from D between two without delay consecutive values of $\beta(x, \cdot)$. This number is still exponentially big in n. And so the machine M cannot go throughout the whole interval (b, c) and locate a contradiction with some axiom, or a "divisor" of N.

Lemma 19. *Let $k \geq 1$ be fixed. There exists $N_0 \in \mathbb{N}$ such that for any $N > N_0$ consider its length of binary notation n, and denote $D = \{0,1\}^{n^{O(1)}}$. Let (D, β) is a structure with the underlying set D with a linear ordering $<$, and a binary operation β. Let $\beta(x, \cdot)$ is monotone with respect to the ordering $<$ for all $x \in D$. Then, in time*

n^k, it is impossible to find a pair $(x, y) \in D \times D$ such that $\beta(x, y) = N$, even if we have oracle accesses to β and $<$.

Proof. For any $x, y \in D$ the oracle does not say that $\beta(x, y) = N$. This is possible thanks to the argumentation above in the asymptotic case for N large enough. ❑

Contemporary algorithms for integer factorisation are based on one idea: find a number t, $1 < t < N$, which is not relatively prime to N. Then their great common divisor produces a factor of N.

Consider the typical RSA case when $N = pq$ for some prime numbers p and q. These numbers are usually of similar lengths. This is the hardest situation for factoring algorithms, because the set of all t's satisfying the condition of the previous paragraph is tiny. Its size is precisely $p + q - 1$.

Since the interesting instances are for N very large, we compute a ratio

$$\lim_{p,q\to\infty} \frac{|\{t \mid 1 < t < N, \gcd(t, N) > 1\}|}{N} = \lim_{p,q\to\infty} \frac{p+q-1}{pq} =$$
$$= \lim_{p,q\to\infty} \frac{1 + \frac{q}{p} - \frac{1}{p}}{q} = \lim_{p,q\to\infty} \left(\frac{1}{q} + \frac{1}{p} - \frac{1}{pq}\right) = 0\,.$$

Hence the probability of finding the solution at random is negligible. Furthermore there would be no problem when we added an oracle for the greatest common "divisor" γ into our model. For a "lucky" input it returns immediately a pair of elements (d_1, d_2) such that $\beta(d_1, d_2) = N$. On the rest of the set D it returns the greatest common "divisor" of the given numbers with respect to β. Note that the "lucky" domain of γ is very limited. Its size is roughly $\sqrt{N} \approx 2^{n/2}$.

We claim that after the computation of M we are able to define a set of "non-touched" elements of size $\sqrt{N}$.[4] It suffices to have n such that $n^k < 2^{n/2}$ or equivalently $k < n/2 \log n$. Since k is a constant, such n must exist.

The construction is now easy. Choose the least element r and the second least s for which the machine M has not asked yet. Define r's "multiples" as every odd consequent number; for s take the even ones. Here by odd and even we mean their order with respect to the ordering $<$. Repeat this by we have $\sqrt{N}$ non-touched elements.

Having these two sets of the same size $\sqrt{N}/2$ we can define $\beta(i, -i) = N$ where symbol i means the i-th least r's "multiple" and $-i$ represents the i-th largest s'es "multiple".

[4] Consider the even number closest to $\sqrt{N}$.

Notice, that for some ith multiple of r there exist two elements v, w, such that $v < ir < w$ and there is not any other element among them. Suppose the machine M has asked for values $\beta(v,x)$ and $\beta(w,x)$ for some $x \in D$. Then, by monotonicity, it must hold $\beta(v,x) < \beta(ir,x) < \beta(w,x)$. Its existence after our construction follows from the large gap between any pair of consecutive values.

Lemma 20. *For any $k \geq 1$ it can be found an integer N_0 such that for any $N > N_0$, $n = |N|$, there exists a set D of size $2^{n^{O(1)}}$ with a linear ordering $<$, and two functions β and γ on D. The function β is commutative, associative and bilinear. The function γ on input (a,b) returns the largest element $c \in D$ such that $\beta(c,A) = a$ and $\beta(c,B) = b$ for some $A, B \in D$, or it returns 1, if such element does not exist. Then, considering functions β, γ, and $<$ as oracles, in time n^k it is impossible to find a pair $(x,y) \in D \times D$ such that $\beta(x,y) = N$.*

Proof. It follows directly from the construction discussed before the lemma. ❑

We can also generalize the oracle β in the following manner. Suppose we allow questions to β in the form "$\beta(x,?) = y$". In other words, this asks for the "ratio" y/x. It is obvious that due to this generalization we get closer to the model of natural numbers, since we can divide integers as well. However, not every combination of dividend and divisor is allowed. Hence the oracle must have competence to refuse the input, and say these numbers are not divisible.

Let us summarize the possible queries to β for arbitrary $x, y < N$.

$$\text{“}\beta(x,y) = ?\text{”} \ \ldots \ \text{return } z,$$
$$\text{“}\beta(x,?) = y\text{”} \ \ldots \ \begin{cases} \text{return } z \in D, \text{ if “}\beta(x,z) = ?\text{” has answer } y, \\ \text{or return NO.} \end{cases}$$

Proposition 21. *For any $k \geq 1$, there is an N_0 such that for all $N > N_0$ the following holds. Let us denote $n = |N|$ and D of size $2^{n^{O(1)}}$ the underlying set. There are two oracles β and γ which can answer questions as above for any $x, y < N$, and a linear ordering $<$ oracle to compare elements from D. The function defined by β is bilinear, associative and commutative, the function defined by γ is commutative. In time n^k it is impossible to find a pair $(x,y) \in D \times D$ such that $\beta(x,y) = N$.*

Proof. We should only explain how to define the "quotients", because the rest is clear from the two lemmas before.

The first obvious rule is very simple. For any x the queries $\beta(x, ?) = N$ must be answered NO. Of course, there exists an element y such that $\beta(x, y) = N$, but there are exponentially many (in n) numbers in D, whereas we the machine M has only polynomial amount of time. Thus the machine seeking for y must always overpass it.

Other results of the form $\beta(a, b) = c \neq N$ are useless to M, since there is no metrics to measure distance between two elements. It is irrelevant if the relation was obtained as a "product" or "quotient". The oracle β should situate its answers far between as it was described earlier. ❑

Chapter V

Further Related Problems

Every finite field is of size $N = p^r$ for some prime number p and a positive integer r. We will use the set $\{0, 1, \ldots, N-1\}$ as a universe of the field. This can be identified with N. Then there exist two binary functions s, t (addition and multiplication), two unary functions u, v (additive and multiplicative inverses), and two constants $0, 1$ in the field. These functions must satisfy the following axioms:

a) $(\forall x)(\forall y)\ s(x,y) < N$,
b) $(\forall x)(\forall y)\ t(x,y) < N$,
c) $(\forall x)\ u(x) < N$,
d) $(\forall x)\ x = 0 \vee 0 < v(x) < N$,
e) $(\forall x)(\forall y)(\forall z)\ s(x, s(y,z)) = s(s(x,y), z)$,
f) $(\forall x)(\forall y)(\forall z)\ t(x, t(y,z)) = t(t(x,y), z)$,
g) $(\forall x)\ s(x,0) = s(0,x) = x$,
h) $(\forall x)\ t(x,1) = t(1,x) = x$,
i) $(\forall x)\ s(x, u(x)) = s(u(x), x) = 0$,
j) $(\forall x)\ x = 0 \vee t(x, v(x)) = t(v(x), x) = 1$,
k) $(\forall x)(\forall y)(\forall z)\ t(x, s(y,z)) = s(t(x,y), t(x,z))$,
l) $(\forall x)(\forall y)(\forall z)\ t(s(x,y), z) = s(t(x,z), t(y,z))$.

Clearly, all of these quantifiers are bounded by N.

Suppose we are given an integer N which is not of the form p^r. Then no finite field of cardinality N can exist. Thus it is possible to find elements which violate one of the axioms above. Precisely, we look for a four-tuple (i, b, c, d) where i stands for the index of axiom which is not satisfied, and b, c and d are the witnesses. When for example axiom c is violated, only the second parameter is used and the rest is an empty word λ.

Obviously this is an oracle-**NP**-search problem, since the length of the tuple is at most $O(1)+3\log N$. Its input is an integer N not of the form p^r (this can be witnessed by its two different divisors and verified in polynomial time as well as it is possible to verify the result in polynomial time), and an access to an oracle Φ which defines the functions s, t, u and v. We may consider that constants 0 and 1 are interpreted as usual. The task is to find a counterexample to the claim that the oracle defines a field. Call this problem simply ***Field***.

We shall prove that ***Field*** is not solvable by a p-time machine which has an access to a **PLS**-oracle. The proof is based on a similar result of Chiari and Krajíček of 1998 who showed that the weak pigeonhole principle is not solvable in **PLS** [CK98].

Proposition 22. *There is not an oracle* **PLS** *problem* L^Φ *such that for any oracle* Φ *defining a* ***Field*** *problem every local optimum of* L^Φ *contains a solution to* Φ.

Proof. Suppose the contrary. Let L be a $\mathbf{PLS}^\Phi$-problem such that for every field functions s, t, u and v of F it gives us a solution to the problem ***Field***. Whenever $S = (x, y, z)$ is a projection of some locally optimal solution for the instance N of the problem $L(s, t, u, v)$, then one of the following holds:

a) $s(x, y) \geq N$,
b) $t(x, y) \geq N$,
c) $u(x) \geq N$,
d) $x \neq 0 \wedge 0 = v(x)$ or $x \neq 0 \wedge v(x) \geq N$,
e) $s(x, s(y, z)) \neq s(s(x, y), z)$,
f) $t(x, t(y, z)) \neq t(t(x, y), z)$,
g) $s(x, 0) \neq s(0, x)$ or $s(x, 0) \neq x$ or $s(0, x) \neq x$,
h) $t(x, 1) \neq t(1, x)$ or $s(x, 1) \neq x$ or $s(1, x) \neq x$,
i) $s(x, u(x)) \neq s(u(x), x)$ or $s(x, u(x)) \neq 0$ or $s(u(x), x) \neq 0$,
j) $x \neq 0 \wedge t(x, v(x)) \neq t(v(x), x)$ or $x \neq 0 \wedge t(x, v(x)) \neq 1$
or $x \neq 0 \wedge t(v(x), x) \neq 1$,
k) $t(x, s(y, z)) \neq s(t(x, y), t(x, z))$,
l) $t(s(x, y), z) \neq s(t(x, z), t(y, z))$.

We claim that no such problem L can exist. We will continue in this manner: Fix an arbitrary L and find some N and suitable functions s, t, u and v such that none of the conditions above is satisfied for all locally optimal solutions.

Consider the cost function c_L and the neighbourhood function n_L associated to L. We identify these functions with the oracle p-time Turing machines computing them.

The machine computing L asks for values of the field functions. These functions have to be defined in such manner that the returned values do not contradict with the axioms of the field. This is done by progressive definition by the field oracle Φ.

This oracle uses some infinite field K and defines a partial isomorphism ι between it and the hypothetical one with universe $\{0, \ldots, N-1\}$ denoted F,

$$\iota : F \to K .$$

When it is asked for value of a term on some elements, it represents these elements as members of the infinite field, calculate the term, and finally it translates the result back to the original field.

Note that the partial isomorphism defines also several values which have not been asked during the computation, because of the axioms. Suppose that the machine has already known values $s(a,b) = d$, $s(c,d) = e$, and $s(b,c) = f$. Then it must hold $s(a,f) = e$, since $s(a,f) = s(s(a,b),c) = s(c,d) = e$, and a different result would witness the violation of the axiom e.

Let c_0 be the minimal possible cost for all possible computations of machine c_L where the oracle calls are answered as it was just explained. Fix some computation leading to the solution of cost c_0.

After at most $m = (\log N)^{O(1)}$ steps the computation must halt and the machine L outputs a locally optimal solution. Since the oracle Φ answers all the queries in a way preserving the isomorphism ι, the machine L cannot know any witness for ***Field***. It could have asked for at most m function values. These were defined as images of ι in infinite field K, and thus they do not violate any axiom of fields. If there is an element b in L's result such that $\iota(b)$ was not defined during the computation, define its image under ι in order that no axiom of fields is violated. Since these elements can be at most three, this can be done if $m + 3 < N$. Because there exists N sufficiently larger than m, there must exist instances of the problem ***Field***, which cannot be solved by L.

Now it is clear that the L's answer (i, b, c, d) is always wrong because we are able to define function values in i-th axiom how we want, and there is not any violence of the field axioms among the previously answered values.

All what we need is to choose sufficiently large N. That is a number for which there is an exponential gap between m and N itself. Since we have considered arbitrary **PLS**-oracle L we are done. ❑

It would be nice to establish a relationship between previously defined ***Field*** problem and commonly known ***Integer-Factoring*** problem. Recall that this stands for the question of finding some non-trivial factor of a given integer. However, we do not know such a formal relationship.

Now we shall consider another problem which is in some sense similar to the ***Field*** problem, and thus it is expected that they will have analogous properties.

From the theory of finite fields it is known that the only subfields of GF_{p^r} are finite fields of cardinalities p^s where $s|r$. This structure is well-studied, hence no surprise can be hidden there. To prove that some structure A is a substructure of a larger one B, we need to verify two things:

a) All functions defined on B are extensions of those defined on A.
b) Every function in A is closed in A, i.e. given arguments from A the function has value still in A.

Suppose for a while that we are given two finite fields of different sizes. Let us denote them F_1, F_2, and their cardinalities N_1, N_2, respectively. Without loss of generality we may suppose that $N_2 > N_1$. Since both structures form a finite field, we deduce that $N_1 = p^s$, and $N_2 = q^r$ for some prime numbers p, q and natural numbers s, r. If $p = q$, we require $s \not| r$ in addition.

It is claimed that F_1 is a subfield of F_2. From the text above we know that this is impossible, and thus we should be able to find a witness such that F_1 or F_2 is not a field, or F_1 is not a subfield of F_2 (and hence one of the conditions above is violated).

Let us define a new **NP**-search problem ***Subfield***: Given two integers N_1, N_2 as above, and accesses to oracles Φ_1 and Φ_2 defining F_1 and F_2, find a witness that one of F_1, F_2 is not a field, or that F_1 does not form a subfield of F_2.

Like in the first case we prove that this problem is not solvable easily.

Proposition 23. *There is not an oracle* **PLS** *problem L^{Φ_1,Φ_2} such that for any oracles Φ_1, Φ_2 defining a* ***Subfield*** *problem every local optimum of L^{Φ_1,Φ_2} contains a solution to* ***Subfield*** (Φ_1, Φ_2).

Proof. Similarly, in the first proof of this chapter we show that it is possible to define the oracle answers in a way that no contradiction can be found in polynomial time with respect to the length of input. Here we consider as the input the sizes of both fields. Since F_2 is supposed to be larger, it is sufficient to compute only with $\log N_2$ as the length of input.

The proof is very similar to the previous one. First, we should list all the possible inconsistencies with the assumptions. These are nearly the same as before; replace N by N_2, the cardinality of the largest field, and add some more schemes for the smaller field F_1 whose functions are denoted with bar.

a) $x, y < N_1 \wedge (\bar{s}(x, y) \neq s(x, y) \vee \bar{t}(x, y) \neq t(x, y))$,
b) $x < N_1 \wedge (\bar{u}(x) \neq u(x) \vee \bar{v}(x) \neq v(x))$,
c) $x, y < N_1 \wedge (\bar{s}(x, y) \geq N_1 \vee \bar{t}(x, y) \geq N_1)$,

d) $x < N_1 \wedge (\bar{u}(x) \geq N_1 \vee \bar{v}(x) \geq N_1)$.

Suppose there is a $\mathbf{PLS}^{\Phi_1,\Phi_2}$-problem L such that for any definition of the field functions $s, t, u, v, \bar{s}, \bar{t}, \bar{u}, \bar{v}$ it is able to find a solution to the ***Subfield*** problem. If $S = (x, y, z)$ is a projection of a locally optimal point for the instance (N_1, N_2), then one of the conditions described above must arise.

Fix an arbitrary cost function c_L, and a neighbourhood function n_L of L. These functions are computed by polynomial time Turing machines C_L and N_L. The machine computing L has an access to both these algorithms as well as to the oracles Φ_1 and Φ_2 computing the values of the field functions.

Now we describe a way how to construct their responses. The procedure is a simple generalization of the method used in the first proposition.

Fix two arbitrary infinite fields K and T such that T is a proper subfield of K, denoted by $T < K$. Then define partial isomorphisms

$$\eta : F_1 \to T \,,$$
$$\iota : F_2 \to K \,.$$

The isomorphism ι extends η. During the computation the oracles Φ_1 and Φ_2 are asked for values of the form $\bar{s}(a, b)$, $s(a, b)$ etc. The questions may repeat, in which case they must be responded always in the same way. If a new element h is mentioned in the query to Φ_1, the oracle defines its image under the partial isomorphism $\eta(h) \in T$, and evaluates the function in the field T. The isomorphism ι is created by Φ_2 in the same manner.

For instance, the machine wants to know a sum of $a, b \in F_1$, i.e. $\bar{s}(a, b)$. It uses the oracle Φ_1 which first looks in its database of questions responded so far. If there is some question on a or b, it uses the value $\eta(a)$ or $\eta(b)$ from the database. On the other hand, if such a question has not been put, it chooses elements $\alpha, \beta \in T$, and map $\eta(a) = \alpha$, $\eta(b) = \beta$. Then it calculates $\alpha + \beta$ in T. Say it is γ. If γ is in the database, the oracle uses its preimage from the database, in other case it can define it arbitrarily $\eta^{-1}(\gamma) = c$, and return back c.

Denote the minimal possible cost of all feasible computations of L by c_0 and fix a path leading to that cost. Because L must halt after at most $m = (\log N_2)^{O(1)}$ steps, it cannot ask for all possible combinations of functions and elements. In fact, it is allowed to ask for only negligible fraction of these values when N_2 is sufficiently large.

Since the field functions are defined as partial copies of some infinite fields, all the responses of the oracles are valid, and thus there cannot be found any contradiction with the axioms of fields or subfields. ❑

Again, since the fact that the problem ***Subfield*** is well-defined strongly depends on divisibility of integers, there might be a relationship with the ***Integer-Factoring*** problem. It is surely possible to define more problems like these two examples, because the realm of finite algebraic structures is significantly larger than only theory of finite fields.

Chapter VI

Concluding Remarks and Open Problems

We have presented several results that do in a sense estimate the complexity of integer factorization which is one of the most important problems in contemporary theoretical cryptography. Even though our bounds are not very tight, and work only in the relativized world of **NP**-search problems, they seem to be of interest. To our knowledge these are the first results of their kind.

In particular, the method of Chapter IV might be an inspiration for further research. There are many axioms which should be added to our hypothetical structure. First, in the domain of whole numbers we have another binary operation – addition, and also its inverse. Due to this we are able to define a distance measure and a predecessor and successor functions as well. These properties of the ring of integers should be added, and then one should verify, if the proof still works.

Note that there is a property which makes multiplication of naturals interesting. It has not any inverse function in fact, because in general a quotient of two integers is a fraction, and so not an integer. This is not true about addition of integers. But consider the addition of prime numbers. According to the famous Goldbach's conjecture every even integer greater than 2 can be written as a sum of two prime numbers. This has not been proved, but some weaker results have. Suppose for a moment it were true, then we could define a new computational problem. Call it ***Goldbach***: Given an even positive integer $N > 2$, find two prime numbers p and q such that $p + q = N$.

As well as the ***Integer-Factoring*** problem, ***Goldbach*** has not a unique solution[5], and no effective algorithm is known to us. The third parallel is in the non-existence of an inverse function (division of naturals, difference of an integer and a prime must not be a prime, respectively).

Another question we have left open asks whether the general case of ***Integer-Factoring*** is in some of the classes defined under **TFNP**. We have seen a special case is in **PPA** and also a randomized version in **PPP**, and we believe that these results can be improved.

For the sake of completeness of this text we only remark that the best known algorithm to ***Integer-Factoring*** is based on general number field sieve and its complexity is

$$\exp\left(c\sqrt[3]{n\log^2 n}\right)$$

where n is the length of the number to be factored and $c > 0$ is a constant [Po96].

On the other hand, in the world of quantum algorithms, the problem of factoring an integer is surprisingly easy. Shor's algorithm takes only $O(n^3)$ steps [S94].

There are also many related structural questions which have not been successfully solved yet. For instance, as we have mentioned, whether $\mathbf{PLS}^\Gamma \subseteq \mathbf{PPP}^\Gamma$ is not known. Also a relationship between the randomized class $\mathbf{FZPP}^{\mathbf{PPP}}$ and some other subclass of **TFNP** (for example **PPA**) is not clear.

Reader should look into [P94] to see some more open problems.

5) For example $20 = 3 + 17 = 7 + 13$, and $20 = 2 \cdot 10 = 4 \cdot 5$ etc.

References

[B10] Josh BURESH-OPPENHEIM, *On the TFNP Complexity of Factoring*, preprint, 2006, updated 2010.

[B90] Eric BACH, *Explicit Bounds for Primality Testing and Related Problems*, Mathematics of Computation, Vol. 55(191), pp. 353–380, 1990.

[BCEIP97] Paul BEAME, Stephen COOK, Jeff EDMONDS, Russell IMPAGLIAZZO, Toniann PITASSI, *The Relative Complexity of NP Search Problems*, Journal of Computer and System Sciences, Vol. 57(1), pp. 3–19, 1998.

[BM04] Josh BURESH-OPPENHEIM, Tsuyoshi MORIOKA, *Relativized NP Search Problems and Propositional Proof Systems*, IEEE Conference on Computational Complexity 2004: 54–67, 2004.

[C71] Stephen COOK, *The Complexity of Theorem Proving Procedures*, Proceedings of the Third Annual ACM Symposium on Theory of Computing, pp. 151-158, 1971.

[CIY97] Stephen COOK, Russell IMPAGLIAZZO, Tomoyuki YAMAKAMI, *A Tight Relationship between Generic Oracles and Type-2 Complexity Theory*, Information and Computation, Vol. 137, pp. 159–170, 1997.

[CK98] Mario CHIARI, Jan KRAJÍČEK, *Witnessing Functions in Bounded Arithmetic and Search Problems*, Journal of Symbolic Logic, Vol. 63(3), 1095–1115, 1998.

[ET10] Robert ELSÄSSER, Tobias TSCHEUSCHNER, *Settling the Complexity of Local Max-cut (almost) Completely*, arXiv:1004.5329v2 [cs.CC], 2010.

[G01] Michelangelo GRIGNI, *A Sperner Lemma Complete for PPA*, Information Processing Letters 77: 255–259, 2001.

[GJ79] Michael GAREY, David JOHNSON, *Computers and Intractability – A Guide to the Theory of NP-completeness*, Freeman, New York, 1979.

[JPY88] David JOHNSON, Christos PAPADIMITRIOU, Mihalis YANNAKAKIAS, *How Easy Is Local Search?*, Journal of Computer and System Sciences, vol. 37, no. 1, pp. 79–100, 1988.

[K10] Joseph KIZZA, *Fiege-Fiat-Shamir Revisited*, Journal of Computing and ICT Research, Vol. 4, No. 1, pp. 9–19, 2010.

[K95] Jan KRAJÍČEK, *Bounded Arithmetic, Propositional Logic, and Complexity Theory*, Cambridge University Press, 1995.

[M01] Tsuyoshi MORIOKA, *Classification of Search Problems and Their Definability in Bounded Arithmetic*, Master's thesis, University of Toronto, 2001.

[M75] Gary MILLER, *Riemann's Hypothesis and Tests for Primality*, STOC '75 Proceedings of seventh annual ACM symposium on Theory of computing, 1975.

[P93] Christos PAPADIMITRIOU, *Computational Complexity*, Addison Wesley, 1993.

[P94] Christos PAPADIMITRIOU, *On the Complexity of the Parity Argument and Other Inefficient Proofs of Existence*, Journal of Computer and System Sciences, 1994.

[Po96] Carl POMERANCE, *A Tale of Two Sieves*, Notices of the AMS 43 (12): pp. 1473–1485, 1996.

[PT09] Pavel PUDLÁK, Neil THAPEN, *Alternating Minima and Maxima, Nash Equilibria and Bounded Arithmetic*, preprint, 2009.

[S94] Peter SHOR, *Algorithms for Quantum Computation: Discrete Logarithms and Factoring*, 1994 Symposium on Foundations of Computer Science, 1994.

[SY91] Alejandro SCHÄFFER, Mihalis YANNAKAKIS, *Simple Local Search Problems That are Hard to Solve*, SIAM Journal on Computing, Vol.20(1), pp.56-87, 1991.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de