Approximation Complexity of Optimization Problems: Structural Foundations and Steiner Tree Problems

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Mathias Hauptmann

aus

Oldenburg

Bonn, April 2004

Vorwort

Der Ausgangspunkt für die vorliegende Arbeit was das Steinerbaum-Problem, welches mich schon in meiner Diplomarbeit beschäftigt hatte. Während ich mich dort auf exakte Algorithmen für geometrische Probleminstanzen konzentriert hatte, war es mein Promotionsbetreuer Prof. Dr. Marek Karpinski, der meine Aufmerksamkeit auf das allgemeine Steiner Tree Problem in Graphen bzw. endlichen metrischen Räumen lenkte. Die Basis bildeten zum einen seine Resultate zusammen mit Alexander Zelikovsky über neuartige Approximationsalgorithmen und die Arbeit von Robins und Zelikovsky, die den derzeit besten bekannten Steiner Tree Approximationsalgorithmus entwickelten, zum anderen neue Härteresultaten von Karpinski und Berman über gradbeschränkte Optimierungsprobleme.

Zu dieser Zeit kam Piotr Berman für ein Jahr als Gastprofessor nach Bonn, und so konnten wir zahlreiche Ansätze insbesondere zur Verbesserung der Approximationsgüte von Steiner Tree Algorithmen diskutieren. Nun begann ich, mich für das Steiner Forest Problem und die zu seiner Approximation verwandte Primal-Dual-Methode zu interessieren, insbesondere in dem Gewand der Local Ratio Technik von Reuven Bar-Yehuda. Mit Hilfe dieser Technik konnte ich Primal-Dual-Algorithmen für verschiedene Steiner Tree Probleme sehr kompakt darstellen.

Ein anderes, aber sehr eng mit dem bisherigen verzahntes Themengebiet griff ich nun wieder auf, nämlich Fragen der Strukturellen Komplexitätstheorie von Optimierungsproblemen, insbesondere im Bereich der Approximationsschemata. Ähnliche Fragestellungen interessierten mich bereits in meiner Diplom-Phase, doch konnte ich sie dort nur am Rande behandeln. Nun fand ich die Möglichkeit, mich mit Fragen der Reduzierbarkeit innerhalb der Klasse PTAS zu befassen und zu untersuchen, wie komplex die Laufzeit von Approximationsschemata von der Approximationsgüte abhängen kann. Sozusagen als Beiprodukt erhielt ich hier die Separation $EPTAS \neq PTAS$ unter einer vernünftigen komplexitätstheoretischen Annahme. Cesati und Trevisan hatten schon 1997 dasselbe Resultat erhalten, allerdings unter einer anderen Annahme.

In dieser Zeit widmete ich mich, angeregt durch meinen Betreuer Prof. Dr. Karpinski und auch motiviert durch die intensive Befassung mit Approximationsschemata, den dichten Steiner Tree Problemen. Ausgehend von dem Approximationsschema für das dichte Steiner Tree Problem von Karpinski und Zelikovsky konnte ich nun die von ihnen entwickelten Methoden auf verwandte Probleme anwenden.

In der Endphase des Promotionsprojektes griff ich dann nochmals strukturelle Fragestellungen auf. Unbefriedigend an dem Resultat über effiziente Approximationsschemata war, daß mir das Verhältnis der von mir zugrundegelegten komplexitätstheoretischen Annahme zu der von Cesati und Trevisan bisher nicht klar war. Schlechtestenfalls wäre meine Annahme einfach eine stärkere gewesen und hätte ihre impliziert. Glücklicherweise konnte ich im März/April 2004 kurz vor Fertigstellung dieser Arbeit durch eine Orakelkonstruktion mit einem Zählargument zeigen, daß letzteres nicht mit relativierenden Beweistechniken gezeigt werden kann.

Ich möchte Dank sagen: Meinem Betreuer Professor Dr. Marek Karpinski für die

Anregung und stete Förderung dieser Arbeit, für zahlreiche Gespräche und sein weitreichendes Interesse an vielen verschiedenen Themen der Mathematik und Informatik. Er war einerseits stets bemüht, mein Interesse in Richtung interessanter und vielversprechender offener Fragen zu lenken, gab mir andererseits aber auch die Freiheit, eigene Zielsetzungen zu verfolgen. Er war stets offen für Gespräche und interessiert an den Themen, die mich beschäftigten, gab zahlreiche wertvolle Anregungen und Hinweise auch weit über die jeweils spezifischen Fragestellungen hinaus.

Darüber hinaus ist maßgeblich er es, der den Kontakt seiner Arbeitsgruppe zu Forschern aus aller Welt ermöglicht. So holte er Piotr Berman und später Wenceslas Fernandez de la Vega im Rahmen von DFG-Professuren jeweils für ein Jahr nach Bonn und lud zahlreiche weitere Gäste zu Aufenthalten in Bonn ein.

Er ermöglichte mir die Teilnahme an den RAND-APX Workshops über Randomized and Approximate Computation in Edinburgh im September 2000, über Design and Analysis of Randomized and Approximation Algorithms in Schloß Dagstuhl im Juni 2001, über Randomized and Approximation Algorithms in Paris im April 2002 und am Dagstuhl Seminar 03291 über Algorithmic Game Theory and the Internet im Juli 2003 sowie mehrere Aufenthalte in der Arbeitsgruppe von Miklosh Santa am LRI in Paris-Orsay.

Sein breitgestreutes Interesse und sein Engagement belebt nicht nur die Forschungsatmosphäre innerhalb seiner Arbeitsgruppe, sondern ermöglicht unter anderem auch einen engen Kontakt der Gruppe zur Bonner Mathematik, insbesondere auch im Rahmen von BIGS, der Bonn International Graduate School in Mathematics, Physics and Astronomy.

An dieser Stelle möchte ich ganz herzlich Herrn Professor Dr. Bödigheimer für sein Engagement im Rahmen von BIGS und sein Interesse, mit dem er mein Promotionsprojekt begleitete, danken.

Dank gebührt auch Herrn Professor Dr. Piotr Berman, der mir während seines Aufenthaltes in Bonn zahlreiche Gesprächsmöglichkeiten bot, Anregungen gab, meine Intuition schärfte und mich so entscheidend mit an Forschungsthemen und Forschungsmethoden heranführte. Danken möchte ich Wenceslas Fernandez de la Vega, mit dem ich zahlreiche Gespräche führen konnte und der mich während meiner Aufenthalte am LRI hervorragend betreute. Hier gebührt auch ein herzlicher Dank Miklosh Santha und Claire Kenyon sowie Christina Bazgan, die reges Interesse an meinen Forschungsthemen hat und mir zahlreiche wertvolle Anregungen gab.

Danken möchte ich Elias Dahlhaus, mit dem ich eine Vielzahl anregender und hilfreicher Gespräche führen konnte und der während seiner Aufenthalte in Bonn stets ein offenes Ohr hatte.

Dank gesagt sei Herrn Prof. Dr. Norbert Blum, mit dem ich zahlreiche Gespräche führen konnte und der viele wertvolle Anregungen und Hinweise gab.

Danken möchte ich meinen Kollegen in Bonn: An erster Stelle Peter Wegner für eine sehr intensive und vertrauensvolle Zusammenarbeit, aus der sich eine enge Freundschaft entwickelt hat. Weiterer Dank gilt Hans-Hermann Leinen und Martin Löhnertz für zahlreiche anregende Diskussionen und ihre stete Hilfsbereitschaft. Dank gebührt auch Leszek Paszkiet und Ignatios Souvatsis, die stets hilfsbereit waren und in tech-

nischen Fragen und darüber hinaus mit Rat und Tat beiseite standen. Besonders danken möchte ich Christine Marikar. Sie trug über Jahre maßgeblich zu einer sehr angenehmen Atmosphäre in der Arbeitsgruppe bei, nahm Anteil an dem, was einen bewegt, war stets hilfsbereit und engagiert. Mit ihr verbindet mich eine Freundschaft, die mit zum Gelingen dieser Arbeit beigetragen hat.

Danken möchte ich meinen Schwiegereltern Ulrike und Achim Wiese für stetes Interesse und Förderung. Sie geben Halt und Hilfe und nahmen stets Anteil auch am Fortkommen dieser Arbeit. Dank gebührt ebenso meinem Schwager Martin Wiese und meinen Schwiegergroßeltern Erich und Erika Wiese, die mich stets unterstützt haben.

Diese Arbeit ist meiner Frau Annette und meiner Tochter Nicola Anna in Dank und Liebe gewidmet. Mit ihnen ist Leben so wunderbar.

Contents

Chapter 1

Introduction

Consider the following problem:

> *Given three points A, B, C in the Euclidean Plane, find another point D such that the sum of distances to D*
>
> $$d(A, D) + d(B, D) + d(C, D)$$
>
> *is minimized.*

This question is quite old. It has already been asked by Fermat (1601-1665), hence it is usually called the **Fermat Problem**. A geometric solution was given by Toricelli before 1640. Nevertheless, the natural generalization of this question yields one of the most important and most extensively studied problems in Combinatorial Optimization, namely the so called **Steiner Tree Problem**. Here is a slightly informal definition:

> **Steiner Tree Problem:** *Given a finite metric space (V, c), where V is the point set and $c: V \times V \to \mathbb{R}_+$ is a metric on V, and additionally given a subset S of V, find the shortest tree consisting of vertices from V that connects S.*

The set S of points which have to be connected is usually called **the Terminal Set**. If $T = (V_T, E_T)$ is a solution with vertex set $V_T \subseteq V$ and edge set $E_T \subseteq P_2(V)$ (the set of two-element subsets of V), then the cost of T (or its *length*) is just the sum of endpoint distances of all edges from E_T.

The Steiner Tree Problem was the starting point for this thesis. Much work was already done before on this problem and its numerous variants, special cases and generalizations. The Steiner Tree Problem is well known to be **NP-hard** [Kar72], which means its decision version (*Given an instance of the Steiner Tree Problem and some $k > 0$, is there a solution of cost at most k*) is **NP-complete**. Hence polynomial-time algorithms solving the problem to optimality are unlikely to exist. Furthermore the problem is MAXSNP-hard even for very restricted cases [BP89], hence polynomial time approximation schemes are unlikely to exist. One may therefore try to design constant factor approximation algorithms, which still have polynomial running time and compute a solution which is guaranteed to be only slightly more expensive. Currently,

most of the research concerning the Steiner Tree Problem concentrates on approximation algorithms. The state of the art is as follows: On the one hand, Robins and Zelikovsky [RZ00a] have given a family of polynomial time algorithms $\mathcal{A}_\epsilon$ (for $\epsilon > 0$) (a so called **approximation scheme**) with approximation ratio of $\mathcal{A}_\epsilon$ being bounded by $1+\ln(3)/2+\epsilon$. On the other hand Chlebik and Chlebikova [CC02] have shown that unless $P = NP$ no polynomial time approximation algorithm can achieve ratio better than 1.01063.

It is then interesting to look at special cases of the problem. Let us mention two important cases: The Geometric Case in fixed dimension and the Dense Case. In the **Geometric Case**, the direct generalization of Fermat's Problem, we are given a finite point set S in some $\mathbb{R}^d$, together with some L_p metric. The task is to find a tree T consisting of points in $\mathbb{R}^d$ connectingt S such that the length of T is minimal. Length here means the sum of L_p distances of endpoints of edges from T.

Although even the Geometric Steiner Tree Problems are known to be NP-hard for exact solutions, the precise complexity status of the Geometric Case was a longstanding open problem, and finally Sanjeev Arora [Aro98] gave the following answer: There is an algorithm $\mathcal{A}$ which gets as input an instance I of the Geometric Steiner Tree Problem (let us fix for a moment d to 2 and p to 2) together with some parameter $\epsilon > 0$, and returns a solution being at most by factor $(1 + \epsilon)$ longer than the optimal solution (we say its **approximation ratio is bounded by** $1 + \epsilon$). The running time of the algorithm is $O(n^{f\left(\frac{1}{\epsilon}\right)})$ for some function $f(1/\epsilon)$, where n denotes the size of the input I (basically the number of points given). Algorithms with this property are called **Polynomial Time Approximation Schemes** (**PTAS**). Later on, Rao and Smith [RS] could extend the Arora approach and replace the running time by $O(f(1/\epsilon) \cdot n \log n)$.

In the **ϵ-Dense Case**, instead the instance consists of a graph $G = (V, E)$ with edge costs equal to 1, the Terminal Set $S \subseteq V$ and the additional property that each terminal $s \in S$ has at least $\epsilon \cdot |\mathbf{V} \setminus \mathbf{S}|$ neighbours in $V \setminus S$. For this case, Karpinski and Zelikovsky [KZ97a] obtained the following result: For each fixed $\epsilon > 0$, there is a Polynomial Time Approximation Scheme for the **ϵ-Dense Steiner Tree Problem**, i.e. an algorithm $\mathcal{A}$ which gets as input an instance I of the ϵ-Dense Steiner Tree Problem and some $\delta > 0$ and constructs a $(1 + \delta)$-approximate solution in running time $O\left(|I|^{f(1/\delta)}\right)$ for some function f.

The two cases can be considered complementary to each other: In the geometric case the underlying graph on which a solution is constructed is **sparse**, i.e. every node has small degree and hence there are only few edges in the graph. Actually, for the L_1 case one can work with grid graphs [Aro98]. In the dense case, every terminal has high degree and therefore the graph contains many edges. Both are special cases of **Degree-Bounded Optimization Problems**.

An important generalization of the Steiner Tree Problem is the **Steiner Forest Problem** considered in [AKR91]. Here we are given several pairwise disjoint terminal sets $S_1, \ldots, S_n$. The task is to construct a minimum cost forest T such that each S_i becomes connected by T. One is allowed to connect more than one terminal set by

the same component, hence intuitively we have additionally to decide how to arrange the terminal sets in connect components. When this is done, the remaining problem is to compute Steiner Trees for the components. The best known approximation ratio achievable by polynmial time algorithms for the Steiner Forest Problem is 2, see [AKR91, GW92]. The algorithms are based on the Primal-Dual method. The **Prize Collecting Steiner Tree Problem** is another important problem generalizing the Steiner Tree Problem: Additionally to the terminal set S we are here given prizes to the terminals. We are allowed to connect only a subset of S, but the resulting cost is the connection cost plus the prizes for all terminals being left out. Goemans and Williamson [GW92] gave a 2-approximation algorithm again based on the Primal Dual Method. A different formulation of the Primal Dual approach was worked out by Bar-Yehuda [BY98]. His **Local Ratio Technique** provides a very compact formulation and analysis of primal-dual algorithms. In his paper [BY98] he already gave an application to the Steiner Forest Problem, obtaining the same ratio 2. We will in chapter 6 apply the Local Ratio Framework to the Prize Collecting Steiner Tree Problem and further generalizations in graphs and bounded hypergraphs.

Considering the problems mentioned so far, connection requirements are from the range $\{0, 1\}$, either two terminals are in the same set and have to be connected or not. The **Steiner Network Problem** captures the case when there are nonnegative integer requirements (e.g. two terminals have to be connected by the amount of 3), and edges are capacitated. While Ravi and Williamson obtained an approximation algorithm with logarithmic ratio [RW95], attemts to obtain a constant factor approximation using primal-dual methods failed. It was Kamal Jain [Jai98] who finally could provide a 2-approximation algorithm based on a linear programming and iterative rounding approach. His approach is based on the Ellipsoid Method, which was originally developed by Nemirovskii and Shor [Sho77] in the context of nonlinear optimization. Khachiyan [Kha79] modified the method to make it work for linear optimization as well. Unfortunately the running time of the Ellipsoid Method is large, which makes the method quite impractical. Hence it would be desirable to replace it by some purely combinatorial approach for the Steiner Network Problem. We will consider the Steiner Network Problem in chapter 7, providing combinatorial approximation algorithms for a special case of the problem, the Uncapacitated Uniform Steiner Network problem.

In order to compare problems with respect to their computational difficulty, the concept of reductions provides a poweful tool. While the notion of reduction and completeness was originally introduced in mathematical logic and recursion theory, it has successfully been used in the context of computational complexity both for decision and optimization problems. A reduction between two optimization problems A and B basically consists of two ingredients: a mapping from instances of problem A to instances of problem B and a mapping from solutions of B to solutions of problem A. The precise reduction concept must be carefully chosen such that approximation properties are preserved. Another closely related issue is to explore the structure of the class NPO. NPO is the class of all optimization problems whose decision version is in NP. The subclass PTAS contains all problems for which a polynomial time approximation scheme exists, and if additionally the running tiome bound is $O(f(1/\epsilon) \cdot n^c)$ for

some constant c and a function $f(1/\epsilon)$, the approximation scheme is called **efficient**. the according subclass of PTAS is denoted EPTAS [CT97a]. We will extensively discuss structural issues in chapter 4.

The thesis is organized as follows:
In **chapter 2** we give the basic notions and definitions concerning decision problems, function problems and optimization problems. The concept of reducibility and completeness as well as a brief introduction into probabilistic and randomized algorithms are given as well.

Chapter 3 deals with Fixed Parameter Complexity. After giving a brief intrduction containing basic definitions and concepts as well as brief list of problems concerning their parameterized complexity status, we investigate structural properties of the various fixed parameter complexity classes. In section 3.3 we give parameterized analogs of the well known Union and Speedup theorems from classical complexity theory. Section 3.4 deals with Levin's Lower Bound Theorem, which basically states that for every recursive language there exists a tight recursive lower bound for the space complexity of that language. Here we concern possible extensions of the Lower Bound Theorem to a wide and naturally defined class of Blum Complexity Measures (3.4.1), to Fixed Parameter Complexity where we ask for lower bounds on the dependence on the parameter (3.4.2), and for Randomized Space Complexity (3.4.3).

In **chapter 4** we consider structural aspects of the class NPO with special emphasis on polynomial tiem approximation schemes. We discuss reducibility concepts in section 3.2, where we introduce a new kind of approximation scheme- preserving reductions and prove existence of complete problems with respect to this reduction type. In sections 4.2 and 4.2 we take a closer look at the class PTAS. Our interest here is to investigate how running times of polynomial time approximation schemes depend on the approximation ratio. In general, for fixed ϵ the running time of a polynomial time approximation scheme $\mathcal{A}(x, \epsilon)$ depends polynomially on the input length $|x|$, hence the time bound is of the form $O(|x|^{f(1/\epsilon)})$ for some function $f: \mathbb{N} \to \mathbb{N}$. The ptas is called **efficient** if the time bound does not exponentially depend on ϵ, hence is of the form $f(1/\epsilon) \cdot |x|^{O(1)}$. Accordingly the class **EPTAS (Efficient PTAS)** is defined. Besides considering exponential versus multiplicative dependence of the running time on ϵ on may also ask for the computational complexity of the function f. A polynomial time approximation scheme is called **strongly uniform** if the running time dependence on ϵ can be bounded by some **recursive function** $f(1/\epsilon)$. In section 4.2 we prove that unless P equals NP, there exist problems in $PTAS$ for which a strongly uniform approximation scheme does not exist. Somewhat curious, we are able to construct such problems which nevertheless provide an efficient polytime approximation scheme (of course non-uniform).

In section 4.3 we consider the problem of separation of $EPTAS$ from $PTAS$. So far it is unknown how to prove the strictness of the inclusion $EPTAS \subseteq PTAS$ under assumption $P \neq NP$. Cesati and Trevisan were able to give a proof under the stronger assumption $FPT \neq W[P]$ from fixed parameter complexity; here "stronger" means that it implies $P \neq NP$. We give an alternative proof for $EPTAS \neq PTAS$ under

some different complexity theoretic assumption, namely (roughly spoken) existence of problems in NP with exponential lower bound on the deterministic time complexity (subsections 4.3.2 and 4.3.3). It is then natural to ask the question how these two assumptions are related to each other. In subsection 4.3.4 we give at least a partial answer: We construct an oracle relative to which our assumption is true and the assumption $FPT \neq W[P]$ not. This implies that using only relativizing proof techniques (i.e. those that still work under oracles) one cannot show that our assumption implies the other one.

In section 4.4 we consider efficiency of **randomized approximation schemes**. Assuming existence of a language which has exponential upper and lower bounds on the strongly randomized time complexity, we are able to separate $REPTAS$ from $RPTAS$. The methods used there are similar to those from section 4.3, but some careful additional considerations are needed.

Then we turn from structural considerations to algorithmic questions centered around the **Steiner Tree Problem** and its numerous variants and generalizations.

In **chapter 5** we consider the Steiner Tree Problem, concentrating on its general network version. We give a precise problem formulation, an introduction into the basic terminology, consider lower bounds for approximability and give a survey on well known approximation algorithms for the problem.

In **chapter 6** we consider the Steiner Forest Problem. We give brief descriptions of the Primal-Dual Method in section 6.2 and the Local Ratio Framework in section 6.3. Bar-Yehuda described how to apply the Local Ratiop Technique to the Steiner Forest Problem. In sections 6.4 - 6.7 we extend this approach to bounded hypergraphs and to generalizations of the Steiner Forest Problem, namely various **Prize Collecting** variants. In the prize collecting setting one is given several connection requirements in a network, together with a price for each requirement. One is now allowed to leave several requirements unsatisfied, but then has to pay the price. Hence the objective is the cost of the connection network one constructs plus the sum of prices for all unsatisfied requirements. In the **Price Collecting Steiner Tree Problem** the requirements are just terminals, i.e. we are given a terminal set S together with a price function $p\colon S \to \mathbb{R}_+$. The task is to construct a tree T connecting a subset S' of the terminal set such as to minimize the $c(T)$ (the cost of the tree) plus $p(S \setminus S')$ (sum of prices of the remaining terminals).

Chapter 7 deals with the Steiner Network Problem. We describe Jain's algorithm [Jai98] in section 7.2 and provide purely combinatorial approximation algorithms for the Uniform Uncapacitated Case in section 7.3 and to the Prize Collecting UniformUncapacitated Case in section 7.4.

Chapter 8 deals with directed Steiner problems. Giving an introduction with precise problem formulations and references to previous work in section 8.1, we then consider the Directed Zero Skew Tree problem in section 8.2 and the Directed Weighted Path Tree Problem in section 8.3. Both problems are motivated by applications in VLSI design, and the undirected Zero Skew Tree Problem has been considered in the literature before [ZM01, CKK$^+$99].

In **chapter 9** we consider **Dense Steiner Problems**. Pointing to previous work in sections 9.1-9.3, we will then consider the Dense Steiner Tree Problem in section 9.4. In subsection 9.4.2 we take a step towards an approximation scheme for the average-dense case, showing that the Karpinski - Zelikovsky approach [KZ97a] can be extended from the everywhere dense case to the log-dense case. In section 9.5 we consider the dense Steiner Forest problem, obtaining an approximation algorithm that has good performance in case the number of terminals is large compared to the number of terminal sets. In sections 9.6-9.8 we provide polynomial time approximation schemes for the dense versions of the k-Steiner Tree Problem, the Prize Collecting Steiner Tree Problem and the Class Steiner Tree Problem.

Chapter 2

Decision and Optimization Problems

In this chapter we will give the basic definitions and notations concerning resource-bounded decision problems and optimization problems.

2.1 Decision Problems

Given a finite alphabet Σ, subsets $L \subseteq \Sigma^*$ are usually called *decision problems* or *languages*. As usual P denotes the class of languages which can be decided in deterministic polynomial time while NP is the class of languages for which a nondeteministic polynomial-time bounded Turing machine (TM) exists. In general we use the terms $DTIME(t(n))$, $NTIME(t(n))$ for determainistic and nondeterministic time complexity classes, accordingly $DSPACE(s(n))$ and $NSPACE(s(n))$ for space complexity classes.

2.1.1 Reductions

Computable reductions are used for the purpose of comparing computational problems according to their computatioal difficulty. There is a huge number of different computational reductions used for the setting of decision problems. At this point we only mention two of them, namely the polynomial Karp reduction and the polynomial Turing reduction. The former is the resource-bounded variant of the m-reduction (*many-one reduction*) from recursion theory while the latter is the resource-bounded variant of the *oracle reduction*. We assume the reader being familar with the notion of *Oracle Turing Machines*.

Definition 1 *Let Σ be a finite alphabet and $A, B \subseteq \Sigma^*$.*

(a) A is **polynomial-time Karp reducible** *($A \leq_p B$) iff there exists a polynomial-time computable function $f\colon \Sigma^* \to \Sigma^*$ such that for all $x \in \Sigma^*$ $x \in A$ if and only if $f(x) \in B$.*

(b) *A is* **polynomial-time Turing reducible ($A \leq_T B$)** *iff there exists a polynomial-time bounded oracle TM M such that $L(M, B) = A$.*

At this point we only mention that so far it is not known how to separate the NP-completeness notions implied by these two kinds of reductions under the assumption $P \neq NP$, i.e. how to prove existence of a problem $A \in NP$ with is Turing-complete for NP but not Karp-complete. In the next subsection we will give precise definitions of the completeness notions.

2.1.2 Complete Sets

Given a class $\mathcal{C} \subseteq P(\Sigma^*)$ of languages, problem $A \in P(\Sigma^*)$ is called $\mathcal{C}$-hard with respect to polynomial-time Karp reductions iff every $L \in \mathcal{C}$ is polynomial-time Karp reducible to A, and $\mathcal{C}$-complete if additionally $A \in \mathcal{C}$. Hardness with respect to other types of reducibilities (e.g. polynomial-time Turing reducibility) is defined accordingly.

In 1971 Steve Cook gave the first NP-completeness proof for a *natural problem*, namely satisfiability of Boolean formulas in CNF (*conjunctive normal form*):

Theorem 1 (Cook's Theorem) *SAT is NP-complete with respect to polynomial-time reductions.*

Note that this was not the first proof of existence of complete problems for NP, since it was already known how to construct generic NP-complete problems, e.g.

$$TM - COMP := \{(\overline{M}, x, 0^k) | \overline{M} \text{ is the coding of a nondeterministic TM and exists exist an accepting computation of } M \text{ on input } x \text{ of length at most } k\}$$

A good reference is the book of Garey and Johnson, containing a comprehensive list of NP-complete problems from basically all areas of computer science, ranging from logic and algebra to graph theory, number theory and VLSI-design.

2.2 Function Problems

The sceanario of decision problems is not sufficient to work with in case of functional problems where, given some input x the task is to generate a solution y and not only a yes/no answer. E.g. in the case of SAT, the satisfiability problem for Boolean formulas in conjunctive normal form, one might be interested not only in the one-bit information wether a given formula is satisfiable but (in case the answer is "yes") also in a witness for that fact, namely a satisfying assignment. In general we have the following straight forward characterization of the class NP:

Definition 2 *Let $: \mathbb{N} \to \mathbb{N}$ be a function. A binary relation $R \subseteq \Sigma^* \times \Sigma^*$ is called* **$f(n)$-balanced** *iff*

$$\text{for all } x, y \in \Sigma^*\text{: } (x, y) \in R \text{ implies } |y| \leq f(|x|)$$

or equivalently

$$R \subseteq \bigcup_{n \geq 0} \Sigma^n \times \Sigma^{\leq f(n)}.$$

R is called **strongly $f(n)$-balanced** *iff*

$$R \subseteq \bigcup_{n \geq 0} \Sigma^n \times \Sigma^{f(n)}.$$

R is called **polynomially balanced/ strongly polynomially balanced** *iff there exists a polynomial $p(n)$ such that R is $p(n)$-balanced/ strongly $p(n)$-balanced.*

Lemma 2.2.1 (Characterization of NP)
For every $L \subseteq \Sigma^*$ the following are equivalent:

(1) $L \in NP$

(2) There exists a polynomially balanced relation $R \in P$ such that

$$L = R_L := \{x \in \Sigma^* | \exists\, y \in \Sigma^* \text{ such that } (x, y) \in R\}$$

Given such a polynomially balanced relation $R \in P$, the computational problem of generating witnesses in case they exist is called a *Function NP Problem*:

Definition 3 *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a binary relation. The associated computational problem Π_R is defined as follows:*

> *Given $x \in \Sigma^*$, either compute some $y \in \Sigma^*$ such that $(x, y) \in R$*
> *or return "NO" in case such y does not exist.*

The class of all problems Π_R with R being polynomially balanced and $R \in P$ is denoted FNP. The subclass of FNP all such problems Π_R that can be solved in deterministic polynomial time is called FP.

Obviously $P = NP$ if and only if $FP = FNP$. Besides that FNP provides an interesting structure: There are functional problems in FNP which are not believed to be in FP but have the property that solutions always exist.

Definition 4 (Total Relations)
A binary relation $R \subseteq \Sigma^* \times \Sigma^*$ is called **total** *iff*

$$\text{for every } x \in \Sigma^* \ \{y \in \Sigma^* | (x, y) \in R\} \neq \emptyset.$$

$TFNP$ denotes the class of function problems $\Pi_R \in FNP$ such that R is total.

We will now generalize the notion of *function problems* to arbitrary time and space bounds.

Definition 5

$$\begin{aligned} FTIME(t(n)) &:= \{\Pi_R | \Pi_R \textit{ solvable by a deterministic algorithm in time } O(t(n))\} \\ FNTIME(t(n)) &:= \{\Pi_R | \Pi_R \textit{ solvable by a nondeterministic alg. in time } O(t(n))\} \end{aligned}$$

The classes $FSPACE(s(n))$, $FNSPACE(s(n))$ are defined accordingly.

Note that if Π_R can be solved in time $O(t(n))$, then for every $x \in \Sigma^*$, either the answe is "NO" or there exists at least one y with $(x, y) \in R$ such that $|y| = O(t(n))$.

Definition 6 *A relation $R \subseteq \Sigma^* \times \Sigma^*$ is called* **weakly $f(n)$-balanced** *iff for every $x \in \Sigma^*$ one of the followingalternatives holds:*

(i) $\{y | (x, y) \in R\} = \emptyset$.

(ii) There exists $y \in \Sigma^*$ with $(x, y) \in R$ and $|y| \leq f(|x|)$.

2.3 Probabilistic and Randomized Classes

We assume the reader being familiar with the notion of probabilistic Turing machines (which we abreviate "PTM"). Given a PTM $\mathcal{M}$, the **induced function** $\Phi_{\mathcal{M}}: \Sigma^* \times \{0,1\}^* \to \Sigma^*$ is defined by

$$\Phi_{\mathcal{M}}(x, \rho) := \text{the output of } \mathcal{M} \text{ on input } x \text{ with string } \rho \text{ on the random tape}$$

(assumed ρ is sufficiently long). The **function computed by** $\mathcal{M}$ is the partial function $\varphi_{\mathcal{M}}: \Sigma^* \to \Sigma^*$ defined by

$$\varphi_{\mathcal{M}}(x) := \begin{cases} y & \text{if } Pr_\rho\{\Phi_{\mathcal{M}}(x, \rho) = y\} > \frac{1}{2} \\ \text{undefined} & \text{if such } y \text{ does not exist} \end{cases}$$

We will use the following abreviation:

$$Pr\{\mathcal{M}(x) = y\} \quad := \quad Pr_\rho\{\Phi_{\mathcal{M}}(x, \rho) = y\}.$$

Given a PTM $\mathcal{M}$, the language $L(\mathcal{M})$ accepted by $\mathcal{M}$ is precisely the set of strings on which $\mathcal{M}$ answers "yes" with probability strictly greater than $1/2$ (i.e. $\varphi_{\mathcal{M}}(x) = 1$). A PTM with bounded error probability is called a **Monte Carlo TM** or **Randomized TM (RTM)**:

Definition 7 *Let M be a Probabilistic Turing Machine (PTM).*

(a) The **error probability** *e_M is the partial function $e_M: \Sigma^* \to [0, 1/2)$ defined by*

$$e_M(x) := \begin{cases} 1 - Pr\{M(x) = \phi_M(x)\}, & \varphi_M(x) \textit{ is defined} \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

(b) *M is called* **Randomized TM (RTM, Monte Carlo TM)** *iff there exists* $\epsilon > 0$ *such that*

$$e_M(x) \leq \frac{1}{2} - \epsilon$$

for all $x \in D(\varphi_M)$.

(c) *M is called* **Total RTM** *iff additionally* ϕ_M *is a total function.*

RTMs accepting languages which have error probability 0 in case the input is not in the language are called **strong randomized TMs ($\mathbf{R}_S$TMs)**:

Definition 8 *A PTM M accepting a language L is called* **strong randomized** *iff there exists* $\epsilon > 0$ *such that for all* $x \in \Sigma^*$ *the following holds:*

$$\begin{array}{rcl} x \in L & \Longrightarrow & e_M(x) \leq \frac{1}{2} - \epsilon, \\ x \notin L & \Longrightarrow & e_M(x) = 0 \end{array}$$

2.4 Optimization Problems

In this section we will give the basic definitions and notions concerning optimization problems and approximation algorithms. For a more comprehensive treatment the interested reader is referred to [...]. Recall that an **NP optimization problem (NPO problem)** is a four-tuple $F = (I, S, c, g)$ where $I \subseteq \Sigma^*$ is the set of instances, $S \subseteq \Sigma^* \times \Sigma^*$ is the solution relation (i.e. for given $x \in I$ $\{y|S(x,y)\}$ is the set of feasible solutions for instance x of F), $c\colon \Sigma^* \times \Sigma^* \to \Sigma^*$ is the cost function and $g \in \{\min, \max\}$ is the optimization goal, and furthermore the following conditions hold:

1. I and S are polynomial-time decidable.
2. There exists some polynomial p such that for all $x \in I$ $\{y|S(x,y)\} \subseteq \Sigma^{\leq p(|x|)}$.
3. The cost function c is polynomial time computable.

For F an NPO problem and $p(n)$ some polynomial, we will call F **$p(n)$-time bounded** iff the time complexity of I, S and c is bounded by $p(n)$. We will furthermore use the following notations: $S(x) = \{y|S(x,y)\}$ is the set of feasibe solutions for instance x of F, $\mathrm{OPT}_F(x)$ denotes some optimum solution for x and $\mathrm{opt}_F(x)$ its cost. Furthermore we assume (w.l.o.g.) that for all $x \in I$ $S(x) \neq \emptyset$. For $y \in S(x)$

$$R_F(x,y) \quad := \quad \max\left\{\frac{c(x,y)}{\mathrm{opt}_F(x)}, \frac{\mathrm{opt}_F(x)}{c(x,y)}\right\} \tag{2.1}$$

is the **performance ratio of y with respect to** x or simply the ratio of solution y for x. An **approximation algorithm** $\mathcal{A}$ for NPO problem $F = (I, S, c, g)$ is an algorithm $\mathcal{A}$ such that for every instance $x \in I$ of F $\mathcal{A}$ computes a feasible solution $\mathcal{A}(x) \in S(x)$. For a function $r(n)$ $\mathcal{A}$ has performance ratio $r(n)$ if for all $x \in I$ $R_F(x, \mathcal{A}(x)) \leq r(|x|)$.

We are now going to define some of the most important subclasses of **NPO** which are well known in the literature.

Definition 9 (Subclasses of NPO)

1. **PO** *is the class of NPO problems F that are* **solvable to optimality in polynomial time**, *which means there exists a polynomial time algorithm which for each instance x of problem F computes a feasible solution y such that $R_F(x, y) = 1$.*

2. **PTAS** *is the class of NPO problems F that admit a* **polynomial time approximation scheme (ptas)**, *that means there exists an algorithm $\mathcal{A}$ such that for every instance x of F and every $\epsilon > 0$ $\mathcal{A}(x, \epsilon)$ returns a feasible solution y to instance x of problem F such that $R_F(x, y) \leq 1+\epsilon$, and furthermore the running time of algorithm $\mathcal{A}$ is $O\left(|x|^{f(1/\epsilon)}\right)$ for some function f.*

3. **APX** *is the class of NPO problems F that admit a polynomial time constant factor approximation algorithm $\mathcal{A}$ (i.e. there is some $c \geq 1$ such that $\mathcal{A}$ has ratio c).*

4. $\log -$**APX** *is the class of NPO problems F that admit a polynomial time approximation algorithm with approximation ratio $O(\log n)$.*

Remark: To be precise, in the definition of the class **PTAS** we have to replace "for every $\epsilon > 0$" by the restriction to positive rational values of ϵ. We are now going to further simplify the notation and only consider values of the form $\epsilon = \frac{1}{n}$. We get the following redefinition of the class **PTAS**:

Definition 10 (The Class PTAS)
PTAS *is the class of NPO problems F that admit a* **polynomial time approximation scheme (ptas)**, *which means there exists an algorithm $\mathcal{A}$ such that for every instance x of F and every $n \in \mathbb{N}$ $\mathcal{A}(x, n)$ returns a feasible solution y to instance x of problem F such that*

$$\mathbf{R_F(x, y)} \quad \leq \quad \mathbf{1 + \frac{1}{n}}$$

and the running time of algorithm $\mathcal{A}$ is $O\left(|x|^{f(n)}\right)$ for some function $f\colon \mathbb{N} \to \mathbb{N}$.

It is straight forward to check that both definitions are equivalent, i.e. define the same class of optimization problems. The following subclasses of **PTAS** are well-established in the literature:

Definition 11 (Subclasses of PTAS)

1. **FPTAS** *is the class of NPO problems F that admit a* **fully polynomial time approximation scheme**, *i.e. an algorithm $\mathcal{A}$ which for each instance x of F and $n \in \mathbb{N}$ computes a $\left(1 + \frac{1}{n}\right)$-approximate solution y to instance x and such that the running time of algorithm $\mathcal{A}$ on input x, n is bounded by $O\left(p(|x|, n)\right)$ for some polynomial $p(m, n)$.*

2. **EPTAS** *is the class of NPO problems F that admit an* **efficient polynomial time approximation scheme**, *i.e. a* **polynomial time approximation scheme** *$\mathcal{A}$ with running time $O\left(f(n) \cdot |x|^{O(1)}\right)$ for some function $f(n)$.*

An important super-class of **PTAS** is the class of problems that admit an **asymptotic approximation scheme**:

Definition 12 *Given an NPO problem F, an* **asymptotic polynomial time approximation scheme** $\mathcal{A}$ *for F is an algorithm which for each instance x of F and every $n \in \mathbb{N}$ computes a feasible solution y in time $O\left(|x|^{f(n)}\right)$ for some function $f(n)$ such that*

$$R_F(x,y) \quad \leq \quad 1 + \frac{1}{n} + \frac{1}{opt_F(x)},$$

where $opt_F(x)$ denotes the optimum value to instance x of F (we assume without loss of generality that $opt_F(x) \neq 0$).

PTAS$^\infty$ *is the class of NPO problems that admit an* **asymptotic polynomial time approximation scheme**.

Obviously the following chain of inclusion holds:

$$\mathbf{PO} \subseteq \mathbf{FPTAS} \subseteq \mathbf{EPTAS} \subseteq \mathbf{PTAS} \subseteq \mathbf{PTAS}^\infty \subseteq \mathbf{APX} \subseteq \log - \mathbf{APX} \subseteq \mathbf{NPO}.$$

It is natural to ask whether these inclusions are strict, under some natural complexity theoretic assumption. Under assumption $\mathbf{P} \neq \mathbf{NP}$, one obtains

$$\mathbf{PO} \subsetneq \mathbf{FPTAS} \subsetneq \mathbf{PTAS} \subsetneq \mathbf{PTAS}^\infty \subsetneq \mathbf{APX} \subsetneq \log - \mathbf{APX} \subsetneq \mathbf{NPO}$$

and furthermore

$$FPTAS \subsetneq EPTAS.$$

Under assumption $\mathbf{P} \neq \mathbf{NP}$ nothing is known about the strictness of inclusion

$$EPTAS \subseteq PTAS.$$

Nevertheless, Cesati and Trevisan were able to prove strictness under a somewhat stronger assumption from fixed parameter complexity, which translates into a natural assumption about the amount of nondeterminism needed to solve NP problems in polynomial time. In chapter 4, section 4.4 we give an alternative separation proof under some different assumption about lower bounds for deterministic time complexity of NP problems. This assumption is as well stronger than $P \neq NP$ (in the sense it implies the latter), and we will in section 4.4 construct a recursive oracle under which our assumption becomes true and that used by Cesati and Trevisan becomes false. This implies that using relativizing proof techniqhes one can not show that our assumption implies theirs.

Concerning polynomial time approximation schemes, another interesting question arrises: What kind of functions may occur in the exponent of the running time bound ? It is worth mentioning that for all natural problems we are aware of which provide a polynomial time approximation scheme, the dependence of the running time on $1/\epsilon$ is by a polynomial or some exponential function.

But are there problems F in **PTAS** *such that any polytime approximation scheme for F has a running time with* **nonrecursive dependence on** ϵ *?*

We will answer this question affirmative in chapter 4, section 4.2.

Let us point here that in some sense this is astonishing: On the one hand, for an optimization problem to be in **NPO** means polytime computability of the cost function, solution length being bounded polynomially in the input size and so on (cf. the definition above), which implies the problem can be solved to optimality in exponential time $O\left(2^{n^{O(1)}}\right)$, on the other hand we have a non-recursiveness property.

In order to prepare our considerations in chapter 4, we will now define the according subclasses of the class **PTAS**.

Definition 13

Uniform-PTAS *is the class of NPO problems that admit a ptas with running time bounded by $|x|^{f(n)}$ for some recursive function f.*

Uniform-EPTAS *is the class of NPO problems that admit an eptas with running time bounded by $g(n) \cdot |x|^{\alpha}$ for some constant α and some recursive funtion g.*

Chapter 3

Fixed Parameter Tractability

In this Chapter we give a brief introduction into Fixed Parameter Complexity Theory and study some structural aspects of classes of the W-Hierarchy. Fixed Parameter Complexity is motivated by the observation that for various algorithmic problems one is able to identify some parameter of the problem such that fixing this parameter to constant makes the problem algorithmically easier. The notion of **fixed parameter tractability** was invented by Downey and Fellows [DF95a, DF95b]. It has close connections to the NP completeness theory [CCDF95, BG94] and complexity theory of optimization problems [CC97].

Our specific interest in Fixed Parameter Complexity is due to its close connection with approximation complexity of NP-hard optimization problems [CC97, CT97a]. In chapter 4 we will study structural properties of the class NPO with some focus on the existence of efficient polynomial time approximation schemes. Recall that a **Polynomial Time Approximation Scheme** for some optimization problem X has running time $O\left(|I|^{f(1/\epsilon)}\right)$ for some function f (**ptas**), while **Efficient Polynomial Time Approximation Schemes (eptas)** have running time $O\left(f\left(\frac{1}{\epsilon}\right) \cdot |I|^c\right)$ for some constant c. hence the exponent of the running time bound does not depend on ϵ. Accordingly **PTAS** and **EPTAS** denote the classes of **NPO** problems with polynomial time approximation scheme and efficient polytime approximation scheme respectively.

While for some optimization problems which were previously known to fall into class **PTAS**, the existence of an efficient polynomial time approximation scheme could finally be established (e.g. the geometric versions of the **Steiner Tree Problem**, **Travelling Salesman Problem** and similar routing problems, see the paper by Sanjeev Arora [Aro98]), for other problems like the **dense versions** of the **Steiner Tree Problem** and its variants which we consider in chapter 9, existence of an efficient polytime approximation scheme remains an open problem.

It is then natural to ask whether these classes are distinct, under some natural complexity theoretic assumption like $\mathbf{P} \neq \mathbf{NP}$. Unfortunately, under this standard assumption it is open how to prove $\mathbf{EPTAS} \neq \mathbf{PTAS}$. Cesati and Trevisan [CT97a] were able to prove the following:

$$\mathbf{FPT} \neq \mathbf{W[P]} \implies \mathbf{EPTAS} \neq \mathbf{PTAS} \implies \mathbf{FPT} \neq \mathbf{SP}$$

Here

$$\mathbf{FPT} \subseteq \mathbf{W}[\mathbf{1}] \subseteq \mathbf{W}[\mathbf{2}] \subseteq \ldots \subseteq \mathbf{W}[\mathbf{t}] \subseteq \ldots \subseteq \mathbf{W}[\mathbf{P}] \subseteq \mathbf{SP}$$

denotes the W-hierarchy introduced by Downey and fellows.

In chapter 4, section 4.3 we separate **EPTAS** from **PTAS** under some different complexity theoretic assumption, namely existence of problems in NP with a super-polynomial lower bound for the deterministic time complexity. Now, having two proofs of the same result under different assumptions, it is natural to ask **how different** they are, e.g. whether one implies the other. Fortunately, in chapter 4, section 4.4 we give a proof that using **relativizing proof techniques** one can not show that our assumtion implies the above one from Cesati and Trevisan.

It is also interesting to investigate the fixed parameter complexity of Steiner Tree Problems which we will consider in chapters 5-8 of this thesis. The Steiner Tree Problem itself is known to be $W[2]$-complete if we take as a parameter the number of Steiner points in the tree [BfHW00] On the other hand, if we parameterize the number of terminals, the problem falls easily into class FPT, using the Dreyfus Wagner algorithm. We are not aware of fixed-parameter results on the generalizations like the **Steiner Forest Problem** and the other Steiner like problems we consider in subsequent chapters. For the Steiner Forest problem, the besk known hardness result is the same as for the **Steiner Tree Problem** while the best known upper bounds differ drastically (factor 1.55 for the Steiner Tree Problem, cf. chapter 5 and factor 2 for the **Steiner Forest Problem**, cf. chapter 6). Hence a fixed parameter result different from that for the Steiner Tree Problem would be very desirable.

This is motivation enough for us to take a closer look at the field of **Fixed Parameter Complexity**. In this chapter we will give a very brief introduction into the basic notions and concepts. We will define the **W-Hierarchy**, the reducibility concepts being in use and give a very selective list of complete problems for the various levels of the W-Hierarchy.

Furthermore, we take a look at some structural aspects of **Fixed Parameter Complexity**. There are some fundamental results on complexity measures well known in the literature, including the famous **Speedup Theorem** by M. Blum [Blu67], the Union Theorem of McCreight and Meyer [MM69] and Borodin,s **Gap theorem** [Bor72].

The Speedup Theorem [Blu67] states that there are recursive languages without asymtotically fastest algorithm, e.g. such that for each algorithm deciding the language there is another algorithm which decides the language as well and has running time bounded by the squareroot of the former running time. Instead of squareroot on can place here any recursive function.

The Union Theorem [MM69] allows to "give names to complexity classes", in the following sense: If $f_i(n), i \in \mathbb{N}$ is a recursive family of functions then the union $\bigcup_i DTIME(f_i(n))$ can be written as a single class $DTIME(f(n))$ for some recursive function $f(n)$. As a direct application, there is a recursive function $f(n)$ such that $P = DTIME(f(n))$. In the above formulation, $DTIME$ can be replaced by some arbitrary Blum complexity measure.

Borodin's Gap Theorem [Bor72] states that within the deterministic time hierarchy (and any hierarchy provided by a Blum complexity measure) there are arbitrary recursive gaps.

Another fundamental result is Levin's **Lower Bound Theorem** [Lev74] (in russian, see [Lev96] and [All99] for formulations and proofs in english). The theorem states that for every recursive language L there is a recursive function $g(n)$ such that for all space complexity functions $s(n)$, $L \in DSPACE(s(n))$ if and only if $s(n) = \Omega(g(n))$.

Our structural results are the following: In section 3.3.1 we prove an analog of Blum's Speedup Theorem for the class **SP** of parameterized problems solvable in time $O(|x|^{f(n)})$ for some function $f(n)$. Our result allows to **speed up** $f(n)$, the running time dependence on the parameter.

In section 3.4 we prove an analog of Levin's Lower Bound Theorem for the classes **SP** and **FPT**, again providing lower bounds for the dependence on the parameter. Here our lower bound functions are in general not recursive anymore, but provide some weaker computational property: They are guaranteed to be **recursively approximable from below**. Furthermore we consider the question whether the original Lower Bound Theorem for the language case holds for all Blum complexity measures. We leave this question open but are able to identify a reasonable subclass of Blum complexity measures for which it holds. Finally we prove a version of the Lower Bound Theorem for **Randomized Space Complexity**.

3.1 Basic Definitions

The definitions and notions which we list in this section are taken from [DF92, DF95a, DF95b].

Definition 14 (Parameterized Languages, Parameterized Decision Problems)
A parameterized language or parameterized decision problem is a set $L \subseteq \Sigma^* \times \mathbb{N}$. Here for $(x,k) \in L$ the string k is interpreted as a parameter. For a given parameterized language L and a number $k \in \mathbb{N}$, $L_k = \{x \in \Sigma^* : (x,k) \in L\}$ is called the k-th slice of L.

Definition 15 (Fixed Parameter Tractability)
A parameterized problem L is called

(a) nonuniformly fixed-parameter tractable if there is a constant α and a family $T_k, k \in \Sigma^*$ of algorithms (i.e. deterministic Turing machines) such that for each $k \in \Sigma^*$ algorithm T_k recognizes the language L_k in time $O(n^\alpha)$.

(b) uniformly fixed-parameter tractable if there is a constant α and an algorithm T such that T decides if $(x,k) \in L$ in time $f(|k|) \cdot |x|^\alpha$ for some function $f\colon \mathbb{N} \to \mathbb{N}$.

(c) strongly uniformly fixed-parameter tractable if there is a constant α and an algorithm T such that T decides if $(x,k) \in L$ in time $f(|k|) \cdot |x|^\alpha$ for some recursive function $f\colon \mathbb{N} \to \mathbb{N}$.

Definition 16 (Complexity Classes of Parameterized Problems)
FPT *is the class of parameterized languages* $L \subseteq \Sigma^* \times \mathbb{N}$ *for which there exixsts a deterministic algorithm* $\mathcal{A}$ *that decides* L *and has running time bounded by* $f(n) \cdot |x|^{O(1)}$ *for some function* $f\colon \mathbb{N} \to \mathbb{N}$*. Problems in* **FPT** *are called* **fixed parameter tractable**.
SP *is the class parameterized languages* $L \subseteq \Sigma^* \times \mathbb{N}$ *for which there exixsts a deterministic algorithm* $\mathcal{A}$ *that decides* L *and has running time bounded by* $O\left(|x|^{f(n)}\right)$ *for some function* $f\colon \mathbb{N} \to \mathbb{N}$*.*

Definition 17 (Parameter-Reducibility, P-Reducibility)
Let A *and* B *be two parameterized problems. For* $y \in \Sigma^*$ *we let*

$$B^{(y)} := \bigcup_{k \leq y} B_k = \{(x,k) : k \leq y \text{ and } (x,k) \in L\}.$$

(a) A *is nonuniformly P-reducible to* B *(*$A \leq_P^n B$*) if there is a constant* α*, a function* $f\colon \mathbb{N} \to \mathbb{N}$ *and a family of oracle algorithms* $T_k, k \in \Sigma^*$ *(oracle TM's) such that for each* $k \in \Sigma^*$ $L(T_k, B^{(f(|k|))}) = A_k$ *with running time* $f(k)|x|^\alpha$*.*

(b) A *is uniformly P-reducible to* B *(*$A \leq_P^u B$*) if there is a constant* α*, a function* $f\colon \mathbb{N} \to \mathbb{N}$ *and an oracle algorithm* T *such that* $T(B)$ *(algorithm* T *with oracle* B*) decides* L*, the running time of* $T(B)(x,k)$ *is bounded by* $f(k)|x|^\alpha$ *and on input* (x,k) T *only asks oracle questions to* $B^{(f(k))}$*.*

(c) A *is strongly uniformly P-reducible to* B *(*$A \leq_P^s B$*) if* $A \leq_P^u B$ *with* T, f *and* α *as in (b) and furthermore function* f *is recursive.*

The W-Hierarchy
The W-Hierarchy was introduced by Downey and Fellows [DF92] in order to study fixed-parameter intractability. It is defined in terms of Boolean circuits *of mixed type*:

Definition 18 *A* **Boolean circuit of mixed type** *is a Boolean circuits having gates the following kind:*

(1) **Small gates: not** *gates,* **and** *gates and* **or** *gates with bounded fan-in (fan-in* 2 *for* **and** *and* **or***, fan-in* 1 *for* **not***.*

(2) **Large gates: and** *gates and* **or** *gates with unbounded fan-in.*

Furthermore the following notions are used in Fixed Parameter Complexity: The **depth** of a Boolean circuit C of mixed type is the maximum number of gates on a directed path in C. The **weft** of C is the maximum number of large gates on an input-output path in C. A family $\{C_n, n \in \mathbb{N}\}$ of circuits of mixed type has **bounded depth** if there exists some constant h such that each C_n has depth at most h. Similarly families of **bounded weft** are defined.

Definition 19 (The Classes $W[t]$ and $W[P]$)
(a) *A parameterized problem* L *belongs to* $W[t]$ *if there exists a constant* $h \in \mathbb{N}$ *such*

that L is P-reducible to the parameterized circuit problem $L_{F(t,h)}$ for the family $F(t,h)$ of mixed type decision circuits of weft at most t and weft at most h.
(b) *A parameterized problem L belongs to $W[P]$ if L is P-reducible to the parameterized circuit problem L_F for the family F of mixed-type circuits (with no restriction on weft and depth).*

3.2 Complete Problems

In this section we list some complete parameterized problems for some levels of the W-Hierarchy. More comprehensive lists can be found in [DF95a, DF95b].

Generic Complete Problems for $W[t]$

Bounded Weight t-Normalized Satisfiability
Instance: t-normalized Boolean expression X ($t \geq 2$), positive integer k

Question: Does X have a satisfying truth assignment of weight at most k ?

Parameter: k : $W[t]$-complete

Remark: X is called **t-normalized** if it is of the form

$$X = \underbrace{\bigwedge_{j_1 \in J_1} \bigvee_{j_2 \in J_2} \cdots \bigwedge_{j_t \in J_t}}_{t \text{ alternations}} L(j_1, \ldots, j_t)$$

in the case when t is odd, where for each sequence $j_1, \ldots, j_t$ of indices $L(j_1, \ldots, j_t)$ is a literal, and

$$X = \underbrace{\bigwedge_{j_1 \in J_1} \bigvee_{j_2 \in J_2} \cdots \bigvee_{j_t \in J_t}}_{t \text{ alternations}} L(j_1, \ldots, j_t)$$

in case t is even.

Complete Problems for $W[2]$

Dominating Set

Instance: Graph $G = (V, E)$, positive integer k

Question: Is there a set of k vertices $V' \subseteq V$ such that every vertex of $V \setminus V'$ has a neighbor in V' ?

Parameter : k
$W[2]$-complete (membership is obvious, hardness by a reduction from Weighted CNF

Satisfiability)

Steiner Tree Problem

Instance: Graph $G = (V, E)$, subset $S \subseteq V$ with $|S| \leq k$, an integer $m \in [0, |V|\,]$

Question: Does there exist a set of vertices $T \subseteq V \setminus S$ such that $|T| \leq m$ and the induced subgraph $G[S \cup T]$ is connected ?

Parameter : m
$W[2]$-complete (reduction to Short Multi-Tape NTM Computation, reduction from Dominating Set)

Parameter : k
In FPT (solvable in time $O(3^k n + 2^k n^2 + n^3)$ by the Dreyfus-Wagner algorithm)

Complete Problems for $W[P]$

Bounded Nondeterminism Turing Machine Computation

Instance: Single-tape single-head nondeterministic Turing machine T, an input $x \in \Sigma^*$, positive integers k, m

Question: Does there exist an accepting computation path of $T(x)$ having at most m steps and at most k nondeterministic steps ?

Parameter: k : $W[P]$-complete

3.3 Structural Properties

In this section we are concerned with various structural properties of the W-Hierarchy. Previous work on structural aspects of Fixed Parameter Complexity includes the work of Downey and Fellows [DF95a, DF95b] and especially [DF93], who studied several reducibility concepts, separation and density aspects and uniformity questions.

In the classical complexity theory of decision problems the following three results are well-known: The **Speedup Theorem** given by Manuel Blum [Blu67], Borodin's **Gap Theorem** [Bor69, Bor72] and the **Union Theorem** by McCreight and Meyer [MM69].

The **Speedup Theorem** [Blu67] roughly tells us that there exist computable problems without fastest algorithm: Given a recursive monotone increasing function $r\colon \mathbb{N} \to \mathbb{N}$ (the **speedup**), there exists a recursive language L such that the following holds: When $\mathcal{A}$ is an algorithm deciding L with running time $t(n)$, then there exists an algorithm $\mathcal{B}$ deciding L as well such that for the running time $t'(n)$ of algorithm $\mathcal{B}$

the following holds:

$$r(t'(n)) \quad \leq \quad t(n).$$

If we choose for example $r(n) = n^2$, the result reads as follows: There exists a computable problem L such that whenever we have an algorithm for L with running time $t(n)$, then there also exists an algorithm with running time $\sqrt{t(n)}$, and then also some with running time $\sqrt[4]{t(n)} = \sqrt{\sqrt{t(n)}}$ and so on.

Borodin's **Gap theorem** [Bor69, Bor72] states that the deterministic time hierarchy (and indeed each hierarchy according to some **Blum Complexity Measure**) provides gaps of arbitrary recursive width: Given a recursive function $g(n)$ (assume it to be strictly monotone increasing), there exists a recursive function $S(n)$ such that

$$DTIME(g(S(n))) \quad = \quad DTIME(S(n)),$$

hence we have a gap with width g. This somehow seems to contradict the well known **Hierarchy Theorem**, which guarantees some kind of density inside the Deteministic Time Hierarchy. The **Hierarchy Theorem** imlies that - for example - for every $\epsilon > 0$ we have

$$DTIME\left(n^c\right) \subsetneq DTIME\left(n^{c+\epsilon}\right).$$

So how to solve this seemingly contradictory statements ? The answer is: there is no contradiction at all. The **Hierarchy Theorem** establishes density for **time complexity functions**, i.e. those functions $t(n)$ which are computable in time $t(n)$, while the gap providing function $S(n)$ is recursive but in general not a time complexity function.

The **Union Theorem** of McCreight and Meyer [MM69] allows us "to give names" to complexity classes: If $t_i(n)$ is a recursive family of functions (assume each function to be monotone increasing and $t_i(n) \leq t_{i+1}(n)$ for all i, n), then the union of the according time complexity classes can be written as a single time complexity class: There exists a recursive function such that

$$\bigcup_{\mathbf{i} \in \mathbb{N}} \mathbf{DTIME}(\mathbf{t_i}(\mathbf{n})) \quad = \quad \mathbf{DTIME}(\mathbf{t}(\mathbf{n})).$$

As a direct application, there exists a recursive function $t(n)$ such that

$$P \quad = \quad DTIME(t(n)).$$

Again, the Union Theorem holds for general **Blum Complexity Measures** as well.

Let us mention a fourth result, being as fundamental as the three described so far but seemingly much less known: **Levin's Lower Bound Theorem** [Lev74, Lev96]. It states that for every recursive function f there exists a recursive lower bound for the deterministic space complexity of f, namely a recursive function $g(n)$ such that for all space complexity functions $s(n) = \Omega(\log(n))$ the foloowing holds:

$$f \in DSPACE(s(n)) \text{ if and only if } s(n) = \Omega(g(n)).$$

The theorem can be stated in terms of languages instead of functions as well, furthewrmore the same result holds for deterministic time complexity.

Again, there is seemingly a contradiction: What about languages L that provide some speedup ? Again the answer is given in terms of space complexity functions: The lower bound is in general not a space complexity function which means it is never reached.

In this section we consider variants of the Speedup Theorem and Levin's Lower Bound Theorem for the classes **FPT** and **SP** of Fixed Parameter Hierarchy, namely speedup and lower bound phenomena for the dependence of the running time on the parameter. Thus, we say a parameterized language $L \in SP$ **provides r-speedup**, for $r(n)$ being some monotone increasing recursive function, if the following holds: Whenever $\mathcal{A}$ is an algorithm deciding L with running time $O\left(|x|^{t(n)}\right)$, then there also exists an algorithm $\mathcal{B}$ with running time $O\left(|x|^{t'(n)}\right)$ deciding L such that

$$r(t'(n)) \quad \leq \quad t(n).$$

We will prove this result in subsection 3.3.1 Similarly concerning lower bounds, we will prove that for each problem in SP there is a function $g(n)$ such that

$$L \in SP\left(|x|^{t(n)}\right) \quad \Longleftrightarrow \quad t(n) = \Omega(g(n))$$

holds. Unfortunately the lower bound function $g(n)$ which we construct is in general not recursive, nevertheless we can guarantee some slightly weaker property: The function $g(n)$ can be choosen such that

$$\text{the set } \{(n, L) \mid g(n) \geq L\} \text{ is recursively enumerable}$$

(we call such g **r.e. approximable from below**).

3.3.1 Speedup

In this section we use Blum's approach from [Blu67] to prove speedup theorems for parameterized classes FPT and SP.

Let M_j, c_j be some effective enumeration of pairs where M_j is some TM and $c_j > 0$ some constant such that every pair (M, c) occurs infinitely often in that sequence. Let $t_j(n) := \min\{t \in \mathbb{N} | time_{M_j}(x, n) \leq c_j \cdot |x|^t \text{ for all } x \in \Sigma^*\}$ if this minimum exists and $t_j(n) = \infty$ otherwise.

Theorem 2 (Speedup for SP)
Let $r: \mathbb{N} \to \mathbb{N}$ some monotone increasing recursive function such that $r(n) \geq n^2$ for all n. Then there exists a parameterized language $L \in FPT$ such that for any i, if $L = L(M_i)$ and $t_i(n)$ is defined for all n (i.e. M_j is some SP-algorithm for L), there exists some j such that $L = L(M_j)$ with $t_j(n)$ being defined for all n such that $r(t_j(n)) \geq t_i(n)$ for almost all n.

Proof: Let $h: \mathbb{N} \to \mathbb{N}$ be defined by $h(1) = 2, h(n+1) = r(h(n))$. Then h is a recursive function. As in the case of decision problems (c.f. []), for parameterized language L in order to provide r-speedup it is sufficient to have the following two properties:

(1) For all $j \in \mathbb{N}$, if $L = L(M_j)$ then $t_j(n) \geq h(n-j)$ for almost all n.

(2) For all $k \in \mathbb{N}$ there exists $i \in \mathbb{N}$ such that $L = L(M_i)$ and $t_i(n) \leq h(n-k)$ for almost all n.

The difference between the language case and the parameterized language case is that here for some given $n \in \mathbb{N}$ and $t \in \mathbb{N}$ we cannot decide (recursively) whether $t_j(n) \geq t$ but only enumerate those (n, t) by simulating M_j on instances $< x, n >$ for increasing x.

Let $t_j(x,n) := \min\{t \in \mathbb{N} : \text{time}_{M_j}(x,n) \leq |x|^t\}$ if this minimum exists, then $t_j(n) = \min_{x \in \Sigma^*} t_j(x,n)$. We construct L in stages in terms of a lexicographically increasing sequence of strings $x_n, n \in \mathbb{N}$. We maintain an initially empty list $\mathcal{L}$ of pairs (M_j, m), where M_j is a Turing machine and $m \in \mathbb{N}$. In stage n of the construction we scan the list in lexicographic order (first by m and then by j) and search for violations of (1) on the interval $[0, x_{n-2}]$, i.e. for pairs (M_j, m) in $\mathcal{L}$ such that $t_j(x,m) < h(m-j)$ for all $x \in [0, x_{n-2}]$. If we find such a pair (M_j, m) we check for violation at x_{n-1}: if $t_j(x_{n-1}, m) < h(m-j)$ we enforce $L(M_j) \neq L$ by defining $(x_{n-1}, m) \in L$ iff $M_j(x_{n-1}, m) = 0$ and remove (M_j, m) from $\mathcal{L}$, otherwise we replace (M_j, m) by $(M_j, m+1)$ in $\mathcal{L}$. Let us now give the construction in detail:

Construction of parameterized language L
  Initialization: $\mathcal{L} := \emptyset$, $L := \emptyset, x_0 = 0$
  Stage 1**:** Let $x_1 = 1$ and add $(M_1, 1)$ to $\mathcal{L}$.
  Stage $n > 1$**:**
    for $(M_j, m) \in \mathcal{L}$ (in lex. increasing order)
      if for all $x \in [0, x_{n-2}]$ $t_j(x,m) < h(m-j)$
        if $t_j(x_{n-1}, m) < h(m-j)$
          /* *Diagonalize against* $L = L(M_j)$ */
          Remove (M_j, m) from $\mathcal{L}$
          Let $(x_{n-1}, m) \in L$ **iff** $M_j(x_{n-1}, m) = 0$
        else
          Replace (M_j, m) by $(M_j, m+1)$ in $\mathcal{L}$
        Break for-loop
    Add (M_n, n) to $\mathcal{L}$.
    Let $T := \sum_{(M_j,m) \in \mathcal{L}} |x_{n-1}|^{h(m-j)}$+ time needed to simulate stages 1 to n
    $x_n := 0^T$
  End of Stage n
End of Construction

L **satisfies (1):** Assume $L = L(M_j)$ and $t_j(n) < h(n-j)$ infinitely often. In stage j of the construction it is added to $\mathcal{L}$. Since there are only finite many M_i with $i < j$

and $t_i(n) < h(n-i)$ i.o., after finite many stages M_j is the first entry in $\mathcal{L}$ with that property. Hence at some stage M_j is removed from $\mathcal{L}$ and $L \neq L(M_j)$.
L **satisfies (2):** Let $k \in \mathbb{N}$ be given, we will give an algorithm $\mathcal{A}_k$ for L with running time bounded by $|x|^{h(n-k)}$ for almost all n. Let $J_k \subseteq \{1, \ldots, k\}$ be the set of indices j such that at some stage of the construction some tuple (M_j, m) is removed from $\mathcal{L}$. Let $n(k)$ be the latest stage number at which some of those tuples is removed. Algorithm $\mathcal{A}_k$ will decide $\{(x,n) \in L | x \leq x_{n(k)}\}$ by finite control. For instances (x,n) with $x > x_{n(k)}$ $\mathcal{A}_k$ computes $m \in \mathbb{N}$ such that $x_{m-1} \leq x < x_m$ and checks whether in stage m of the construction of language L it is diagonalized against $L = L(M_j)$ for some j using (x,n). If this is not the case, $\mathcal{A}_k$ returns 0, otherwise $j \geq k+1$ and simulation of $M_j(x,n)$ can be done in $|x|^{h(n-j)} \leq |x|^{h(n-k)}$. Hence the running time of $\mathcal{A}_k$ is bounded by $|x|^{h(n-k)}$ for almost all n, which completes the proof. □

3.4 Excurs: Levin's Lower Bound Theorem

In 1974 Leonid A. Levin [Lev74] proved a theorem which roughly states that for every recursive language L there is a tight lower bound for the deterministic space complexity of L. At the first glance this seems to contradict (the deterministic space complexity version of) Blum's Speedup Theorem which tells us that there are recursive languages without best (in the sense of space-complexity) accepting programs. The solution is as follows: Not every recursive function is a *space function* (i.e. space-computable; a function f is space-computable iff there is a Turing machine M that on input x needs precisely $f(|x|)$ space. Especially the lower bound functions in Levin's theorem are usually not space-computable.

Although being of very similar falvour as Blum's Speedup Theorem [Blu67], Borodin's Gap theorem [Bor72] and the Union Theorem of McCreight and Meyer [MM69] discussed at the beginning of this section, Levin's Lower Bound Theorem seems to be much less popular. The only english description of the Lower Bound Theorem we could find is an unpublished, but online available three-page manuscript by Eric Allender [All99] and as a short summary with appendix by Leonid A. Levin himself [Lev96].

In this subsection we will initially consider the question of how to generalize the theorem to other Blum complexity measures. We show that there are Blum complexity measures (Bcm's) without such a Lower Bound result and identify a class of Bcm's to which the theorem generalizes. Furthermore we prove a variant of the theorem for Fixed Parameter Classes, namely a Lower Bound result for the functional dependence of the exponent on the parameter. We leave as an open problem the complete characterization of the class of all Bcm's providing a Lower Bound result in the sense of L. A. Levin.

The subsection is organized as follows: We will first state Levin's original theorem for deterministic space complexity. Afterwards we give a formulation and proof of the theorem for a reasonable subclass of the class of Blum's complexity measures, namely those that provide efficient simulation of a constant number of Turing machines on the same input. Finally we will prove a version of the Lower Bound Theorem for the

parameterized classes FPT and $W[P]$, namely for the functional dependence of the running time on the parameter.

Theorem 3 (Levin's Lower Bound Theorem [Lev74])
Let f be a total recursive function. Then there exists a total recursive function g such that for any space function s such that $s(n) = \Omega(\log(n))$

$$f \in DSPACE(s) \Leftrightarrow s = \Omega(g).$$

A rather similar result can be obtained for deterministic time complexity instead of space complexity. In his draft [Lev96] L. A. Levin states:

> *"We formulate the theorems in terms of Turing Machine space. But it is clear how to generalize them, since any complexity measure is bounded by a total recursive function (t.r.f.) of any other one. Of course, the accurary of a constant factor will turn in to the accuracy of some other t.r.f." [Lev96]*

We say a Blum complexity measure φ **provides an Ω-Lower Bound** iff for each recursive language L there exists a recursive function $g = g_L$ such that for every φ-complexity function h the following are equivalent: (1) There exists a TM M_i with $L(M_i) = L$ and $\varphi_i(n) = h(n)$.
(2) $h(n) = \Omega(g(n))$.

One may now ask which Blum complexity measures provide an Ω-Lower Bound. The above statement is only a partial answer to that question. Let t be the deterministic time complexity measure. Let f be some recursive function and $t_f := f \circ t$, i.e. $t_f(i,n) := f(t(i,n))$. Then t_f is a Blum complexity measure. Now assume L is a recursive language and g an Ω-lower bound function for L wrto time complexity t. We may now ask whether there exists an Ω-lower bound function for L wrto t_f. Assume g_f is such a function. Let $L = L(M_i)$ for some Turing machine M_i. Then by the lower bound properties we obtain $t(i,n) = \Omega(g(n))$ and $f(t(i,n)) = \Omega(g_f(n))$, i.e. there exist constants $C, C_f > 0$ such that for almost all n

$$\begin{array}{rcl} t(i,n) & \geq & C \cdot g(n) \\ f(t(i,n)) & \geq & C_f \cdot g_f(n) \end{array} \quad \Longleftrightarrow \quad t(i,n) \geq f^{-1}(C_f \cdot g_f(n))$$

t-Complexity Functions vs. t_f-Complexity Functions. If h is a t-complexity function then there exists some TM M_j such that $t(j,n) = h(n)$, hence $t_f(j,n) = f(t(j,n)) = f(h(n))$ and $f \circ h$ is a t_f-complexity function. Versa, if H is a t_f-complexity function then $f^{-1} \circ H$ is a t-complexity function.

Now assume H is such a function, then the following are equivalent:

(i) $H(n) = \Omega(g_f(n))$, i.e. $H(n) \geq C \cdot g_f(n)$ *a.e.* for some $C > 0$

(ii) There is a TM M_i with $t_f(i,n) = f(t(i,n)) \leq H(n)$ and $L(M_i) = L$.

Furthermore for $f^{-1} \circ H$ the following are equivalent:

(iii) $f^{-1}(H(n)) = \Omega(g(n))$, i.e. $f^{-1}(H(n)) \geq c \cdot g(n)$ a.e. for some $c > 0$
(which means $H(n) \geq f(c \cdot g(n))$ a.e.)

(iv) There is a TM M_i with $t(i,n) \leq f^{-1}(H(n))$ and $L(M_i) = L$
(which means $f(t(i,n)) \leq H(n)$).

Now it is natural to ask the following

Question: Are there Blum Complexity Measures φ and recursive languages L such that φ does not provide Ω-Lower Bound to L, which means are there φ and L such that for every recursive function g at least one of the following is true:

(1) There is an M such that $L(M) = L$ and $\varphi_M(n) \neq \Omega(g(n))$.

(2) There is a φ-complexity function $t(n)$ such that $t(n) = \Omega(g(n))$ and $L \notin C_\varphi(t(n))$.

Unfortunately, so far we do not know the answer yet. Let us give a brief sketch of an approach we had in mind in order to give a positve answer and then ppoint out where the difficulties are. One could have the following approach in mind in order to give a positive answer to that question: We take a recursive infinite language L and let $M = M_L$ be a Turing machine deciding this language. Then we try to define a Blum complexity measure (BCM) φ such that φ does not provide Ω-Lower Bound to L. In order to do this we try to enforce L not having **constant** φ-complexity. Then we diagonalize against every total recursive function g which is **not constant**.
But then one problem occurs: We cannot algorithmically enumerate all recursive functions neither all non-constant recursive functions (this is an elementary and well known result from computability theory).

However in the next subsection we will at least identify a class of Blum complexity measures for which existence of Ω-Lower Bounds can be proved.

3.4.1 Ω-Lower Bound for Proper Measures

We will now present a generalized version of Levin's Lower Bound Theorem for a quite reasonable subclass of the Blum complexity measures, namely those for which doing one-step simulation of a constant number of Turing machines on the same input x until the first of these machines terminates is asymptotically of the same complexity as the computation of this machine on input x (plus some small overhead for simulation). We call such complexity measures *proper*.

Let us first recall some standard definitions and notations and then give a precise definition of proper complexity measures. Let

$$M_i,\ i \in \mathbb{N}$$

be a fixed *standard numbering* (Gödel numbering) of the Turing machines, which means there is a universal TM M such that $M(i,x) = M_i(x)$ for all i, x, and the sms theorem holds. A *Blum's complexity measure* for $M_i, i \in \mathbb{N}$ is a partial function $\varphi\colon \mathbb{N} \times \Sigma^* \to \mathbb{N}$ such that *Blum's axioms* hold:

(B1) For all $i \in \mathbb{N}$ and $x \in \Sigma^*$: $\varphi(i,x)$ is defined iff $M_i(x)$ is defined.

(B2) The set $\{(i,x,U) : \varphi(i,x) \leq U\}$ is recursive.

We are now going to introduce a notion for complexity classes of general Blum complexity measures as well as **complexity function**, which are the analog of **time computable functions** and **space computable functions** in the case of general complexity measures.

Definition 20 (Complexity Classses, Complexity Functions)
Let φ a Blum complexity measure. For a given function $t\colon \mathbb{N} \to \mathbb{N}$ we define the **complexity class $\mathbf{C}_\varphi(\mathbf{t}(\mathbf{n}))$** *by*

$$\mathbf{C}_\varphi(\mathbf{t}(\mathbf{n})) \;:=\; \{\mathbf{L} \in \mathbb{N} | \exists \mathbf{i} \in \mathbb{N}\, (\, \mathbf{L}(\mathbf{M_i}) = \mathbf{L} \;\&\; \forall \mathbf{x}\, \varphi(\mathbf{i},\mathbf{x}) = \mathbf{O}(\mathbf{t}(|\mathbf{x}|))\,)\}.$$

Furthermore a function $t\colon \mathbb{N} \to \mathbb{N}$ is called **φ-complexity function** *iff there exists $i \in \mathbb{N}$ such that for all $n \in \mathbb{N}$*

$$\max_{|\mathbf{x}|=\mathbf{n}} \varphi(\mathbf{i},\mathbf{x}) \;=\; \mathbf{t}(\mathbf{n}).$$

We are now ready to give the precise definition of proper complexity measures.

Definition 21 (Proper Complexity Measures)
A Blum's complexity measure φ is called **proper** *iff there exists a recursive function $f = f_\varphi\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that the following holds: For every $k \in \mathbb{N}$ and Turing machines $M_{i_1}, \ldots, M_{i_k}$ there is a Turing machine M_i with the following two properties:*

(1) For every input x,

$$M_i(x) = \begin{cases} \textit{undefined} & \textit{if for all } 1 \leq j \leq k, \quad \varphi(i_j,x) = \infty \\ M_{i_j}(x) & \textit{if } m := \min\{\varphi(i_j,x) | j = 1,\ldots,k\} < \infty \\ & \textit{and } j \textit{ is minimum such that } \varphi(i_j,x) = m \end{cases}$$

(2) $\varphi(i,x) \;\leq\; f(k, < i_1, \ldots, i_k >) \cdot \min\{\varphi(i_j,x) | 1 \leq j \leq k\}$.

Note that the standard measures $DTIME$ and $DSPACE$ are proper in this sense.

Theorem 4 (Lower Bound Theorem for Proper Complexity Measures)
Let $M_i, i \in \mathbb{N}$ be a standard numbering of all Turing machines and φ be a proper Blum's complexity measure for that numbering. Let $l(n)$ be some unbounded increasing recursive function. Then the following holds:

For every recursive language L there exists a recursive function $g\colon \mathbb{N} \to \mathbb{N}$ such that for all φ-complexity functions $t\colon \mathbb{N} \to \mathbb{N}$ with $t(n) = \Omega(l(n))$,

$$\mathbf{L} \in \mathbf{C}_\varphi(\mathbf{t}(\mathbf{n})) \iff \mathbf{t}(\mathbf{n}) = \mathbf{\Omega}(\mathbf{g}(\mathbf{n})).$$

Proof of Theorem 4: In the proof of this result we will follow the lines of Levin's original proof as it is described in [All99]. First we construct a family of functions $p_i\colon \mathbb{N} \to \mathbb{N}$, $i \in \mathbb{N}$ with the following properties:

(1) The function $p\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, $< i, n > \mapsto p_i(n)$ is recursive.

(2) For all i, n: $p_i(n) \geq p_{i+1}(n)$.

(3) For all i: $L \in C_\varphi(p_i(n))$.

(4) For all φ-complexity functions $t\colon \mathbb{N} \to \mathbb{N}$:
$L \in C_\varphi(t(n)) \iff \exists i : \ t(n) = \Omega(p_i(n))$.

The computation of the function $p(i, n) := p_i(n)$ works as follows:

Computation of function p:
Input: i, n
Output: $p_i(n)$
For φ-complexity bounded by $l(n)$ **do**

/⋆ *As usual,* $\chi_L\colon \Sigma^* \to \{0, 1\}$ *denotes the characteristic function of* L. /⋆
Search for some $j \leq i, y \in \Sigma^*$ such that $M_j(y) \neq \chi_L(y)$.
Each such j found in this process is *cancelled*.

Let $A := \{j \leq i : j \text{ not cancelled}\} \cup \{k\}$.
For $c = 1$ **to** ∞ **do**
Check if there is some $j \in A$ such that
for all $x \in \Sigma^n$ $\varphi(j, x) \leq c$.
If so, **return** c.

Obviously the function p is recursive and satisfies $p_i(n) \geq p_{i+1}(n)$ for all i, n. In order to show that property (3) holds, for given $i \in \mathbb{N}$ we let

$$A_f(i) := \quad \text{the set of indices in } \{1, \ldots, i\} \cup \{k\} \text{ that will never be cancelled in any computation of a value } p_i(n).$$

Note that by the definition of $A_f(i)$, $p_i(n) = \min\{\varphi(j, n) | j \in A_f(i)\}$ for almost all n. Furthermore for each $j \in A_f(i)$ we have $L(M_j) = L$. Consider the following algorithm for L:

Algorithm A_i:
Input: x
Do step-by-step simulation of $M_j(x), j \in A_f(i)$
until one of them terminates, say M_{j_0}.
Return the result $M_{j_0}(x)$.

By the definition of $A_f(i)$, this algorithm decides L. Now let $a_i := |A_f(i)|$ and $A_f(i) = \{j_1, \ldots, j_{a_i}\}$. Since φ is proper, there is a function $f_\varphi: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that there exists a Turing machine M_h implementing algorithm A_i such that

$$\begin{aligned} \varphi(h,x) &\leq f_\varphi(a_i, (j_1, \ldots, j_{a_i})) \cdot \min\{\varphi(j,x) | j \in A_f(i)\} \\ &= f_\varphi(a_i, (j_1, \ldots, j_{a_i})) \cdot p_i(|x|) \;=\; O(p_i(|x|)) \end{aligned}$$

since i and hence $A_f(i)$ are fixed. Hence property (3) holds. In order to prove (4) we first observe that "$\Longleftarrow$" directly follows from (2) and the definition of the complexity class $C_\varphi(p_i(n))$. Now suppose $L \in C_\varphi(t(n))$ for a function $t: \mathbb{N} \to \mathbb{N}$. Then there exists a Turing maschine M_i such that $L = L(M_i)$ and $\varphi(i,n) = O(t(n))$. Hence by the definition of p_i we have $p_i(n) = O(t(n))$ which implies $t(n) = \Omega(p_i(n))$.

Let $t_i: \mathbb{N} \to \mathbb{N}$ be the (partial) φ-complexity function defined by $t_i(n) := \max_{|x|=n} \varphi(i,x)$. Assume that our enumeration of Turing machines is such that each machine occurs infinitely often (a standard property of such enumerations which is easily to achieve). Now we construct a function $g: \mathbb{N} \to \mathbb{N}$ with the following properties:

(5) For all i $p_i \geq g$ a.e.

(6) For all φ-complexity functions $t_i: \mathbb{N} \to \mathbb{N}$ with $t_i(n) = \Omega(l(n))$ and all $i \in \mathbb{N}$:
$\exists^\infty n \;\; t(n) < p_i(n) \longrightarrow \exists n > i \;\; t(n) < g(n)$.

Let us first show that (5) and (6) are sufficient in order to prove the theorem. Indeed, let L and $p_i, i \in \mathbb{N}$ be as above and assume (5) and (6) hold. Let $L \in C_\varphi(t(n))$ for some function $t: \mathbb{N} \to \mathbb{N}$. Then there is some $i \in \mathbb{N}$ such that $t(n) = \Omega(p_i(n))$. Then $t(n) = \Omega(g(n))$ by (5). This proves "$\Longrightarrow$" in Theorem 4. It remains to show "$\Longleftarrow$". Assume $t_i(n) = \Omega(g(n))$ and let $M_i = M_{i_2} = M_{i_3} = \ldots$ according to the above assumption. Assume that for all $j \in \mathbb{N}$ $t_i(n) \neq \Omega(p_j(n))$. This means for each l and each constant $c > 0$ there are infinitely many $n \in \mathbb{N}$ such that $t_i(n) = t_{i_l}(n) < c \cdot p_{i_l}(n)$. Hence using (6) we conclude that for each $c > 0$ there exist infinitely many n such that $t_i(n) < c \cdot g(n)$, contradicting our assumption $t_i(n) = \Omega(g(n))$.

We will now describe how to compute a function g with properties (5) and (6) recursively:

Computation of function g:
For $l(n)$ steps **do**
  Check if $\exists\, i < m < n$ with $t_i(m) < g(m)$
  If so, cancel i.
Pick the least uncanceled $i < n$ with $t_i(n) < p_i(n)$
Cancel i, set $g(n) := p_i(n)$.
If no uncanceled $i < n$ exists, set $g(n) := p_{l(n)}(n)$.

It remains to show that g computed as above satisfies (5) and (6). Here property (5) directly follows from the computation of $g(n)$. Now let $t_i(n)$ be a φ-complexity function such that $t_i(n) = \Omega(l(n))$, and assume $\exists^\infty n\, t_i(n) < p_i(n)$. Choose $n > i$ such

that $t_i(n) < p_i(n)$ and within $l(n)$ steps all indices $j < i$ that will ever be canceled in the computation of g are already canceled. Then either i is cancelled as well because there was some $m > i$ with $g(m) > t_i(m)$ or it is not cancelled so far and some $j < i$ will be cancelled implying $g(n) = p_j(n) \geq p_i(n) > t_i(n)$. □(Theorem 4)

3.4.2 Lower Bound Theorem for Parameterized Classes

We will now consider versions of the Lower Bound Theorem for the parameterized classes FPT and SP. Here we are interested in obtaining lower bounds on the dependence of the running time on the parameter, e.g. in the case of $L \in SP$ with an algorithm of running time $O(|x|^{f(n)})$ we ask for lower bounds on $f(n)$.

Let us start with introducing some notations.

Definition 22 *For a function $f\colon \mathbb{N} \to \mathbb{N}$ let $SP\left(|x|^{f(n)}\right)$ be the class of all parameterized languages $L \subseteq \Sigma^* \times \mathbb{N}_0$ which can be decided in time $O\left(|x|^{f(n)}\right)$.*

Recall that a function $g\colon \mathbb{N} \to \mathbb{N}$ is called **r.e. approximable from below** iff the set $\{(n, L) : f(n) \geq L\}$ is recursively enumerable (r.e.). The definition of being **r.e. approximable from above** is similar.

Theorem 5 (Lower Bound Theorem for SP)
Let $l(n)$ be some monotone unbounded recursive function. Let L be in $SP\left(|x|^{f(n)}\right)$ for some function $f\colon \mathbb{N} \to \mathbb{N}$. Then there exists a function $g\colon \mathbb{N} \to \mathbb{N}$ such that

(1) g is r.e. approximable from below.

(2) For all $h\colon \mathbb{N} \to \mathbb{N}$ such that $h(n) = \Omega(l(n))$:
$L \in SP\left(|x|^{h(n)}\right) \Leftrightarrow h(n) = \Omega(p(n))$.

Proof: Let L be decided by TM M_k in time $O\left(|x|^{f(n)}\right)$. First we construct a family of functions $p_i\colon \mathbb{N} \to \mathbb{N}, i \in \mathbb{N}$ with the following properties:

(1) $\{p_i(n)\}$ is r.e. approximable from above, i.e. $\{(i, n, U) : p_i(n) \leq U\}$ is recursively enumerable. Furthermore $L \in SP(|x|^{p_i(n)})$ for all $i \in \mathbb{N}$.

(2) For all $h\colon \mathbb{N} \to \mathbb{N}$ such that $h(n) = \Omega(l(n))$:
$L \in SP\left(|x|^{h(n)}\right) \Leftrightarrow \exists i : h(n) = \Omega(p_i(n))$.

(3) $p_i(n) \geq p_{i+1}(n)$ for all i, n.

We will define $p_i(n) := \sup\{p_i(x, n) | x \in \Sigma^*\}$ for recursive functions $p_i(x, n)$ for which the supremum will be guaranteed to be finite. As in the preceeding subsection in the case of proper measures, we will as well perform bounded search for machines not deciding L and afterwards simulate the remaining machines on input (x, n) with increasing budget on the exponent of running time:

Computation of $p_i(x,n)$:
Input: i, n, x
For $l(n)$ steps do:
  For each $j \leq i$ search for y, m such that
  $M_j(x,m)$ proves that M_j does not decide L.
  Cancel all indices j for which the search succeeds.
$A := \{j \leq i : j \text{ was not cancelled}\} \cup \{k\}$
for $c = 1$ **to** ∞
  Simulate the computations $M_j(x,n), j \in A$
  each for at most $j \cdot |x|^c$ steps.
  If one of them terminates, return $\max\{c, l(n)\}$.

Let us first show that for all i and n the supremum

$$p_i(n) \quad := \quad \sup\{p_i(x,n) | x \in \Sigma^*\}$$

is finite,which directly implies $p_i(n)$ being r.e. approximable from below (first part of property (1)). Therefore note that for index k we have $L(M_k) = L$ (by definition of k) and $k \in A$ (by the above construction). Hence $p_i(x,n) \leq f_k(n)$ for all x, n, and the supremum is finite.

We show now that the second part holds as well, i.e. for all $i \in \mathbb{N}$ $L \in SP(|x|^{p_i(n)})$. Note that we can decide L in time $O\left(|x|^{p_i(x,n)}\right)$ by simulating the comutation of $p_i(x,n)$ and instead of returning c return the result of the first terminating machine (note that for i being fixed, only a constant number of machines whose index was not cancelled have to be simulated). Let $A^i_f(n)$ be the set of indices from $\{j \leq i\} \cup \{k\}$ that will never be cancelled in any computation of a value $p_i(x,n), x \in \Sigma^*$. Note that $k \in A^i_f(n)$. Furthermore, directly from the definition we obtain

$$p_i(n) \; = \; \sup\{p_i(x,n) | x \in \Sigma^*\} = \inf\{t \in \mathbb{N} | \exists\, j \in A^i_f(n)\; (\forall x \; time_{M_j}(x,n) \leq i \cdot |x|^t)\}$$

and from the first equality the second part of property (1) follows. Furthermore

$$m \leq n \quad \Longrightarrow \quad A^i_f(n) \; \subseteq A^i_f(m)$$

since with n growing the search range for violations increases. Let

$$A^i_f \quad := \quad \bigcap\nolimits_{n \in \mathbb{N}} A^i_f(n), \quad \text{note that } k \in A^i_f.$$

In order to prove (2), assume $h(n) = \Omega(l(n))$ and $L \in SP(|x|^{h(n)})$ via machine M_j with running time bounded by $|x|^{h(n)}$. Then j will not be cancelled and hence $p_j(n) \leq h(n)$. Versa, if $h(n) = \Omega(l(n))$ and $h(n) = \Omega(p_i(n))$, we observe that $L \in SP(|x|^{p_i(n)}) \subseteq SP(|x|^{h(n)})$.

Property (3) follows directly from the definition of $p_i(n)$.

Still following the lines of the preceeding proof for the language case, we will now define function $g(n)$ being r.e. approximable from below such that the following conditions are satisfied:

(4) For all $i \in \mathbb{N}$ $p_i(n) \geq g(n)$ a.e.

(5) For all indices i such that $t_i(n) := \inf\{t \in \mathbb{N} | \text{for all } x \; time_{M_i}(x,n) \leq i \cdot |x|^t\}$ is finite for all n and $t_i(n) = \Omega(l(n))$,

$$\exists^\infty n \, t_i(n) < p_i(n) \longrightarrow \exists n > i \, t_i(n) < g(n)$$

(4) and (5) suffice: If $L \in SP(|x|^{h(n)})$ for some $h(n)$ r.e. approx. from below, then there exists some TM M_i with according function $t_i(n) \leq h(n)$ such that $L \in SP(|x|^{t_i(n)})$, hence $h(n) \geq t_i(n) \geq p_j(n)$ for some j and we use property (4)to obtain $t_i(n) = \Omega(g(n))$. Versa, if $t_i(n) = \Omega(g(n))$ then if $L \notin SP(|x|^{t_i(n)})$ we use property (2) for functions $p_i(n)$ combined with property (5) for $g(n)$ in order to obtain a contradiction (again assuming that each machine M_i occurs infinitely often in our enumeration). It now remains to define $g(n)$ and to establish (4) and (5). Again in the definition of $g(n)$, while being structurally completely adapted from the language case, there is some central difference: The functions $t_i(n)$ are not recursive anymore, hence when checking conditions $t_i(m) < B$ for some bound B, the result depends on the size of the input (we have to perform bounded search !), and we cannot even be sure that $t_i(n)$ is well-defined. Hence instead of obtaining recursive g as before, we will define recursive function $g(x,n)$ and let

$$g(n) \quad := \quad \sup\{g(x,n) \mid x \in \Sigma^*\}$$

guaranteeing that the supremum is finite. This then directly implies $g(n)$ being r.e. approximable from below. Hence let us now define $g(x,n)$.

Computation of $g(x,n)$
Input x, n. **For** $l(n)$ steps **do**
Search for some $i < m < n$ such that $t_i(m) < g(x,m)$ so far
(considering values for y from a bounded range $|y| \leq |x|$)

/⋆ *Formally:* $\max_{|y| \leq |x|} t_i(y,m) < g(x,m)$ ⋆/
/⋆ *where* $t_i(y,m) = \inf\{t \in \mathbb{N} | time_{M_i}(y,m) \leq i \cdot |y|^t\}$ ⋆/

Cancel each such i.
endfor
Let i be the least uncancelled index such that $t_i(n) < p_i(n)$ so far.

/⋆ *Formally:* $\max_{|y| \leq |x|} t_i(y,n) < p_i(y,n)$ ⋆/
Cancel i and set $g(x,n) := p_i(x,n)$.
If no such i exists: Set $g(x,n) := p_{l(n)}(x,n)$.

Again property (4) follows directly from the definition of $g(n) = \sup_x g(x,n)$. Assume $t_i(n) = \Omega(l(n))$ is finite for all n and $\exists^\infty n \, t_i(n) < p_i(n)$. Recall that $p_i(n) =$

$\sup_x p_i(x,n), t_i(n) = \sup_x(t_i(x,n)$ and $g(n) = \sup_x g(x,n)$. Since the suprema are finite, let $x_n \in \Sigma^*$ be chosen such that for all n

$$t_i(n) = t_i(x_n, n), \quad p_i(n) = p_i(x_n, n).$$

Furthermore we define according sets of uncancelled indices $G_f^i(x,n)$ and $G_f^i(n)$ by

$$\begin{array}{lcl} G_f^i(x,n) & := & \{j \leq i | j \text{ is not cancelled in the comp. of } g(x,n)\} \\ G_f^i(n) & := & \{j \leq i | j \text{ is never cancelled in any comp. of some } g(x,n)\} \end{array}$$

Hence we have $G_f^i(n) = \bigcap_x G_f^i(x,n)$ and $x < y \rightarrow G_f^i(x,n) \supseteq G_f^i(y,n)$ since the search range increases. If index i is in $G_f^i(n)$ then for no x it will be cancelled in a computation of $g(x,n)$, and for $x > x_n$ such that all indices $j \leq i$ with $j \notin G_f^i(n)$ are cancelled and $t_i(x,n) = t_i(n) < p_i(n) = p_i(x,n)$ we have $g(n) \geq g(x,n) = p_i(x,n) = p_i(n) > t_i(n)$.

If index i is not in $G_f^i(n)$ then it will be cancelled in computations of $g(x,n)$ for all $x > x_0(n)$ for some $x_0(n)$, and hence for n such that $t_i(n) < p_i(n)$ and $i < l(n)$ we will have $t_i(n) < g(n)$ by definition. □

In a similar manner one obtains the following result, we omit the details here.

Theorem 6 (Lower Bound Theorem for FPT)
For every monotone unbounded recursive function $l(n)$ the following holds: For each $L \in FPT$ there exists a function $g(n)$ r.e. approximable from below such that for all functions $h(n) \geq l(n)$ which are r.e. approximable from below,

$$L \in FPT\left(h(n) \cdot |x|^{O(1)}\right) \quad \Longleftrightarrow \quad h(n) = \Omega(g(n)).$$

3.4.3 A Lower Bound Theorem for Randomized Space Complexity

We consider now a version of the lower bound theorem for randomized space complexity. Let us start by recalling some definitions. A probabilistic Turing machine (PTM) is an offline deterministic TM with separate read-only input tape and an additional one-direction read-only "random tape", hence for given input x and string ρ on the random tape M computes an output $\Phi_M(x,\rho)$. Now the *function computed by* M is the partial function $\varphi_M : \Sigma^* \rightarrow \Sigma^*$ defined by

$$\varphi(x) := \begin{cases} y, & Pr\{\Phi_M(x,\rho) = y\} > 1/2 \\ \text{undefined} & \text{if no such } y \text{ exists} \end{cases}$$

The *error probability* e_M of M is the partial function with the same domain as φ and defined by $e_M(x) = Pr\{\Phi_M(x,\rho) \neq \varphi_M(x)\}$. The *probabilistic time and space complexity* of M are given by

$$\begin{array}{lcl} ptime_M(x) & := & \begin{cases} \min\{t \in \mathbb{N}_0 | Pr\{M(x) = \varphi_M(x) \text{ in} \leq t \text{ steps}\} > 1/2\}, & x \in D(\varphi_M) \\ \infty & & \text{otherwise} \end{cases} \\ pspace_M(x) & := & \begin{cases} \min\{s \in \mathbb{N}_0 | Pr\{M(x) = \varphi_M(x) \text{ in} \leq s \text{ space}\} > 1/2\}, & x \in D(\varphi_M) \\ \infty & & \text{otherwise} \end{cases} \end{array}$$

where we let

$$Pr\{M(x) = \varphi_M(x) \text{ in } \leq t \text{ steps}\} := \frac{|\{\rho \in \{0,1\}^t | M(x,\rho) \text{ stops in at most } t \text{ steps}\}|}{2^t}$$

and $Pr\{M(x) = \varphi_M(x) \text{ in } \leq s \text{ space}\}$ is defined accordingly. Then we have the probabilistic time and space complexity classes

$$\begin{aligned} PTIME(t(n)) &:= \{L \subseteq \{0,1\}^* | \exists \text{ PTM } M \text{ such that } \varphi_M = \chi_L \text{ and } ptime_M = O(t(n))\} \\ PSPACE(s(n)) &:= \{L \subseteq \{0,1\}^* | \exists \text{ PTM } M \text{ such that } \varphi_M = \chi_L \text{ and } pspace_M = O(s))\} \end{aligned}$$

M is called a (total) *Randomized Turing Machine* (RTM, also called *Monte Carlo TM*) if φ_M is a total function (i.e. a majority always exists) and furthermore the error probability is bounded (away from 1/2), i.e. there exists a constant $c > 0$ such that $e_M(x) \leq 1/2 - c$ for every x. It is a well known fact that for randomized machines we can decrease error probability to arbitrary small constant while staying within the same time complexity class. In order to define randomized complexity classes, instead of PTMs M we will now consider tuples $\mathcal{M} = (M, c)$ where M is a PTM and $c > 0$ is a constant. We will call the tuple an "RTM" if M is an RTM in the usual sense with error probability bouded by $1/2 - c$. Now for an RTM $\mathcal{M} = (M, c)$ we let

$$\begin{aligned} rtime_{\mathcal{M}}(x) &:= \min\{t \in \mathbb{N}_0 | Pr\{M(x) = \varphi_M(x) \text{ in } \leq t \text{ steps}\} \geq 1/2 + c\}, \\ rspace_{\mathcal{M}}(x) &:= \min\{s \in \mathbb{N}_0 | Pr\{M(x) = \varphi_M(x) \text{ in } \leq s \text{ space}\} \geq 1/2 + c\} \end{aligned}$$

and define the randomized time and space complexity classes

$$\begin{aligned} RTIME(t(n)) &:= \{L \subseteq \{0,1\}^* | \exists \text{ RTM } \mathcal{M} = (M,c) \, [\, \varphi_M = \chi_L \text{ and } rtime_M = O(t(n)) \,]\} \\ RSPACE(s(n)) &:= \{L \subseteq \{0,1\}^* | \exists \text{ RTM } \mathcal{M} = (M,c) \, [\, \varphi_M = \chi_L \text{ and } rspace_M = O(s(n)) \,]\} \end{aligned}$$

Definition 23 *A function $s: \mathbb{N} \to \mathbb{N}$ is called* **randomized space complexity function** *iff there exists a RTM $\mathcal{M}$ such that*

$$s(n) \;=\; rspace_{\mathcal{M}}(n) \quad \textit{for all } n \in \mathbb{N}.$$

Theorem 7 (Lower Bound Theorem for Randomized Space Complexity) *For every recursive language L there exists a recursive function $g: \mathbb{N} \to \mathbb{N}$ such that for every randomized space complexity function $s(n) \geq log(n)$*

$$L \in \; RSPACE(s(n)) \quad \Leftrightarrow \quad s(n) = \Omega(g(n)).$$

Proof. We give a modified version of the proof for the deterministic case. The main difference is that for a given PTM M we do not know how to decide if it is randomized or not. Hence instead of PTMs we will work with pairs

$$\mathcal{M}_i = (M_i, c(i))$$

where M_i is a PTM and $1/2 > c(i) > 0$ is a *guess* for M_i being randomized. The guess is called *correct* if

$$e_M(n) \leq c(i) \text{ for all } n.$$

We are not aware of any procedure how to decide the correctness of the guess, but if a guess is incorrect, we will recognize after finite amount of time. We will now argue that this is still enough in order to prove lower bound theorem.

In the first step we construct a family of functions $p_i\colon \mathbb{N} \to \mathbb{N}$, $i \in \mathbb{N}$ with the following properties:

(1) For every $i \in \mathbb{N}$ $L \in RSPACE(p_i(n))$.

(2) For every space complexity function $s(n)$, $L \in RSPACE(s(n))$ iff there exists an $i \in \mathbb{N}$ such that
$$s(n) = \Omega(p_i(n)).$$

(3) The total function $p\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ defined by $(i, n) \mapsto p(i, n) := p_i(n)$ is recursive.

(4) $p_i(n) \geq p_{i+1}(n)$ for for every $i, n \in \mathbb{N}$.

Computation of $p_i(x)$
Using at most $\log(|x|)$ space
  for $l = 1$ **to** i
    Search for some y such that $e_{M_l}(y) > c(j)$.
    If such y is found, cancel $(M_l, c(l))$.
Let $A = \{l \leq i | (M_l, c(l))$ not canceled$\} \cup \{k\}$.
For $S = 0$ **to** ∞
  if there is some $l \in A$ such that $M_l(x)$ terminates
    within space S **then return** S.

For functions $p_i(x)$ properties (2)-(4) obviously hold. It remains to show (1). Let $A_f(i)$ be the set of indices l such that ($l \leq i$ and l will never be cancelled) or $l = k$. Without loss of generality we can assume $A_f(i) = \{1, \ldots, k\}$. Furthermore we assume that k is odd (if not, add some index $l > k$ such that M_l has precisely the same behaviour as M_1, for any reasonable numbering of TMs such l can be easily constructed). We will now construct an RTM T_i such that $L(T_i) = L$ and $space_{T_i}(n) = O(p_i(n))$.

Computation of $T_i(x)$**:**
  Independently perform step by step simulation of the computations
  $M_j(x), j \in A_f(i)$ starting with space amount $c = 1$.
  Whenever one of the machines wants to use a new tape cell, first
  simulate all other computations until every of them wants to use another cell,
  then increase c by 1 and proceed.
  If one of the simulations has terminated,
    let's say $M_j(x)$, then **return** $M_j(x)$
  else let $c := c + 1$ and proceed.

In order to analyze the algorithm's behaviour we first give a precise definition of the underlying probability space and especially of the probability of events concerning

the behaviour of a single machine M_j on input x. For given input x, for each finite sequence $\rho \in \{0,1\}^m$ for some $m \in \mathbb{N}$ let $s(\rho, j), 1 \leq j \leq k$ be the number of steps up to which machine M_j on input x is simulated. The underlying space is the set Ω of all mappings $\rho\colon \mathbb{N}_0 \to \{0,1\}$ with σ-algebra generated by the events $S_\rho := \{\rho' \in \Omega | \rho \text{ is initial segment of } \rho'\}$ with probabilities $\Pr(\rho) := \Pr(S_\rho) := 2^{-|\rho|}$, $\rho \in \bigcup_{n\in\mathbb{N}_0}\{0,1\}^n$.

We will now estimate the error probability of T_i as follows: Let input x be given and

$$c := \min\{rspace_{M_i}(x) | 1 \leq i \leq k\}$$

the minimum randomized space amout of machines $M_1, \ldots M_k$ on input x. Without loss of generality let

$$rspace_{M_1}(x) \;=\; c.$$

We obtain

$$\begin{aligned}
& Pr\,\{\text{one of the machines } M_1, \ldots, M_k \text{ terminates in space } c \\
& \qquad \text{and returns the correct answer to input } x\} \\
\geq\; & Pr\,\{M_1 \text{ terminates in space } c \\
& \qquad \text{and returns the correct answer to input } x\} \\
=\; & (1 - e_{M_1}(x)) \cdot Pr\{\text{no other machine disturbs this}\} \\
\geq\; & (1 - e_{max}(x)) \cdot (1 - e_{max}(x))^{k-1} \\
=\; & (1 - e_{max}(x))^k
\end{aligned}$$

where

$$e_{max}(x) \;:=\; \max\{e_{M_1}(x), \ldots, e_{M_k}(x)\}$$

is the maximum of error probabilities of machines $M_i, 1 \leq i \leq k$ on input x. Now observe that due to the fact that we have a logarithmic lower bound on the randomized space complexity of L, we can decrease error probabilities of machines M_i to arbitrary small positive constant without spending extra space amount (just repeat the simulations M_i sufficiently often and take majority decision), hence we may assume

$$e_{max}(x) \;\leq\; 1 - \sqrt[k]{\frac{3}{4}}$$

which implies

$$(1 - e_{max}(x))^k \;\geq\; \frac{3}{4},$$

hence T_i is an RTM with error probability bounded by $\frac{1}{4}$ such that

$$L(T_i) = L \quad \text{and} \quad rspace_{T_i}(x) \leq \min\{rspace_{M_l}(x) | 1 \leq l \leq k\}.$$

In the second step we construct function $g\colon \mathbb{N} \to \mathbb{N}$ with the following properties:

(4) $\forall\, i \in \mathbb{N}\; p_i(n) \geq g(n)$ a.e.

(5) For every $i \in \mathbb{N}$: If M_i is randomized with $e_M(n) \leq c(i)$ (which means **the guess** $\mathcal{M}_i = (M_i, c(i))$ is **correct**) and randomized space complexity $s_i(n)$, then

$$\exists^{\infty} n \quad s_i(n) < p_i(n) \implies \exists\, n > i \text{ with } s_i(n) < g(n)$$

Similar to the case of deterministic space complexity, (4) and (5) are sufficient in order to prove the theorem: If $L \in RSPACE(s(n))$ for some randomized space complexity function $s(n)$ then by property (2) there exists $i \in \mathbb{N}$ such that $s(n) = \Omega(p_i(n))$, and from $p_i(n) \geq g(n)$ a.e. we conclude $s(n) = \Omega(g(n))$. On the other hand, assume $s(n)$ is a randomized space complexity function such that $L \notin RSPACE(s(n))$. It suffices to show that for all $k \in \mathbb{N} \quad k \cdot s(n) < g(n)$ infinitely often (i.o.). In analog to the deterministic case, randomized space complexity provides **linear speedup**:

Lemma 3.4.1 *Let $L \in RSPACE(s(n))$ for a randomized space complexity function $s(n) \geq \log(n)$, and let $c > 0$. Then it follows*

$$L \in RSPACE(c \cdot s(n)).$$

Proof of Lemma 3.4.1: Let $L = L(M)$ for some randomized TM such that

$$Pr\{M(x) \text{ returns the correct answer in space bounded by } s(|x|)\} \geq \frac{3}{4}.$$

Encode k tape symbols into one and simulate the comutation of M on blocks each consisting of k tape positions without using extra space. This reduces space amount to $\lceil s(n)/k \rceil \leq s(n)/k + 1$. Choosing $k \geq 2/c$, this term gets bounded by $c \cdot s(n)$ for almost all n. □(Proof of Lemma 3.4.1)

Hence we obtain $L \notin RSPACE(k \cdot s(n))$. Using (2) we obtain:

$$\text{For all } i \text{ there exist infinitely many } x \text{ such that } k \cdot s(x) < p_i(x).$$

In order to construct g, it suffices to mention that we take the approach from the language case (see subsection 3.4.1) and add in the cancelling procedure the check whether M_i looks as being randomized so far (i.e. on inputs $m < n$ in the computation of $g(n)$). Properties (4),(5) are then verified in analog to the deterministic case.

□(Theorem 7)

Chapter 4

On the Structure of NPO

In this chapter we will consider various structural aspects of complexity classes of optimization problems. Our focus will be on the class **PTAS** of NP optimization problems providing polynomial time approximation schemes. Recall that a polynomial time approximation scheme $\mathcal{A}$ for an NP optimization problem X has running time bounded by $\mathbf{O}\left(|\mathbf{x}|^{\mathbf{f(n)}}\right)$ for some function $f(n)$, where we use the notation $\epsilon = 1/n, n \in \mathbb{N}$. $\mathcal{A}$ is called **efficient polynomial time approximation scheme** if instead the running time bound does not exponentially depend on ϵ but is o the form $\mathbf{O}\left(\mathbf{f(n)} \cdot |\mathbf{x}|^{\alpha}\right)$ for some fixed α and some function $f(n)$. The according subclass of $PTAS$ is called **EPTAS**. Now given a problem in $PTAS$ and having established some running time bound $O(|x|^{f(n)})$, we might ask for running time improvements, by decreasing $f(n)$, i.e. replacing it by some more slowly growing function $f'(n)$, or even obtaining an efficient approximation scheme for the problem, replacing exponential dependence of running time on ϵ by multiplicative dependence. For various natural optimization problems this goal has been successfully achieved:

Concerning the **Travelling Salesman Problem** it was a longstanding open question wether the geometric version of the problem in the plane provides a polynomial time approximation scheme. It was Sanjeev Arora [Aro98] who could answer the question affirmatively, giving a polynomial time approximation scheme with running time $\mathbf{O}\left(\mathbf{n} \cdot (\log \mathbf{n})^{\mathbf{f(n)}}\right)$ (called **quasi-linear running time**), see the paper of Mitchell [Mit99] for an independently obtained result. Nevertheless the dependence on $\epsilon = 1/n$ was so drastic that initial attempts to use the approach in practice were hardly promising. But then Rao and Smith [RS] were able to replace running time by $\mathbf{O}\left(\mathbf{f(n)} \cdot |\mathbf{x}| \cdot \log(|\mathbf{x}|)\right)$, obtaining an efficient polytime approximation scheme for geometric TSP (and various related geometric problems). Their approach relies on the concept of *sparse spanner graphs*, see their paper for additional information.

In chapter 9 we will consider **Dense Optimization Problems**, especially dense versions of various **Steiner Tree Problems**. For the dense Steiner Tree Problem, which is roughly spoken the Steiner Tree Problem restricted to graphs in which terminals have high degree, Karpinski and Zelikovsky [KZ97a] obtained a polynomial time approximation scheme with running time $O(|V|^{1/\epsilon})$, where V is the vertes set of the

input graph. To our knowledge it is open whether an efficient approximation scheme for this problem exists.

In the first section we consider PTAS-preserving reductions, i.e. those for which problem A being reducible to problem B and B being in **PTAS** implies A being in **PTAS** as well. Our motivation here is two-fold: First, in chapter 9 we will be concerned with **Dense Optimization Problems**, many of whose provide polynomial time approximation schemes and for some it is unknown wether **efficient polytime approximation schemes** exist. Hence it would be very interesting to obtain some completeness results for those problems, explaining why efficient approximation schemes are unlikely to exist. Our second motivation is a structural one. Observe that in the case of decision problems one has the notion of polynomial time many-one reductions $\leq_m^p$ (also called Karp reductions), on which the NP-completeness theory is based. There we have the following situation: On the one hand the polynomial time Karp reduction provides a rich structure of complete and intermediate sets (assuming $P \neq NP$), on the hand $\leq_m^p$ does not only preserve membership in P but also in NP. In the case of optimization problems and concerning the class $PTAS$, there is already a huge literature on PTAS-preserving reductions and completeness results [PY91, CP91, KST96, KMSV94, KM96], see subsection 4.1.1 for a survey. All of the reductions being defined before have one of the following properties: Either they are PTAS-preserving but do not provide any structure inside class PTAS (namely all PTAS problems are pairwise reducible to each other) or they provide complete and intermediate problems inside class PTAS but are not PTAS-preserving. In section 4.1 we will define a new type of reduction which we call **parameter-reductions** (notion:$\leq_{\mathrm{PAR}}$). This reduction turns out to be PTAS-preserving and provides complete and incomplete problems inside class PTAS. This is achieved to the following effect: parameter-reducibility is not transitive, and there exist problems in PO which are $\leq_{\mathrm{PAR}}$-complete for the class $PTAS$.

In section 4.2 we discuss questions concerning the uniformity (recursiveness) of the running time dependence of approximation schemes on ϵ. We show that there exist problems in NPO that provide a ptas but no ptas with running time depending recursively on ϵ. We also obtain the following combined result: There is a problem in EPTAS that does not have an approximation scheme (efficient or not) with running time recursive in ϵ.

In section 4.3 we will discuss the **EPTAS** versus **PTAS** question. Cesati and Trevisan [CT97b] proved **EPTAS** $\neq$ **PTAS** under the assumtion $FPT \neq W[P]$ from fixed parameter complexity. We will obtain the same separation under a different assumption, namely existence of problem in NP with superpolynomial lower bound on deterministic time complexity. We make use of this assumtion in a diagonal construction of a problem $U \in PTAS \setminus EPTAS$, such that for each $n \in \mathbb{N}$ approximability within $1+1/n$ remains hard for increasing input size. Let us give some explanation: A very simple diagonal construction of a problem in $PTAS$ is as follows: Take a problem $L \in NP \setminus P$. Define optimization problem U_n to have polynomial proofs for membership $x \in L$ as solutions of cost $1 + 1/n$, all other strings (of pol. length in the input) cost 1. Use U_1 to diagonalize against machine M_1, using the fact $L \notin P$. Then

take U_2 to diagonalize against M_2 and so on. Hence the resluting problem U becomes easier and easier to approximate within some given $1+\epsilon$ with input size increasing. An approximation scheme for U works as follows: For input x, n and the task to construct an $1+1/n$ approximate solution, first compute m such that U on input x is defined as U_m. if $m \geq n$ return some arbitrary string of pol. length, otherwise solve the problem by brute force. Hence for fixed $\epsilon = 1/n$, only on an initial interval we have to perform brute force, later on the problem becomes trivial. This approach is not successful in order to prove $EPTAS \neq PTAS$ by showing $U \notin EPTAS$, and the reason is precisely that for each fixed $\epsilon = 1/n$, there is some x_n such that for inputs $x > x_n$ $1+\epsilon$ approximation becomes easy. Making use of our assumtion we can perform a more sophisticated diagonal construction avoiding this effect described above.

It is then natural to ask how the different assumptions under which $EPTAS \neq PTAS$ was proved are related to each other. Does one of them easily imply the other ? Fortunately, we are able in section 4.4 to show that in some sense this is not the case: We construct a recursive oracle X relative to which our assumption holds but that of Cesati and Trevisan does not. Hence using relativizing proof techniques, one can not show that our assumtion implies theirs.

Section 4.4 starts with an introduction into **Bounded Nondeterminism** (subsection 4.4.1). The reason is that Cai and Chen [CCDF95] proved the assumtion $FPT \neq W[P]$ being equivalent to a natural assumption on bounded nondeterminism, namely that whenever we take amount of nondeterminism asymptotically faster than logarithmic the according class is not contained in P anymore. See subsection 4.4.5 for the details. We will briefly describe the result of Beigel and Goldsmith [BG94] which proves that using relativized proof techniques one can not obtain separation results for the Kintala-Fischer hierarchy. The Beigel Goldsmith results provide (besides others) a recursive oracle X such that relative to X the Kintala-Fischer hierarchy [KF84] becomes strict:

$$\mathbf{P^X} = \mathbf{NP^X}[\log \mathbf{n}] \neq \mathbf{NP^X}[\log^{\mathbf{2}} \mathbf{n}] \neq \ldots \neq \mathbf{PSPACE^X}$$

In some sense, we obtain a refinement of their result concerning the first and second level: In subsection 4.4.6 we construct a recursive oracle X such that for all unbounded polynomial time computable functions $s(n)$

$$\mathbf{P^X} \quad = \quad \mathbf{NP^X}[\log(\mathbf{n})] \quad \neq \quad \mathbf{NP^X}[\mathbf{s(n)} \cdot \log(\mathbf{n})].$$

Furthermore at the end of subsection 4.4.4 we indicate how this result might be further extended, towards a refinement of the Beigel Goldsmith results along the whole Kintala Fischer hierarchy.

4.1 PTAS-Preserving Reductions and PTAS-Completeness

In order to compare problems with respect to their computational difficulty, the concept of reductions builds a poweful tool. While the notion of reduction and completeness

was originally introduced in mathematical logic and recursion theory, we will here focus on **reduction concepts for NP Optimization Problems**.

A reduction between two optimization problems A and B basically consists of two ingredients: 1. a mapping from instances of problem A to instances of problem B, 2. a mapping from solutions of B to solutions of problem A. In order to make computational properties of one problem imply computational properties of the other, the mappings have to be resoure-bounded computable (e.g. computable in polynomial time). Furthermore the precise reduction concept must be carefully chosen such that approximation properties are preserved. Accordingly, reductions that preserve the existence of constant-factor approximation algorithms are called **APX-preserving**, reductions that preserve the existence of polynomial time approximation schemes are called **PTAS-preserving** and so on.

In this section we will consider reductions which preserve existence of polynomial time approximation schemes, i.e. when problem A is reducible to problem B and problem B is already known to fall into class **PTAS**, then problem a falls into class **PTAS** as well.

There does already exist a huge literature on reductions between **NP Optimization problems**. Let us mention some of the work which was already done before:

4.1.1 Previous Work

In this subsection we give a list of previously defined approximation preserving reducibilities and their main properties. A good survey on some of the various reducibilities and their structural properties can be found in [CKST96]. The first reduction we will consider wa sdefined by Crescenzi and Panconesi [CP91] in order to provide structure inside PTAS.

Definition 24 (F-Reductions [CP91])
Let $A, B \in NPO$. A is F-reducible to B, in symbols $A \leq_F B$, if there exist functions f, g, r such that for all $x \in I_A$ $f(x) \in I_B$ and f is polynomial-time computable, for every $x \in I_A$ and $y \in S_B(f(x))$ $g(x, y) \in S_A(x)$ and g is polynomial-time computable, $r\colon \mathbb{N} \times I_A \to \mathbb{N}$ and $r(n, x)$is computable in time $p(n, |x|)$ for some polynomial p, $r(n, x) \leq q(n, |x|)$ for some polynomial q and furthermore for any $x \in I_A, y \in S_B(f(x))$

$$\mathcal{E}_B(f(x), y) \leq \frac{1}{r(n, x)} \implies \mathcal{E}_A(x, g(x, y)) \leq \frac{1}{n}. \tag{4.1}$$

F-reductions are FPTAS-preserving, and in [CP91] existence of complete and intermediate problems in PTAS are shown. Note that F-reductions do not preserve existence of polynomial time approximation schemes, see the original paper for a discussion.

L-Reductions were defined by Papadimitriou and Yannakakis:

Definition 25 (L-Reductions [PY91])
Let $A, B \in NPO$. A is L-reducible to B, in symbols $A \leq_L B$, if there exist functions f, g and constants $\alpha, \beta > 0$ such that for any $x \in I_A$ $f(x) \in I_B$ and f is polynomial-time

computable, for any $x \in I_A$ and $y \in S_B(f(x))$ $g(x,y) \in S_A(x)$ and g is polynomial-time computable and furthermore the following conditions hold:

(a) For any $x \in I_A$ $opt_B(f(x)) \leq \alpha \cdot opt_A(x)$.

(b) For any $x \in I_A$ and $y \in S_B(f(x))$

$$|opt_A(x) - c_A(x, g(x,y))| \leq \beta \cdot |opt_B(f(x)) - c_B(f(x), y)|.$$

L-Reductions are PTAS-preserving, but all problems inside PTAS are pairwise L-reducible to each other. Papadimitriou and Yannakakis proved MAXSNP-hardness with respect to L-reductions for many natural problems, including Bounded Degree Vertex Cover, Bounded Degree Dominating Set, Bounded Degree Maximum Independent Set and Maximum Cut.

Khanna, Motwani, Sudan and Vazirani [KMSV94] introduced E-reductions in order to study structural properties of syntactic classes like MAXSNP and computationally defined classes as APX.

Definition 26 (E-Reductions [KMSV94])
Let $A, B \in NPO$. A is E-reducible to B, in symbols $A \leq_E B$, if there exist functions f, g and a constant $\alpha > 0$ such that for any $x \in I_A$ $f(x) \in I_B$ and f is polynomial-time computable, for any $x \in I_A$ and $y \in S_B(f(x))$ $g(x,y) \in S_A(x)$ and g is polynomial-time computable and furthermore the following condition holds: For any $x \in I_A$ and $y \in S_B(f(x))$

$$R_A(x, g(x,y)) \quad \leq \quad 1 + \alpha \cdot (R_B(f(x), y) - 1). \tag{4.2}$$

E-reductions are FPTAS-preserving, but not PTAS-preserving.

Crescenzi and Trevisan [CT00] introduced PTAS-reductions which turn out to be PTAS-preserving, but PTAS problems are pairwise reducible to each other.

Definition 27 ($PTAS$-Reductions [CT00])
Let $A, B \in NPO$. A is $PTAS$-reducible to B, in symbols $A \leq_{PTAS} B$, if there exist computable functions f, g and r such that the following conditions hold:

(a) For every $x \in I_A$ and $n \in \mathbb{N}$ $f(x,n) \in I_B$ and for any fixed n $f(x,n)$ is computable in time polynomial in $|x|$.

(b) For every $x \in I_A$, $n \in \mathbb{N}$ and $y \in S_B(f(x,n))$ $g(x,y,n) \in S_A(x)$ and for any fixed n $g(x,y,n)$ is computable in time polynomial in $|x|$ and $|y|$.

(c) $r\colon \mathbb{N} \to \mathbb{N}$ and for any $x \in I_A, n \in \mathbb{N}$ and $y \in S_B(f(x,n))$

$$R_B(f(x,n), y) \leq 1 + \frac{1}{r(n)} \implies R_A(x, g(x,y,n)) \leq 1 + \frac{1}{n}.$$

AP-reductions were defined by Crescenzi et al. [CKST96]. They preserve existence of polynomial time approximation schemes but again do not provide any structure inside PTAS.

Definition 28 (*AP*-Reductions [CKST96])
Let $A, B \in NPO$. A is AP-reducible to B, in symbols $A \leq_{AP} B$, if there exist computable functions f and g and a constant $\alpha > 0$ such that the following conditions hold:

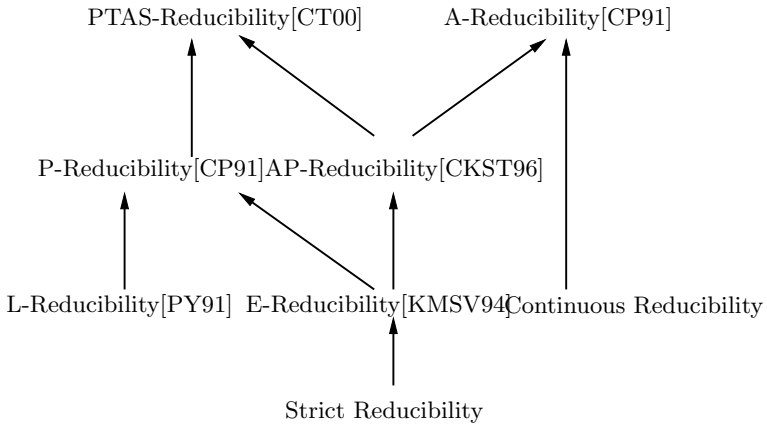
(a) For every $x \in I_A$ and $n \in \mathbb{N}$ $f(x,n) \in I_B$ and for any fixed n $f(x,n)$ is computable in time polynomial in $|x|$.

(b) For every $x \in I_A$, $n \in \mathbb{N}$ and $y \in S_B(f(x,n))$ $g(x,y,n) \in S_A(x)$ and for any fixed n $g(x,y,n)$ is computable in time polynomial in $|x|$ and $|y|$.

(c) For any $x \in I_A, n \in \mathbb{N}$ and $y \in S_B(f(x,n))$

$$R_B(f(x,n),y) \leq 1 + \frac{1}{n} \implies R_A(x, g(x,y,n)) \leq 1 + \frac{\alpha}{n}.$$

Figure 4.1: Previously defined Reducibilities



In the next section we will define a new reducibility concept, the so called **PAR-Reductions**, and we deal with completeness and intermediateness questions with respect to this type of reductions. **PAR-Reductions** differ from PTAS-reduction in that the backward solution to solution mapping g is not allowed anymore to depend on $\epsilon = 1/n$. Note that n general, if we have a backward mapping of the form

$$x, y, n \quad \mapsto \quad g(x,y,n)$$

such that g is polytime computable for fixed n, then dealing with problems $A, B \in PTAS$ we can always use the backward mapping to simply apply a ptas to x, n, obtaining problems in PTAS being pairwise equivalent.

4.1.2 PAR-Reductions

Recall that all the reducibility concepts we listed in that last subsection have one lack in common: Either they are **PTAS-preserving** or they are and the whole class **PTAS** falls inside a single equivalence class with respect to the reduction. Let us now define a new type of reduction which will on the one hand turn out to preserve existence of polynomial time approximation schemes and on the other hand provide some nontrivial structure inside the class **PTAS**.

Definition 29 (PAR-Reductions, Parameter-Reductions)
Let A and B be two NPO problems. A is PAR-reducible (parameter-reducible) to B, in signs $A \leq_{PAR} B$, if there exists a tuple (f, g, r) where f and g are recursive functions $f\colon \Sigma^* \times \Sigma^* \to \Sigma^*$, $g\colon \Sigma^* \times \Sigma^* \to \Sigma^*$ and $r\colon \mathbb{N} \to \mathbb{N}$ is a recursive function with $\lim_{n\to\infty} r(n) = \infty$ such that

(a) For all $x \in I_A$ and $n \in \mathbb{N}$ $f(x, n) \in I_B$ and for every fixed n the function $f(\cdot, n)$ is polynomial time computable.

(b) For all $x \in I_A$: if $y \in S_B(f(x, n))$ then $g(x, y) \in S_A(x)$, $g(x, y)$ is computable in time polynomial in $|x|$ and $|y|$.

(c) $r(n)$ is polynomial-time computable and for every $n \in \mathbb{N}$, $x \in I_A$ and $y \in S_B(f(x, n))$

$$R_B(f(x, n), y) \leq 1 + \frac{1}{r(n)} \implies R_A(x, g(x, y)) \leq 1 + \frac{1}{n}.$$

The last part of property (a) is equivalent to the following: There exists an algorithm $\mathcal{A}_f$ computing f whose running time is bounded by $|x|^{t_f(n)}$ for some function $t_f\colon \mathbb{N} \to \mathbb{N}$.

Lemma 4.1.1 *Let A and B be NPO problems. If $A \leq_{PAR} B$ and $B \in PTAS$ then $A \in PTAS$.*

Proof: Let T_B be a polynomial time approximation scheme for B and $A \leq_{\mathrm{PAR}} B$ via (f, g, r). Let $|x|^{t_B(n)}$ and $|x|^{t_f(n)}$ denote the time bounds for T_B and an algorithm computing f respectively. Consider the following algorithm T_A for A:

Algorithm T_A:
Input: instance x of A, $n \in \mathbb{N}$
Output: $(1 + 1/n)$-approximate solution y for x
(1) Compute $m \in \mathbb{N}$ such that $r(m) \geq 2n$. Let $l := 2n + 1$.
(2) Let y be a $\left(1 + \frac{1}{l}\right)$-approximate solution to $f(x, m)$.
(3) **Return** $g(x, y)$.

We compute a bound on the performance ratio of T_A as follows:

$$\begin{aligned} R_A(x, T_A(x,n)) &\leq \left(1 + \frac{1}{r(m)}\right) \cdot R_B(f(x,m), y) \text{ (Def. of } \leq_{PAR}) \\ &\leq \left(1 + \frac{1}{r(m)}\right) \cdot \left(1 + \frac{1}{l}\right) \text{ (Def. of } y) \\ &\leq \left(1 + \frac{1}{2n}\right) \cdot \left(1 + \frac{1}{2n+1}\right) \\ &= \frac{2n+1}{2n} \cdot \frac{2n+2}{2n+1} = 1 + \frac{1}{n}. \end{aligned} \tag{4.3}$$

Step (1) needs at most $h(n)$ for some recursive function h, since r is recursive and unbounded. Step (2) can be performed in time $|f(x,m)|^{t_B(l)} = |f(x, r_1(n))|^{t_B(r_2(n))}$ for recursive functions r_1, r_2. Step (3) needs at most $(|x| \cdot p_B(|f(x,m)|))^{t_g(n)}$ steps (where p_B is a polynomial bound for the length of solutions to problem B) and hence can also be bounded by $|x|^{r_3(n)}$ for some function r_3. Hence T_A is a ptas for A. □

In order to study closure properties of the subclasses EPTAS, UPTAS and UEPTAS we need the following refinements of definition 29.

Definition 30 *Let A and B be NPO problems.*

A is uniformly PAR-reducible to B ($A \leq^u_{PAR} B$) iff $A \leq_{PAR} B$ by some (f, g, r) such that f and g are computable in time bounded by $|x|^{t_f(n)}$ and $(|x| \cdot |y|)^{t_g(n)}$ for some recursive functions $t_f, t_g \colon \mathbb{N} \to \mathbb{N}$.

A is efficiently PAR-reducible to B ($A \leq^e_{PAR} B$) iff $A \leq_{PAR} B$ by some (f, g, r) such that f is computable in time bounded by $t_f(n) \cdot |x|^\alpha$ for some constant α and some function $t_f \colon \mathbb{N} \to \mathbb{N}$ and g is computable in time $t_g(n) \cdot p_g(|x|, |y|)$ for some function t_g and some polynomial p_g.

A is uniformly efficiently PAR-reducible to B ($A \leq^{ue}_{PAR} B$) iff $A \leq^e_{PAR} B$ by some (f, g, r) with time bounds $t_f(n) \cdot |x|^\alpha$ and $t_g(n) \cdot p_g(|x|, |y|)$ for f and g and furthermore t_f and t_g are recursive.

Lemma 4.1.2 *Let A and B be NPO problems.*

(a) If $A \leq^u_{PAR} B$ and $B \in$ UPTAS then $A \in$ UPTAS.

(b) If $A \leq^e_{PAR} B$ and $B \in$ EPTAS then $A \in$ EPTAS.

(c) If $A \leq^{ue}_{PAR} B$ and $B \in$ UEPTAS then $A \in$ UEPTAS.

Proof of (a): Let $A \leq_{\mathrm{PAR}}{}^u B$ via some tuple (f, g, r). Consider the algorithm T_A from the proof of lemma 4.1.1. If t_B, t_f and t_g are recursive, then obviously $t_B(r_2(n))$ and $r_3(n)$ are recursive, hence the running time of T_A can be bounded by $|x|^{r'(n)}$ for some recursive function r' and A is in UPTAS.
Proof of (b) and (c): If the running time for computing T_B, f and g can be bounded

by $t_B(n) \cdot |x|^\alpha$, $t_f(n) \cdot |x|^\beta$ and $t_g(n) \cdot p_g(|x|, |y|)$ respectively, then in step (2) of algorithm T_A the computation of $f(x, m)$ can be performed in $t_f(h(n)) \cdot |x|^\beta$, $t_B \circ h$ is recursive, $T_B(f(x, m), l)$ can be computed in $t_B(h(n)) \cdot |f(x, m)|^\alpha = t_B(h(n)) \cdot \left(t_f(h(n)) \cdot |x|^\beta\right)^\alpha$ and $g(x, y, n)$ in step (3) can be computed in $t_g(n) \cdot p_g(|x|, |y|) = t_g(n) \cdot p_g(|x|, p_B(f(x, m)))$, hence the total running time of algorithm T_A can be bounded by $H(n) \cdot |x|^\gamma$ for some function $H: \mathbb{N} \to \mathbb{N}$ and some constant γ. If furthermore t_B, t_f and t_g are recursive then H is recursive as well. □

4.1.3 Complete Problems

We will now consider structural properties of $(PTAS, \leq_{\text{PAR}})$. First we will show that problems in **PO** are pairwise reducible to each other:

Lemma 4.1.3 *Let A, B be two problems in PO. Then $A \leq_{PAR} B$.*

Proof: Let x_0 be some fixed instance of B and T be some polynomial time algorithm solving A to optimality. Let $f(x, n) := x_0$ and $g(x, y) := T(x)$, then for $r(n) := n$ the triple (f, g, r) is a PAR-reduction from A to B. □
The next lemma shows existence of **PTAS**-complete problems in PO with respect to PAR-reductions.

Lemma 4.1.4 *There exists $X \in PO$ such that X is $PTAS$-complete with respect to $\leq_{PAR}$.*

Proof: Define X as follows: Instances are tuples $\chi = (x, n, T, F, 0^k)$ as in the proof of ... with $S_X(\chi) = \{T(x, n)\}$ and $c_X(\chi, T(x, n)) = c_F(x, T(x, n))$. Let $A \in PTAS$, consider $f(x, n) := (x, n, T, A, 0^k)$ with T a polynomial time approximation scheme for A and k sufficiently large to compute $T(x, n)$. Let $g(x, T(x, n)) = T(x, n)$ and $r(n) = n$, then $A \leq_{\text{PAR}} X$ via (f, g, r). □

Lemma 4.1.5 *Assume $P \neq NP$. Then there exists a problem $X \in PO$ such that X is not $\leq_{PAR}$-complete for $PTAS$.*

Proof: We let problem X be very simple, namely having solutions of constant size only. We define $X = (I, S, c, \max)$ with instance set $I = \Sigma^*$, where Σ is the underlying alphabet, for each $x \in I$ the set of feasible solutions $S(x) = \{0\}$ and cost function c defined by $c(x, 0) = 1$. Now assume X is $\leq_{\text{PAR}}$-complete for PTAS, and let Y be a problem in $PTAS \setminus PO$ (whose existence is guaranteed under assumption $P \neq NP$). Then there exists a PAR-reduction (f, g, r) from Y to X, where g is polytime computable. But this would imply that the following polynomial time algorithm

$$x, n \quad \mapsto \quad g(x, 0)$$

approximates Y to arbitrary precision (and would hence solve it to optimality), a contradiction. □

The preceeding results seem somehow curious: On the one hand, problems in PO are pairwise reducible to each other, and there exists a problem in PO being $\leq_{\mathrm{PAR}}$-complete for class $PTAS$, on the other hand there exists another problem in PO which is not $PTAS$-complete.

The reason for this seemingly strange situation is that, unfortunately, PAR-reductions do not compose in general, hence $\leq_{\mathrm{PAR}}$ is not transitive. Formally this already follows from the preceeding lemmata (under asssumption $P \neq NP$). Intuitively, one can observe this directly, recognizing the following difficulty:

Assume $A \leq_{\mathrm{PAR}} B$ via (f, g, r) and $B \leq_{\mathrm{PAR}} C$ via (F, G, R). Then how would one try to compose the two reductions in order to get $A \leq_{\mathrm{PAR}} C$? Well, one would try the following most natural approach:

$$\begin{array}{ccccccc} x, n & \longmapsto & f(x,n) & \longmapsto & F(f(x,n), r(n)) \\ & & & & \Downarrow \\ g(x, G(f(x,n), y)) & \longleftarrow & G(f(x,n), y) & \longleftarrow & \text{solution } y \end{array}$$

But then it is obvious that the backward solution-to-solution mapping

$$g(x, G(f(x,n), y)) \qquad \longleftarrow \qquad (x, y)$$

is not polynomial-time computable and not even well-defined since by the definition of PAR-reductions it is not allowed to be dependent on n.

Recall that our motivation for choosing PAR-reductions as they are defined is that with respect to previously defined PTAS-preserving reductions where the backward-mapping is allowed to depend (non-polynomially) on n, the whole class $PTAS$ falls into one equivalence class, i.e. PTAS-problems are pairwise reducible to each other.

We will now define a generic problem U_P which will turn out to be PTAS-complete with respect to PAR-reductions. In the construction we follow the lines of the approach of Crescenzi and Panconesi [CP91], but we have to modify their construction to make it work for PAR-reductions and such that problem U_P becomes complete for both minimization and maximization problems.

Definition of maximization problem U_P:

Instance: $X = \left(x, n, T, N_F, 0^k\right)$, where $x \in \Sigma^*$, n is a natural number, T is a Turing machine, $F = (I_F, S_F, c_F, g_F)$ is an NPO problem and $0^k = 0 \ldots 0$ (k times) serves as a padding string.

Solutions: Consider the following algorithm $\mathcal{A}_{Tr}$.

Algorithm $\mathcal{A}_{Tr}$:
For k steps simulate the following:
If $I_F(x)$ and $S_F(x, T(x,n)$ **then** $t := c_F(x, T(x,n))$.

If k is too small to run $\mathcal{A}_{Tr}$, instance X of U_P has only one solution: $S_U(X) := \{X\}$. Otherwise we set

$$S_U(X) := \begin{cases} \{y \in S_F(x) : c_F(x,y) \geq t\}, & g_F = \max \\ \{y \in S_F(x) : c_F(x,y) \leq t\}, & g_F = \min \end{cases} \tag{4.4}$$

Costs: If k is too small to run $\mathcal{A}_{Tr}$, $c_U(X,X) := 1$, otherwise for $y \in S_U(X)$ the cost is defined as

$$c_U(X,y) := \begin{cases} A(X) + \min\{(1+1/n)t, c_F(x,y)\}, & g_F = \max \\ A(X) + \min\left\{\frac{1+1/n}{t}, \frac{1}{c_F(x,y)}\right\}, & g_F = \min \end{cases} \tag{4.5}$$

Here $A(X)$ is a function of X which will be specified later.
End of definition.

Remark: The subscribt Tr in $\mathcal{A}_{Tr}$ refers to the original construction of Crescenzi and Panconesi ([CP91], p. 249) where they used to describe NP optimization problems in terms of a *trunk* and **branches**, the branches in their notation correspond to different feasible solutions for a given instance.

Lemma 4.1.6 *Function $A(X)$ can be defined such that U_P is in PTAS.*

Proof: By simulating $\mathcal{A}_{Tr}$ and checking wether it terminates within k steps one can check in polynomial time wether $S_U(X) = \{X\}$ or $S_U(X) = S_F(x)$. It suffices to deal with the latter case. Hence in the sequel assume k is large enough. We will then compute an upper bound $E = E(A(X))$ for the performance ratio of solution $T(x,\delta)$ to instance X of U_P in terms of $A(X)$ and then define $A(X)$ such that the following algorithm $\mathcal{A}$ is a polynomial time approximation scheme for U_P:

Algorithm $\mathcal{A}$
(0) **Input:** $X = (x, n, T, F, 0^k), \quad n'$
(1) **If** k is large enough to run $\mathcal{A}_{Tr}$
  (2) **If** $E \leq 1 + 1/n'$ **return** $T(x,n)$
  (3) **otherwise**
    (4) Compute by brute force an optimum solution y^* to
    (5) instance x of problem F and return y^*.
End.

In order to define $A(X)$ we distinguish two cases.
Case 1: $g_F = \max$. We obtain

$$\begin{aligned} R_U(X, T(x,n)) &= \frac{\mathrm{opt}_U(X)}{c_U(X, T(x,n))} = \frac{A(X) + \min\{(1+1/n)t, \mathrm{opt}_F(x)\}}{A(X)+t} \\ &\leq \frac{A(X) + (1+1/n)t}{A(X)+t} =: E \end{aligned} \tag{4.6}$$

Case 1.1: $E \leq 1 + 1/n'$. In this case algorithm $\mathcal{A}$ returns $T(x,n)$ in line (2).
Case 1.2: $E > 1 + 1/n'$. We will define $A(X)$ such that $n' \geq |X|$ and therefore in

line (4)-(5) algorithm $\mathcal{A}$ needs at most $2^{|X|} \cdot |X| \leq 2^{n'}|X|$ steps to compute y^* by brute-force. Therefore we set

$$E \leq 1 + 1/|X| \iff \frac{t/n}{A(X)+t} \leq \frac{1}{|X|} \iff A(X) \geq t \cdot (|X|/n - 1) \quad (4.7)$$

We set $A(X) = t \cdot |X|/n$ in case 1.
Case 2: $g_F = \min$. We obtain

$$\begin{aligned} R_U(X, T(x,n)) &= \frac{\mathrm{opt}_U(X)}{c_U(X, T(x,n))} = \frac{A(X) + \min\{(1+1/n)/t, 1/\mathrm{opt}_F(x)\}}{A(X) + 1/t} \\ &\leq \frac{A(X) + (1+1/n)/t}{A(X) + 1/t} =: E \end{aligned} \quad (4.8)$$

Case 2.1: $E \leq 1 + 1/n'$. Then $T(x,n)$ is a $(1+1/n')$-approximation for instance X of U, and algorithm $\mathcal{A}$ returns $T(x,n)$ in line (2).

Case 2.2: $E > 1 + 1/n'$. Again, in order to solve instance x of F to optimality in polynomial time, we choose $A(X)$ such that $n' > |X|$, i.e.

$$1/n' < 1/|X| \impliedby \frac{A(X) + (1+1/n)/t}{A(X) + 1/t} < 1 + \frac{1}{|X|} \iff A(X) > \frac{|X|/n - 1}{t} \quad (4.9)$$

We set $A(X) = |X|/(t \cdot n)$ in case 2. □(Lemma 4.1.6)

Lemma 4.1.7 *For $A(X)$ defined as in the proof of lemma 4.1.6, problem U_P is PTAS-complete with respect to PAR-reductions.*

Proof: Let $F = (I_F, S_F, c_F, g_F)$ be in PTAS and T a polynomial time approximation scheme for F such that for $n \in \mathbb{N}$ $T(x,n)$ computes a $\left(1 + \frac{1}{n}\right)$-approximate solution to instance x of F and has running time $|x|^{h(n)}$ for some function h. Let $p(n)$ be a polynomial time bound for F, that means I_F, S_F and c_F are decidable resp. computable in time bounded by $p(n)$. In order to construct a PAR-reduction (f, g, r) from F to U_P a first approach would be $f(x,n) := \left(x, n, T, F, 0^k\right)$ with $k := \max\left\{p(|x|), |x|^{h(n)}\right\}$, in order to be able to simulate algorithm $\mathcal{A}_{Tr}$. Unfortunately, f would not necessarily be recursive (since h might be non-recursive). The following choice of k and f avoids this difficulty and is still sufficient to obtain an PAR-reduction:

Computation of $f(x,n)$:
Let t_x be the running time of $T(x,n)$.
Let $k := \max\{p(|x|), t\}$.
$f(x,n) := (x, n, T, F, 0^k)$.

Obviously f is recursive and for fixed n the running time of the algorithm computing f is bounded by some polynomial in $|x|$. Furthermore, since k is large enough to simulate

the trunk,

$$S_U(f(x,n)) \;=\; \begin{cases} \{y \in S_F(x) : c_F(x,y) \le t\}, & g_F = \min \\ \{y \in S_F(x) : c_F(x,y) \ge t\}, & g_F = \max \end{cases}$$

For $y \in S_U(f(x,n))$ we define $g(x,y) := y$, hence g is polynomial-time computable. As above, we let $t = c_F(x, T(x,\delta))$. We consider two cases:

Case 1: $g_F = \max$. Since T is a ptas for F and $g_F = \max$,

$$\mathrm{opt}_F(x)/(1+1/n) \le t \le \mathrm{opt}_F(x).$$

Let $X := f(x,n)$, then $y \in S_U(X)$ and

$$\begin{aligned} R_U(X,y) &= \frac{A(X) + \min\{(1+1/n)t, \mathrm{opt}_F(x)\}}{A(X) + \min\{(1+1/n)t, c_F(x,y)\}} = \frac{A(X) + \mathrm{opt}_F(x)}{A(X) + c_F(x,y)} \\ &= R_F(x,y) \cdot \frac{c_F(x,y)}{\mathrm{opt}_F(x)} \cdot \frac{A(X) + \mathrm{opt}_F(x)}{A(X) + c_F(x,y)} \\ &= R_F(x,y) \cdot \frac{(t \cdot |X|/n)/\mathrm{opt}_F(x) \;+\; 1}{(t \cdot |X|/n)/c_F(x,y) \;+\; c_F(x,y)/c_F(x,y)} \\ &\ge R_F(x,y) \cdot \frac{\frac{1/n}{1+1/n} \cdot |X| + 1}{|X|/n + 1} \;=\; R_F(x,y) \cdot \frac{\frac{1}{n+1} + 1/|X|}{1/n + 1/|X|} \end{aligned} \tag{4.10}$$

from which we obtain

$$R_F(x,y) \;\le\; \frac{(1/n) \cdot |X| + 1}{(1/n) \cdot |X|/(1+1/n) + 1} \cdot R_U(X,y) \le \left(1 + \frac{1}{n}\right) \cdot R_U(X,y) \tag{4.11}$$

Case 2: $g_F = \min$. Then

$$\begin{aligned} R_U(X,y) &= \frac{A(X) + \min\{(1+1/n)/t, 1/\mathrm{opt}_F(x)\}}{A(X) + \min\{(1+1/n)/t, 1/c_F(x,y)\}} \\ &= R_F(x,y) \cdot \frac{\mathrm{opt}_F(x)}{c_F(x,y)} \cdot \frac{(1/n) \cdot |X|/t + 1/\mathrm{opt}_F(x)}{(1/n) \cdot |X|/t + 1/t} \\ &= R_F(x,y) \cdot \frac{1 + (1/n) \cdot |X| \cdot \mathrm{opt}_F(x)/t}{(|X|/n + 1) \cdot c_F(x,y)/t} \\ &\ge R_F(x,y) \cdot \frac{1 + (1/n) \cdot |X|/(1+1/n)}{(|X|/n + 1)} \\ &= R_F(x,y) \cdot \frac{1/|X| + (1/n)/(1+1/n)}{1/n + 1/|X|} \end{aligned} \tag{4.12}$$

which is equivalent to

$$R_F(x,y) \;\le\; \frac{1/n + 1/|X|}{(1/n)/(1+1/n) + 1/|X|} \;\cdot\; R_U(X,y) \;\le\; \left(1 + \frac{1}{n}\right) \cdot R_U(X,y) \tag{4.13}$$

as in case 1. Hence we set $r(n) := n$ and obtain $F \le_{\mathrm{PAR}} U_P$ via (f,g,r). □(Lemma 4.1.7)

Hence we obtain:

Theorem 8 *The problem U_P as defined above is PTAS-complete with respect to PAR-reductions.*

We can now argue that the problem U_P is not polynomial-time solvable to optimality. The reason is that U_P is defined in a very generic way and hence is also PTAS-complete with respect to PO-preserving reductions. In the proof of the following theorem we make only imlicitly use of this fact.

Theorem 9 *If* $\mathbf{P} \neq \mathbf{NP}$, *then* $\mathbf{U_P} \notin \mathbf{PO}$.

Proof: Assume $U_P \in PO$, and let B be some optiomization problem in $APX \setminus PO$ such that T is a polynomial-time $\left(1 + \frac{1}{2}\right)$-approximation algorithm for B. Without loss of generality let B be a maximization problem. Consider the following mapping of instances from B to instances of U_P:

$$x \quad \mapsto \quad X_x \ := \ \left(x, 2, T, B, 0^{p(|x|}\right),$$

where by B we also denote a code for problem B (cf. the above considerations), by T also a code for T and $p(n)$ is a polynomial bound both for the running time of T and the instance and solution checking and cost computation of problem B.

Now, for a solution y to this instance of problem U_P, the cost is defined as

$$A(X_x) + \min\left\{\left(1 + \frac{1}{2}\right) \cdot c_B(x, T(x)), c_A(x, y)\right\} \ = \ A(X_x) + c_B(x, y),$$

hence solving U_P to optimality in polynomial time yields a polynomial time algorithm computiong the optimum for problem B, a contradiction. □

As an example of a natural $\leq_{\mathrm{PAR}}$-complete problem for the class $PTAS$ we like to mention the following variant of the satisfiability problem for cnf formulas, which was already considered and proved to be F-complete for $PTAS$ in [CP91].

Linear Bounded SAT (LBSAT)

Instance: A boolean formula φ with variables $x_1, \ldots, x_n$, weights $w_1, \ldots, w_n$ and a weight W such that $W \leq \sum_{i=1}^{n} w_i \leq (1 + 1/(n-1))W$.

Solution: assignment $\alpha\colon \{x_1, \ldots, x_n\} \to \{0, 1\}$.

Maximize the function

$$f_{LBSAT}(\varphi, \alpha) = \begin{cases} W & \text{if } \varphi(\alpha(x_1), \ldots, \alpha(x_n)) = 0 \\ \sum_{i=1}^{n} w_i \alpha(x_i) & \text{otherwise} \end{cases}$$

We state the following lemma without proof, which is a variation of the F-completeness proof in [CP91].

Lemma 4.1.8 *LBSAT is PAR-complete for PTAS.*

4.2 The Class PTAS: Uniformity versus Efficiency

Concerning structural aspects of the class **PTAS**, two aspects might be of special interest: The first one is that of **PTAS versus EPTAS**: Given a problem in $PTAS$ and having established some running time bound $O(|x|^{f(n)})$, we might ask for running time improvements, by decreasing $f(n)$, i.e. replacing it by some more slowly growing function $f'(n)$, or even obtaining an efficient approximation scheme for the problem, replacing exponential dependence of running time on ϵ by multiplicative dependence. For various natural optimization problems this goal has been successfully achieved, for others this is still an open problem (see the introduction of the chapter for further information). Concerning structural aspects, it is still open whether these two classes can be separated under assumption $P \neq NP$.

The second question is how difficult in the sense of computaional complexity the dependence of the running time on $\epsilon = 1/n$ can be at all. Recall that for an optimization problem X, to be in NPO means solutions being polynomially bounded in the length of the instance, polynomial time computability of the cost function and poly-time checkability whether some string x is a legal instance of the problem and whether string y is a feasible solution to instance x of problem X. Hence in exponentiall time $2^{n^{O(1)}}$ one can solve the problem X to optimality. The structural question now is: Are there problems in $PTAS$ for which there does not exist an approximation scheme with recursive dependence of the running time on $\epsilon = 1/n$?

In this section we will give some first results concerning uniformity and efficiency of polynomial time approximation schemes, answering the question affirmatively.

Theorem 10 *Unless P=NP, for every recursive function $f\colon \mathbb{N} \to \mathbb{N}$ there is a problem $U \in PTAS$ such that:*

1. *There exists no polynomial time approximation scheme for U with running time $|x|^{O(f(n))}$.*

2. *There exists a polynomial time approximation scheme for U with running time $g(n) \cdot |x|$ for some recursive function g.*

Proof: Let $T_i, i = 1, 2, \ldots$ be an effecive enumeration of Turing machines. Let $U_n, n \in \mathbb{N}$ be the family of APX-complete problems from last section, $p(n)$ the polynomial and T the Turing machine such that T has time complexity bounded by $p(n)$ and for each $n \in \mathbb{N}$ U_n is $p(x)$-bounded and T is a $(1 + 1/n)$-approximation algorithm for U_n. We will construct an NPO problem U with the properties 1 and 2 from the theorem in stages in terms of a lexicographically increasing infinite sequence of strings $x_0, x_1, \ldots, x_n, \ldots$ such that the following holds:

(a) We call $I_n := [x_{n-1}, x_n)$ the *n-th interval*. The problem: Given $x \in \Sigma^*$, compute the number n with $x \in I_n$ is polynomial time solvable.

(b) U restricted to I_n is equal to U_n.

(c) For each n there exist pairs $(y_{n,i}, m_{n,i}), 1 \leq i \leq n$ such that for all $i \in \{1, \ldots, n\}$ $y_{n,i} \in I_n$ and $T_i(y_{n,i}, m_{n,i})$ is not a $(1 + 1/m_{n,i})$-approximation to U_n in time $|y_{n,i}|^{f(m_{n,i})}$.

Stage 0**:** Let $x_0 := 0$.
Stage n **(for** $n > 0$**):** Consider the following recursive algorithm for the construction of x_n:

Algorithm Construct
Input: $n \in \mathbb{N}$ **Output:** $x_n \in \Sigma^*$
(1) **If** $n = 0$ **return** $x_0 = 0$ **else**
Let $x_{n-1} :=$ Construct$(n-1)$. Find pairs $(y_{n,i}, m_{n,i}), 1 \leq i \leq n$ by brute force.
(2) Let t_n be the time used for step (1).
$x_n := 0^{t_n+1}$. **Return** x_n.

Here "brute force" in step (1) means the following: we check pairs (y, m) in lexicographic order starting with $(x_{n-1}, 2)$. For each pair (y, m) the computations $T_i(y, m).i = 1, \ldots, n$ are simulated for at most $|y|^{f(m)}$ steps. If a computation stops within that time, the result is compared to the optimum solution $\mathrm{OPT}_{U_n}(y)$ of instance y for problem U_n, which is computed by brute force, i.e. by trying all strings up to length $p(|y|)$. Note that since U_n is APX-complete, it does not permit a ptas and therefore pairs $(y_{n,i}, m_{n,i}), 1 \leq i \leq n$ exist. In order to show that U is an NPO problem it suffices to prove (a). Given x, we can easily compute n with $x \in I_n$ by simulating for at most $|x|$ steps the construction of $x_0, x_1, \ldots$. Let x_m the last string that was constructed by this process, then $n = m + 1$.
By the diagonal construction U does not have a ptas with running time $|x|^{f(n)}$. We will now argue that the following algorithm is an efficient ptas for U with time complexity $g(n) \cdot |x|$ for some recursive function g.

Algorithm Approximate
Input: $x \in \Sigma^*, n \in \mathbb{N}$ **Output:** $(1 + 1/n)$-approximate solution y
(1) Compute $m \in \mathbb{N}$ with $x \in I_m$.
(2) **If** $m \geq n$ **return** $T(x)$ **otherwise**
Compute an optimum solution y_x to instance x of U_n by brute force, **return** y_x

Obviously **Approximate**(x, n) computes a $(1 + 1/n)$-approximate solution y_x to instance x of U. For given $n \in \mathbb{N}$, for $x \geq x_{n-1}$ the running time is $p(|x|)$, for $x < x_{n-1}$ it is at most $|x_{n-1} \cdot 2^{|x_{n-1}|} =: g(n)$. Hence we obtain a time bound of $g(n) \cdot |x|$, and obviously g is a recursive function, which completes the proof. □

Theorem 11 *If P≠ NP then there exists a problem U ∈NPO such that*

1. *There is a polynomial time approximation scheme for U with time complexity $g(n) \cdot p(|x|)$ for some polynomial p and some function $g: \mathbb{N} \to \mathbb{N}$. (Hence U belongs to the class EPTAS.)*

2. *For every recursive function $f: \mathbb{N} \to \mathbb{N}$, U does not have a ptas with time complexity $|x|^{f(n)}$ (hence U does not belong to Uniform-PTAS).*

Proof: We will first describe the main ideas and afterwards give the precise proof: Again we will construct U in stages in terms of an infinite sequence $x_0, x_1, \ldots, x_n, ..$ of strings. As in the proof of the previous theorem, assume $U_n, n \in \mathbb{N}$ to be a family of APX-complete problems with uniform bound $p(x)$ and T a Turing machine with running time bounded by the same polynomial $p(x)$ such that for every $n \in \mathbb{N}$ T is a $(1+1/n)$-approximation algorithm for U_n. Assume further that for each n the existence of a polynomial time $(1 + 1/h_n)$-approximation algorithm would imply P=NP, such that the function $n \mapsto h_n$ is recursive. Let $(A_i, T_i, \alpha_i)_{i \in \mathbb{N}}$ be an effective enumerations of the set of triples (A, T, α) where A and T are Turing machines and $\alpha \in \mathbb{N}$.
The idea of our construction is as follows: In stage n assume we have already constructed the sequence up to x_{m-1} for some $m \leq n$. We start simulating the computation $A_i(h_n)$ for $i = 1, \ldots, n$. In every stage we maintain a list $\mathcal{L} = \{(A_i, n_j, C_i)\}$ of actually simulated computations $A_i(hn_j)$, where C_i denotes the current configuration up to which the computation is already simulated. In stage n, after adding the pairs $(A_i, n, C_i^0), 1 \leq i \leq n$ with initial configurations C_i^0 of computations $A_i(h_n)$ to list $\mathcal{L}$, for all entries of the list we simulate one step. If none of the computations stop, we go to step $n+1$. If, say, (A_i, n_j, C_i) stops (or has already stopped before stage n) and (i, j) is the lexicographically first pair with that property then U restricted to interval $I_m = [x_{m-1}, x_m)$ will be defined as U_j in order to satisfy the following constraint

$C_{n,i}$: If $A_i(h_n)$ stops then $T_i(\ , h_n)$ is not a $\left(1 + \frac{1}{h_n}\right)$-approximation algorithm for U in time $\alpha_i \cdot |x|^{A_i(h_n)}$.

Constraint $C_{n,i}$ can be satisfied by instances $x \geq x_{m-1}$ using the properties of U_n. The crucial fact that will enable us to give a ptas for U is that for every $n \in \mathbb{N}$, $U|I_m$ will be set to U_n only finitely many times (namely when one of the computations $A_i(h_n), i = 1, \ldots n$ stops. Therefore for every $n \in \mathbb{N}$ after at most finitely many steps U will be defined as U_m for $m \geq n$ and hence $(1 + 1/n)$-approximation is possible.
Let us now give the details. First we describe the construction of problem U in stages.

Initialization: $x_0 := 0, m := 1, \mathcal{L} := \emptyset$ (the empty list).
Stage 0**:** Do nothing.
Stage $n > 0$**:** Add $(A_i, n, C_i^0), i = 1, \ldots, n$ to list $\mathcal{L}$, where C_i^0 is the initial configuration of the computation $A_i(h_n)$. For every triple (A, n', C) such that C is not a stopping configuration simulate one further step. If there is no triple in $\mathcal{L}$ whose computation has already terminated, stage n is done. Otherwise let (A_i, n_j, C_i) be the lexicographically first such triple, ordered first by the value n_j and then by A_i. Remove (A_i, h_j, C_i) from $\mathcal{L}$. Let $x \geq x_m$ be the first instance such that either $T_i(x, h_{n_j})$ does not terminate within time $\alpha_i \cdot |x|^{A_i(h_{n_j})}$ or $T_i(x, h_{n_j})$ is not an $(1+1/h_{n_j})$-approximate solution to instance x of U_{n_j}. Such x exists since $(1 + 1/h_{n_j})$-approximation to U_{n_j} in polynomial time is assumed to be NP-hard. Let t_n be the time needed to recursively

compute x_{m-1} by performing stages 0 to $n-1$ and to find x by brute force. Let $x_m := 0^{t_n+1}$.
End of Construction

As before, there is a linear time algorithm which, for given x, computes numbers m, n with $x_{m-1} \leq x < x_m$ and $U|[x_{m-1}, x_m) = U_n$. Hence U is an NPO problem. Consider the following algorithm:

Algorithm Approximate
Input: instance $x \in \Sigma^*$ of U, $n \in \mathbb{N}$
Output: $(1 + 1/n)$-approximate solution y for x
(1) Compute $m \in \mathbb{N}$ with $x_{m-1} \leq x < x_m$.
Compute $n' \in \mathbb{N}$ with $U|[x_{m-1}, x_m) = U_{n'}$.
(2) **If** $n' \leq n$ **return** $T(x)$ **else**
Compute an optimum solution y^* to instance x of $U_{n'}$.
Return y^*.
End.

Obviously **Approximate**(x, n) is a $(1+1/n)$-approximate solution for instance x of U. Since for every $n' < n$ only tuples (A_i, n', C) with $i \leq n'$ are added to list $\mathcal{L}$ and only those tuples can cause $U|[x_{m-1}, x_m) = U_{n'}$, it follows that for all but finitely many m, $U|[x_{m-1}, x_m) = U_{n''}$ for $n'' \geq n$. Hence the running time of **Approximate** can be bounded by $g(n) \cdot p(|x|)$ for some function g.
Now assume that $f: \mathbb{N} \to \mathbb{N}$ is some recursive function and there exists a ptas for U with running time bounded by $c \cdot |x|^{f(n)}$for some constant c. Let $i \in \mathbb{N}$ be such that this ptas be given by Turing machine T_i, f is computed by Turing machine A_i and $c = \alpha_i$. Then in stages $n \geq i$ computations $A_i(h_n)$ are started. Since A_i is total and by the priority rule in the construction of U, for every $n \in \mathbb{N}$ there exists an interval $I = [x_{m-1}, x_m)$ such that $U|I = U_n$ and I contains an x such that $T_i(x, h_n)$ is not a $(1+1/h_n)$-approximate solution to U_n in time bounded by $\alpha_i \cdot |x|^{A_i(h_n)}$, a contradiction. □

4.3 PTAS versus EPTAS

In this section we deal with the problem of separating EPTAS from PTAS. In [CT97b] this problem was solved using some natural assumption from Fixed Parameter Complexity. Indeed, they showed the following chain of implications:

$$W[P] \neq FPT \Longrightarrow EPTAS \neq PTAS \Longrightarrow FPT \neq SP.$$

Note that also the following inclusion is well-known (cf. [DF92]):

$$FPT \neq W[P] \Longrightarrow P \neq NP.$$

In this section we will separate PTAS from EPTAS using only assumptions from classical complexity theory. For any given recursive function $f(n)$ we will construct an

NPO problem that has a polynomial time approximation scheme with time bound $|x|^{f(n)}$ but not an efficient ptas (even not with time bound $g(n) \cdot |x|^{\alpha}$ for non-recursive g). Our first separation is based on an assumption which is as well slightly stronger than $P \neq NP$ but still reasonable, namely existence of function problems in FNP with a tight superpolynomial polynomial time computable bound on the deterministic time complexity (assumption (A)). In a second step we relax this assumtion slightly and separate $PTAS$ from $EPTAS$ under the assumption of existence of a problem in FNP with an exponential lower bound on the deterministic time complexity (assumption (A')). The reason for this relaxation is two-fold: First it would be preferable to separate $PTAS$ from $EPTAS$ under assumption $P \neq NP$, and this is a step in this direction. Secondly, having a separation result under two different complexity theoretic assumptions, it is natural to ask how different they are. At the end of the section we will discuss this question and compare our assumption (A') with the assumption $W[P] \neq FPT$ used by Cesati and Trevisan. We will construct an oracle X relative to which our assumption (A') is true but that of Cesati and Trevisan not, showing that using methods which still hold in relativized worlds one cannot prove that our assumption implies theirs. Currently we do not know how to do the same with assumption (A) instead of (A').

The section is organized as follows: In subsection 4.3.1 we give a brief sketch of the results and methods of Cesati and Trevisan, in subsection 4.3.2 we separate EPTAS from PTAS using our alternative complexity theoretic assumption (A). In section 4.3.3 we give the separation under relaxed assumption (A'). Finally subsection 4.4 contains a discussion of our assumptions, their relation to the standard assumption $P \neq NP$ as well as the following oracle constructions:

- Oracles X_0, Y_0 such that relative to X_0 our assumption (A') is true and relative to Y_0 it is not true.
- Oracles X_1, Y_1 such that relative to X_0 the assumption $W[P] \neq FPT$ is true and relative to Y_1 it is not true.
- An oracle X such that relative to X our assumption (A') is true but the assumption $W[P] \neq FPT$ is not true.

4.3.1 Separation under Assumption $W[P] \neq FPT$

Let us review the second part of the Cesati Trevisan result [CT97b]: Assume $FPT = SP$. Then $EPTAS = PTAS$ follows: If $X \in PTAS$ via approximation scheme $T(x, n)$, then $L_T = \{(x, y, k) | \exists z \ (\ yz = T(x, k) \ )$ is in SP, hence in FPT which can be used to obtainn $X \in EPTAS$. For the first part and detailed information we refer to the original paper [CT97b].

4.3.2 Separation under Assumption (A)

Recall that the class NP of problems which can be decided by polynomial-time bounded nondeterministic TMs can be characterized in terms of polynomially bounded binary

relations in P and is closely related to the class FNP of function problems solvable in nondeterministic polynomial time. For each language $L \in NP$ and every polytime nondeterministic TM M with $L(M) = L$ the relation

$$R = R_{L,M} = \{(x, y) | y \text{ encodes an accepting computation of } M \text{ on input } x\}$$

is in P and has associated computational problem $\Pi_R \in FNP$, and versa for each polynomially balanced relation $R \in P$ the associated language

$$L = L_R = \{x \in \Sigma^* | \text{ there exists } y \in \Sigma^* \text{ with } (x, y) \in R\}$$

is in NP. Also recall that Π_R is the following computational problem:

Π_R :: *Given $x \in \Sigma^*$, either compute some string π such that $(x, \pi) \in R$ (a proof or witness for x, in case $x \in L$) or return "NO" (in case $x \notin L$).*

Our separation between PTAS and EPTAS is based on the assumption that for at least one problem $L \in NP$ there exists a polynomially bounded relation $R \in P$ with $L = L_R$ and a tight superpolynomial polynomial-time computable bound for the deterministic complexity of Π_R:

> **Assumption (A):** There is a language $L \in NP \setminus P$ and a polynomially bounded binary relation $R \subseteq \Sigma^* \times \Sigma^*$ with $L = R_L$ such that
>
> $$\Pi_R \in DTIME(t(n)) \setminus DTIME(o(t(n)))$$
>
> for some polynomial time computable superpolynomial function $t(n)$.

Theorem 12 *Under assumption (A), for every strictly monotone recursive function $f: \mathbb{N} \to \mathbb{N}$ with $f(1) = 1$ there is an NPO problem U_f such that*

1. *U_f has a ptas with running time $|x|^{f(n)+1}$.*
2. *For every monotone increasing function $g: \mathbb{N} \to \mathbb{N}$ and every $\alpha > 0$, U_f does not have a ptas with running time $g(n) \cdot |x|^{\alpha}$.*

In particular, this implies UPTAS $\neq$ UEPTAS and PTAS $\neq$ EPTAS.

In order to prove Theorem 12 we will first define a family $(U_n : n \in \mathbb{N})$ of NPO problems and then construct U_f in stages in terms of problems U_n. The crucial point will be that although problems U_n will be approximable with better and better ratio (with n increasing), for every fixed $\epsilon > 0$ the time complexity of approximation within $1 + \epsilon$ will stay the same.
Let $f: \mathbb{N} \to \mathbb{N}$ be some monotone increasing recursive function as above such that $f(1) = 1$. Let $A \subseteq \Sigma^*$ be some problem in NP with proofchecker T_A and p_A some polynomial such that

1. For all $x \in \Sigma^*$, $x \in L$ iff there is some string π of length $|\pi| \leq p_A(|x|)$ such that $T_A(x, \pi)$ accepts in time $p_A(|x|)$.

2. $WP(L,T)$ is solvable in time $t(n)$ but not in time $o(t(n))$, where $t(n)$ is some polynomial time computable superpolynomial function.

We define a family (U_n) of optimization problems as follows:

Definition of U_n:
Instances of U_n: $X = (x_1, \ldots, x_n, 0^k)$ such that $k \geq 1$, $x_1, \ldots, x_n \in \Sigma^*$ and

$$t(|x_j|) \;\leq\; |X|^{f(2^j)}, \qquad j = 1, \ldots, n-1. \tag{4.14}$$

Solutions: $\pi = (\pi_1, \ldots, \pi_n)$ with $|\pi_j| \leq p_A(|x_j|), j = 1, \ldots, n$

Costs: $c_n(X, \pi) \;=\; 2^n + \sum_{i=1}^{n} T_A(x_i, \pi_i) \cdot 2^{n-i} \in \left[2^n, 2^n + 2^{n+1} - 1\right]$.
End of definition.

Here 0^k serves as a padding string and furthermore $T_A(x_i, \pi_i) \in \{0, 1\}$ denotes the result of computation of machine T_A on input x_i, π_i (1 for accept, 0 for reject).

Lemma 4.3.1 *For all $n \in \mathbb{N}$ U_n is an NPO problem. There exists a two-variate polynomial $q(x, y)$ such that for all $n \in \mathbb{N}$ U_n is $q(x, t_f(n))$- time bounded, where $t_f(n)$ denotes the time complexity of f.*

Proof: Function $f(n)$ can be computed in time $t_f(n)$. Then conditions (4.14) are equivalent to $\log(t(|x_j|)) \leq f(2^j) \cdot \log(|X|), j = 1, \ldots, n-1$ and therefore can be checked in time $t_f(n) \cdot n \cdot \text{poly}(|X|)$. In time $O(n \cdot p_A(|X|))$ one can check whether a given string $\pi = (\pi_1, \ldots, \pi_n)$ is a solution to X. The cost $c_n(X, \pi)$ can be computed in time $O(n \cdot p_A(|X|))$. □

Lemma 4.3.2 *Assume A, T_A and f are as above and the assumption from theorem 12 holds. Then for all $n \in \mathbb{N}$ problem U_n has the following properties:*

1. *For $j = 1, \ldots n-1$ there is a $\left(1 + 2^{-j}\right)$-approximation algorithm for U_n with running time $|x_1| + |x_2|^{f(2^2)} + \ldots + |x_j|^{f(2^j)} \leq n \cdot |X|^{f(2^j)} = O\left(|X|^{f(2^j)}\right)$ (n being fixed).*

2. *For $j = 1, \ldots, n-1$, every approximation algorithm $\mathcal{B}$ for U_n with running time $o\left(|X|^{f(2^j)}\right)$ has approximation ratio at least $1 + 2^{-(j+2)}$.*

3. *There is no polynomial time approximation algorithm for U_n with ratio better than $1 + 2^{-(n+2)}$.*

Proof: Assume $\mathcal{A}$ is some algorithm which for each instance $X = (x_1, \ldots, x_n, 0^k)$ of U_n computes a feasible solution $\pi_X = (\pi_1, \ldots, \pi_n)$. For $i \in \{1, \ldots, n\}$ we say $\mathcal{A}$ *answers x_i correct* iff for every instance $X = (x_1, \ldots, x_n, 0^k)$ of U_n, if $x_i \in A$ then $T_A(x_i, \pi_i)$ accepts (i.e. $\mathcal{A}$ constructs a proof π_i for the fact $x_i \in A$). We will show:

(a) In time $t(|x_1|) + t(|x_2|) + \ldots + t(|x_j|) \leq n \cdot |X|^{f(2^j)}$ one can answer $x_1, \ldots, x_j$ correct, and every algorithm answering $x_1, \ldots x_j$ correct has approximation ratio at most $1 + 2^{-j}$.

(b) For $j \in \{1, \ldots, n\}$ every approximation algorithm that does not answer x_j correct has approximation ratio at least $1 + \frac{1}{2^{j+2}}$.

(c) For $j = 1, \ldots, n-1$: In running time $o\left(|X|^{f(2^j)}\right)$ an algorithm cannot answer x_j correct.

(d) An algorithm with polynomial running time cannot answer x_n correct.

Obviously, from (a)-(d) the lemma follows.
Proof of (a): The time bound for answering $x_1, \ldots, x_j$ correct follows directly from the definition of U_n and the assumption about the time complexity of $W(A, T_A)$. Assume now that $\pi = (\pi_1, \ldots, \pi_n)$ is a feasible solution for instance X such that $x_1, \ldots, x_j$ are answered correct. Let

$$C(j) \quad := \quad 2^n + \sum_{i=1}^{j} [T_A(x_i, \pi_i)] \cdot 2^{n-i} \in \left[2^n, 2^n + 2^{(n-j)} \cdot (2^{j+1} - 1)\right] \tag{4.15}$$

be the contribution of $\pi_1, \ldots, \pi_j$ to the solution cost. Then

$$\begin{aligned} R_{U_n}(X, \pi) &= \frac{\mathrm{opt}_{U_n}(X)}{c_n(X, \pi)} \leq \frac{C(j) + 2^{n-j-1} + \ldots + 2 + 1}{C(j)} && (4.16) \\ &= 1 + \frac{2^{n-j-1} + \ldots + 2 + 1}{C(j)} \leq 1 + \frac{2^{n-j} - 1}{2^n} \leq 1 + 2^{-j} && (4.17) \end{aligned}$$

Proof of (b): Assume $\mathcal{B}$ is some approximation algorithm for U_n answering x_j incorrect. We compute a lower bound on the approximation ratio of $\mathcal{B}$: Let X be an instance such that x_j is answered incorrect, i.e. $x_j \in A$ and $T_A(x_j, \pi_j) = 0$ for $\mathcal{B}(X) = \pi = (\pi_1, \ldots, \pi_n)$. Let $\pi^* = (\pi_1^*, \ldots, \pi_n^*)$ be an optimum solution to instance X of U_n of cost $c_n(X, \pi^*) = 2^n + 2^{n-j} + R^*$. Let $R := \sum_{i \neq j} T_A(x_i, \pi_i) \cdot 2^{n-i}$. Obviously $c_n(X, \pi) = 2^n + R$ and $R \leq R^* \leq 2^{n+1} - 2^{n-j} - 1$. Then

$$\begin{aligned} R_{U_n}(X, \mathcal{B}(X)) &= \frac{2^n + 2^{n-j} + R^*}{2^n + R} \geq 1 + \frac{2^{n-j}}{2^n + R} \\ &\geq 1 + \frac{2^{n-j}}{2^n + 2^{n+1} - 2^{n-j} - 1} \geq 1 + \frac{1}{2^{j+2}} && (4.18) \end{aligned}$$

Proof of (c): Assume there is some algorithm $\mathcal{A}$ with running time $o\left(|X|^{f(2^j)}\right)$ such that for each input $X = (x_1, \ldots, x_n, 0^k)$ of U_n $\mathcal{A}$ computes a feasible solution $(\pi_1, \ldots, \pi_n)$ where x_j is answered correct. Using the padding property of U_n, for each string $x \in \Sigma^*$ we can construct in polynomial time an instance X of U_n such that

$x = x_j$ and $|X| \in \left[\sqrt[f(2^j)]{t(|x_j|)},\ \sqrt[f(2^j)]{t(|x_j|)} + 1 \right]$. Then using algorithm $\mathcal{A}$ we would solve the problem $W(A, T_A)$ in time

$$o\left(|X|^{f(2^j)}\right) = o\left(f\left(2^j\right) \cdot t(|x|)\right) = o(t(|x|)),$$

a contradiction.
Proof of (d): Such an algorithm could be used to solve $W(A, T)$ in polynomial time (note that $|x_n| = \Theta(|X|)$ is possible), in contradiction to the assumption. This completes the proof of the lemma. □

Proof of Theorem 12: Again we will construct U_f in stages, using a monotone increasing sequence of strings $x_0, x_1, \ldots, x_n, \ldots$. U_f restricted to the n-th interval $I_n := [x_{n-1}, x_n)$ will be equal to U_n. Let $(T_i, c_i, \alpha_i), i \in \mathbb{N}$ be some effective enumeration of triples (T_i, c_i, α_i) where T_i is a Turing machine, $c_i > 0$ some constant and $\alpha_i \in \mathbb{N}$ such that every such triple (T, c, α) occurs infinitely often. We will construct $x_1, \ldots, x_n, \ldots$ such that for every $n \in \mathbb{N}$ the following condition C_n is satisfied:

C_n: For $j = 1, \ldots, n$ both (a) and (b) hold.

(a) For $m = 1, \ldots, n-1$: If $f(2^m) \geq \alpha_j + 1/n$ then there exists some $y_{j,m} \in I_n$ such that $T_j(y_{j,m}, 2^{m+3})$ is not a $(1 + 2^{-(m+3)})$-approximate solution to instance $y_{j,m}$ of U_f in time bounded by $c_j \cdot |y_{j.m}|^{\alpha_j}$.

(b) There is some $z_j \in I_n$ such that $T_j(z_j, 2^{n+3})$ is not a $(1+2^{-(n+3)})$-approximate solution to instance z_j of U_f in time $c_j \cdot |z_j|^{\alpha_j}$.

Construction of U_f:
Stage 0**:** $x_0 := 0$.

Stage $n > 0$**:** Compute $f(2^1), \ldots, f(2^n), f(2^{n+1})$. By Brute Force find the lexicographically first strings $y_{j,m}, j = 1, \ldots, n, m \in \{1, \ldots, n-1\}$ with $f(2^m) \geq \alpha_j + 1$ and z_j with $y_{j,m}, z_j \geq x_{n-1}$ such that C_n becomes true. Such pairs exist because of properties 2 and 3 from Lemma 4.3.2. Let t_n be the time needed to compute $f(j), 1 \leq j \leq n$ and to find strings $y_{j,m}$ and z_j. Let $x_n := 0^{t_n+1}$ and $U_f|I_n := U_n$.
End of construction.

Claim: U_f is an NPO problem.
Proof: Again there is a linear time algorithm to compute for given $x \in \Sigma^*$ the number $n \in \mathbb{N}$ with $x \in I_n$, by computing the number $n-1$ in at most $|x|$ steps. Since in stage $n-1$ we compute $f(n)$, $|x| \geq t_f(n)$, and therefore using Lemma 4.3.1 we conclude that U_f is bounded by some polynomial $p(|x|)$ and hence an NPO problem.
Claim: U_f admits a polynomial time approximation scheme.
Proof: For given x and n, in order to compute a $(1 + 1/n)$-approximate solution to instance x of U_f we first compute the number $m \in \mathbb{N}$ with $x \in I_m$. If $2^{m-1} \geq n$ we compute a $(1+2^{-j})$-approximate solution for $j = \lceil \log(n) \rceil$ in time $|x|^{f(2^j)} \cdot m \leq |x|^{f(2^j)+1}$.

Otherwise we compute an optimum solution y^* to instance x by brute force. Since $2^{m-1} < n$ for only finite many m (and therefore finite many x) and $2^{\lceil log(n) \rceil} < 2n$, the running time is bounded by $O\left(|x|^{f(2n)+1)}\right)$ and hence by $O\left(|x|^{f(2n)+1}\right)$.
Claim: For every function $g\colon \mathbb{N} \to \mathbb{N}$ and every $\alpha \in \mathbb{N}$, U_f does not admit a ptas with running time bounded by $g(n) \cdot |x|^\alpha$ (for $1 + 1/n$)-approximation).
Proof: Assume U_f admits a ptas T with running time $g(n) \cdot |x|^\alpha$ for some function g and some constant α. Then for every $n \in \mathbb{N}$ there exists some $i \in \mathbb{N}$ such that $T_i = T, \alpha_i = \alpha$ and $c_i = g(n)$. But for $m \geq n + 3$, in I_m there exist $y \in I_m$ such that either $T_i\,(y, 2^m)$ is not a $(1 + 2^{-m})$-approximate solution for instance y of U_f or $T_i\,(y, 2^m)$ does not stop after at most $c_i \cdot |y|^{\alpha_i}$ steps, a contradiction ! This proves the claim and hence Theorem 12. □

4.3.3 Separation under some weaker assumption

In the previous subsection we have presented a separation of $PTAS$ from $EPTAS$, i.e. a proof that $EPTAS \neq PTAS$, under the assumption that for at least one problem in NP and a polynomially balanced relation R defining it, we have a **tight superpolynomial polytime computable lower bound** for the deterministic time complexity, namely

$$\Pi_R \in DTIME(t(n)) \setminus DTIME(o(t(n)))$$

for such a function. The natural question which now occurs is how this assumtion is related to that which was used by Cesati and Trevisan in their separation proof. In order to do this, we will compare the two assumptions using oracles. It will turn out that under some oracle X our assumption becomes true but theirs becomes false. This shows that using **relativizing techniques** one cannot prove that our assumption implies theirs.

Unfortunately we do not know how to apply oracle techniques to make assumption (A) become true. It turns out that the requirement of a **sharp threshold** for deterministic time complexity is difficult to handle.

Therefore in this subsection we will slightly relax the assumption, namely replace the sharp threshold by existence of superpolynomial upper and lower bound $T(n)$ and $t(n)$, where "a certain gap" between $t(n)$ and $T(n)$ is allowed. This relaxed version allows us to apply oracle techniques and to obtain the above result.

We show that in order to separate $PTAS$ from $EPTAS$ the following weaker assumption is sufficient:

> **Assumption (A'):** There exists a polynomially bounded binary relation $R \in P$ such that for the associated function problem Π_R the following is true:
>
> $$\Pi_R \in DTIME(T(n)) \setminus DTIME(t(n))$$
>
> for some pair $t(n), T(n)$ of super-polynomial polynomial-time computable functions with $t(n)^2 = O(T(n))$.

Theorem 13 *Under assumption (A') there exists an optimization problem U such that*

$$U \in PTAS \setminus EPTAS.$$

Proof: The proof follows the same line as that of our first separation (cf. the previous subsection). Let $T(n) := t(n)^2$, then $T(n)$ is polynomial-time computable as well. Furthermore, according to assumption (A') let R be some $p(n)$-balanced binary relation such that

$$\Pi_R \in DTIME(T(n)) \setminus DTIME(t(n)).$$

Now problems U_n are defined as follows:

Definition of U_n:
Instance: $X = (x_1, \ldots, x_n, 0^k)$ such that $T(|x_j|) \leq |X|^{2^j}$, $j = 1, \ldots, n-1$

Solutions: $\pi = (\pi_1, \ldots, \pi_n)$ such that $|\pi_i| \leq p(|x_i|), 1 \leq i \leq n$

Costs: $c_n(X, \pi) = 2^n + \sum_{i=1}^{n} R(x_i, \pi_i) \cdot 2^{n-i}$

As in the previous section we use the terms *answering $x_1, \ldots, x_j$ correct* and *answering x_j incorrect*. Problem U_n has the following properties the proof of which is completely analoguous to that of lemma 4.3.1 in subsection 4.3:

(1) Each approximation algorithm $\mathcal{A}$ for U_n answering $x_1, \ldots, x_j$ correct has an approximation ratio

$$A.R._{U_n}(X, \mathcal{A}(X)) \leq 1 + 2^{-j}.$$

(2) Each approximation algorithm $\mathcal{A}$ for U_n answering x_j incorrect has an approximation ratio

$$A.R._{U_n}(X, \mathcal{A}(X)) \geq 1 + 2^{-(j+2)}.$$

(3) There is an approximation algorithm for U_n answering $x_1, \ldots, x_j$ correct with running time

$$O\left(\sum_{i=1}^{j} T(|x_i|)\right) = O\left(\sum_{i=1}^{j} |X|^{2^i}\right) = O\left(|X|^{2^j}\right).$$

Lemma 4.3.3 *Under assumption (A') the following holds:*

(a) In polynomial time an approximation algorithm for U_n cannot answer x_n correct.

(b) In time $O\left(|X|^{2^{j-1}}\right)$ an approximation algorithm for U_n cannot answer x_j correct.

Proof: (a) is clear from the definition of U_n.
(b): Similar to the proof of ... in section ... we give a reduction from Π_R to U_n: Given

some instance x of Π_R we construct an instance $X = F(x)$ of U_n with $x_j = x$, hence $T(|x_j|) \leq |X|^{2^j}$. Using the padding one easily constructs such X with

$$|X| \in \left[\sqrt[2^j]{T(|x|)}\,,\ \sqrt[2^j]{T(|x|)} + 1 \right].$$

Assume $\mathcal{A}$ is some approximation algorithm for U_n answering x_j correct such that $time_{\mathcal{A}}(|X|) = O(h(|X|))$. Then due to assumption (A') it must hold

$$h(|X|) = \omega(t(|x_j|)).$$

Since we can assume h to be at most exponential, we conclude $h(m+1) = O(h(m))$, hence

$$h\left(\sqrt[2^{j-1}]{T(|x_j|)} \right) = \omega(t(|x_j|))$$

hence $h(m) = \omega\left(m^{2^{j-1}}\right)$. □

Proof of Theorem 13: Let (T_i, c_i, α_i) be an enumeration of triples consisting of Turing machine T_i, constant $c_i > 0$ and exponent $\alpha_i \in \mathbb{N}$ such that each triple occurs infinitely often. We construct U in stages, in stage n of the construction strings x_{n-1}, x_n are defined and U restricted to $[x_{n-1}, x_n)$ will be defined as U_n. We will satisfy the following requirements $C_n, n \in \mathbb{N}$:

(C_n) : For $j = 1, \ldots, n$, for machine T_j both (a) and (b) hold:

(a) For $m = 1, \ldots, n-1$: If $2^m \geq \alpha_j$ then there exists a string

$$y = y_{j,m} \in I_n := [x_{n-1}, x_n)$$

such that $T_j(y, 2^{j+3})$ is not a $(1+2^{-(m+3)})$-approximate solution to instance y of U in time $c_j \cdot |y_{j,m}|^{\alpha_j}$.

(b) There exists $x = x_j \in I_n$ such that

$$time_{T_j}(x, 2^{n+3}) > c_j \cdot |x|^{\alpha_j}$$

or

$$A.R.(x, T_j(x, 2^{n+3})) > 1 + 2^{-(n+3)}.$$

By choosing the intervals I_n sufficiently large we guarantee that $U \in NPO$ and hence $U \in PTAS$, cf. the proof of theorem 12 in the previous section. Then from $(C_n), n \in \mathbb{N}$ the theorem follows:

Assume T is an efficient PTAS for U with running time bounded by $f(n) \cdot |x|^{\alpha}$ for some $\alpha > 0$ and $f\colon \mathbb{N} \to \mathbb{N}$ being recursively approximable from below. Then for every $n \in \mathbb{N}$ the tuple $(T, f(n), \alpha)$ occurs infinitely often in the list. But for $m > n+3$ there exists $y \in I_m$ such that at least one of (i) and (ii) holds:

(i) $T(y, 2^m)$ doest not terminate within $f(n) \cdot |y|^{\alpha}$ steps.

(ii) $A.R.(y, T(y, 2^m)) > 1 + 2^{-m}$,

a contradiction.

We construct problem U in stages:

Stage 0**:** $x_0 := 0$.

Stage $n > 0$**:**

Find strings $x_j, y_{j,m}$, $j = 1, \ldots, n$, $m \in \{1, \ldots, n-1\}$ such that
(c_n) becomes true. These strings exist due to Lemma 4.3.3.
Choose x_n sufficiently large, namely
$x_n := 0^T$
where T is the time needed for the construction of stages $1, \ldots, n-1$
and finding $y_{j,m}, x_j$ in stage n by Brute Force.

Due to the choice of x_n in stage n, for given string x we can in linear time compute the interval number n such that $x \in I_n = [x_{n-1}, x_n)$. Hence U is in NPO and hence in $PTAS$. This proves the theorem. □

4.4 Oracle Constructions

In the previous sections we gave constructions of optimization problems yielding the separation of $EPTAS$ from $PTAS$ under certain complexity theoretic assumptions (A) and (A'). As mentioned before, Cesati and Trevisan [CT97b] obtained the same separation result under a different assumption (B) and using a different construction method. In this section we ask how different the assumptions are. They have in common the property of implying $P \neq NP$ (in our words: they are **stronger** than assumption $P \neq NP$, or to be precise: *at least as strong as*).

Our aim here is to show that at least using certain standard proof techniques one can not show that our assumption (A') implies (B), namely those proof techniques which **relativize**, i.e. still hold under **oracles**.

4.4.1 Bounded Nondeterminism

Dealing with problems in NP, two computational resources are of special interest: **deterministic running time** and the **amount of nondeterminism** needed to solve the problem in polynomial time. Here amount of nondeterminism simply means number of nondeterministic computation steps an algorithm solving the problem makes. Trivial observations are that problems in NP can be solved in exponential time $2^{n^{O(1)}}$ and nondeterministically in polynomial time with polynomial number of nondeterministic steps (**guessing steps**). The precise interaction of the two resources time and nondeterminism is far from being well-understood.

Interestingly there are natural problems in NP neither known to be in P nor NP-complete for which a bounded amount of nondeterminism is sufficient in order to solve

the problem. One of the most interesting examles was considered by Papadimitriou and Yannakakis, namely Computing the Vapnik-Chervonenkis dimension of a $0-1$ matrix. Below we will give a very brief introduction into the problem and their results.

In this and the following subsections we give a survey on previous results on bounded nondeterminism as well as an introduction into the basic notions and definitions. Let us start by defining subclasses of NP bounding nondeterminism.

Definition 31 **(a)** *Let $f(n)$ and $t(n)$ be polynomially bounded functions with $f(n) \leq t(n)$ for almost all n. Then*

$$NTIME[f(n)](t(n)) := \{L \in NP | \textit{there exists a } t(n) \textit{ time bounded NTM } \mathcal{M} \textit{ making at most } O(f(n)) \textit{ nondet. computation steps s.t. } L(\mathcal{M}) = L\}$$

(b) *Let $f(n)$ be a polynomially bounded function. Then*

$$NP[f(n)] := \{L \in NP | \textit{there exists a polytime bounded NTM } \mathcal{M} \textit{ making at most } O(f(n)) \textit{ nondeterministic computation steps such that } L(\mathcal{M}) = L\}$$

The following observations are straight forward.

Lemma 4.4.1

$$\begin{array}{lcl} NP & = & NP\left[n^{O(1)}\right], \\ P & = & NP[\log(n)] \quad = \quad NP[1], \\ NP[f(n)] & \subseteq & DTIME\left(2^{O(f(n))}\right). \end{array}$$

Of special interest are the classes defined by polylogarithmically bounded nondeterminism, known as the Kintala-Fischer hierarchy. We will consider them in the next subsection.

4.4.2 The Kintala-Fischer Hierarchy

In 1977 Kintala and Fischer [KF84] defined a hierarchy of subclasses of NP according to increasing amount of nondeterminism:

Definition 32 (The Kintala- Fischer β-Hierarchy [KF84])
The β-Hierarchy is defined as follows:

$$\beta_k := NP[\log^k(n)] = \{L \in NP | L = L(\mathcal{M}) \textit{ for some polytime bounded Oracle NTM that makes only } O(\log^k(n)) \textit{ nondeterministic computation steps}$$

$$\beta := \bigcup_{k \in \mathbb{N}_0} \beta^k$$

Besides others, Papadimitriou and Yannakais [PY93] studied the complexity of natural problems which require only subpolynomial nondeterminism, with special emphasis on complexity of computing the VC Dimension.

4.4.3 Complexity of the VC Dimension

Let U be a set, the so called universe, let $\mathcal{C}$ be a subset of the power set $P(U)$ of U, i.e. a family of subsets of U. The **Vapnik- Chervonenkis dimension** (**VC dimension**) of $\mathcal{C}$, denoted as $d(\mathcal{C})$, is the largest cardinality of a subset S of U such that the following holds: For every subset $T \subseteq S$ there exists a set $C_T \in \mathcal{C}$ such that $S \cap C_T = T$. The V-C dimension can be considered as a measure of the variability or expressive power of $\mathcal{C}$. The most important application is in Learning Theory.

Papadimitriou and Yannakakis [PY93] investigate the computational complexity of computing the VC dimension. They consider the following algorithmic decision problem:

VC DIMENSION

Instance: Finite set U, a family of subsets $\mathcal{C} \subseteq P(U)$, integer $k \geq 0$

Question: Is $d(\mathcal{C}) \geq k$?

Obviously, **VC DIMENSION** is in NP:

Just guess a set $S \subseteq U$ of size k and check by Brute-Force whether

$$\{\mathbf{C} \cap \mathbf{S} | \mathbf{C} \in \mathcal{C}\} = \mathbf{P}(\mathbf{S}).$$

Note that due to the fact that $\mathcal{C}$ is given in explicit form (e.g. as a list of incidence vectors of the sets $C \in \mathcal{C}$, i.e. a $|\mathcal{C}| \times |U|$ $0-1$ matrix $M = M(U, \mathcal{C})$, for $k > \log(|\mathcal{C}|)$ the answer will be **no** and hence the above is indeed a polytime nondeterministic algorithm. This also implies that the problem can be solved in time $O\left(n^{\log(n)}\right)$.

Papadimitriou and Yannakakis [PY93] defined the class **LOGSNP** in analogue to the well known Fagin's class **SNP**. They proved $VC\ DIMENSION$ being complete for **LOGSNP** under polynomial-time reductions. Furthermore they give the same result for several other problems in NP for which bounded nondeterminism suffices, see their paper for the details and proofs.

4.4.4 Downward separation fails: The Beigel - Goldsmith construction

Concerning hierarchies of classes of sets in general, one may ask what the relation of single levels of the hierarchy to each other implies for the whole hierarchy. The **Polynomial Hierarchy PH** and the **Boolean Hierarchy BH** both have the property of **Downward Separation**. Recall the definition of **PH** and **BH**: The Polynomial

Hierarchy is defined by

$$\begin{array}{lcl} PH & := & \bigcup_{k\in\mathbb{N}_0} \Sigma_k^P, \qquad \text{where} \\ \Sigma_0^P & = & \Pi_0^P = \Delta_0^P = P, \\ \Sigma_{k+1}^P & := & NP^{\Sigma_k^P} \\ \Pi_{k+1}^P & := & co-NP^{\Sigma_k^P} = co-\Sigma_{k+1}^P = \{L|\overline{L} \in \Sigma_{k+1}^P \\ \Delta_{k+1}^P & := & P^{\Sigma_k^P} = P^{\Pi_k^P}. \end{array}$$

The Boolean Hierarchy **BH** is the closure of NP under Boolean operations (intersection, complement, union). The levels of **BH** corespond to the depth of the formula defining the set:

$$\begin{array}{lcl} BH & := & \bigcup_{k\in\mathbb{N}} BH(k), \qquad \text{where} \\ BH(1) & := & NP \\ BH(k+1) & := & \{L_1 \setminus L_1 \mid L_1 \in NP,\ L_2 \in BH(k)\} \\ co-BH(k) & := & \{L|\ \overline{L} \in BH(k)\}. \end{array}$$

The following lemma is well-known, the first part can be found in standard text-books about complexity theory.

Lemma 4.4.2 (Downward Separation of PH and BH)

(a) If for some $k \geq 1$, $\Sigma_k^P = \Pi_k^P$, then $\Sigma_{k+j}^P = \Pi_{k+j}^P = \Sigma_k^P$ for all $j \geq 0$.

(b) If for some $k \geq 1$, $BH(k) = co-BH(k)$ then $BH = BH(k)$.

For further information see e.g. [Kad88] and [CGH$^+$88].

The question that arises in our current context is whether such phenomenon as downward separation aslo exists for hierarchies of bounded nondeterminism. For the **Kintala-Fischer Hierarchy**, Beigel and Goldsmith [BG94] gave a partial negative answer. They were able to prove that using **relativizable proof techniques**, namely those that also hold in every oracle setting, one cannot obtain downward separation for $\beta = \bigcup \beta_{\mathbf{k}}$.

Before citing (a rough version of) the main result of Beigel and Goldsmith, let us first indicate the difficulty in showing downward separation. A very natural approach one might have in mind is **padding**. Assume $\beta_n \neq \beta_{n+1}$ for some $n \in \mathbb{N}$, and we now ask for implication of this equality to lower levels of the Kintala-Fischer hierarchy. Let us assume we try to prove $\beta_m \neq \beta_{m+1}$, for some $m < n$. Padding here would mean the following: We take a language $L \in \beta_{n+1} \setminus \beta_n$ and use it to construct a language $L' \in \beta_{m+1} \setminus \beta_m$. Let $L = L_R$ for some $c \cdot \log^{n+1}$-balanced binary relation in P. We set

$$R' := \{(x10^k, y) \mid (x, y) \in R\}$$

and choose k sufficiently large such that proof length is reduced from $O(\log^{n+1}(x))$ to $O(\log^{m+1}(x))$ and such that $x \mapsto x10^k$ is a polynomial reduction from L to $L' = L_{R'}$.

But that way we do not succeed: in order to obtain sufficient decrease of proof length, we must choose

$$k = 2^{(\log(x))^{(n+1)/(m+1)}} = n^{\log^{\delta}(x)}$$

for some $\delta > 0$, i.e. we get a superpolynomial construction and therefore not a polynomial time reduction.

Let us now formulate the main result of Beigel and Goldsmith, explaining why the above approach was not successful.

Theorem 14 (Main Result of Beigel and Goldsmith [BG94])
For any consistent set of collapses and separations of levels of the Kintala-Fischer Hierarchy that respects $P = \beta_1 \subseteq \beta_2 \subseteq \ldots \subseteq NP$, there exists an oracle X such that relative to X those collapses and separations hold.

For a precise formulation of the result and proof techniques we refer to [BG94].

Let us mention that in subsection 4.4.6 we take a much closer look at the bottom of the β-Hierarchy. There, we construct an oracle X such that

$$NP^X[s(n) \cdot \log(n)] \neq P^X$$

for every unbounded polytime computable function $s(n)$. The difficulty arising in such a construction is that although we can give a list of polytime bounded TMs, but by no means we know in advance which of these machines computes a bounded function and which not.

Future Work: Obtain the following further generalisation: An oracle X such that

$$s(n) = o(S(n)) \;\Rightarrow\; GC^X(s(n)\log(n)) \neq GC^X(S(n)\log(n))$$

for all pairs of polytime comutable unbounded functions. The construction we have in mind implies that already

$$\text{Not } s(n) = \Omega(S(n)) \;\Rightarrow\; GC^X(s(n)\log(n)) \neq GC^X(S(n)\log(n))$$

4.4.5 Guess and Check: The W-Hierarchy versus Bounded Nondeterminism

In [CT97b] $PTAS \neq EPTAS$ was proved under assumption $FPT \neq W[P]$. In [CCDF95] this was proved being equivalent to the assumption that classes of decision problems with superlogarithmic proof length are not contained in P. In this subsection we will give a precise formulation of this characterization as well as definitions of the terms and notations being used.

The term *guess and check* was introduced by Cai and Chen [CCDF95] in order to define classes of bounded proof length:

Definition 33 (Guess and Check Classes)
Let $f: \mathbb{N} \to \mathbb{N}$ be some function. The guess and check class $GC(f(n)$ is defined by

$$GC(f(n)) := \{L \subseteq \Sigma^* | \text{there exists a } O(f(n))\text{-bounded relation } R \subseteq \Sigma^* \times \Sigma^* \text{ such that } R \in P \text{ and } L_R = L\}$$

Obviously the following holds:

$$GC(\log(n)) = P, \quad GC\left(n^{O(1)}\right) = NP.$$

Cai and Chen proved that $FPT \neq W[P]$ is equivalent to the assumption that increasing proof length slightly beyond logarightmic already leads us out of class P:

Theorem 15 (Characterization of assumption $FPT \neq W[P]$ [CCDF95])
The following are eqivalent:

(1) $FPT \neq W[P]$

(2) **Assumption (B):** *For every unbounded polynomial-time computable function $s: \mathbb{N} \to \mathbb{N}$,*

$$GC(s(n)\log(n)) \not\subseteq P.$$

We will now consider relativized versions of (B). Let us begin by defining appropriate oracle classes.

Definition 34 *Let X be some subset of Σ^* and $f: \mathbb{N} \to \mathbb{N}$ some function. Then*

$$GC^X(f(n)) := \{L \subseteq \Sigma^* | \text{there exists an } O(f(n))\text{-balanced binary relation } R \subseteq \Sigma^* \times \Sigma^* \text{ such that } L = L_R \text{ and } R \in P^X\}$$

4.4.6 An Oracle relative to which Assumption (B) is true

It is easy to construct an oracle making assumption (B) false, namely by taking a sufficiently powerful oracle.

Theorem 16 (An Oracle relative to which Assumption (B) is false)
For every polynomial-time computable unbounded function $s(n)$ there exists some recursive set $X \subseteq \Sigma^*$ such that

$$GC^X(s(n)\log(n)) \subseteq P^X.$$

Proof: Let X be some $DTIME\left(n^{O(s(n))}\right)$-complete set, then the inclusion obviously holds. For a precise argument (yielding a slightly more general result) see the proof of Lemma 4.4.3 below. □

In order to construct an **oracle making (B) become true**, one needs to diagonalize against polynomially bounded oracle machines accepting inputs (x, y) with $|y|$ bounded

by $s(|x|)\log(|x|)$ for some nonconstant polytime computable function $s(n)$. We can enumerate polynomially bounded TMs M_i computing functions $s_i(n)$, but by no means we know in advance which of these functions is bounded. The only thing we can do is guessing $s_i(n)$ being bounded by some constant C, and if this guess was incorrect, then we will observe the incorrectness after finite amount of time, namely find some n such that $s_i(n) > C$. Our strategy will be to perform such a guessing process, and each time a violation occurs, we will increase the constant C and perform one diagonalization step against $GC^X(s_i(n)\log(n))$ being equal to P^X.

Let us now formulate the result and then give a precise proof.

Theorem 17 (An Oracle relative to which Assumption (B) is true)
There existst some recursive set $X \subseteq \Sigma^*$ such that for every unbounded polynomial-time computable function $s(n)$

$$GC^X(s(n)\log(n)) \not\subseteq P^X.$$

Proof: It is sufficient to consider monotone increasing functions, hence we take a list $\mathcal{L}$ of all polytime bounded TMs M_i, then

$$\{s_i := h_{M_i} | M_i \in \mathcal{L}\} = \{s(n) | s(n) \text{ is polytime computable monotone function}\},$$

where we take

$$h_{M_i}(n) := \max\{M_i(0), \ldots, M_i(n)\}.$$

But then we do not know in advance which of these functions is bounded and which not. Therefore we will alternatively work with guesses of the form

$$(M_i, c) \qquad \text{standing for "}\forall\, n\ \ s_i(n) \leq c\text{".}$$

If a guess is incorrect, we will recognize after finite amount of time and then do the following:

(1) The guess will be updated from (M_i, c) to $(M_i, s_i(n))$, where n is the actual value where we observe $s_i(n) > c$.

(2) We perform one step in the construction of oracle X towards guaranteeing

$$GC^X(s_i(n)\log(n)) \not\subseteq P^X,$$

namely towards

$$L_i := \{x = 0^n | \exists y \text{ such that } |y| = s_i(|x|)\log(|x|) \text{ and } y \in X\} \not\in P^X.$$

Here *performing one step towards $L_i \not\in P^X$* means the following: The oracle X will be constructed in stages. In stage n we define X restricted to strings of length $l(n)$. For each i we maintain a number $j(i)$ of the index of the polytime TM $N_{j(i)}$ to be taken care of next. The guesses (M_i, c) are maintained in a priority queue $\mathcal{Q}$. Whenever in some stage n of the construction the guess (M_i, c) is the first being violated, we define

$X \cap \Sigma^{l(n)}$ such that machine $N_{j(i)}$ with oracle X does not recognize L_i. This is made possible by an appropriate choice of guesses, namely such that if the guess is violated then N_i on input of length $l(n)$ might ask at most a polynomial in $l(n)$ number of questions to X while there are more strings of length $l(n)$ available.

Let $p_{j(i)}(n)$ denote a polynomial bound on the running time of polytime machine $N_{j(i)}$, let $d_{j(i)}$ and $a(j(i))$ denote the degree and maximum coefficient of polynomial $p_{j(i)}(n)$, hence $p_{j(i)}(n) \leq a(j(i)) \cdot (d_{j(i)} + 1) \cdot n^{d_{j(i)}} =: C_{j(i)} \cdot n^{d_{j(i)}}$ for all $n \in \mathbb{N}$.

In order to guarantee that if we identify a guess (M_i, c_i) being incorrect we are able to diagonalize, we need to assure that

$$C_{j(i)} \cdot l(n)^{d_{j(i)}} \; < \; 2^{s_i(l(n)) \cdot \log(l(n))} \; = \; l(n)^{s_i(l(n))}$$

hence machine M_i will enter the queue $\mathcal{Q}$ with guess $(M_i, c_i := d_1 + 1)$, and we will choose

$$l(n+1) \; := \; \max\{l(n)+1, \max\{C_{j(i)} | \text{ there is some guess } (M_i, c_i) \text{ in } \mathcal{Q}\}\}.$$

(This implies $p_{j(i)}(l(n)) \leq C_{j(i)} \cdot l(n)^{d_{j(i)}} < 2^{c_i \cdot \log(l(n))}$ for all i, n, and hence there are more strings of length $l(n)$ than machine $N_{j(i)}$ might ask on input $0^{l(n)}$).

We are now ready to describe the construction of X in detail.

Construction of oracle X in stages

Stage 0**:**
$\mathcal{Q} := \{(M_1, d_1)\}$ (initialization of the priority queue of guesses)
$j(1) := 1$
$l(1) := C_1$

Stage $n > 0$**:**
Set $X \cap \Sigma^l \; := \; \emptyset$ for all $l(n-1) < l < l(n)$.
If for each entry (M_i, c_i) of $\mathcal{Q}$ $s_i(l(n)) \leq c_i$
then $X \cap \Sigma^{l(n)} := \emptyset$ **else**
Remove the first violation (M_i, c) from $\mathcal{Q}$.
/ $X \cap \Sigma^{<l(n)}$ is already defined */*
Let oracle Y be defined as X on strings of length $\leq l(n)$ and $Y \cap \Sigma^{\geq l(n)} := \emptyset$.
If $0^{l(n)} \in L(N_{j(i)}, Y)$ **then** let $X \cap \Sigma^{l(n)} := \emptyset$
otherwise
Let $x \in \Sigma^{l(n)}$ not being asked by the computation $N_{j(i)}(0^{l(n)}, Y)$
/ Such string exists due to the choice of $l(n)$! */*
Let $X \cap \Sigma^{l(n)} := \{x\}$.
endif
/ Update the violated guess and the index $j(i)$ */*
Replace (M_i, c) by $(M_i, s_i(l(n)) + 1)$. Insert (M_i, c) into $\mathcal{Q}$.
$j(i) := j(i) + 1$.
endif

Expand queue of guesses:
Insert (M_n, n) into $\mathcal{Q}$ and set $j(n) := 1$.
Compute $l(n+1)$**:**
$l(n+1) := \max\{\, l(n)+1,\ \max\{C_{j(i)} | \text{some guess } (M_i, c_i) \text{ is in } \mathcal{Q}\}\,\}$
End of Stage n

It follows directly from the construction that

$$GC^X(s_i(n)\log(n)) \not\subseteq P^X$$

for all $i \in \mathbb{N}$, furthermore X is recursive. □

4.4.7 An Oracle relative to which (A') is true but (B) is false

In this section we separate assumptions (A') and (B) by oracle construction, showing that using relativizing techniques one can not prove that our assumption (A') implies assumtion (B). Let us first formulate our result, then give an outline of our proof as well as some difficulties that may arise, finally we give the precise proof.

Theorem 18 (An Oracle relative to which (A') is true but (B) is false) *There exists a recursive set $X \subseteq \Sigma^*$ such that (1) and (2) hold.*

(1) $GC^X(\log^2(x)) \subseteq P^X$ *(hence relative to oracle X assumption (B) is false).*

(2) For the relation $R_X := \{(x,y) | x = 0^n, n \in \mathbb{N},\ |y| = 2n,\ y \in X\}$,

$$\Pi_{R_X} \in FTIME^X\left(2^{n^3}\right) \setminus FTIME^X\left(2^n\right).$$

(Hence relative to oracle X assumption (A') is true.)

Proof Idea: Let $\tilde{X}$ be some $DTIME\left(n^{O(\log n)}\right)$-complete problem with respect to polynomial-time Karp reductions. Then $X' := \{1xx | x \in \tilde{X}\}$ is $DTIME\left(n^{O(\log n)}\right)$-complete as well. We let

$$X \quad = \quad X' \cup X''$$

where X'' consists of strings of even length only. We use X'' for a diagonalization process in order to assure (2).

The following two lemmas indicate how powerful the set may be in order to make (1) become true.

Lemma 4.4.3 *For every $DTIME\left(n^{O(\log n)}\right)$-complete problem Y,*

$$GC^Y(\log^2(n)) \subseteq P^Y.$$

Proof of Lemma 4.4.3: Let R be $c \cdot \log^2(n)$ - balanced for some $c > 0$ with $R \in P^Y$. Let $R = L(M, Y)$ for some polytime bounded oracle machine M. We have to show that $\Pi_R \in FTIME^Y\left(n^{O(1)}\right)$. For input x of length n there are less than $n \cdot 2^{c \cdot \log^2(n)} = n \cdot n^{c \cdot \log(n)} = n^{O(\log(n))}$ strings of length bounded by $c \cdot \log^2(n)$, hence a straight forward algorithm will enumerate all such strings and for each of them ask the question to oracle machine M with oracle Y. The running time of this oracle algorithm is bounded by

$$\underbrace{p(n + c \cdot \log^2(n))}_{\text{running time of } M} \cdot \underbrace{n^{O(\log(n))}}_{\sharp\text{strings of that length}} = n^{O(\log(n))},$$

If we replace the oracle questions by calls of a $n^{O(\log(n))}$-time bounded algorithm M_Y for Y, the total running time can be bounded by

$$n^{O(\log(n))} \cdot \underbrace{q(n)^{O(\log(q(n)))}}_{\text{time for one call of } M_Y} = n^{O(\log(n))}$$

where $q(n) := p(n + c \cdot \log^2(n))$ is a polynomial. Hence we can reduce this problem to Y (due to the $DTIME(n^{O(\log(n))})$- completeness of Y) and therefore $L \in P^Y$.□(Lemma 4.4.3)

Lemma 4.4.4 *For every $DTIME\left(n^{O(\log n)}\right)$-hard problem Y which is complete for a class $\mathcal{C}$ of languages such that $DTIME(t(n)) \subseteq \mathcal{C}$ implies $DTIME\left(n^{O(\log(n))} \cdot t\left(n^{O(1)}\right)\right) \subseteq \mathcal{C}$,*

$$GC^Y(\log^2(n)) \subseteq P^Y.$$

Proof of Lemma 4.4.4: Let $L \in GC^Y(\log^2(n))$ and $Y \in DTIME(t(n))$. We perform the complete-enumeration oracle algorithm as in the proof of claim 1 and obtain a running time bounded by

$$\underbrace{n^{O(\log(n))}}_{\sharp \text{ strings to be enum.}} \cdot \underbrace{p(n + c \cdot \log^2(n))}_{\text{single call of the Oracle TM for } L} \cdot \underbrace{t(p(n + c \cdot \log^2(n)))}_{\text{solving one instance of } Y} = n^{O(\log(n))} \cdot t(n^{O(1)})$$

Hence $L \in \mathcal{C}$, and L is polynomial-time reducible to Y, therefore $L \in P^Y$. □(Lemma 4.4.4)

Now if we take an arbitrary set $X'' \subseteq \bigcup_{n \in \mathbb{N}} \Sigma^{2n}$ which is complete for class $\mathcal{C}$, the disjoint union $X := X' \cup X''$ still satisfies the conditions in claim 2, hence (1) will still hold.

Proof of Theorem 18: Let $X = X' \cup X''$ where X' consists of odd-length strings only and X'' consists of even-length strings only. We choose

$$C^X := \{(i, x, 0^s) | M_i \text{ on input } x \text{ with oracle } X \text{ accepts within } s \text{ steps and } O(\log^2(n)) \text{ nondeterministic steps}\}.$$

Then C^X is $\leq_m^p$-complete for $GC^X(\log^2(n))$. Let $p(n)$ be a polynomial such that

$$C^X \in NTIME[\log^2(n)](p(n)).$$

We will encode C^X into X' in a polynomial manner and use X'' for diagonalisation in order to assure (2). The encoding will be as follows: For all strings $x \in \Sigma^*$,

$$x \in C^X \iff 1^{p(|x|)^2}\, 0\, x \in X.$$

We are now ready to construct X. Let $M_1, \ldots, M_n, \ldots$ denote an enumeration of $O(2^n)$-time bounded Oracle machines such that without loss of generality M_i is $i \cdot 2^n$ - time bounded.

Construction of X in stages

/⋆ At the end of stage s X is defined up to ⋆/
/⋆ strings of length n_s. During stage s diagonalization ⋆/
/⋆ against $L(M_i) = L_X$ is performed by choosing ⋆/
/⋆ $n > n_{s-1}$ and assuring $0^n \in L(M-i)\Delta L_X$ ⋆/

Stage 0**:** $n_0 := 0, \quad X := \emptyset$

Stage $s > 0$**:**
/⋆ X is defined up to length n_{s-1} ⋆/
Choose $n > n_s$ to be a power of 2 and sufficiently large.
Define X up to length $l := n^3 - 1$.
Define C^X up to strings y with $p(|y|)^2 + 1 + |y| \leq n^3 - 1$.

/⋆ $L_X(1^n)$ depends on strings of length n^3. ⋆/

Compute $M_s(1^n, X)$.

If $M_s(1^n, X)$ accepts **then**
Let y be a **legal** string of length n^3.
$X := X \cup \{y\}$.

$n_s := \max\{l, s \cdot 2^n\}$
Freeze X up to length n_s.
Freeze C^X up to strings of length m with $p(m)^2 + 1 + m \leq n_s$.
End of Stage s**.**

It remains to define which strings y of length n^3 are **legal** and what it means to choose n **sufficiently large**. The initial idea is simply to pick a string y which was not asked by the computation $M_s(x, X)$. Such string of length n^3 exists since M_i has running time bounded by $s \cdot 2^n$ and there are 2^{n^3} strings of length n^3 available (we choose n such that $s \cdot 2^n < 2^{n^3}$). The problem is that the encoding of C^X into X may depend

on this choice, and the result of $M_s(x, X)$ may in turn depend on this coding and so on.

We accomplish this difficulty by using an approach from Beigel and Goldberg: We show that due to the way the coding of C^X into X is arranged, there are more strings of length n^3 available than are influencing the coding of C^X into X in the range up to length $s \cdot 2^n$:

$M_s(x, X)$ has running time bounded by $s \cdot 2^n$ and therefore may ask at most $s \cdot 2^n$ oracle questions about strings of length bounded by $s \cdot 2^n$.

Due to the encoding, these strings may encode strings from C^X of length bounded by

$$\sqrt{s \cdot 2^n} \;=\; \sqrt{s} \cdot 2^{n/2}.$$

Each of these encoding strings w in turn may depend on at most

$$p(|w|) \cdot 2^{\log^2(|w|)} \;=\; p\left(s^{2^{-1}} \cdot 2^{n/2^1}\right) \cdot 2^{\log^2\left(s^{2^{-1}} \cdot 2^{n/2^1}\right)}$$

strings of this smaller length and so on.

Since X is already fixed up to length $l-1$, the recursion can be cut off at strings of length $\leq l-1$. Therefore the number of such terms is bounded by

$$\log\log(s \cdot 2^n) \;-\; \log\log(l-1) \;\leq\; \log(\log s \;+ n) - \log\log(n) \;\leq c \cdot \log(n) = \Theta(\log(n))$$

for some constant $c > 0$. Hence the total number of strings being influenced by the choice of y is bounded by

$$\begin{aligned} & s \cdot 2^n \cdot \prod_{i \leq c \cdot \log(n)} p\left(s^{2^{-i}} \cdot 2^{n/2^i}\right) \;\cdot\; 2^{\log^2\left(s^{2^{-i}} \cdot 2^{n/2^i}\right)} \\ = \;& s \cdot 2^n \cdot \prod_{i \leq c \cdot \log(n)} \left(s^{2^{-i}} \cdot 2^{n/2^i}\right)^{O(1)} \;\cdot\; 2^{\log^2\left(s^{2^{-i}} \cdot 2^{n/2^i}\right)}. \end{aligned}$$

We compute

$$\log^2\left(s^{2^{-i}} \cdot 2^{n/2^i}\right) = \left(2^{-i} \cdot \log(s) \;+ \frac{n}{2^i}\right)^2 \;= O(n^2)$$

Hence the total number of strings is bounded by

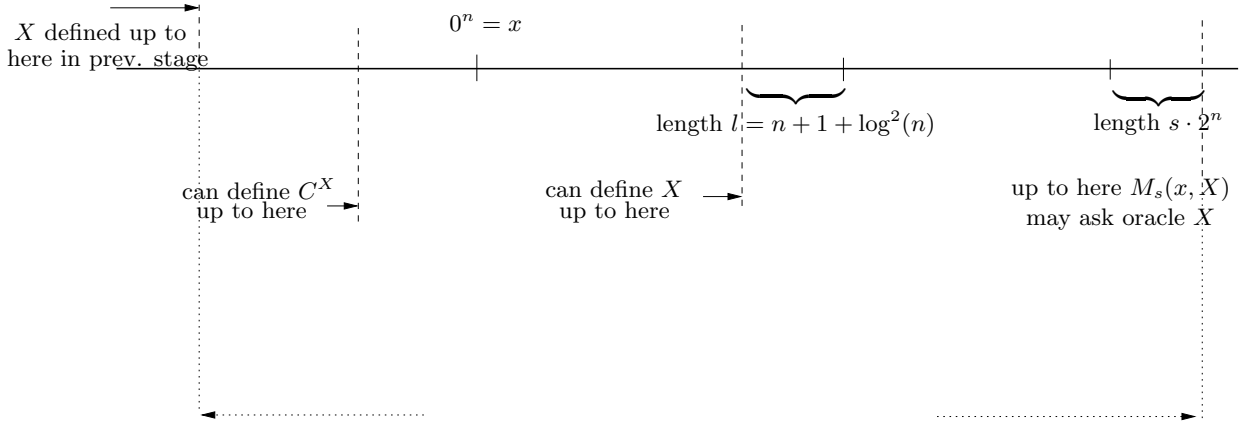
$$s \cdot 2^n \;\cdot\; O(\log(n)) \;\cdot\; 2^{O(n)} \cdot 2^{O(n^2)} \;=\; 2^{O(n^2)}$$

supposed we choose s sufficiently small compared to n, i.e. choose n large, namely

$$s = 2^{\log s},\ \log(s) = O(n^2).$$

Hence we find a string y of length n^3 such that adding y to X does not change the result of computation $M_s(x, X)$, therefore the diagonalization step is well-defined. The whole construction is shown in figure 4.2. □(Theorem 18)

Figure 4.2: **The Construction of Oracle X: Stage s**



4.5 Randomized Approximation Schemes: RPTAS versus REPTAS

In this section we consider the question of efficiency for randomized approximation schemes. Recall that a **randomized approximation scheme** $\mathcal{A}$ for an optimization problem X is a probabilistic approximation algorithm $\mathcal{A}$ for X such that

$$Pr\{\text{A.R. of } \mathcal{A}(x, \epsilon) \text{ is at most } 1 + \epsilon\} \geq \frac{3}{4}$$

and the randomized time complexity of algorithm $\mathcal{A}$ is

$$O\left(|x|^{f(1/\epsilon)}\right)$$

for some function $f: \mathbb{N} \to \mathbb{N}$. $\mathcal{A}$ is called an **efficient randomized approximation scheme** if instead the randomized running time is bounded by

$$O\left(f\left(\frac{1}{\epsilon}\right) \cdot |x|^{O(1)}\right)$$

for some function $f: \mathbb{N} \to \mathbb{N}$. Accordingly the subclasses **RPTAS** and **REPTAS** of NPO are defined.

In the following lemma, the case of **efficient probabilistic approximation schemes** where the **error probability** is **bounded for each fixed** ϵ is considered.

Lemma 4.5.1 *Assume $\mathcal{A}$ is a* **probabilistic approximation scheme** *for NPO problem X, which means*

$$Pr\{A.R. \text{ of } \mathcal{A}(x, \epsilon) \text{ is at most } 1 + \epsilon\} > \frac{1}{2}$$

Assume further that there are functions $f\colon \mathbb{N} \to \mathbb{N}$ *and* $e\colon \mathbb{N} \to [0, 1/2)$ *and* $\alpha > 0$ *such that* e *is recursive and*

$$Pr\{A.R.\ of\ \mathcal{A}(x,n)\ is\ at\ most\ 1 + \tfrac{1}{n}\ in\ time\ bounded\ by\ f(n) \cdot |x|^{\alpha}\} \ \geq\ 1 - e(n)$$

for every x *and* n. *Then* $\mathbf{X} \in \mathbf{REPTAS}$.

Proof: We can decrease the error probability to at most $1/4$ by applying the δ-Lemma for Monte Carlo machines to each n: Compute $e(n)$, then the δ-Lemma gives a computable bound $k = k(n)$ such that repeating $\mathcal{A}(\cdot, n)$ k times and making majority decision decreases error probability to $1/4$. The running time is bounded by $k(n) \cdot f(n) \cdot |x|^{\alpha}$, which proves the lemma. □

We are now going to separate the classes **REPTAS** and **RPTAS** assuming the existence of polynomially balanced relations in P for which the associated functional problems provides a superpolynomial lower bound on the randomized time complexity (assumption (A") in theorem 19 below). This is basically the variant of assumption (A') where deterministic time complexity is replaced by randomized time complexity.

Theorem 19 *Assume there exists a polynomially balanced binary relation* $R \subseteq \Sigma^* \times \Sigma^*$ *such that* $R \in P$ *and*

$$\Pi_R \ \in\ RFTIME\left(t^2(n)\right) \ \setminus\ RFTIME\left(t(n)\right)$$

for some superpolynomial polynomial-time computable function $t(n)$ **(Assumption A")**.
Then there exists an optimization problem $U_R \in NPO$ *such that*

$$U_R \ \in\ RPTAS \ \setminus\ REPTAS.$$

Proof: Assume R to be as in assumption (A"), and let R be $p(n)$-balanced for some polynomial $p(n)$. Let $T(n) := t^2(n)$. We define problem $U_{R,n}$ as follows:

Instance: $X = (x_1, \ldots, x_n, 0^k)$ such that

$$T(|x_j|) \ \leq\ |X|^{2^j},\ 1 \leq j \leq n-1.$$

Solution: $\pi = (\pi_1, \ldots, \pi_n)$ such that

$$|\pi_j| \ \leq\ p(|x_j|),\ 1 \leq j \leq n-1$$

Cost: $\mathrm{cost}_n(X, \pi) \ =\ 2^n + \sum_{j=1}^{n} R(x_j, \pi_j) \cdot 2^{n-j}$

Problems $U_{R,n}$ have the following properties:

(1) There exists a two-variate polynomial $q(n, m)$ such that $U_{R,n}$ is a $q(n, m)$**-bounded NP optimization problem** (recall that this means for given x, y in time $q(n, |x|)$ we can check whether x is a valid instance of problem $U_{R,n}$, y is a feasible solution for instance x of U_n and compute the cost function $\mathrm{cost}_n(x, y)$, c.f. section ..., definition ...).

(2) Every approximation algorithm $\mathcal{A}$ for problem $U_{R,n}$ **answering** $x_1, \ldots, x_j$ **correct** has approximation ratio

$$A.R.(\mathcal{A}) \leq 1 + 2^{-j}.$$

Every approximation algorithm $\mathcal{A}$ for problem $U_{R,n}$ **answering** x_j **incorrect** has approximation ratio

$$A.R.(\mathcal{A}) \geq 1 + 2^{-(j+2)}.$$

(3) For $j = 1, \ldots, n-1$, there exists a randomized approximation algorithm $\mathcal{A}$ with running time

$$O\left(\sum_{i=1}^{j} |X|^{2^i}\right) = O\left(j \cdot |X|^{2^j}\right)$$

such that

$$\Pr\{\mathcal{A} \text{ on input } x \text{ answers } x_1, \ldots, x_j \text{ correct}\} \geq \frac{3}{4}$$

(4) For $j = 1, \ldots, n-1$: There is no randomized approximation algorithm with running time $O\left(|X|^{2^{j-1}}\right)$ for $U_{R,n}$ answering x_j correct.

(5) There is no randomized approximation algorithm with polynomial running time for $U_{R,n}$ answering x_n correct.

The proof of (1) and (2) is identical to that of lemma 4.3.1. Let $\mathcal{A}$ be a $T(n)$-time bounded randomized algorithm for the functional problem Π_R such that

$$\Pr\{(x, \mathcal{A}(x)) \in R\} \geq \frac{3}{4}$$

for every $x \in L_R$. By the δ-Lemma for Monte-Carlo algorithms, there is a randomized algorithm $\mathcal{A}'$ such that

$$\Pr\{x \in L_R \Rightarrow (x, \mathcal{A}'(x)) \in R \text{ in time bounded by } f(j) \cdot T(|x|)\} \geq \left(\frac{3}{4}\right)^{1/j} =: a_j$$

(for some function $f(j)$ of j). Let $\mathcal{B}$ be the approximation algorithm for U_n which for each component x_j independently sets

$$\pi_j := \mathcal{A}'(x_j).$$

We obtain

$$\begin{aligned} & \Pr\{\mathcal{B} \text{ on input } X \text{ answers } x_1, \ldots, x_j \text{ correct}\} \\ = & \prod_{i=1}^{j} \Pr\{x_i \in L_R \Rightarrow (x_i, \mathcal{A}'(x_i)) \in R\} \\ \geq & (a_j)^j = \left(\frac{3}{4}\right)^{j/j} = \frac{3}{4} \end{aligned}$$

which completes the proof of (3).

In order to prove (4) we make use of the same kind of reduction from Π_R to $U_{R,n}$ as before: Given an instance x of Π_R we construct an instance X of $U_{R,n}$ such that

$$X = (x_1, \ldots, x_n, 0^k), \quad x = x_j$$

and

$$|X| \in \left[\sqrt[2^j]{T(|x|)},\ \sqrt[2^j]{T(|x|)} + 1 \right]$$

Now assume $\mathcal{B}$ is a randomized approximation algorithm for $U_{R,n}$ such that

$$\Pr\left\{\mathcal{B} \text{ answers } x_j \text{ correct in time } c \cdot \left(|X|^{2^{j-1}}\right)\right\} \geq \frac{3}{4}$$

for some $c > 0$. Since

$$c \cdot |X|^{2^{j-1}} \leq c \cdot \left(\sqrt[2^j]{T(|x|)} + 1\right)^{2^{j-1}} = O\left(2^{j-1} \cdot T(|x|)^{\frac{2^{j-1}}{2^j}}\right) = O\left(2^{j-1} \cdot t(|x|)\right)$$

this contradicts assumption (A"), hence (4) holds as well.

Let $(M_i, c_i), i \in \mathbb{N}$ be a listing of all pairs consisting of PTMs M_i and constants $c_i > 0$, where we initially guess M_i being an approximation scheme for problem U_R with error probability bounded by $1/4$. Assume that each pair occurs infinitely often in this list. For each such M_i, either the guess is incorrect and we will recognize this after finite amount of time, or we diagonalize against M_i being an efficient randomized approximation scheme for U_R with error probability bounded by $\frac{1}{4}$.

We construct problem U_R in stages. In stage n the n-th interval $I_n = [x_{n-1}, x_n)$ is defined (recall that this refers to lexicographic order on Σ^*), U_R restricted to I_n will be defined as $U_{R,n}$. During the construction the following requirements are satisfied:

(C_n) For $j = 1, \ldots, n, i = 1, \ldots, n$:
There are strings $y = y_{i,j}$ and $x = x_i$ in $I_n = [x_{n-1}, x_n)$ such that

$$Pr\left\{\text{A.R. of } M_i(y, 2^{j+3}) \text{ is} \leq 1 + \tfrac{1}{2^{j+3}} \text{ within time } c_i \cdot |y|^{2^{j-1}}\right\} < \tfrac{3}{4}$$
$$Pr\left\{\text{A.R. of } M_i(x, 2^{n+3}) \leq 1 + 2^{-(n+3)} \text{ within time } c_i \cdot |x|^{2^n}\right\} < \tfrac{3}{4}$$

Let us argue that (assuming we guarantee problem U_R being in $RPTAS$) constraints (C_n) are sufficient in order to prove the theorem: Hence assume $U_R \in REPTAS$ but (C_n) are satisfied. Then due to the fact that we can decrease error probability to $\frac{1}{4}$ increasing running time by at most a constant factor, there exists an efficient randomized polytime approximation scheme M for problem U_R with error probability bounded by $\frac{1}{4}$ and randomized running time bounded by

$$f(n) \cdot |x|^{\alpha} \text{ for some } \alpha > 0 \text{ and some function } f(n).$$

Now choose j such that $\alpha < 2^{j-1}$, and let i be choosen such that $(M_i, c_i) = (M, f(2^{j+3}))$. From some stage n on (namely $n \geq \max\{i, \log(\alpha)\}$) there exist strings y in interval I_n such that

$$Pr\left\{\text{A.R. of } M_i(y, 2^{j+3}) \text{ is } \leq 1 + \tfrac{1}{2^{j+3}} \text{ within time } f(2^{j+3}) \cdot |x|^{\alpha}\right\} < \frac{3}{4},$$

hence we obtain a contradiction and thus M is not an efficient randomized polytime approximation scheme for problem U_R.

Construction of Problem U_R in Stages.

Stage 0**:** Set $x_0 := 0$.

Stage $n > 0$**:** Let x_{n-1} and U_R restricted to the interval $[x_0, x_{n-1})$ already be defined. We will construct x_n and define

$$U_R \quad \text{restricted to interval } I_n \text{ be equal to } U_{R,n}.$$

Now by brute force find strings $y_{i,j}, x_i, 1 \leq i, j \leq n$ satisfying constraint (C_n). Such strings are guaranteed to exist due to properties (1)-(5) of problem $U_{R,n}$. Let T_n be the total time needed to (deterministically) construct problem U_R up to stage $n-1$ and to find strings $y_{i,j}, x_i, 1 \leq i, j \leq n$ Let $x_n := 0^{T_n}$.
End of Stage n

By the very same reason as in the construction in the proof of theorem 12 in section 4.3, problem U_R has polynomial time decidable sets of instances and solutions and polynomial cost functions, furthermore we obtain a randomized approximation scheme for U_R by first computing the interval number of string x, if it is too small solve the instance to optimality by brute force and otherwise answering sufficiently many components of instance x corrcectly by the randomized algorithm for Π_R. Hence $U_R \in RPTAS \setminus REPTAS$, which completes the proof. □

Chapter 5

The Steiner Tree Problem

In this chapter we deal with the Steiner Tree Problem, which asks for a shortest network connecting a given finite set of points. This problem is one of the most intensively studied problems in combinatorial optimization and comutational complexity, due to its relevance in various applications ranging from Traffic Routing and Transportation Network Problems to VLSI design. It is closely connected to another famous and intensively studied optimization problem, the *Travelling Salesman Problem* (TSP, also known as the Travelling Salesperson Problem), which asks for a minimum length tour visiting each of n given points precisely once.

The history of the *Steiner Tree Problem* goes back to Fermat (1601-1665), who considered the following question:

> **Fermat's Problem:** *Given three points a, b, c in the plane, find a point d in the plane minimizing the sum of distances to the given points.*

A geometric solution was given by Toricelli before 1640.

This is already a special case of the *Euclidean Steiner Tree Problem* in the plane which was proposed by Jarnik and Kössler in 1934: Given a finite set of points S in the Euclidean plane, construct a minimum length network T consisting of a finite set of points in the plane such that S is connected by that network. Formally T is a connected graph $T = (V_T, E_T)$ with $S \subseteq V$, and the length (or cost) of the tree is defined as the sum of lengths of its edges, where the length of an edge is the Euclidean distance of its endpoints.

In 1965 Hanan formulated the rectilinear version of the problem (i.e. we take the L_1 distance instead of the Euclidean or L_2 distance.

In 1971 Hakimi [Hak71] and Levin [Lev71] independently proposed the graph or network version of the problem, usually known as the **Network Steiner Tree Problem**. The L_1-version can be directly seen to be a special case of the network version.

In this chapter we give an introduction into the **Network Steiner Tree Problem**. Throughout this chapter and the rest of the thesis, if nothing different is esplicitly mentioned, we deal with the Network Steiner Tree Problem and simply call it the *Steiner Tree Problem*. While it is easy to see that minimum spanning trees

provide 2-approximations to the Steiner Tree problem (see [TM80] for implementational issues), the first nontrivial polynomial time approximation algorithm for the *Steiner Tree Problem* was given by Zelikovsky [Zel93] in 1993, achieving an approximation ratio of 11/6, based on a greedy strategy. Since then, several improvements were obtained. Let us only mention some of the most important of them: Berman and Ramaiyer [BR94] gave a polynomial time 1.78 approximation algorithm based on a two phase approach. In 1995 Zelikovsky [Zel95] provided a 1.69 approximation algorithm, the so called **Relative greedy Heuristic (RGH)**. Based on this result, Karpinski and Zelikovsky [KZ97b] gave a 1.644 approximation algorithm which consists of one call of the **RGH** algorithm preprocessed by another call with a modified gain function. Hougardy and Prömel [HP99] iterated the Karpinski Zelikovsky algorithm, obtaining an approximation ratio of 1.59. Robins and Zelikovsky [RZ00b] obtained a 1.55 approximation algorithm, which is based on amodified relative greedy approach using concepts from [KZ97b].

The chapter is organized as follows: In section 5.1 we give precise definitions of the different versions of the Steiner Tree Problem mentioned above. We will also list some very basic facts about Minimum Steiner Trees, e.g. the equivalence of the weighted graph case and the metric case. Section 5.2 deals with known lower bounds for the approximability of the Steiner Tree Problem. In section 5.3 we will briefly describe two well-known exact algorithms for the Steiner Tree Problem, i.e. exponential time algorithms always computing optimum solutions. These algorithms will play an essential role in applications to **Dense Steiner Tree Problems** in chapter 9 of this thesis. Finally in section 5.5 we will give a brief survey on approximation algorithms for the Steiner Tree Problem, indicating the underlying ideas and methods.

We will not discuss here the numerous results on special cases of the Steiner Tree Problem, but only mention one of the most important ones: It was a longstanding open problem whether there is a polynomial time approximation scheme for the Euclidean Steiner Tree Problem (and in general for the Geometric Steiner Tree Problem in fixed dimension). This question was answered affirmatively by Sanjeev Arora [Aro98] (Mitchell [Mit99] independently obtained similar results).

5.1 Problem Formulation

Let us now give the precise problem formulation for the graph and metric Steiner tree problems and then argue why they are equivalent.

Steiner Tree Problem
Instance: graph $G = (V, E)$, a subset $S \subseteq V$ of *terminals* and a cost function $c\colon E \to \mathbb{R}_+$

Solution: a tree $T = (V(T), E(T)) \subseteq G$ such that $S \subseteq V(T)$

Cost: $c(T) = \sum_{e \in E(T)} c(e)$

Metric Steiner Tree Problem
Instance: A finite metric space (V, d) (i.e. V is a finite set and $d: V \times V \to \mathbb{R}_+$ is symmetric and satisfies the triangle inequality) and a terminal set $S \subseteq V$.

Solution: tree $T = (V(T), E(T))$ with $S \subseteq V(T) \subseteq V$ and $E(T) \subseteq P_2(V(T))$.

Cost: $d(T) := \sum_{e=\{u,v\} \in E(T)} d(u, v)$

Lemma 5.1.1 *The Steiner Tree Problem and the Metric Steiner Tree Problem are equivalent with respect to L-reductions with parameters $\alpha = \beta = 1$ in both directions.*

Proof: On the one hand, the metric case is a special case of the Steiner Tree Problem in graphs with G being the complete graph on V and edge weights equal to metric distances. On the other hand, given an instance of the Steiner Tree Problem in edge-weighted graphs consisting of graph $G = (V, E)$ with edge costs $c: E \to \mathbb{R}_+$ and terminal set $S \subseteq V$, one can in polynomial time construct the distance graph on V, i.e. the complete graph on V with costs equaling shortest path distances in G, c, and solving the problem in this metric space directly yields a solution for the graph problem with no cost increase. □

Hence from now on, let the **Steiner Tree Problem (SMT)** denote the problem in edge weighted graphs (networks) **or** equivalently the metric Steiner Tree Problem. If not esplicitly stated, we will no longer distinguish between the two cases.

5.2 Lower Bounds

The decision version of the **Steiner Tree problem** was already proved being NP-complete by Karp [Kar72], using the following reduction from **SAT**: Given an instance

$$\varphi = C_1 \wedge \ldots \wedge C_m$$

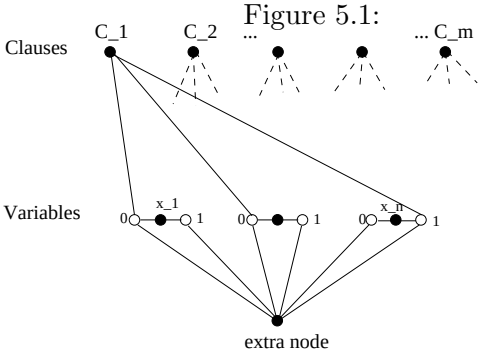
of **Max SAT** with clauses $C_1, \ldots, C_m$ and variables $x_1, \ldots, x_n$, we construct graph $G = G_\varphi = (V, E)$ as follows: For each clause C_j we take a terminal t_j, for each variable x_i we take a three-vertex path $P_i = v_{i,0} - v_i - v_{i,1}$ and let v_i be a terminal as well, furthermore we take one more vertex u. We connect all vertices $v_{i,0}, v_{i,1}$ to u by one edge, furthermore we take all edges $\{v_{i,\alpha}, t_j\}$ such that literal x_i^α ($\alpha \in \{0, 1\}$) occurs in clause C_j. The construction is shown in figure 5.1.

If we start from the problem **Max-3-OCC-MAX-3SAT**, the construction yields an L-reduction. **Max-3-OCC-MAX-3SAT** is the special case of **Max SAT** where clauses have length bounded by 3 and each variable occurs at most three times in the formula.

Theorem 20 *The construction of Karp as described above yields an L-reduction*

Max-3-OCC-MAX-3SAT $\leq_L$ *Steiner Tree Problem*

Figure 5.1:



with parameters $\alpha = 15$ *and* $\beta = 1$.

Proof: The proof consists in following the proof of Papadimitriou and Yannakakis and then analysing what happens if we use Max-3-OCC-MAX-3SAT as a starting point of the reduction: Given an assignment $\beta: \{x_1, .., x_n\} \to \{0, 1\}$, let the tree T_β in G_φ consist of all edges $\{u, v_{i,\beta(x_i)}\}$, $\{v_{i,\beta(x_i)}, v_i\}$, for each satisfied clause C_j an edge connecting it to the vertex corresponding to a satisfying true literal and for each unsatisfied clause a path of length 2 connecting it to u. Then obviously T_β is a Steiner Tree for the terminal set

$$S_\varphi \;=\; \{t_1, \ldots, t_m, v_1, \ldots, v_n\}$$

in graph G_φ of cost

$$c(T_\beta) = 2n + 2m - |\{j \in \{1, \ldots, m\} : \beta(C_j) = 1\}|.$$

On the other hand, let T be a Steiner Tree for terminal set S_φ in graph G_f. We call such tree a **normal form tree** if it satisfies the following conditions:

1. Each clause vertex t_j has degree 1.
2. Each variable vertex v_i is connected to exactly one of the vertices $v_{i,0}, v_{i,1}$.
3. If $\{v_i, v_{i,j}\}$ is an edge of T, then $\{v_{i,j}, u\}$ is an edge of T as well.

In polynomial time each tree T can be transformed into a no longer normal form tree T'. Hence we now assume T to be in normal form. We will now construct an assignment β_T as follows:

$$\beta_T(x_i) = \begin{cases} 0 & if \text{ only } \{v_i, v_{i,0}\} \text{ is an edge of } T \\ 1 & if \text{ only } \{v_i, v_{i,1}\} \text{ is an edge of } T \\ \alpha & if \text{ both } \{v_i, v_{i,0}\},\ \{v_i, v_{i,1}\} \text{ are edges of } T \end{cases}$$

where $\alpha \in \{0, 1\}$ is such that x_i^α occures more often than $x_i^{1-\alpha}$ in formula φ. For MAX-SAT we have $\frac{m}{2} \le \text{opt} \le m$. For **Max-3-OCC-MAX-3SAT** we obtain

$$n/3 \le m \le n, \;\; \text{therefore} \;\; \text{smt} = 2n + 2m - \text{opt}(f) \le 8m - \text{opt}(f) \le 16\text{opt} - \text{opt} = 15\text{opt},$$

which establishes $\alpha = 15$. Furthermore we have

$$\begin{aligned} \text{opt}(f) - |\{C \text{ clause}{:}\beta(C) = 1\}| &\le m - \big(2(n+m) - c(T)\big) \\ &= c(T) - (2n+m) \le c(T) - \text{opt}(G_f, S_f) \end{aligned}$$

which directly implies $\beta = 1$, completing the proof. □

One can now use well known results on the approximation hardness of Max-3OCC-MAX-3SAT in order to obtain hardness results for the Steiner Tree Problem. One starts from the following fundamental result from Hastad on hardness of the problem **Max-E3-Lin-2**, which is maximum satisfiability of linear equations modulo 2 with exactly 3 variables per equation.

Theorem 21 (Hastad 1997 [Has97]) *For every $\epsilon \in (0, 1/4)$ and sufficiently large integer $k \geq k(\epsilon)$ the following problem is NP-hard: Given an instance of* **Max-E3-Lin-2** *consisting of n equations with exactly $2k$ occurrences of each variable, decide if at least $(1-\epsilon)n$ or at most $(1/2+\epsilon)n$ equations are satisfied by the optimum assignment. Equivalently: For E3-Lin-2 with $2n$ equations and n variables it is hard whether $MaxLin(f) \leq (1+\epsilon)n$ or $MaxLin(f) \geq (2-\epsilon)n$.*

Berman and Karpinski obtained improved hardness results for the problem **3OCC-E2-Lin-2**:

Theorem 22 (Berman, Karpinski 1998 [BK98b]) *It is NP-hard for 3OCC-E2-Lin2 instances with $336n$ equations to decide whether opt $\leq (331+\epsilon)n$ or opt $\geq (332+\epsilon)n$.*

Recently they were able to further improve on this, obtaining the following result.

Theorem 23 (Berman, Karpinski 2003 [BK03]) *It is NP-hard to approximate* **E3-OCC-E2-Lin-2** *within $\frac{112}{111} - \epsilon$.*

Now one can use a reduction from **3OCC-E2-Lin-2** to **12OCC-E2-SAT** by replacing each linear equation $x + y = 0/1$ by a set of 4 clauses. Starting from the first result of Berman and Karpinski [BK98b], it is NP-hard for **3OCC-E2-Lin2** instances with $336n$ equations and n variables to decide whether opt $\leq (331+\epsilon)n$ or opt $\geq (332+\epsilon)n$. This yields $672n$ clauses and n variables in the according 12OCC-E2-SAT instance, and using the above reduction one obtains a Steiner Tree instance with $4n+672n*2 = 1348n$ edges, $(672+3)n+1 = 675n+1$ nodes and $673n+1$ terminals, where the cost of an optimum Steiner Tree is $\text{smt} = 2n + (2^1+1)m - MaxLin(f) = 2n + 3*336n - MaxLin(f) = 1010n - MaxLin(f)$, f being the 3OCC-E2-Lin2 instance we start from.

Therefore is NP-hard for SMT instances with $1348n$ edges, $675n+1$ nodes and $673n+1$ terminals to decide whether $\text{smt} \leq 1010n - (332-\epsilon)n = (678+\epsilon)n$ or $\text{smt} \geq 1010n - (331+\epsilon)n = (679-\epsilon)n$. Hence one obtains the following hardness result for the **Steiner Tree Problem**.

Corollary 5.2.1 *It is NP-hard to approximate the Steiner Tree Problem within $A.R. \frac{679}{678} - \epsilon \approx 1.0014 - \epsilon$.*

In 1989, Bern and Plassmann [BP89] considered the following special case of the Steiner Tree Problem where edge lengths are restricted to 1 and 2:

$(1,2)-$Steiner Tree Problem (**$(1,2)$-STP**)
Instance: finite vertex set V, cost function $c : P_2(V) \to \{1,2\}$,
Terminal set $S \subseteq V$

Solution: Steiner Tree T for S in (V,c)

Cost: $c(T)$, the length of the tree

This problem is well known to be a special case of the **Metric Steiner Tree Problem**, as is formulated in the following lemma:

Lemma 5.2.1 *If V is a finite set and c is a function $c: P_2(V) \to \{1,2\}$, then (V,c) is a finite metric space.*

Proof: It suffices to prove that the triangle inequality holds, but this is obvious: Let $a,b,c \in V$, then $c(a,c) \leq 2$ but $c(a,b) + c(b,c) \geq 1+1 = 2$. □

Bern and Plassmann [BP89] were able to give an L-reduction from the **Bounded Degree vertex Cover Problem** to the **$1-2-$Steiner Tree Problem**, implying MAX SNP-hardness of the latter. We will first define the **Bounded Degree Vertex Cover Problem** and then state the Bern-Plassmann result.

B$-$Vertex Cover Problem (**B-VC**)
Instance: Graph $G = (V,E)$ such that for all
vertices $v \in V$, $d_G(v) \leq B$
Solution: a vetex cover C for G, i.e. a set of vertices
$C \subseteq V$ such that $\forall e \in E\, e \cap C \neq \emptyset$

Cost: $|C|$, the cardinality of the vertex cover

Theorem 24 *[BP89] There is an L-Reduction from* **B$-$Vertex Cover Problem** *to the* **$(1,2)-$Steiner Tree Problem** *with parameters $\alpha = B/2, \beta = 1$.*

Proof: Given a graph $G = (V,E)$ with vertex degree bounded by B, construct an instance of the Steiner Tree Problem consisting of graph $H = (V_H, E_H)$ and terminal set $S \subseteq V_H$ as follows: For each edge e of G we introduce a vertex v_e, furthermore for each vertex u of G a vertex v_u, hence $V_H = \{v_e | e \in E\} \cup \{v_u | u \in V\}$. For each $u \in e \in E$ we let $\{v_u, v_e\}$ be an edge of H, furthermore we add all the edges $\{v_u, v_w\}$ for $u,w \in V$. The terminal set is defined as $S := \{v_e | e \in E\}$.

Let us analyse the construction: For a vertex cover $U \subseteq V$ in graph G we can construct a Steiner Tree T_U by taking $\{v_u | u \in U\}$ as set of Steiner Points, connecting each edge vertex v_e to a node v_u such that $u \in U$ covers edge e and adding the edges of a spanning tree of length $|U| - 1$ for the set $\{v_u | u \in U\}$. We get

$$\mathrm{smt}(G', E) = |E| + |VC| - 1.$$

Furthermore, if T is a Steiner Tree with $|T|-1$ edges, we can assume that each edge of G has degree 1 in T (if the degree is 2, neighbours are u, v and $e = \{u, v\}$ then replace one of the edges $\{e, u\}, \{e, v\}$ by $\{u, v\}$). Then the Steiner nodes of T define a cover U_T in G, we get

$$|U_T| = \text{ smt}(G', E) - |E| - 1.$$

If G has $\Delta_G = B$ then $|E| \leq B/2\,|V|$, and $|VC| \leq |E| \leq (B/2)\,|V|$ shows $\text{opt}(G', E) \leq (B/2)|V| + |VC| - 1 \leq (B/2)|V| + (B/2)|V| - 1 = B|V| - 1 \leq B\text{opt}(G)$. Furthermore if T is a Steiner tree then $\big||U_T| - \text{opt}(G)\big| = \big|c(T) - |E| - 1 - (\text{ smt} - |E| - 1) = c(T) - \text{ smt}$. Therefore we have

$$\mathbf{B-VC} \leq_L \mathbf{STP} \quad \text{with parameters } \alpha = B/2, \beta = 1.$$

□

Berman and Karpinski [BK98a, BK98b] also obtained hardness results for bounded-degree versions of the Minimum Vertex Cover Problem:

Theorem 25 (Berman, Karpinski 1998 [BK98a, BK98b])
Minimum Vertex Cover is NP-hard to approximate within $\frac{79}{78} - \epsilon$ *in graphs* G *with maximum degree* $\Delta_G = 4$ *and within* $\frac{145}{144} - \epsilon$ *in graphs* G *with* $\Delta_G = 3$.

More precisely, they proved the following: For $\epsilon \in (0, 1/2)$ it is NP-hard to decide whether an instance of the problem **3MIS** (Maximum Independent Set in graphs with maximum degree 3) with $284n$ nodes has maximum independent set of size below $(139 + \epsilon)n$ or above $(140 - \epsilon)n$. Since independent sets are complements of vertex covers, one obtains the following equivalent formulation for the Vertex Cover Problem: It is NP-hard to decide whether an instance of the problem **3VC** with $284n$ nodes has a Minimum Vertex Cover of size below $(144 + \epsilon)n$ or above $(145 - \epsilon)n$?

Their result for graphs with $\Delta_G = 4$ is the following: For $\epsilon \in (0, 1/2)$ it is hard to decide whether an instance of 4MIS with $152n$ nodes has max independent set of size below $(73 + \epsilon)n$ or above $(74 - \epsilon)n$. Equivalently it is hard to decide whether the minimum VC is of size below $(78 + \epsilon)n$ or above $(79 - \epsilon)n$?

Combining these results with the Bern-Plassmann reduction, one obtains: The Berman-Karpinski graph for 3MIS has at most $3 * 284n/2 = 432n$ edges, therefore it is hard whether $\text{ smt} \leq 432n + (144 + \epsilon)n - 1 = (576 + \epsilon)n - 1$ or $\text{ smt} \geq 432n + (145 - \epsilon)n - 1 = (577 - \epsilon)n - 1$. Therefore the resulting $A.R$ being NP-hard for SMT is at least
$577/576 - \epsilon \approx 1.0013 - \epsilon$.

The graph for 4MIS has at most $4 * 152n/2 = 304n$ edges, therefore it is hard whether $\text{ smt} \leq 304n + (78 + \epsilon)n - 1 = (382 + \epsilon)n - 1$ or $\text{ smt} \geq 304n + (79 - \epsilon)n - 1 = (383 - \epsilon)n - 1$. Therefore the resulting $A.R$ being NP-hard for SMT is at least $\frac{383}{382} - \epsilon \approx 1.0026 - \epsilon$.

Corollary 5.2.2 *It is NP-hard to approximate the* $(1, 2)$*-Steiner Tree Problem within A.R.* $\frac{383}{382} - \epsilon \approx 1.0026 - \epsilon$.

In 2001 Thimm [Thi01] was able to obtain the following hardness result for the Steiner Tree Problem which is based on starting directly from the Hastad result and using special expander constructions:

Theorem 26 (Thimm 2001 [Thi01])
The Steiner Tree Problem is NP-hard to approximate within 1.00617.

In 2003 Chlebik and Chlebikova [CC02] obtained the following improved hardness result which is based upon ideas from Thimm and improved expander constructions.

Theorem 27 (Chlebik, Chlebikova 2003 [CC02])
The Steiner Tree Problem is NP-hard to approximate within 1.01063.

5.3 Two Exact Algorithms

For the sake of completeness and for later purpose (see chapter 9 on Dense Steiner Problems) we will now describe two well-known exact algorithms for the Steiner Tree Problems, namely the Dreyfus-Wagner algorithm and the Spanning Tree Enumeration Algorithm due to Hakimi. The first one has running time polynomial in the number of non-terminals and exponentialin the number of terminals, while the second runs in time polynomial in the number of terminals but exponential in the number of non-terminals in the graph.

5.3.1 The Dreyfus-Wagner Algorithm

The Dreyfus-Wagner Algorithm uses a dynamic-programming strategy to solve the Steiner Tree Problem. In this subsection we give a description of this algorithm based on (Steiner Tree book). The idea is as follows: Let T be some optimum Steiner tree for a terminal set S in a given graph $G = (V, E)$ with edge weights $c\colon E \to \mathbb{R}_+$. Let v be a vertex in T of degree at least two. Partition the set of neighbors $N_T(v)$ of v in T into two subsets N_1, N_2. This splits T into two subtrees T_1, T_2 which have the vertex v in common. For $i = 1, 2$ let $S_i \subset S$ be the subset of terminals which belong to T_i. Then T_1 is an optimum Steiner tree for the set S_i $(i = 1, 2)$. The Dreyfus-Wagner Algorithm is based on this *optimal deecomposition property*. For $\emptyset \neq S' \subset S$ and $v \in V \setminus S'$ let $T_G(\{v\} \cup S')$ denote an optimum Steiner Tree for the terminal set $\{v\} \cup S'$ in G and $T_G^2(\{v\} \cup S')$ denote a minimum length union of two Steiner trees, one for a terminal set $\{v\} \cup S''$ for some $\emptyset \neq S'' \subset S'$ and one for the terminal set $\{v\} \cup (S' \setminus S'')$. Hence the following recurrences hold:

- $\text{cost}(T_G^2(\{v\} \cup S) \;=\; \min_{\emptyset \neq S'' \subset S'} \{\text{cost}(T_G(\{v\} \cup S'')) + \text{cost}(T_G(\{v\} \cup (S' \setminus S')))\}.$
- $$\begin{aligned}\text{cost}(T_G(\{v\} \cup S') \;=\; \min\{\; & \text{cost}(T_G(\{v\} \cup S),\\ & \min_{w \notin S'}\{\text{dist}_G(v,w) + \text{cost}(T_G^2(\{w\} \cup S'))\},\\ & \min_{w \in S'}\{\text{dist}_G(v,w) + T_G(S')\} \qquad \}\end{aligned}$$

It is now straightforward to use these recurrences in order to build a dynamic programming algorithm.

Theorem 28 *The Dreyfus-Wagner algorithm has running time bounded by* $O(3^n|S| + 2^n|S|^2 + n^3)$.

Note that the running time is polynomial in the number of terminals but exponential in the number of non-terminals in G. Then deleting v from T splits T into $d_T(v)$ many disjoint subtrees.

5.3.2 The Spanning Tree Enumeration Algorithm

In this subsection we describe an exact algorithm for the Steiner tree Problem which has running time exponential in the number of terminals but polynomial in the number of non-terminals in G. Since the edge-weighted graph version of the Steiner Tree Problem is equivalent to the metric case we can restrict ourselves to the latter. Hence assume G is a complete graph and c satisfies triangle inequality. Then it is obvious that a Minimum Steiner Tree for S in G can be assumed to have at most $|S| - 2$ Steiner points (since we can assume each Steiner point to have degree at least 3 in T). Note that a Minimum Steiner Tree is a minimum spanning tree for its vertex set (i.e. the union of the terminal set and the set of Steiner points). Enumerating all supersets of S in G of size at most $2|S| - 2$ and computing a minimum spanning tree for each of them obviously yiels an exact algorithm for the Steiner Tree Problem. The running time is determined by the number of supersets of S being considered, which is given by

$$\sum_{i=0}^{|S|-2} \binom{n-|S|}{i} \leq 2^{n-|S|}.$$

Theorem 29 *The time complexity of the Spanning Tree Enumeration Algorithm is* $O(|S|^2 2^{|S|-n} + n^3)$.

5.4 The k-Steiner Ratio

Let V, c, S be an instance of the Metric Steiner Tree Problem, let T be some Steiner Tree for S in (V, c). Then T naturally splits into **full components**, where a full component is a subtree K of T such that all leaves of K are terminals and all internal nodes of T are non-terminals. The **size of a full component** K defined as the number of terminals of K.

Definition 35 (Full Steiner Trees, k-Restricted Steiner Trees)

(a) A **full Steiner Tree** *for S in (V, c) is a Steiner tree for S in (V, c) such that S is the set of leaves of T (and hence every internal node of T is not a terminal).*

(b) A Steiner Tree T for terminal set S is **k-restricted** *iff the size of each full component of T is bounded by k.*

Restricting the size of full components has two interesting features: On the one hand, if we bound the full-component size by a constant k, then for a given subset S' of S of size at most k we can compute a minimum-length k-restricted full Steiner Tree $K_{S'}$ for S' in polynomial time, using the ... algorithm described above.

On the other hand, size-restricted Steiner Trees have very nice approximation properties uniform in k. Namely, one can give a tight bound for the ratio length of an optimum k-restricted Steiner tree to length of an optimum Steiner Tree which only depends on k and not on the size of the input. This ratio is called **the k-Steiner Ratio**.

Definition 36 *The* **k-Steiner Ratio** *is defined as*

$$\rho_k \quad := \quad \sup\left\{ \frac{smt_k(M,S)}{smt(M,S)} : M = (V,c) \text{ a finite metric space, } S \subseteq V \right\}.$$

For a fixed instance $M = (V,c), S$ of the Metric Steiner Tree Problem, we denote

$$\rho_k(M,S) \quad := \quad \frac{smt_k(M,S)}{smt(M,S)},$$

hence

$$\rho_k \quad := \quad \sup\left\{\rho_k(M,S) | M = (V,c) \text{ a finite metric space, } S \subseteq V\right\}.$$

In 1997, A.Borchers and D.-Z. Du [BD97] were able to finally give the precise values for ρ_k. We cite their result and refer to their paper for the proof and further information on the history of research concerning the k-Steiner Ratio.

Theorem 30 (A.Borchers, D.-Z.Du 1997 [BD97])
For $k = 2^r + s$, $0 \leq s < 2^r$ the k-Steiner Ratio is

$$\rho_k = \frac{(r+1)2^r + s}{r2^r + s}.$$

Especially, for $k \longrightarrow \infty$, $\rho_k \longrightarrow 1$.

5.5 Approximation Algorithms for the Steiner Tree Problem

The lower bound results for the Steiner Tree Problem surveyed in section 5.2 indicate that for the Steiner Tree Problems polynomial time approximation schemes are unlikely to exist. Hence one is interested in polynomial time algorithms which approximate the problem to a constant factor. We will in this section survey the most important results that have been obtained, starting from a very simple 2-approximation algorithm due to Takahashi and Matsuyama [TM80], based on a Minimum Spanning Tree approach. We will then describe the 1.78 approximation algorithm of Berman and Ramaiyer [BR94], the Greedy Contraction Framework of Zelikovsky [Zel95] using which he achieved an approximation ratio of 1.69, the 1.644 approximation ratio of Karpinski and Zelikovsky [KZ97c], the 1.59 algorithm of Hougardy and Pr"omel [HP99] and finally the 1.55 approximation algorithm due to Robins and Zelikovsky [RZ00a] which is up to our

knowledge currently the best known approximation algorithm for the Steiner Tree Problem.

In spite of the fact that the k-Steiner Tree Ratio tends to 1 when k goes to infinity, in order to approximate the Steiner Tree Problem it suffices to approximate the optimum k-restricted Steiner Tree. Indeed all the polynomial time approximation algorithms for the general (metric) Steiner Tree Problem we are aware of are based on this fact.

A Simple 2-Approximation Algorithm
Consider an instance of the Steiner Tree Problem consiting of finite metric space (V, c) and terminal set $S \subseteq V$. Let M be a minimum spanning tree for S with respect to c. Then M is a 2-approximative Steiner Tree for S. The reason is the following: Consider an optimum Steiner tree T, and do the following for each full component K of T separately: Draw K in the plane, then we have a clockwise sorting of the terminals $s_1, \ldots, s_k$ where $k = |K|$ is the size of the component. Now replace K by the path $s_1, s-2, \ldots s_k$, it is clear that we have to take each edge of K at most twice. This approach was already suggested by Takahashi and Matsuyama [TM80].

Minimum Spanning Tree based approaches, gain of components [Zel93, KZ97b, BR94]
Let $G = (V, d)$ be a finite metric space and $S \subseteq V$ a nonempty set of terminals. Let $M = (S, E, c)$ be an edge weighted tree with vertex set S, called *terminal spanning tree*. Let K be a full Steiner tree for a subset $S' \subseteq S$ of the terminal set. Let $R(M, K) \subseteq E$ be a set of edges such that $M - R(M, K) \cup K$ is a tree again. $R(M, K)$ is called a removal set of K w.r.to M. The gain of K is defined as

$$\text{gain}_M(K) = c(R(M, K)) - d(K).$$

The following theorem assures that under certain conditions a full component of not too small gain does exist.

Theorem 31
Let $M = (S, E, c)$ be a terminal spanning tree of cost $c(M) = (1 + x) \cdot \ smt_k(S)$. Then there is a k-restricted full Steiner tree K for a subset of S with $gain_M(K) \geq (1 + x) \cdot d(K)$.

Theorem 32
Let $M = (S, E, c)$ be a terminal spanning tree such that no k-restricted full component has positive gain w.r.to M. Then $c(M) \leq \ smt_k(S)$.

The Algorithm of Berman and Ramaiyer [BR94]
The approach of Berman and Ramaiyer is a 2-phase algorithm which we call **BR** in the sequel, achieving an approximation ratio of 1.78. In a first phase all possible full components from size 3 to k are considered, their gain with respect to the current tree (starting with a minimum spanning tree) is estimated. In the second phase the

decision is made which of these components to take into the Steiner tree. Let us give some details.

In a first phase all full components from size 3 to k are considered. At each step **BR** holds a spanning tree M for S, initially $M = \text{Mst}(S)$. If a component K has positive gain compared to M, then roughly said the cost of edges in M which would be deleted when inserting K is decreased by the gain of K. The effect is that for further components it gets more difficult to also throw out these edges. Thus in the first phase **BR** constructs a sequence of trees $\mathbf{M_0}, .., \mathbf{M_k}$, each of them being a spanning tree for S. The positive gain components are stored in stacks $\sigma_\mathbf{3}, .., \sigma_\mathbf{k}$. In the second phase components are popped from the stacks in reverse order and the tree $M_k = N_k$ is modified in steps $N_k, .., N_3$. N_3 is the output tree. Directly from the definition of the algorithm one obtains $m_{j-1} - m_j = (j-1)\sum_{K\in\sigma_j} g_K \quad (3 \le j \le k)$. Let n_i denote the cost of tree N_i, then the cost decrease per step can be bounded as $n_{j-1} - n_j \le (j-2)\sum_{K\in\sigma_j} g_K$ for $3 \le j \le k$. Therefore one gets the following upper bound for the length of the output tree N_2 in terms of lengths of the trees M_i from the first phase:

$$\begin{aligned} \text{c}(BR(S)) = n_2 &= n_k + \sum_{j=3}^{k}(j-2)\sum_{K\in\sigma_j}\text{gain}(K) \\ &\le m_k + \sum_{j=3}^{k}\frac{j-2}{j-1}(m_{j-1}-m_j) = m_2 - \sum_{j=3}^{k}\frac{m_{j-1}-m_j}{j-1} \end{aligned}$$

The crucial step is then to show that $m_j \le \text{opt}_j$ for $j = 3, .., k$. In order to do this, the above result bounding the length of tree for which no k-restricted component has positive gain is used.

Sketch of the Proof of $m_j \le \mathbf{opt}_j$, $j = 3, .., k$
(1) Let T be a Steiner Tree with full components $K_1, .., K_p$, M a spanning tree such that no K_i has positive gain, then $\text{cost}(M) \le \text{cost}(T)$.
(2) For the tree M_j after considering all j-size components, no j-component has positive gain, i.e. For all $K \in P_j(S)$: $\text{gain}_T(K) \le 0$ This combined with result (1) directly implies $m_j \le \text{opt}_j$, $j = 3, .., k$.
(3) To prove (2) the following subresults are used:

(3a) For every s-element subset K of $S \quad (3 \le s \le k)$ it happens somewhere during the evaluation phase step s that $\text{gain}_M(K) \le 0$.

(3b) For all $K \subset S$: No modification of M during the evaluation phase increases $\text{gain}_M(K)$.

(2) follows from (3a) and (3b): Since M is a min spanning tree, no 2-component (=edge) has positive gain w.r.to M_2. The rest follows immediately.
Proof of (3a): Consider the moment at the first phase when K is considered. If $\text{gain}_M(K) \le 0$ then (3a) holds for K. Otherwise M is modified to $M - R \cup A$. We

show $\text{gain}_{M':=M-R\cup A}(K) \leq 0$. The reason is: A is a spanning tree for the terminals of K. So it is the only removal set for K in M'. The cost of A is $(2-s)\text{gain}_M(K)$ because we have $\text{gain}_M(K) = \text{cost}(R) - \text{cost}(K)$, $\text{cost}(R) - \text{cost}(A) = (j-1)\text{gain}_M(K)$.
Proof of (3b): Therefore two results are shown.

(3b1) If $K \subseteq S$, $K = B \cup \{v\}$ ($v \notin B, B \neq \emptyset$), then we can compute the cost of a removal set for K in M as $\text{rcost}(M,K) = \text{rcost}(M,B) + \min_{u \in B} \text{lcost}(M,u,v)$, where $\text{lcost}(M,u,v)$ is the largest cost of an edge in the M-path between u and v. This yields an algorithm for the construction of a removal set by using bottleneck edges.

(3b2) A basic exchange for M is defined to be the insertion of an edge f and removal of the maximum cost edge in the cycle in $M \cup f$ created by f. (3b1) yields a way to construct removal sets by basic exchanges. The following is crucial: A basic exchange never increases the gains. In detail: Let $M' = M - e \cup f$ be a basic exchange. Then for any $K \subseteq S \quad \text{gain}_{M'}(K) \leq \text{gain}_M(K)$.

For the remaining details of the proof we refer to the original paper [BR94].

The Greedy Contraction Framework [Zel95]
There is a quite general paradigm used in many approximation algorithms for the Steiner Tree Problem, first formulated by Alexander Zelikovsky [Zel95]. The idea is as follows: Let $\mathcal{C}$ be a class of full Steiner trees for subsets of S such that a minimum length Steiner tree for S exists whose components are in $\mathcal{C}$. Start with a minimum spanning tree $M = \text{MST}(S)$ for the terminals. Then iteratively choose a component K from $\mathcal{C}$ maximizing a certain criterion function $f(M,K)$, add this component to an initially empty list L and modify M by performing certain contractions. Iteration stops if for all components $K \in \mathcal{C}$ $f(M,K) < 0$. Then a Steiner tree for S is constructed from the elements of L.

Greedy Contraction Framework (GCF)
(0) Start with $M = \text{MST}(S)$
(1) **repeat until** $\text{c}(M) = 0$
  (a) find a full Steiner Tree K^* in a class C
    minimizing a criterion function f
  (b) insert K^* in LIST
  (c) contract a subgraph $K' \subseteq K^*$
    (i.e. update M)
(2) reconstruct an output Steiner Tree from M and LIST

To design a specific algorithm, the class $\mathcal{C}$, the criterion function f and the operation of "contraction" have to be defined. A general analysis technique for algorithms of GCF type was told to us by P. Berman [Ber01], working as follows: Let M_0 be the initial terminal spanning tree and M_i denote the terminal spanning tree after i steps of the algorithms (i.e. i executions of the loop). Let K_i be the full component chosen

in the ith iteration of the loop. Let us consider "contractions" of the following form: Find a component $K_i \in \mathcal{C}$ of minimum value $f(M_{i-1}, K_i)$. Let $R_i := R(M_{i-1}, K_i)$ be a maximum cost set of edges from M_{i-1} such that the graph $M_{i-1} \setminus R_i \cup K_i$ is a tree. Let M_i be a terminal spanning tree constructed from $M_{i-1} \setminus R_i \cup K_i$ by contracting some subgraph C_i of K_i. Assume that if after t steps the algorithm stops, then $\mathrm{c}(M_t) \leq Z$. Using $M_0 = \mathrm{MST}(S)$ as an initial tree, it always holds $\mathrm{c}(M_0) \leq 2 \cdot \mathrm{smt}(S) \leq 2 \cdot \mathrm{smt}_k(S)$. Assume further that if M is a terminal spanning tree of cost $\mathrm{c}(M) \geq (1+x) \cdot Z$, then we can always find a component $K \in \mathcal{C}$ such that $\mathrm{gain}_M(K) \geq (1+x)\mathrm{c}(K)$ and $\mathrm{c}(C) \leq \alpha \cdot \mathrm{c}(K)$, where C is the subgraph of K to be contracted and α is some fixed number from $[0,1]$. From the definition of gain it follows $\mathrm{gain}_M(K_i) = \mathrm{c}(R_i) - \mathrm{c}(K)$. Further, the cost of M is reduced by $\mathrm{gain}_M(K_i) + \mathrm{c}(C_i)$. Then the cost of the Steiner tree T constructed by GCF can be bounded as

$$\begin{aligned} \mathrm{c}(T) &\leq \mathrm{c}(M_t) + \sum_{i=1}^{t} \mathrm{c}(C_i) \leq Z + \sum_{i=1}^{t} \frac{\mathrm{contract}_i}{\mathrm{gain}_i + \mathrm{contract}_i} \cdot \mathrm{reduce}_i \\ &\leq Z + \int_0^1 \frac{\mathrm{contract}}{\mathrm{gain} + \mathrm{contract}} \leq Z + \int_0^1 \frac{\alpha}{1 + x + \alpha} \end{aligned}$$

In the sequel we will discuss some examples of the GCF method and application of the analysis method described above. The first example is an approximation algorithm of Zelikovsky [Zel95] for the graph Steiner tree problem with ratio ≈ 1.69.

Zelikovsky's Relative Greedy Heuristic [Zel95]
The algorithm of Zelikovsky, called Relative Greedy Heuristic (**RGH**) is best described in a very compact way in terms of the Greedy Contraction Framework. This description is taken from Zelikovsky's paper: Define $\mathcal{C}$ to be the class of all k-restricted full Steiner trees for subsets of S. The criterion function is defined as

$$f(M, K) = \frac{\mathrm{c}(K)}{\mathrm{gain}_M(K).}$$

Further, in each choice step, the whole component is contracted. We give an explicit description of the algorithm:

Relative Greedy Heuristic (k-RGH)
Input: Weighted Graph $G = (V, E, \mathrm{c})$, Terminal Set $S \subseteq V$
$M := \mathrm{MST}(S)$
$H := S$
while $\exists$ k-restricted K with $\mathrm{gain}_M(K) > 0$
  Choose K of maximum value $\mathrm{gain}_M(K)/\mathrm{c}(K)$
  $H := H \cup \{\text{Steiner points of } K\}$
  $M := M/K$
Output the tree $\mathrm{MST}(S \cup H)$.

For the analysis, first note that $\text{contract}_i = \text{c}(K_i)$ and $\text{reduce}_i = \text{c}(K_i) + \text{gain}_{M_{i-1}}(K_i)$. Normalizing $\text{opt}_k = 1$, the approximation ratio then is

$$A.R.(RGH) \;\leq\; 1 + \int_0^1 \frac{1}{1+x}\,dx \;=\; 1 + \int_1^2 \frac{1}{x}\,dx \;=\; 1 + \ln(2) \;\approx\; 1.69$$

The 1.644 **Approximation Algorithm of Karpinski, Zelikovsky [KZ97b]**
Karpinski and Zelikovsky [KZ97b] obtained a polynomial time approximaion algorithm with ratio ≈ 1.644 for the Steiner Tree Problem, based on adding preprocessing phases to the relative greedy heuristic k-**RGH** which we have described above. The novel approach of Karpinski and Zelikovsky was to better estimate the use of full components and the extra cost they produce, in terms of the so called *loss of full components*. The loss, to be described in detail in the next part of this section, estimates the extra cost a full component K which it will produce if badly being chosen. Roughly spoken, if we pick a component K but will later make no real use of it, we have to connect the Steiner points of component K to the rest of our solution. The loss of K, denoted as $l(K)$, precisely estimates this extra connection cost. Now the algorithm of Karpinski and Zelikovsky, based on (a variant of) k-**RGH** combined with a preprocessing phase works as follows:

Algorithm Preproc-RGH

Input: instance of the metric Steiner Tree problem consisting
of finite metric space (V, c), and terminal set $S \subseteq V$

Preprocessing Phase:
Run k-**RGK**(α) which is k-RGH with the modified gain function
$p(K) := (c(K) + \alpha \cdot l(K))/m(K)$, where
$m(K) :=$ mst$(S) -$ mst$(S \cup E(K))$ and $E(K)$ is a set of
zero-cost edges between the terminals of K.
Add all the Steiner points of the constructed tree to S to obtain set S'.
Final Phase:
Run k-**RGH** with the enlarged terminal set S'
and return the resulting tree.

Theorem 33 *[KZ97b] Choosing $\alpha \approx 0.5$, the algorithm* **Preproc-RGH** *has a performance ratio ≈ 1.644.*

The 1.59**-Approximation Algorithm of Hougardy and Pr"omel [HP99]**
Hougardy and Pr"omel improved upon the preceedingly described result by simply iterating the Karpinski-Zelikovsky algorithm several times and giving a (quite technical and nontrivial) estimate of the total improvement that can be obtained. We cite their result and refer to the original paper for details and further information.

Theorem 34 *[HP99] There is a polynomial tiem approximation algorithm for the Steiner Tree Problem with approximation ratio* ≈ 1.59.

The Loss Contracting Algorithm of Robins, Zelikovsky [RZ00a]
The algorithm of Robins,Zelikovsky, called Loss Contracting Algorithm (k-LCA), is based on the following idea: Instead of contracting the whole component, only contract a subgraph of minimum cost such that even if none of the Steiner points of the component would be of any use, the contracted subgraph would collect them all. Formally, for a given full component K the Loss of K, denoted as $\mathrm{Loss}(K)$ is definded to be a minimum cost subgraph of K that contains all Steiner points of K and such that each connected component contains at least one terminal. The cost of $\mathrm{Loss}(K)$ is denoted as $\mathrm{loss}(K)$. It is not difficult to see that $\mathrm{loss}(K) \leq \frac{1}{2}\mathrm{c}(K)$. In each step the algorithm chooses a k-restricted full component K of maximum value $\frac{gain_M(K)}{\mathrm{loss}(K)}$ - equivalently: of maximum value $\frac{gain_M(K)+\mathrm{loss}(K)}{\mathrm{loss}(K)}$ - and contracts the subgraph $\mathrm{Loss}(K)$. Therefore k-LCA can be described in terms of the GCF in the following way: $\mathcal{C}$ is the class of all k-restricted full Steiner trees for subsets of S, the criterion function is

$$f(M,K) \;=\; \frac{\mathrm{loss}(K)}{\mathrm{gain}_M(K)+\mathrm{loss}(K)},$$

and the subgraph to be contracted is $C = C_K = \mathrm{Loss}(K)$. Here is an explicit description of the algorithm:

Loss Contracting Algorithm (k-LCA)
Input: Weighted Graph $G = (V, E, \mathrm{c})$, Terminal Set $S \subseteq V$
$M :=$ MST(S)
$H := S$
while $\exists$ k-restricted K with $\mathrm{gain}_M(K) > 0$
  Choose K of maximum value $\mathrm{gain}_M(K)/\mathrm{loss}(K)$.
  $H := H \cup \{\text{Steiner points of } K\}$
  $M := C[\ \mathrm{MST}(M \cup K)]$
Output the tree MST$(S \cup H)$.

Analysis of k-LCA in terms of the GCF described above:

$$\begin{aligned} A.R.(\ k\text{-LCA}) \;&\leq\; 1 + \int_0^1 \frac{1/2}{1/2+x}\,dx \;=\; 1 + \int_0^1 \frac{1}{1+2x}\,dx \\ &=\; 1 + \frac{1}{2}\int_1^3 \frac{1}{x}\,dx \;=\; 1 + \frac{\ln(3)}{2} \;\approx\; 1.55 \end{aligned}$$

Chapter 6

The Steiner Forest Problem

6.1 Introduction

In this chapter we consider two important generalizations of the Steiner Tree Problem: In the *Steiner Forest Problem* we are given a family of pairwise disjoint terminal sets $S_1, \ldots, S_n$ and ask for a minimum-cost subnetwork F of the underlying graph or metric space such that each set S_i is connected by F. Since we only consider the case of non-negative costs, F will always be a forest.

In the *Prize Collecting Steiner Tree Problem* we are given a single terminal set S as in the Steiner Tree Problem, but additionally each terminal $s \in S$ has a prize $p(s) \geq 0$. The task is to find a tree T connecting a subset $S' \subseteq S$ of the terminal set such as to minimize the cost of T plus the prizes for all terminals not connected by T (i.e. minimize $c(T) + \sum_{s \in S \setminus S'} p(s)$).

The Steiner Forest was already considered by Agrawal, Klein and Ravi in [AKR91], where they provided an approximation algorithm obtaining constant approximation ratio. Implicitly the approach contains already the whole machinery which builds the state-of-the-art for Steiner Forest Problems today, namely making use of the Primal Dual Method.

In the paper of Goemans and Williamson [GW92], this approach is made completely esplicit. They obtain 2-approximation algorithms for the Steiner Forest Problem and its generalization to **propewr constraint functions** as well as for the **Prize Collecting Steiner Tree Problem** and the **Prize Collecting TSP**.

Another formulation of the primal-dual approach was given by Bar-Yehuda [BY98] in terms of the **Local Ratio Framework**. This method allows for very compact formulations of algorithms and analysis. We will in this chapter give application of the Local Ratio Technique to a bunch of related problems in graphs and bounded hypergraphs, including a prize-collecting variant where prizes are given for connection requirements instead of terminals, yielding a 3-approxiamtion algorithm. This is due to the fact that for a given terminal, speaking in terms of local-ratio analysis, we pay cost 1 in order to connect it to the outside but we might be forced to pay the prize $O(n)$ when not doing so.

Further work on the topic and related problems was already done, including [Rav94, RW95, HRS00].

The chapter is organized as follows: In the next section we describe the approximation algorithms for the Steiner Forest Problem and the Prize Collecting Steiner Tree Problem due to Goemans and Williamson [GW92], based on the Primal-Dual Method. In section 6.3 we give a brief review of Bar-Yehuda's Local Ratio Framework. In section 4 we consider the Steiner Forest Problem in k-bounded hypergraphs. In section 5 we deal with the Prize Collecting Steiner Tree Problem in graphs and k-bounded hypergraphs. For the graph case Goemans and Williamson [GW92] gave a 2-approximation based on a primal-dual approach. We will give a short and simple description of a variant of this algorithm, based on the Local Ratio Framework [BY98]. Our algorithm will be rootless, therefore improving running time. In section 6 we generalize this approach to obtain a k-approximation algorithm for the Prize Collecting Steiner Pair Problem in k-bounded hypergraphs, a natural variant where prizes are given for connection requirements (pairs of vertices) instead of vertices. Although the pairs do not have to be disjoint and hence the number of pairs can be quadratic in the number of edges needed to connect them all we are able, using a carefull prepaying scheme, to get a 3-approximation algorithm for this problem. In section 6 we generalize this result to obtain a 3-approximation algorithm for the Prize Collecting Steiner Forest Problem in graphs.

6.2 The Primal-Dual Method

We give a very informal description of the Primal Dual approach as it was used for the Steiner Forest Problem by Goemans and Williamson [GW92], in terms of **growing balls**. Suppose we start with terminal sets $S_1, \ldots S_n$. Consider each terminal as a single component. Call a component C **active** if it has to be connected to the outside, which means there exists some set S_i such that $S_i \cap C \neq \emptyset$ and $S_i \setminus C \neq \emptyset$. Initially, all terminals are active (assuming $|S_i| \geq 2$ for all i. Now we start growing balls around the active components, each at the same rate. Whenever two balls meet, we take an according edge connecting them and shrink the components into one, eventually updating whether it still remains active. At the end we prune the set of collected edges in order to obtain a forest F, which we return.

What do we get ? Well, consider the execution of the algorithms between two events of components meeting. When component C is grown by amount Δ, we **know** that we have to spend at least amount Δ in order to connect it to the outside. When two components meet, say two active components, we are therefore willing to spend cost Δ for each of them,and due to the fact that we end up with a forest, on average we will spend 2Δ for each of them (the average degree in a tree is bounded by 2. This establishes approximation ratio 2.

In the case of **Prize Collecting Problems**, think of growing components in the plane and blowing up also in the third dimension which stands for the prizes, then the same argument works.

6.3 The Local-Ratio Framework of Bar-Yehuda

In this section we will briefly review the notations and results of Bar-Yehuda's Local Ratio framework [BY98]. An instance (X, f, ω) of the *Minimum Cover Problem* consists of a finite set X, a *monotone increasing* polynomial time computable function $f: P(X) \to \{0,1\}$ (i.e. $A \subseteq B \subseteq X$ implies $f(A) \leq f(B)$) and a weight function $\omega: X \to \mathbb{R}_+$. The subsets Y of X with $f(Y) = 1$ are called *(feasible) covers*. The problem is to compute a minimum weight cover Y:

- **Minimum Cover Problem:**
  Given (X, f, ω), find a cover $C \subseteq X$ of minimum weight $\omega(C) = \sum_{e \in C} \omega(e)$.

A function $\delta: X \to \mathbb{R}^+$ is called *weight reduction* for ω if for all $e \in X$ $0 \leq \delta(e) \leq \omega(e)$. The weight reduction δ is called *r-effective* if furthermore $\delta(X) \leq r \cdot \mathrm{OPT}(\delta)$. Consider the following recursive algorithm for the **Minimum Cover Problem:** based on weight reductions:

Algorithm $\mathcal{A}_1(X, f, \omega)$:
(1) **if** $C := \{e \in X : \omega(e) = 0\}$ is a cover
(2) **then** return C **else**
(3) Choose weight reduction δ.
(4) $C := \mathcal{A}_1(X, f, \omega - \delta)$.
(5) **return** C.

Lemma 6.3.1 *[BY98] If δ in line (3) of the algorithm $\mathcal{A}_1$ is allways r-effective with respect to the actual weigth function ω then $\mathcal{A}_1$ has approximatin ratio bounded by r.*

Sometimes it is not known how to find r-effective weight reductions since the requirement $\delta(X) \leq r \cdot \mathrm{OPT}(\delta)$ turns out to be quite restrictive. One way to come up with such situations was first applied to the Weighted Feedback Vertex Set Problem by Bafna, Berman and Fujito [BBF95] (see also [BY98]): Instead of requireing that every feasible cover has good approximation properties it suffices to consider only *minimal* covers. A cover $C \subseteq X$ is *ω-minimal* if for all $e \in C$ with $\omega(e) \neq 0$ $f(C \setminus \{e\}) = 0$. A weight reduction $\delta: X \to \mathbb{R}^+$ is called *called r-minimal effective* iff for every δ-minimal cover C $\delta(C) \leq r \cdot \mathrm{OPT}(\delta)$. This yields the following extension of algorithm $\mathcal{A}_1$:

Algorithm $\mathcal{A}_2(X, f, \omega)$:
(1) **if** $C := \{e \in X : \omega(e) = 0\}$ is a cover
(2) **then** return C **else**
(3) Choose weight reduction δ.
(4) $C := \mathcal{A}_2(X, f, \omega - \delta)$.
(5) **for all** $e \in C$ with $\delta(e) > 0$
(6) **if** $f(C \setminus \{e\}) = 1$ **then** $C := C \setminus \{e\}$.
(7) **return** C.

Lemma 6.3.2 *[BY98] If in each recursive call of algorithm $\mathcal{A}_2$ the weight reduction δ is r-minimal effecive with respect to the current weight function ω then algorithm $\mathcal{A}_2$ has approximation ratio bounded by r.*

6.4 The Steiner Forest Problem in k-bounded hypergraphs

In the Steiner Forest Problem one is given a graph $G = (V, E)$ with edge weights $c: E \to \mathbb{R}_+$ and pairwise disjoint terminal sets $S_1, \ldots, S_n$, and one has to find a minimum cost set of edges $F \subseteq E$ such that for each $1 \leq i \leq n$ the set S_i is connected by F. This problem generalizes the well known Steiner Tree Problem and has applications in VLSI design, electrical network design and many other areas. Agrawal, Klein and Ravi [AKR91] obtained a 2-approximation algorithm for the Steiner Forest problem in graphs. Ravi [Rav94] gave a primal-dual algorithm with the same ratio. Goemans and Williamson [GW92] generalized this approach to a wider class of forest construction problems described by proper functions. Bar-Yehuda [BY98] gave a description of the Steiner Forest algorithm in terms of the Local Ratio Framework. We generalize the method of Bar-Yehuda and obtain a k-approximation algorithm for the Steiner Forest Problem in k-bounded hypergraphs. Our result was independently obtained by Takeshita, Fujito and Watanabe ([TFW99], in Japanese) A hypergraph $G = (V, E)$ is called *k-bounded* iff for all $e \in E$ $|e| \leq k$. We start with the problem formulation.

- **Steiner Forest Problem in k-bounded hypergraphs:**
  Given a k-bounded hypergraph $G = (V, E)$, edge costs $c: E \to \mathbb{R}_+$ and pairwise disjoint nonempty sets $S_1, \ldots, S_n$, find a set of edges $F \subseteq E$ such that for each $1 \leq i \leq n$ S_i is connected by F.

We can reformulate this problem in terms of Bar-Yehuda's framework: Let $X = E$, $\omega(x) = c(x)$ for $x \in E$ and $f(C) = 1$ iff for every $1 \leq i \leq n$ S_i is connected by C. Using Bar-Yehudas generic framework we get the following algorithm.

Algorithm $\mathcal{A}_3$:
(1) **if** $F := \{e \in F : \omega(e) = 0\}$ satisfies $f(F) = 1$
(2) **then** return F **else**
(3) Choose weight reduction δ.
(4) $F' := \{e \in E : (\omega - \delta)(e) = 0\}$
(5) **for all** $e \in F'$ contract e.
(6) Recursively apply $\mathcal{A}_3$ to the contracted instance
to obtain a solution F''. Let $F := F' \cup F''$.
(7) **while** there is $e \in F$ with $(\omega - \delta)(e) = 0, f(F \setminus \{e\}) = 1$
Let $F := F \setminus \{e\}$.
(8) **return** F.

It remains to specify the weight reduction δ: Let $d: E \to \mathbb{R}_+$ be defined by $d(e) = |e \cap (\bigcup_{i=1}^{n} S_i)|$ and $\delta(e) := d(e) \cdot \min\{w(f)/d(f) | f \in E, d(f) \neq 0\}$. We note that if d is r-minimal effective for some $r > 0$ then δ is also r-minimal effective. The next lemma shows that d is minimal k-effective.

Lemma 6.4.1 *Let $C \subseteq E$ be a d-minimal solution. Let C_S denote the set of hyperedges $e \in C$ with $e \cap S \neq \emptyset$. Then $d(C) \in \left[\sum_{i=1}^{n} |S_i|, k \cdot \sum_{i=1}^{n} |S_i|\right]$.*

Proof: Let C be a d-minimal solution. It suffices to show that for every connected component C' of C $d(C') \in [\,|S'|, k \cdot |S'|\,]$, where S' is the set of terminals of component C'. Let $H(C')$ denote the subhypergraph induced by edge set C'. For $s \in S'$ let $\deg(s)$ denote the number of edges from C' which contain s. By the choice of d we have $d(C') = \sum_{e \in C'} |e \cap S'| = \sum_{s \in S'} \deg(s)$. Since each terminal has to be connected by at least one edge, the lower bound follows. For the upper bound we build a rooted tree $T_{C'}$ whose vertices are the edges of C' as follows: Pick an edge $e \in C'$ as the root of $T_{C'}$. Since C' is a minimal solution, the hypergraph $H(C' \setminus \{e\})$ consists of connected components $C_1, \ldots, C_j$ for some $j \geq 2$. Iteratively build subtrees T_i with roots e_i for the sets C_i and connect each e_i to the root e. We show by induction on the depth of $T_{C'}$:

$$\sum_{e \in C'} |e \cap S'| \quad \leq \quad k \cdot (|S'| - 1). \tag{6.1}$$

If the depth of $T_{C'}$ is 0 then either $C' = \emptyset$ and $|S'| = 1$ or $C' = \{e\}$ and $|e \cap S'| = |S'| \leq k \cdot (|S'| - 1)$ (since $|S'| \geq 2$ and $k \geq 2$). For the induction step let $C_1, \ldots, C_j$ be the connected components of $C' \setminus \{e\}$ with terminal sets $S_1, \ldots, S_j$. By inductive assumption, for $1 \leq i \leq j$ we have $\sum_{e \in C_i} |e \cap S_i| \leq k \cdot (|S_i| - 1)$. Hence

$$\begin{aligned}
\sum_{e \in C'} |e \cap S'| \quad &\leq \quad k \cdot \left(\sum_{i=1}^{j} (|S_i| - 1) \right) + k && (6.2)\\
&= \quad k \cdot \Big(\sum |S_j| \; - (j-1) \Big) && (6.3)\\
&\leq \quad k \cdot \Big(\sum |S_j| \; - 1 \Big) = k \cdot \big(|S'| - 1 \big). && (6.4)
\end{aligned}$$

□

6.5 The Prize Collecting Steiner Tree Problem

In this section we consider the prize collecting variant of the Steiner Tree Problem:

- **The Prize Collecting Steiner Tree Problem (PCStP)**
  Given a graph $G = (V, E)$ with edge weights $c\colon E \to \mathbb{R}_+$, a set of terminals $S \subseteq V$ and a prize function $p\colon S \to \mathbb{R}_+$, find a tree $T \subseteq G$ connecting a subset $S' \subseteq S$ of minimum value $c(T) + p(S \setminus S')$.

This problem has application in the design of local access networks. Goemans and Williamson gave a 2-approximation algorithm based on the primal-dual method. The approach is based on fixing a root r as part of the tree, considering an LP formulation of this rooted version and running the algorithm for all possible choices of r. In this section we give a simplified presentation of this algorithm using the Local Ratio Framework. It is worth mentioning that this yields a rootless version of the algorithmic approach, thus improving running time. For an instance G, c, S, p of the **PCStP** we get the following formulation as a cover problem: Let $X = E \cup S$, $\omega\colon X \to \mathbb{R}_+$ with $\omega(e) = c(e)$ for $e \in E$ and $\omega(s) = p(s)$ for $s \in S$

Algorithm $\mathcal{A}_4$:
(1) **if** $C := \{x \in X : \omega(e) = 0\}$ satisfies $f(C) = 1$
(2) **then** return C **else**
(3) Choose weight reduction δ.
(4) $C' := \{x \in X : (\omega - \delta)(x) = 0\}$
(5) *Combinatorial Reduction:*
for all $e \in C' \cap E$ contract e.
for all $s \in C' \cap S$ remove s from S.
(6) Recursively apply $\mathcal{A}_3$ to the reduced instance
to obtain a solution C''. Let $C := C' \cup C''$.
(7) **while** there is $x \in C$ with $(\omega - \delta)(x) = 0, f(C \setminus \{x\}) = 1$
remove x from C
(8) **return** F.

We choose $\delta(x) := d(x) \cdot \min\left\{\frac{\omega(y)}{d(y)} : d(y) \neq 0\right\}$ with $d(e) = |e \cap S|$ for $e \in E$ and $d(s) = 1$ for $s \in S$. If for some $r > 0$ d is r-minimal effective then δ is minimal r-effective as well. The following lemma shows that d is minimal 2-effective.

Lemma 6.5.1 *For every d-minimal cover C $d(C) \in [|S|, 2(|S| - 1)]$.*

Proof: Let C be a d-minimal cover. For $s \in S$ we call $\mathrm{con}(s, C) = |\{e \in C | s \in e\}| + |C \cap \{s\}|$ the *contribution* of s to $d(C)$. Note that $d(C) = \sum_{s \in S} \mathrm{con}(s, C)$. For every terminal s at least one of the following must be true: 1. $s \in C$. 2. There is some edge $e \in C$ with $s \in e$. Hence $\mathrm{con}(s, C) \geq 1$ for every terminal $s \in S$ and hence $d(C) \geq |S|$. On the other hand let $S' = C \cap S$. Since C is d-minimal, S' is the set of terminals not being connected by $C \cap E$. Since $d(C \cap E) \in [\, |S \setminus S'|, 2(|S \setminus S'| - 1)\,]$ as in the Steiner Forest case, we conclude that $d(C) \in [\, |S|, 2(|S| - 1)\,]$. □

Corollary 6.5.1 *Algorithm $\mathcal{A}_4$ is a $2 \cdot \left(1 - \frac{1}{n}\right)$-approximation algorithm for the* **PCStP**.

We consider now the extension of the **PCStP** to k-bounded hypergraphs. For an instance $G = (V, E), c: E \to \mathbb{R}_+, S \subseteq V, p: S \to \mathbb{R}_+$ where G is a k-bounded hypergraph we define d essentially as above: $d(s) = 1$ for $s \in S$ and $d(e) = |e \cap S|$ for $e \in E$. Consider a d-minimal solution $C = E_C \cup S_C$ where $E_C \subseteq E$ and $S_C \subseteq S$. Then E_C connects $S \setminus S_C$. Applying lemma ... from section ... to E_C and the set $S \setminus S_C$ we conclude that $d(E_C) \in [\, |S \setminus S_C|, k \cdot |S \setminus S_C|\,]$ and therefore

$$d(C) \quad \in \quad \left[\, |S|, |S'| + k \cdot |S \setminus S_C|\,\right] \;\subseteq\; [\, |S|, k \cdot |S|\,]\,. \tag{6.5}$$

Hence d is minimal k-effective and algorithm $\mathcal{A}_4$ applied to k-bounded hypergraphs has approximation ratio k.

6.6 The Prize Collecting Steiner Pair Problem

In this section we consider another quite natural variant of the Prize Collecting Problem where prizes are given for (undirected) pairs instead of vertices:

- **The Prize Collecting Steiner Pair Problem**
  Given a graph $G = (V, E)$ with edge costs $c\colon E \to \mathbb{R}_+$, a set of undirected pairs $P = \{\{u_1, v_1\}, \ldots, \{u_n, v_n\}\}$ and a prize function $p\colon P \to \mathbb{R}_+$, find a set of edges $F \subseteq E$ and a subset $P' \subseteq P$ of minimum value $c(F) + p(P')$ such that for every pair $\{u_i, v_i\} \in P \setminus P'$ there is a path of F-edges connecting u_i and v_i..

If the pairs are pairwise disjoint, this is a special case of the Prize Collecting Steiner Forest Problem discussed in the previous section. The main difficulty in the general case is that in a graph with n vertices there might exist $\Theta(n^2)$ pairs while $O(n)$ edges suffice to connect the whole vertex set. We illustrate this fact by giving a straight forward formulation in terms of the Local Ratio Technique and arguing that such a formulation only gives a linear approximation ratio.
Let $X = E \cup P$ with weight function ω given by $\omega(e) = c(e), e \in E$ and $\omega(\pi) = p(\pi)$ for $\pi \in P$. For $E' \subseteq E$ and $P' \subseteq P$ let $f(E' \cup P') = 1$ iff E' connects every pair from $P \setminus P'$. Using a weight reduction defined by $d(e) = |e \cap S|$, $S = \{u_1, v_1, \ldots, u_n, v_n\}$ and $d(\pi) = \Omega(1)$ for $\pi \in P$, we can only prove a linear approximation ratio: A set E' of edges connecting S forms a d-minimal solution with $d(E') = O(|S|)$ while the set P of all pairs is d-minimal and $d(P) = \Omega\left(|S|^2\right)$ might be the case.
In order to avoid such problems we replace each undirected pair $\pi = \{u, v\}$ from P by the two directed pairs (u, v) and (v, u), each of prize equal to half the prize of π and require that u and v are connected by egdes of the solution or the prize for both (u, v) and (v, u) is paid.

More formally: Let $\tilde{P} := \left\{(x, y) \in V^2 : \{u, v\} \in P\right\}$ and $\tilde{p}((u, v)) = \tilde{p}((v, u)) = p(\{u, v\})/2$ for $\{u, v\} \in P$. Let $\tilde{X} := E \cup \tilde{P}$ with weight function ω defined by $\omega(e) = c(e)$ for $e \in E$ and $\omega(\pi) = \tilde{p}(\pi)$ for $\pi \in \tilde{P}$. For a set $C = E' \cup P' \subseteq \tilde{X}$ with $E' \subseteq E$ and $P' \subseteq \tilde{P}$ we let $f(C) = 1$ iff for every pair $\pi = \{u, v\} \in P$, at least one of the following two conditions holds:

1. E' contains a path connecting u and v.
2. $(u, v) \in P'$ and $(v, u) \in P'$.

Thus we have defined a reduction from the **PCStPP** to the following somewhat more artificial covering problem:

- **Prize Collecting Directed Pair Problem (PCDPP):**
  Given an undirected graph $G = (V, E)$ with edge weights $c\colon E \to \mathbb{R}_+$ and a set $P = \{(u_1, v_1), \ldots, (u_n, v_n)\} \subseteq V^2$ of directed pairs of vertices with prize function $p\colon P \to \mathbb{R}_+$, find a set of edges $F \subseteq E$ and a set of pairs $P' \subseteq P$ of minimum value $c(F) + p(P')$ such that for every pair $(u_i, v_i) \in P \setminus P'$ there is a path of edges from F connecting u and v.

Lemma 6.6.1 *The* **PCStPP** *and the* **PCDPP** *are equivalent with respect to L-reductions with parameters* $\alpha = \beta = 1$ *in both directions.*

Proof: 1. PCStPP$\leq_L$ **PCDPP:** For an instance $I = (G, c, P, p)$ of the **PCStPP** let $f(I) = (G, c, \tilde{P}, \tilde{P})$ where $\tilde{P}$ and $\tilde{p}$ are as described above. For a solution $F \subseteq E, \tilde{P}' \subseteq \tilde{P}$ to $f(I)$ let $C := F \cup P'$ with $P' := \left\{\{u, v\} \in P : (u, v), (v, u) \in \tilde{P}'\right\}$.
2. PCDPP$\leq_L$ **PCStPP:** If at least one of the pairs $(u, v), (v, u)$ is in P, introduce an undirected pair $\{u, v\}$ with prize equal to $p(u, v) + p(v, u)$ if both $(u, v), (v, u)$ exist and $p(u, v)$ if only (u, v) exists. Given a solution $F \cup \tilde{P}'$ to the **PCStPP**-instance obtained in this way, for each pair $\{u, v\} \in \tilde{P}'$ take all directed pairs with vertices u, v that exists in P to obtain a solution to the **PCDPP**-instance. □

Theorem 35 *There is a polynomial time approximation algorithm for the* **PCDPP** *with approximation ratio* 3*.*

In order to prove theorem 35 we consider the straight forward formulation of the **PCDPP** as a cover problem: For an instance $I = (G, c, P, p)$ of the former let $X := E \cup P$ with weights $\omega(e) = c(e), e \in E$ and $\omega(\pi) = p(\pi), \pi \in P$. For $C \subseteq X$ let $f(C) = 1$ iff C forms a solution to instance I. Applying the generic framework of Bar-Yehuda we obtain the following recursive algorithm:

Algorithm $\mathcal{A}_4$:
Input: instance (X, ω, f) of the cover problem formulation of **PCDPP**
with $X = E \cup P$
Output: $E' \cup P'$ with $E' \subseteq E, P' \subseteq P$ and $f(E' \cup P') = 1$
(1) **if** $C = \{x \in X | \omega(x) = 0\}$ forms a cover **then return** C.
(2) Choose weight reduction δ.
Combinatorial Reduction:
(3) **For** every edge $e \in E$ such that $(\omega - \delta)(e) = 0$
Contract (e) into a single point v_e.
(4) **For** every $\pi \in P$ such that $(\omega - \delta)(\pi) = 0$
Remove (π) from P.
(5) Let $\left(\bar{X}, \bar{\omega}, \bar{f}\right)$ be the reduced instance.
$C := \mathcal{A}_4\left(\bar{X}, \bar{\omega}, \bar{f}\right)$
(6) Add to C all contracted edges and all removed pairs.
(7) **While** there is some $x \in C$ with $\delta(x) > 0$ and $f(C \setminus \{x\}) = 1$
Choose such x and remove it from C.
(8) **return** C.

It remains to specify the weight reduction δ. As before we choose $\delta(x) = \epsilon \cdot d(x)$ for a function $d: X \to \mathbb{R}_+$ and $\epsilon = \min\{\omega(x)/d(x) | d(x) \neq 0\}$. We call $V_a := \{v \in V | v$ appears in at least one pair$\}$ the set of active vertices. For an active vertex v we define the outdegree of v as $\deg(v) := |\{\pi \in P : \pi = (v, u)$ for some $u \in V\}|$. We choose

$$\begin{aligned} d(e) &:= |e \cap V_a| && \text{for } e \in E \\ d((u, v)) &:= \tfrac{1}{\deg(u)} && \text{for } (u, v) \in P \end{aligned}$$

Lemma 6.6.2 *Every d-minimal cover $C \subseteq X$ satisfies $d(C) \leq 3 \cdot OPT(d)$.*

Proof: Consider a d-minimal cover $C = E' \cup P'$ with $E' \subseteq E$ and $P' \subseteq P$. For $v \in V_a$ the amount v adds to the total cost $d(C)$ is given by

$$\mathrm{add}(C,v) \quad = \quad \underbrace{\sum_{v \in e \in C} 1}_{=:\mathrm{add}_E(C,v)} \quad + \quad \underbrace{\sum_{u:(u,v)\in C} d((u,v))}_{=:\mathrm{add}_P(C,v)} \tag{6.6}$$

Since C is a cover, $\mathrm{add}(C,v) \geq 1$ for every $v \in V_a$. Furthermore

$$\sum_{v \in V_a} \mathrm{add}_E(C,v) \quad \leq \quad 2 \cdot (|V_a| - 1) \tag{6.7}$$

where the maximum appears when E' forms a spanning tree for V_a and

$$\sum_{v \in V_a} \mathrm{add}_P(C,v) \quad \leq \quad |V_a| \tag{6.8}$$

since by the choice of d $\mathrm{add}_P(C,v) \leq 1$ for every $v \in V_a$. Since

$$\begin{aligned} d(C) \quad &= \quad \sum_{v \in V_a} \mathrm{add}(C,v) = \sum_{v \in V_a} (\mathrm{add}_E(C,v) + \mathrm{add}_P(C,v) \\ &\leq \quad 2 \cdot (|V_a| - 1) \; + \; |V_a| = 3 \cdot |V_a| \cdot \left(1 - \frac{2}{3 \cdot |V_a|}\right) \\ &\leq \quad 3 \cdot |V_a| \cdot \left(1 - \frac{2}{3n}\right). \end{aligned} \tag{6.9}$$

We conclude that for every d-minimal cover C

$$d(C) \quad \in \quad [\,|V_a|, 3 \cdot |V_a|\,] \tag{6.10}$$

and therefore d is 3-minimal effective. □

Proof of Theorem 35: Directly from Lemma 6.3.2 and Lemma 6.6.2. □

Remark: We give an example that shows the tightness of (6.10) in the following sense: For every $\epsilon > 0$ there are instances (X, ω, f) with d-minimal covers C_1, C_2 such that $d(C_1) = |V_a|$ and $d(C_2) \geq (1 - \epsilon) \cdot 3 \cdot |V_a|$. Let instance $I(n,m)$ consist of n vertex-disjoint paths $P_1, \ldots, P_n$ each consisting of m vertices $v_{i,1}, \ldots, v_{i,m}$ and edges $\{v_{i,j}, v_{i,j+1}\}, j = 1, \ldots, m-1$. The set of pairs P contains pair $\pi_i = (v_{i,1}, v_{i,m})$ for each path P_i, furthermore all pairs $(v_{i,j}, v_{k,l})$ for $i \neq k$. Hence $I(n,m)$ has $|V_a| = n \cdot m$ active vertices and $n + \frac{n(n-1)}{2} \cdot m^2$ pairs. The set C_1 of all pairs is d-minimal and has cost $d(C_1) = |V_a|$ while the edges of the n paths together with all the pairs between vertices of different paths form a d-minimal set C_2 of cost

$$\begin{aligned} d(C_2) = &\underbrace{n \cdot 2 \cdot (m-1)}_{\text{edge costs}} + \underbrace{n(m-2)}_{\text{pairs starting from inner points}} + \underbrace{2n \cdot \frac{(n-1)m}{(n-1)m+1}}_{\text{pairs starting from endpoints}} \\ &\geq 3 \cdot n \cdot m \cdot \left(1 - \tfrac{4}{3m}\right), \end{aligned}$$

hence choosing $m \geq 4/(3\epsilon)$ is sufficient.

6.7 The Prize Collecting Steiner Forest Problem

In this section we consider a natural generalization of the Prize Collecting Steiner Tree Problem to the case of possibly several pairwise disjoint terminal sets with prizes. This problem also generalizes the Steiner Forest Problem in graphs.

- **Prize Collecting Steiner Forest Problem (PCStF)**
  Given a graph $G = (V, E)$ with edge weights $c\colon E \to \mathbb{R}_+$ and pairwise disjoint nonempty terminal sets $S_1, \ldots, S_n \subseteq V$ with prizes $p\colon \bigcup_{i=1}^n S_i \to \mathbb{R}_+$, construct a set of edges $F \subseteq E$ and a set of terminals $S' \subseteq S := \bigcup_{i=1}^n S_i$ of minimum value $c(F) + p(S')$ such that for every $i \in \{1, \ldots, n\}$ F connects $S_i \setminus S'$.

If we apply Bar-Yehuda's covering algorithm to this problem the following might happen: Although initially the terminal sets were pairwise disjoint, terminals of different sets S_i, S_j might be contracted into the same supernode. Hence in order to apply the recursive algorithm $\mathcal{A}\ldots$ we have to consider the following generalization of the **PCStF**:

- **Multi-Prize Problem (MPP):**
  Given a graph $G = (V, E)$ with edge weights $c\colon E \to \mathbb{R}_+$ and prize functions $p_1, \ldots, p_n\colon V \to \mathbb{R}_+$. Let $S_i := \{v \in V | p_i(v) \neq 0\}, 1 \leq i \leq n$. Find a set of edges F and for each $1 \leq i \leq n$ a subset $S_i' \subseteq S_i$ of minimum $c(F) + \sum_{i=1}^n p_i(S_i')$ such that for every i either $S_i' = S_i$ or $|S_i \setminus S_i'| \geq 2$ and $S_i \setminus S_i'$ is connected by F.

The Multi-Prize Problem is also a generalization of the Prize Collecting Directed Pair Problem (PCDPP): For an instance of the latter with pairs (x_i, y_i) of prizes p_i $(1 \leq i \leq n)$ let $p_i'(x_i) = p_i(y_i) = p_i/2$ and $p_i'(z) = 0$ for $z \in V \setminus \{x_i, y_i\}$.

Theorem 36 *There is a polynomial time approximation algorithm for the* **MPP** *with approximation ratio* 3.

Proof: We let $X = E \cup \{(v, i) | 1 \leq i \leq n, v \in S_i\}$ with costs $\omega(e) = c(e)$ for $e \in E$ and $\omega((v, i)) = p_i(v)$. The definition of f is obvious. For $v \in V$ let $\deg(v)$ denote the number of sets S_i containing vertex v. We choose a function $d\colon X \to \mathbb{R}_+$ by

$$\begin{aligned} d(e) &= |e \cap S|, && e \in E \\ d((v,i)) &= 1/\deg(v), && 1 \leq i \leq n, v \in S_i \end{aligned}$$

Again each terminal pays at least one (at least one edge or prizes for all terminal sets it belongs to), and taking a spanning tree for S plus paying all prizes yields d-cost $2\,(|S| - 1) + |S|$, hence d is 3-minimal effective. Let $\delta\colon X \to \mathbb{R}_+$ be defined by

$$\delta(x) := d(x) \cdot \min \left\{ \frac{\mathrm{c}(e)}{d(e)}, d(e) \neq 0, \omega((v,i)), (v,i) \in X \right\}.$$

The problem $(X, f, \omega - \delta)$ is then combinatorially reduced as follows: First all pairs (v, i) with $(\omega - \delta)((v, i)) = 0$ are removed from X. Then every edge $e \in X$ with $(\omega - \delta)(e) = 0$ is contracted as follows: $e = \{u, v\}$ is contracted into a new vertex v_e in the underlying graph, and e and all existing pairs $(u, i), (v, i)$ are removed from X. For each i with $S_i \cap e \neq \emptyset$ a pair (v_e, i) of cost $\sum_{x \in S_i \cap e}(\omega - \delta)(x)$ is added to X. Obviously if C' is a solution to the reduced problem then the union of C, the set of pairs (v, i) with $(\omega - \delta)((v, i)) = 0$ and the set of edges e with $(\omega - \delta)(e) = 0$ form a solution for (X, f, ω).
Hence using Bar-Yehuda's framework with weight reduction δ and the combinatorial reduction as defined above, the theorem follows. □

Theorem 37 *There is a polynomial time approximation algorithm for the* **MPP** *in k-bounded hypergraphs with approximation ratio $k + 1$*

Proof: Define $d(e) = |e \cap S|, e \in E$ and $d((v, i)) = 1/\deg(v)$ as before, then $d(C) \in [\,|S|, k|S| + |S|\,]$. □

Chapter 7

The Steiner Network Problem

7.1 Introduction

In this section we consider an important generalization of the Steiner Forest Problem, the **Steiner Network Problem**. Recall that the Steiner Forest Problem can be formulated using a $0-1-$valued requirement function r, where $r(u,v) = 1$ iff u and v belong to the same terminal set S_i. Let us now assume that connection requirements are non-negative integer values, where a connection requirement $r(u,v) \in \mathbb{Z}_+$ means that in the solution network each $u-v-$cut is supposed to contain at least $r(u,v)$ edges, and additionally we are allowed to pick multiple edges, i.e. each edge e has a capacity $u(e) \in \mathbb{Z}_+$ and we are allowed to pick e up to $u(e)$ times (paying its cost $c(e)$ as many times as we pick it). Hence a solution now is a function $x\colon E \to \mathbb{Z}_+$, and its cost is $\sum_e c(e) \cdot x(e)$. Here is a formal definition of the Steiner Network Problem:

Steiner Network Problem

Given: Graph $G = (V,E)$ with edge costs $c\colon E \to \mathbb{R}_+$, capacities $u\colon E \to \mathbb{Z}_+$, requirement function $r\colon V \times V \to \mathbb{Z}_+$

Solution: mapping $x\colon E \to \mathbb{Z}_+$ with $0 \le x(e) \le u(e)$ for all $e \in E$ such that for all $u,v \in V$: for each $u-v-$cut $U, V \setminus U$ it holds $\sum_{e \in E(U,V\setminus U)} x(e) \ge r(u,v)$

Cost: $\sum_{e \in E} c(e) \cdot x(e)$

It was for some time an open question to obtain a constant factor approximation algorithm for this problem. Goemans et al. [GGP$^+$94] obtained a logarithmic approximation ratio based on a direct primal dual approach adapted from the Steiner Forest Case. It was finally Kamal Jain [Jai98] who gave a 2-approximation algorithm. His approach is based on iterative rounding based on an LP relaxation of the problem. See [JMVW99] for further work on the topic.

Unfortunately, the LP formulation of the bf Steiner Network Problem turns out

to be exponential in the size of the input, hence Jain's algorithm relys on using the Ellipsoid Method for solving linear programs, resulting in very large running times even in practice. Hence it would be desirable to obtain a combinatorial algorithm for the problem which is not based on the Ellipsoid Method, hopefully yielding better running times.

We are so far not successful in this attempt. Nevertheless we give combinatorial algorithms for two special cases: 1. for the uncapacitated version (each edge might be used to arbitrary capacity) with uniform requirements (section 7.3) and for the Prize Collecting version of this problem in section 7.4 In both cases, although the optimum solution is in general not a tree, the primal dual approach works fine.

7.2 Jain's 2-Approximation Algorithm

In this section we describe Jain's 2-Approximation Algorithm [Jai98] for the Steiner Network Problem. The algorithm uses the method of iterated rounding based on a Linear Programming Relaxation of the problem. Although the size of this LP-relaxation is exponential in the input size, the LP can be solved in polynomial time using the equivalence of separation and optimization, a well-known and central paradigm in the theory of Linear Programming. The crucial property of the LP-Relaxation is that for each extreme-point solution x there exists at least one edge e in G such that $x_e \geq 1/2$. Jain's algorithm iteratively generates such an extreme point solution x, takes all edges e with $x_e \geq 1/2$ as part of the solution and reduces the problem.

The rest of the section is organized as follows: First we give a reformulation of the problems in terms of a set requirement function $f = f_r$and state the important properties of this function. Next we give the LP-Relaxation of the Steiner Network Problem based on this reformulation, then in Theorem 38 we formulate the crucial property of this LP, then we give a pseudo-code description of Jain's algorithm, finally we describe the main steps in the proof of Theorem 38.

Reformulation. Consider an arbitrary subset $S \subseteq V$ of the vertex set of G. If $\emptyset \neq S \neq V$, then each feasible solution must cross the cut defined by S at least $\max\{r(u,v) : u \in S, v \in V \setminus S\}$ times, i.e. contain such many edges from $\delta(S) := \{e \in E : e \cap S \neq \emptyset \neq e \setminus S\}$. Hence we define the *set requirement function* $f = f_r \colon P(V) \to \mathbb{Z}_+$ BY

$$f(S) \ := \ \max\{r(u,v) : u \in S, v \in V \setminus S\}.$$

Obviously a function $x \colon E \to \mathbb{Z}_+$ is a solution to the Steiner Network problem iff it satisfies all the inequalities

$$x(\delta(S)) \ \geq \ f(S), \quad \emptyset \neq S \subset V.$$

The most important property of the function f is its *weak super-modularity*.

Definition 37 (Sub-, Super- and Weak Super-Modularity) *Let $g: P(V) \to \mathbb{R}$ be a set function.*

(a) g is called submodular *iff for all subsets A, B of V the following inequalities hold:*

$$f(A) + f(B) \leq f(A \cap B) + f(A \cup B),$$
$$f(A) + f(B) \leq f(A \setminus B) + f(B \setminus A).$$

(b) g is called supermodular *iff $-g$ is submodular, i.e. if for all subsets A, B of V the following inequalities hold:*

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B),$$
$$f(A) + f(B) \geq f(A \setminus B) + f(B \setminus A).$$

(c) g is called weakly supermodular *if for all subsets A, B of V at least one of the inequalities*

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B),$$
$$f(A) + f(B) \geq f(A \setminus B) + f(B \setminus A)$$

holds.

Lemma 7.2.1 *The set requirement function f as defined above is weakly supermodular.*

LP-Formulation of the Steiner Network Problem:

$$\begin{array}{lll} LP(G,c,u,f): & \min \sum_{e\in E} c(e)\cdot x_e \text{ subject to} & \\ & \sum_{e\in\delta(S)} x_e \;\geq\; f(S), & S \subseteq V \\ & 0 \leq x_e \leq u(e), & e \in E \end{array}$$

Definition 38 (Extreme Point Solution)
Given a linear program $\min\{cx|Ax \leq b\}$, a feasible solution x is called extreme point solution *iff it is not a convex combination of two other feasible solutions.*

A well-known result from the Linear Programming theor states that a solution $x \in \mathbb{R}^n$ is an extreme point solution for $\min\{cx|Ax \leq b\}$ iff there are n linear independent inequalities $a_i x \leq b_i$ from $Ax \leq b$ such that each of them is tight (i.e. satisfied with equality).

Theorem 38 (Crucial Fact)
Let x be an extreme point solution to the above LP. Then there exists an edge e such that $x(e) \geq 1/2$.

For Jain's algorithm it is important to be able to generate extreme point solutions in polynomial time. This can be done for the special case of set requirement functions $f: P(V) \to \mathbb{Z}_+$ comming from requirement functions $r: V \times V \to \mathbb{Z}$. The paradigm used here is one of the most important results from Linear Programming Theory, known as

the equivalence of optimization and separation.

Optimization versus Separation. Consider a linear program $\min\{cx|Ax \leq b\}$ with n variables and m linear inequalities, where m might be exponential in n (as in the above LP formulation of the Steiner Network Problem). In order to make use of the Linear Programming Framework, one is interested in solving the LP. But here the input size is roughly spoken the size of the network, hence in polynomial time we even cannot **write down** the complete LP.

Jain's Algorithm. We are now ready to describe Jain's algorithm. Given an instance $G = (V, E), c\colon E \to \mathbb{R}_+, f\colon P(V) \to \mathbb{Z}_+$ of the Steiner Network Problem it constructs a solution $X\colon E \to \mathbb{Z}_+$.

Jain's Algorithm $\mathcal{A}_{Jain}$:
Input: $G = (V, E),\ c\colon E \to \mathbb{R}_+,\ f\colon P(V) \to \mathbb{Z}_+$
Output: $X\colon E \to \mathbb{Z}_+$
Initialization:
  Let $X(e) := 0$ for all $e \in E$. *(set of picked edges)*
  Let $f' := f$. *(residual requirement function)*
While there exists $S \subseteq V$ with $f'(S) \neq 0$ **do**
  Find an extreme point solution x for $LP(G, c, f')$.
  For each $e \in E$
    If $x(e) \geq 1/2$ **then**
      Add $\lceil x(e) \rceil$ copies of e to F.
      Let $u(e) := u(e) - \lceil x(e) \rceil$.
    Update f'.

Lemma 7.2.2 *Jain's algorithm can be implemented to run in polynomial time (in the size of graph G).*

Proof: It is sufficient to give a separation procedure for the linear program $LP(G, c, f)$. This can be done using standard max-flow techniques: For a given vector x, violated inequalities exist iff in the graph G with edge capacities $x_e, e \in E$ there exists a pair of vertices u, v such that the maximum $u - v-$flow has value less than $r(u, v)$. In such a case a min-cut corresponds to a violated inequality. □

7.3 The Uniform Uncapacitated Case

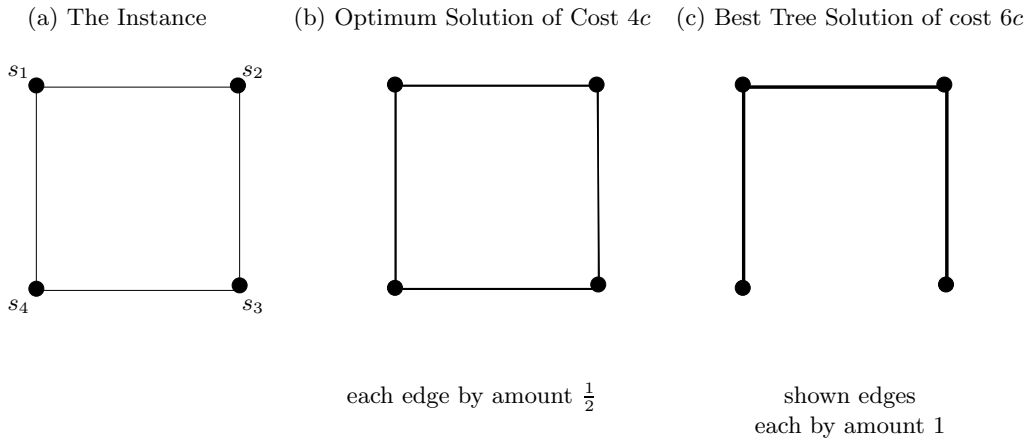
In this section we consider the following special case of the Generalized Steiner Network Problem where we have unbounded capacities on the edges and uniform requirements:

Uniform Uncapacitated Steiner Network Problem

Instance:	finite metric space (V, c), requirement function $r\colon P_2(V) \to \{0, r_1\}$ for some $r_1 > 0$
Solution:	$x\colon P_2(V) \to \mathbb{Z}_+$ satisfying the requirements
Cost:	$c(x) := \sum_{\{u,v\} \in P_2(V)} x(\{u,v\}) \cdot c(u,v)$

In general the optimum solution to the Uniform Uncapacitated Steiner Network Problem is not a tree. Consider the following example: $V = \{s_0, \ldots, s_3\}$ with $c(s_i, s_{(i+1) \bmod 4}) = c > 0$ and requirement $r(s_0, s_2) = r(s_1, s_3) = r_1 = 2$. Then the optimum cost equals $4 \cdot c$ while the optimum tree solution has cost $3 \cdot 2 \cdot c = 6 \cdot c$.

Figure 7.1: **The Optimum is not a tree in general**



Nevertheless in this special case the Primal-Dual approach provides a combinatorial polytime 2-approximation algorithm. The reason is that due to infinite capacitis of the edges we can always produce a solution that is basically a tree, where each edge of the tree is taken r_1 times, r_1 being the uniform non-zero requirement. Hence the averaging argument in the standard analysis method for growing-ball algorithms works. Let us now give the algorithm and analysis.

Algorithm $\mathcal{A}_U$
Input: finite metric space (V, c), requirement function $r: V \times V \to \{0, r_1\}$ for some $r_1 \in \mathbb{Z}_+$
Output: $x: V \times V \to \mathbb{Z}_+$ satisfying all requirements
Initialization:
$\mathcal{C} := \{\{v\} | v \in V\}$ set of components
$\mathcal{C}_a := \{\{v\} | v \in V, \max_{u \in V} r(v,u) \neq 0\}$ set of active components
$E := P_2(V)$ (consider the complete graph induced by (V, c))
$x(e) := 0$ for all $e \in E$ (finally building the solution)
While $\mathcal{C}_a \neq \emptyset$ **do**
$d(e) := |e \cap \mathcal{C}_a|$ for all $e \in E_{act}$
$\Delta := \min \left\{ \frac{c(e)}{d(e)} : d(e) > 0 \right\}$
For all $e \in E$ **do**
$c(e) := c(e) - \Delta \cdot d(e)$
For all $e \in E$ with $c(e) = 0$ **do**
$x(e) := r_1$.
Contract e.
Cleaning Step:
For all $e \in E$ with $x(e) \neq 0$ **do**
If x' defined by

$$x'(f) := \begin{cases} 0, & f = e, \\ x(f) & \text{otherwise} \end{cases}$$

is still feasible then $x := x'$.
Return x.

Theorem 39
Algorithm $\mathcal{A}_U$ provides Approximation Ratio 2 for the Uncapacitated Uniform Steiner Network Problem.

Proof: In order to argue by local-ratio analysis, we need to show that

$$OPT(I) \ \geq \ \sum_j OPT(I_j),$$

where I denotes an instance of the problem, j runs through iterations of the while-loop of the algorithm and I_j denotes the current local-cost instance in the j-th execution of the while-loop.

In order to do so, it suffices to show that in each single iteration with initial costs c on the edges,

$$OPT(c) \ \geq \ OPT(c - \Delta \cdot d) \ + \ OPT(\Delta \cdot d).$$

Assume x is an optimum solution to instance I, hence $x: E \to \mathbb{Z}_+$. Let

$$E_x \ := \ \{e \in E | x(e) > 0\}$$

be the set of edges that contribute to solution x. We partition E_x into

$$\begin{aligned} E_1 &:= \{e \in E_x | c(e) = \Delta \cdot d(e)\}, \\ E_2 &:= \{e \in E_x | c(e) > \Delta \cdot d(e) > 0\}, \\ E_3 &:= \{e \in E_x | d(e) = 0\}. \end{aligned}$$

Then

$$\begin{aligned} \text{cost}(x) &= \sum_{e \in E_1} \Delta \cdot d(e) \cdot x(e) \\ &\quad + \sum_{e \in E_2} (\Delta \cdot d(e) \ + \ (c(e) - \Delta \cdot d(e))) \cdot x(e) \\ &\quad + \sum_{e \in E_3} c(e) \cdot x(e) \\ &= \sum_{e \in E_1 \cup E_2} \Delta \cdot d(e) \cdot x(e) \\ &\quad + \sum_{e \in E_2 \cup E_3} (c(e) - \Delta \cdot d(e)) \cdot x(e) \\ &= \sum_{e \in E} \Delta \cdot d(e) \cdot x(e) \quad + \quad \sum_{e \in E} (c(e) - \Delta \cdot d(e)) \cdot x(e) \\ &\geq OPT(V, \Delta \cdot d, r) \quad + \quad OPT(V, c - \Delta \cdot d, r) \end{aligned}$$

since x is feasible with respect to both cost functions. Now we observe that in the j-th run through the while-loop we have a lower bound of

$$r_1 \cdot \Delta_j \text{ times number of active components}$$

(the value of Δ in that loop), and due to the fact that our solution will be a forest, we pay on average $2 \cdot \Delta_j \cdot r_1$ per active component (namely $2 \cdot r_1 \cdot \Delta_j \cdot (|\mathcal{C}_a| - 1)$ in total). This completes the proof. □

7.4 The Prize-Collecting Uniform Uncapacitated Case

In this section we extend the results from the last section to the **Prize-Collecting Uniform Uncapacitated Steiner Network Problem**, where additionally to the requirement function we have a prize function for the requirements. Here is the precise definition:

Prize-Collecting Uniform Uncapacitated Steiner Network Problem

Instance: finite metric space (V, c),
requirement function $r\colon P_2(V) \to \{0, r_1\}$ for some integer $r_1 > 0$,
prize function $p\colon P_2(V) \to \mathbb{Q}_{\geq 0}$ with $p(a,b) > 0$ implying $r(a,b) > 0$
(i.e. we have prizes only for non-zero requirements)

Solution: $x\colon P_2(V) \to \mathbb{Z}_+$

Cost: $\sum_{\{u,v\} \in P_2(V)} x(\{u,v\}) \cdot c(u,v) \; + \sum_{\{u,v\} \in P_2(V)} \max\{(r - r_x)(u,v), 0\} \cdot p(u,v)$,
where r_x is defined by

$$r_x(u,v) := \min \left\{ x(U, V \setminus U) = \sum_{a \in U, b \notin U} x(a,b) | u \in U, v \notin U \right\}$$

Let us now formulate our algorithm, which again is an application of the primal dual approach.

Algorithm $\mathcal{A}_{UP}$
Input: finite metric space (V, c), requirement function $r\colon V \times V \to \{0, r_1\}$ with $r_1 > 0$,
prize function $p\colon P_2(V) \to \mathbb{Q}_{\geq 0}$ being nonzero only on pairs with
non-vanishing requirement (i.e. $r(u,v) = 0 \;\Rightarrow\; p(u,v) = 0$)
Output: $x\colon V \times V \to \mathbb{Z}_+$
Initialization:
$\mathcal{C} := \{\{v\} | v \in V\}$ set of components
$\mathcal{C}_a := \{\{v\} | v \in V, \max_{u \in V} r(v,u) \neq 0\}$ set of active components
$E := P_2(V)$ (consider the complete graph induced by (V, c))
$x(e) := 0$ for all $e \in E$ (finally building the solution)
While $\mathcal{C}_a \neq \emptyset$ **do**
/⋆ Find minimal weight reduction. ⋆/
$d(e) := |e \cap \mathcal{C}_a|$ for all $e \in E_{act}$
$\Delta := \min \left\{ \left\{ \frac{c(e)}{d(e)} : d(e) > 0 \right\}, \min\{p(C) | C \in \mathcal{C}_a\} \right\}$
/⋆ Reduce Edge Costs and Prizes of Active Components. ⋆/
For all $e \in E$ **do** $c(e) := c(e) - \Delta \cdot d(e)$
For all $C \in \mathcal{C}_a$ **do** $p(C) := p(C) - \Delta$.
/⋆ Collect Zero-Cost Edges and Components.. ⋆/
For all $e \in E$ with $c(e) = 0$ **do** { $x(e) := r_1$. Contract e. }
For all $C \in \mathcal{C}_a$ with $p(C) = 0$ **do** $\mathcal{C}_a := \mathcal{C}_a \setminus \{C\}$.
Cleaning Step:
For all $e \in E$ with $x(e) \neq 0$ **do**
If x' defined by

$$x'(f) := \begin{cases} 0, & f = e, \\ x(f) & \text{otherwise} \end{cases}$$

is still feasible then $x := x'$.
Return x.

Theorem 40 *The algorithm described above has approximation ratio* 3 *for the Prize Collecting Uncapacitated Steiner Network Problem.*

Sketch of Proof: Analog to the previous section, but here in each local step the following might happen: We have edges being fully paid and take them, but they reduce the prize only by a very small amount, hence the lower bound is 1 and the upper bound is -pessimistically - 3: For each component, on average we have connection cost 2 per round but still have to pay prize amount of 1. □

We leave as an open problem the question for a completely combinatorial approximation algorithm for the **Steiner Network Problem**, as well as obtaining approximation ratios better than 2.

Chapter 8

Steiner Problems in Directed Graphs

8.1 Introduction

In this section we consider the directed version of the Steiner Tree Problem, known as the **Steiner Arborescence Problem** (**StAP**). In this problem we are given a directed graph G with edge weigts, a root r and a set of sinks (*terminals*) S in G. The task is to construct a minimum cost arborescence T rooted at r in G such that S is contained in T. As usual the cost of a subgraph of G here means the sum of the costs of edges.

Although the formulation of the Steiner Arborescence Problem is quite similar to that of the Steiner Tree Problem, its complexity status is quite different. The **Set Cover Problem** can be basically seen as a special case of the Steiner Arborescence Problem, hence Feige's hardness result for Set Cover [Fei98] implies a logarithmic lower bound on approximability of the **StAP**. On the other hand Zelikovsky [Zel97] gives an approximation scheme $A_i, i \in \mathbb{N}$ for the **StAP**, where for each i A_i is a polynomial time approximation algorithm with ratio $O(|S|^{1/i})$ for the **StAP**. The approach is based on the idea of *level-restricted trees*. While Zelikovsky's approach only works for the case of acyclic direted graphs, Charikar et al. citecharikar98approximation were able to generalize it to the general directed case, obtaining the same approximation ratio. Furthermore they considered a directed variant of the *Steiner Forest Problem* and gave a polynomial time approximation algorithm for this problem with polynomial approximation ratio.

In this chapter we consider directed variants of the following two problems: The **Directed Zero Skew Tree Problem** and the **Direchted Weighted Path Tree Problem**. Both problems are strongly motivated by applications in VLSI design, where trees have to be constructed in order to satisfy specific physical constraints. Assume a signal has to be sent from a source to a given set of sinks in an electrical network, e.g. a computer hardware. Since computers are synchronized by a clock, signals ought arrive at the sinks within a certain time window or at a specific time

point. In a very rough first approximation, signal running times can be seen as being linear in the length of the path in the network they are taking. Hence we are now concerned with path-length constraints in a network layout problem. Both problems mentioned above take a step in order to deal with this task. We will now give informal problem descriptions for the undirected versions of both of these problems, which have already been considered to some extend in the literature.

In the **Zero Skew Tree Problem** we are give an undirected edge weighted graph or equivalently a finite metric space (V, c). (it will turn out in subsequent section that both versions are equivalent). Furthermore we are given a vertex r in V, called the **root**, as well as a set $S \subseteq V$ of **sinks**. The task is to construct a tree T in (V, c) connecting the root to the sinks such that T has **zero delay**, i.e. for each sink $s \in S$ the length of the path from r to s in T is the same. Since in this form a solution does not necessarily exist, we are allowed to make use of **delay segments**, which we can place on the vertices in V. The cot of a segment yielding delay d is precisely d, and the total cost is the length of the tree plus the cost of all delay segments being used. We will give a formal problem formulation below.

There is a huge literature on heuristic approaches and computational experiences concerning the Zero Skew Tree Problem and its variants. We are only aware of two papers in which approximation issues of the problem are concerned: Charikar et al. [CKK$^+$99] give a polynomial time approximation algorithm with ratio $2e \approx 5.44$ for the **Zero Skew Tree Problem**. Zelikovsky and Mandoiu [ZM01] improve om this result, obtaining a 4-approximation algorithm for the **Zero Skew Tree Problem** and a 14-approximation algorithm for the **Bounded Skew Tree Problem**, where instead of zero skew the signal arrival times are allowed to differ by a prespecified bound. Both algorithms are conceptually much easier than the approach of Charikar et al. [CKK$^+$99].

8.1.1 Level-Restricted Trees

The notions and results in this subsection are taken from [Zel97]. We have already discussed the use of **restricted trees** for approximating the **Steiner Tree Problem**. There the restriction was on the maximum size of **full components**, a concept all Steiner Tree approcximation algorithms we are aware of are based on. The analog tool in the setting of directed Steiner Tree Problems are **level-restricted trees**. Let T be a directed tree (also called arborescence) with root r. Hence $T = (VT, E_T)$ where V_T is the vertex set of T, $E_T \subseteq V_T \times V_T$ is the set of (directed) edges of T, and $r \in V_T$ is the only vertex in T with indegree 0 in T. All other vertices in T have indegree 1. The vertices with outdegree 0 are the leaves of T. The root r is the only vertex in t at level 0. The level-1 vertices of T are precisely the children of r, i.e. those vertices v such that there is an edge $(r, v) \in E_T$. The level-2 vertices are the children of level-1 vertices and so on. A directed tree is called **l-level restricted** iff it has at most $l + 1$ levels $0, \ldots, l$.

Accordingly, for a given instance of the Steiner Arborescence Problem with root r and terminal set S, let $SMT(r, S)$ denote a shortest directed tree rooted at r that

contains S, let $smt(r,S)$ denote its cost. Let $SMT_k(r,S)$ denote a shortest directed$k$-level-restricted tree rooted at r that contains S, let $smt_k(r,S)$ denote its cost. Hence we are interested in the ratio

$$\frac{smt_k}{smt} := \sup\left\{\frac{smt_k(r,S)}{smt(r,S)} : G,S,r \text{ instance of the Steiner Arborescence Problem}\right\}$$

Theorem 41 (Zelikovsky [Zel97])
For any instance G,S,r of the Steiner Arborescence Problem and any $k \in \mathbb{N}$,

$$\frac{smt_k(r,S)}{smt(r,S)} \leq |S|^{1/k}$$

and this bound is tight.

All the algorithms we will consider in this chapter are based on this ratio, i.e. they attemt to approximate the optimum k-level restricted tree and run in time exponential in k which -roughly spoken- implies at least a polynomial approximation ratio. We are not aware of algorithmic approaches that avoid this lack.

8.2 The Directed Zero Skew Tree Problem

Before considering the directed version of the Zero Skew Tree Problem and giving a precise problem formulation, let us first take a look at the undirected version which was already extensively considered in the literature and builds the starting point and main motivation for our investigations in this section. The following definition as well as the problem formulation of the **Zero and Bounded Skew Tree Problems** are taken from the Zelikovsky-Mandoiou paper [ZM01]:

Definition 39 *Let (V,c) be a metric space. A* **stretched tree** *for a set of sinks $S \subseteq V$ is a quadruple $T = (V_T, E_T, \pi, cost)$ such that (V_T, E_T) is a rooted tree, π is a mapping $\pi: V_T \to V$ and cost$: E \to \mathbb{R}_+$ such that π is a 1-1 mapping from the leaves of T to S and for every edge $e = (u,v) \in E_T$, cost$(e) \geq c(\pi(u), \pi(v))$.*

The underlying intuition is to put additional delay on the edges in order to obtain equal signal running times. The skew of T is the maximum cost difference of any two root-to-leaf paths in T.

Zero Skew Tree Problem
Instance: finite metric space (V,c), set of sinks $S \subseteq V$

Solution: stretched zero skew tree $T = (V_T, E_T, \pi, \text{cost})$ for S

Cost: cost(T)

If instead one allows skew bounded by some prespecified value b, one obtains the

b-Bounded Skew Tree Problem
Instance: finite metric space (V, c), set of sinks $S \subseteq V$
skew bound $b \in \mathbb{R}_{\geq 0}$

Solution: stretched tree $T = (V_T, E_T, \pi, \text{cost})$ for S
with $\text{skew}(T) \leq b$

Cost: $\text{cost}(T)$

Charikar, Kleinberg et al. [CKK+99] obtained a constant factor algorithm for the more general problem with prespecified root, which we call the **Rooted Zero Skew Tree Problem** in order to distinguish between the two variants:

Rooted Zero Skew Tree Problem
Instance: finite metric space (V, c), root $r \in V$, set of sinks $S \subseteq V$
Solution: stretched zero skew tree $T = (V_T, E_T, \pi, \text{cost})$ for S
rooted at r

Cost: $\text{cost}(T)$

Theorem 42 (Charikar, Kleinberg et al. [CKK+99])
There is a polynomial time approximation algorithm with ratio $2e \approx 5.44$ *for the* **Rooted Zero Skew Tree Problem**.

Mandoiu and Zelikovsky [ZM01] improved on this, but their result only works for the Zero Skew Tree Problem without prespecified root. Their algorithms are heavily based on the freedom to choose the root.

Theorem 43 (Mandoiu, Zelikovsky [ZM01])
There are polynomial time approximation algorithms with ratio 4 *for the* **Zero Skew Tree problem** *and* 14 *for the* **Bounded Skew Tree Problem**.

We refer to the original paper [ZM01] for more detailed information, algorithms and proofs. In this section we will now consider the following directed variant of the problem:

Directed B-Bounded Skew Tree Problem
Instance: directed graph $G = (V, E)$, edge costs $c\colon E \to \mathbb{R}_+$
a root $r \in V$ and a set of sinks (terminals) $S \subseteq V \setminus \{r\}$
and a number $B \in \mathbb{R}_{\geq 0}$ bounding the skew

Solution: stretched directed tree $T = (V_T, E_T, \pi, \text{cost})$ rooted at r for S
such that the skew of T with respect to S is bounded by B

Cost: $\text{cost}(T)$

The **Directed Zero Skew Tree Problem** is the special case with $B = 0$. Let us start by proving the analog hardness result as is already known for the Directed Steiner Tree Problem.

Theorem 44 *Unless $NP \subseteq DTIME\left(n^{\log\log n}\right)$, there is no polynomial-time approximation algorithm for the Directed B-Bounded Skew Tree Problem with ratio*

$$(1-\epsilon)\cdot\log(n).$$

Proof: For $B = \max\{\text{dist}(r,s)|s \in S\}$ the reduction from the Set Cover Problem to the Directed Steiner Tree Problem already given in [CCC+98, Zel97] is also a reduction to the Directed B-Bounded Skew Tree Problem. □

8.2.1 Stretched Arborescences, Skew and Delay

Stretched arborescences are the directed analog of stretched trees, i.e. formally a tuple $T = (V, E, \pi, \text{cost})$. Given a stretched arborescence T rooted at r spanning the set of sinks X which is not necessarily a zero-skew arborescence, we can obtain a stretched zero-skew arborescence by -informally spoken- inserting delay edges at certain vertices of T (this was called *tree stretching* in [Zelikovsky, Mandoiu]). The cost increase by this operation can be bounded in terms of the *delay* and *skew* of vertices of T:

Definition 40 *Let T be a stretched arborescence rooted at r spanning the set of sinks X. For a vertex v of T let T_v denote the subtree of T rooted at v.*

The skew of v in T is defined as $skew_{\mathbf{T}}(\mathbf{v})$:= the maximum difference of costs of any two v-to-leaf paths in T_v.

The delay of v in T is defined as $delay_{\mathbf{T}}(\mathbf{v})$:= the maximum cost of a v-to-leaf path in T_v.

The skew of T with respect to X is defined as $skew_{\mathbf{X}}(\mathbf{T}) := \sum_{v\in T\setminus X} skew_T(v)$.

The delay of T with respect to X is defined as $delay_{\mathbf{X}}(\mathbf{T}) := \sum_{v\in T\cap X} delay_T(v)$.

Lemma 8.2.1 *There is a polynomial time algorithm that, given a stretched arborescence T rooted at r spanning a set of sinks X in G, constructs a stretched zero-skew arborescence $\tilde{T}$ rooted at r spanning X such that*

$$cost(\tilde{T}) \;\leq\; c(T) \;+\; delay_X(T) \;+\; skew_X(T).$$

Definition 41 *For a stretched arborescence T in G rooted at r spanning the set of sinks X let*

$$cds_{\mathbf{X}}(\mathbf{T}) \;:=\; c(\mathbf{T}) \;+\; delay_{\mathbf{X}}(\mathbf{T}) \;+\; skew_{\mathbf{X}}(\mathbf{T}).$$

Hence the Directed Zero Skew Tree Problem is equivalent to the problem of finding minimum-cds stretched arborescences for a given root and set of sink:

> **Minimum cds Arborescence Problem** Given a root r and a set of sinks X in a transitive directed graph $G = (V, E)$ with edge weights $c: E \to \mathbb{R}_+$, find a minimum-cds stretched arborescence with root r for X.

Based on Feige's result and using the standard reduction to the Directed Steiner Tree Problem as above we obtain the following hardness result for the **Minimum cds Arborescence Problem** and hence as well for the Directed Zero Skew Tree Problem:

Theorem 45 *Unless* $NP \subseteq DTIME\left(n^{\log\log n}\right)$*, for every* $\epsilon > 0$ *there is no polynomial time approximation algorithm for the Minimum cds Arborescence Problem with ratio* $(1-\epsilon)\cdot\ln(n)$*.*

Proof: Consider an instance $(U, \mathcal{S})$ be an instance of the Set Cover Problem with $U = \{1, \ldots, n\}$ and $\mathcal{S} = \{S_1, \ldots, S_m\} \subseteq P(U)$. Let the tree $T(U, \mathcal{S})$ be defined as follows: The vertex set consists of vertices v_u for each $u \in U$ (*element nodes*), v_S for each $S \in \mathcal{S}$ (*set nodes*) and an additional vertex r. There are edges (r, v_S) of cost 1 from r to the set nodes v_S and edges of cost 0 from each set node v_S to all element nodes v_u with $u \in S$. Consider now the instance for the Minimum cds Arborescence Problem with directed graph $T(U, \mathcal{S})$, root r and set of sinks $\{v_u | u \in U\}$. An optimum solution T corresponds to an optimum cover $\mathcal{S}' \subseteq \mathcal{S}$ of cost $\mathrm{cds}(T) - 1$. □

8.2.2 Level-Restricted Stretched Trees

Recall that for the Steiner Arborescence Problem Zelikovsky obtained a tight bound on the k-level ratio. We show that the same ratio holds for the Directed Zero Skew Problem. Let $\mathrm{zmt}(\mathbf{G}, \mathbf{r}, \mathbf{X})$ denote the minimum cds of an arborescence rooted at r spanning X, let $\mathrm{zmt}_i(G, r, X)$ denote the minimum cds of an i-level-restricted arborescence rooted at r spanning X.

Lemma 8.2.2 *For all directed transitive graphs* $G = (V, E)$ *with edge costs* $c: E \to \mathbb{R}_+$*, root* r *and set of sinks* X

$$zmt_i(G, r, X)/zmt(G, r, X) \;\leq\; |X|^{1/i}.$$

Proof: Let T be an arborescence for r, X in G of optimum value $\mathrm{cds}_X(T)$. Then there is an i-level-restricted arborescence $\tilde{T}$ for r, X of cost $\mathrm{c}(\tilde{T}) \leq |X|^{1/i} \cdot \mathrm{c}(T)$. Furthermore from the proof of the k-level ratio in ... it directly follows that $\tilde{T}$ can be constructed such that for any vertex v of $\tilde{T}$ the skew and delay of v in $\tilde{T}$ is not larger than the skew and delay of v in T. Thus by the definition of cds it follows

$$\mathrm{cds}_X(\tilde{T}) \leq |X|^{1/i} \cdot \mathrm{c}(T) + \mathrm{skew}_X(T) + \mathrm{delay}_X(T) \leq |X|^{1/i} \cdot \mathrm{cds}_X(T).$$

8.2.3 An Approximation Algorithm

We will now give a recursive approximation scheme for the Minimum cds Arborescence Problem that approximates the optimum solution by an i-level-restricted solution and has running time $O(f(i) \cdot k^i)$ for some function f, thus polynomial in k if i is fixed. As in the Charikar et al. approach [CCC+98] we will work with a slightly more general problem:

> **D-Skew($\mathbf{k}, \mathbf{r}, \mathbf{X}, \mathbf{a}, \mathbf{b}$):** Given $k \in \mathbb{N}$, a root r and a set of sinks $X \subseteq \mathcal{N}$, positive integer numbers $a \leq b$, find a tree T of minimum value $\mathrm{cds}_X(T)$ rooted at r spanning k terminals from X such that for all $u \in T$ $a \leq \mathrm{dist}_T(r, u) \leq b$.

For technical reasons we will assume that all terminals from X are leaves in the directed graph G (if not then add vertices s' and 0-cost edges (s, s') for each $s \in X$ and consider the D-Skew problem for set $\{s' : s \in X\}$ in the transitive closure of this extended graph).

The **density** of a stretched tree T rooted at r with respect to X is the ratio

$$d_X(T) \quad := \quad \mathrm{cds}_X(T)/|T \cap X|.$$

Following the lines of the Charikar et al. paper [CCC+98], we say algorithm $\mathcal{A}$ is an **$\mathbf{f(k)}$-partial approximation algorithm** for D-Skew($\mathbf{k}, \mathbf{r}, \mathbf{X}, \mathbf{a}, \mathbf{b}$) if it constructs a stretched tree T' rooted at r spanning a set of $1 \leq k' \leq k$ terminals from X such that

$$d_X(T') \quad \leq \quad f(k) \cdot \frac{\mathrm{cds}_X(\mathrm{T}^*)}{k}$$

Lemma 8.2.3 *If $\mathcal{A}$ is an $f(k)$-partial approximation algorithm for D-Skew($\mathbf{k}, \mathbf{r}, \mathbf{X}, \mathbf{a}, \mathbf{b}$) and $f(x)/x$ is decreasing, then the algorithm $\mathcal{B}$ obtained by repeated application of algorithm $\mathcal{A}$ and removal of the terminals already collected from X yields a $g(k)$-approximation algorithm for D-Skew($\mathbf{k}, \mathbf{r}, \mathbf{X}, \mathbf{a}, \mathbf{b}$) with $g(k) = \int_0^k \frac{f(x)}{x} dx$.*

Proof: Induction on k. For $k = 1$ we observe $f(1) \leq \int_0^k \frac{f(x)}{x} dx$. For $k > 1$: Let $\mathcal{A}(k, r, X, a, b)$ return T_1, then we have $d_X(T_1) = \mathrm{cds}_X(T_1)/k_1 \leq f(k) \cdot \mathrm{cds}_X(\mathrm{T}^*)/k$, hence

$$\mathrm{cds}_X(T_1) \leq k_1 \cdot \frac{f(k)}{k} \cdot \mathrm{cds}_X(\mathrm{T}^*) \leq \left(\int_{k-k_1}^{k} \frac{f(x)}{x} dx \right) \cdot \mathrm{cds}_X(\mathrm{T}^*).$$

If $k = k_1$ then T_1 is returned. Otherwise $k_1 < k$. Let $\mathcal{B}(k - k_1, r, X \setminus X_1, a, b)$ return tree T_2. By induction hypothesis

$$\mathrm{cds}_{X \setminus X_1}(T_2) \leq \left(\int_0^{k-k_1} \frac{f(x)}{x} dx \right) \cdot \mathrm{cds}_{X \setminus X_1}(\mathrm{T}^*).$$

Now using the result of lemma 8.2.4 below we can replace $X \setminus X_1$ by X in this chain of inequalities, and we obtain $\mathrm{cds}_X(T_1) + \mathrm{cds}_X(T_2) \leq g(k) \cdot \mathrm{cds}_X(\mathrm{T}^*)$. □

Now we will describe our algorithm. The basic idea is to proceed as in the Charikar

et al. paper [CCC+98], solving the problem D-Skew(k, r, X, a, b) by iterated picks of good-density partial solutions as long as we still have to collect terminals from X. In our setting, density of a tree T with respect to a terminal set X is defined as $d_X(T) := cds_X(T)/|T \cap X|$ i.e. cds value per terminal being collected from X. Hence our algorithm looks as follows:

Algorithm $\mathbf{A_i(k, r, X, a, b)}$
Output: i-level-restricted feasible solution to D-Skew(k, r, X, a, b)
Check whether a solution exists.
If $i = 1$: Take the k nearest terminals $v_1, \ldots, v_k$ with distances
$a \leq \text{dist}(r, v_1) \leq \ldots \leq \text{dist}(r, v_k) \leq b$ from the root,
let T consist of edges $(r, v_i), 1 \leq i \leq k$.
If $i \geq 2$: initially $T := \emptyset$.
While $k > 0$
/⋆ *Search for a good-density partial solution.* ⋆/
$\mathrm{T_b} := \emptyset$
For $v \in V$, $1 \leq k' \leq k$, $a - \text{dist}(r, v) \leq a' \leq b' \leq b - \text{dist}(r, v)$
$\mathrm{T_{act}} := A_{i-1}(k', v, X, a', b') \cup \{(r, v)\}$.
If $\frac{\mathrm{cds}_X(\mathrm{T_{act}})}{k'} < \frac{\mathrm{cds}_X(\mathrm{T_b})}{k(\mathrm{T_b})}$ **then** $\mathrm{T_b} := \mathrm{T_{act}}$;

/⋆ *Add the best tree T_b to T, update.* ⋆/
$T := T \cup \mathrm{T_b}$; $X := X \setminus X(\mathrm{T_b})$; $k := k - k(\mathrm{T_b})$;
Return T;

Lemma 8.2.4 *Let T be a stretched arborescence rooted at r and $X \subseteq Y \subseteq V$ such that all leaves of T are in X. Then $cds_X(T) \leq cds_Y(T)$.*

Proof: For all $v \in T$ we have $\text{delay}_T(v) \geq \text{skew}_T(v)$. Now

$$\begin{aligned} \mathrm{cds}_X(T) &= \mathrm{c}(T) + \sum_{v \in X \cap T} \text{delay}_T(v) + \sum_{v \in T \setminus X} \text{skew}_T(v) \\ &\leq \mathrm{c}(T) + \sum_{v \in Y \cap T} \text{delay}_T(v) + \sum_{v \in T \setminus Y} \text{skew}_T(v) = \mathrm{cds}_Y(T). \end{aligned}$$

□

We consider now one call of $A_i(k, r, X, a, b)$. Let $X(j)$ and $k(j)$ denote the actual values of X and k at the beginning of the j-th iteration of the while-loop in the execution of $A_i(k, r, X, a, b)$, and let $\mathrm{T_b}(\mathbf{j})$ denote the tree $\mathrm{T_b}$ found in the j-th iteration and $k_b(j) := |X \cap \mathrm{T_b}(j)|$ the number of terminals it covers. Let $\mathrm{T}^{(i)}_{\mathrm{opt}}(j)$ denote an optimum i-level-restricted solution to D-Skew$(k(j), r, X(j), a, b)$ and $\mathrm{T}^{(i)}_{\mathrm{opt}} = \mathrm{T}^{(i)}_{\mathrm{opt}}(0)$ an optimum i-level-restricted solution to D-Skew(k, r, X, a, b). We claim that for the best tree found

in iteration j we have

$$d_{X_j}(\mathrm{T_b}(j)) = \frac{\mathrm{cds}_{X_j}(\mathrm{T_b}(j))}{k(j)} \quad \leq \quad (i-1)\cdot \frac{\mathrm{cds}_{X_j}(\mathrm{T}^{(i)}_{\mathrm{opt}}(j))}{k(\mathrm{T}^{(i)}_{\mathrm{opt}}(j))} = (i-1)\cdot d_{X_j}(\mathrm{T}^{(i)}_{\mathrm{opt}}(j))$$

First we compute a lower bound for the density of the optimum solution:

$$\begin{aligned} d_X(\mathrm{T}^{(i)}_{\mathrm{opt}}) &= \frac{1}{k}\cdot \mathrm{cds}_X(\mathrm{T}^{(i)}_{\mathrm{opt}}) \\ &= \frac{1}{k}\cdot\left(\mathrm{ds}_{\mathrm{T}^{(i)}_{\mathrm{opt}}}(r) + \sum\nolimits_{(r,v)\in E(\mathrm{T}^{(i)}_{\mathrm{opt}})} (\alpha_v + \mathrm{cds}_X(T_v))\right) \\ &\geq \min\nolimits_{(r,v)\in E(\mathrm{T}^{(i)}_{\mathrm{opt}})} \frac{\alpha_v + \mathrm{cds}_X(T_v)}{k_v} \end{aligned} \tag{8.1}$$

where α_v is the cost of edge (r,v) and k_v is the number of leaves in the subtree rooted at v. Let us fix a vertex v at which the minimum in (8.1) is achieved. Let $a-\alpha_v \leq \tilde{a} \leq \tilde{b} \leq b-\alpha_v$ denote the precise values for the tree T_v, i.e.

$$\tilde{a} = \min_{w\in T_v} \mathrm{dist}_{T_v}(v,w), \tag{8.2}$$

$$\tilde{b} = \max_{w\in T_v} \mathrm{dist}_{T_v}(v,w). \tag{8.3}$$

Consider the execution of $A_{i-1}(k_v, v, X_j, \tilde{a}, \tilde{b})$: Let $\tilde{X}_{j'}$ and $\tilde{k}_{j'}$ denote the values of X and k at the beginning of the j'-th iteration of the while-loop in the execution of $A_{i-1}(k_v, v, X_j, \tilde{a}, \tilde{b})$. Let $\tilde{T}_{\mathrm{b}}(j')$ denote the tree constructed in the j'-th iteration of the while-loop. Let L be the number of while-loop iterations performed by $A_{i-1}(k_v, v, X_j, \tilde{a}, \tilde{b})$. Let $s_0 = 0$ and $s_l =$ the number of terminals from X_j covered by the tree $\mathbf{T(l)} := \tilde{T}_{\mathrm{b}}(1) \cup \ldots \cup \tilde{T}_{\mathrm{b}}(l)$ for $l \in \{1,\ldots,L\}$. There is an $l \in \{0,\ldots,L-1\}$ such that $s_l < k_v/(i-1) \leq s_{l+1}$. Since $\tilde{k}_{l+1} > k_v - \frac{k_v}{i-1} = \frac{i-2}{i-1}\cdot k_v$ and

$$|T_v \cap \tilde{X}_{l+1}| \geq k_v - |T_v \cap X(T(l))| \geq k_v - s_l = \tilde{k}_{l+1} \leq \frac{i-2}{i-1}\cdot k_v$$

we conclude that

1. T_v is a solution for D-Skew$(\tilde{k}_{l+1}, v, \tilde{X}_{l+1}, \tilde{a}.\tilde{b})$ and
2. T_v has density $d_{\tilde{X}_{l+1}}(T_v) \leq \frac{\mathrm{cds}_{\tilde{X}_{l+1}}(T_v)}{k_v}\cdot\frac{i-1}{i-2}$.

Thus by inductive assumption the tree $\tilde{T}_{\mathrm{b}}(l+1)$ constructed in the $(l+1)$rst execution of the while loop has density with respect to the remaining terminal set $\tilde{X}_{l+1}$ bounded as

$$d_{\tilde{X}_{l+1}}(\tilde{T}_{\mathrm{b}}(l+1)) \quad \leq \quad (i-2)\cdot\frac{i-1}{i-2}\cdot\frac{\mathrm{cds}_{\tilde{X}_{l+1}}(T_v)}{k_v} \leq (i-1)\cdot\cdot\frac{\mathrm{cds}_{X(j)}(T_v)}{k_v}.$$

Furthermore for iterations $1,\ldots,l$ of the while-loop we obtain the same bound. Hence $T(l+1)$ has density bounded by $(i-1)\cdot d_{X_j}(\mathrm{T}^{(i)}_{\mathrm{opt}})$. Now applying lemma 8.2.3 we obtain our main result:

Theorem 46 *Algorithm A_i provides an $i(i-1)k^{1/i}$ approximation algorithm to the problem D-Skew(k, r, X, a, b).*

Proof: The cds cost of the tree T_i constructed by algorithm A_i can be bounded as

$$\mathrm{cds}_X(T_i) \leq (i-1) \cdot \mathrm{cds}(\mathrm{T}^*) \cdot \int_0^k \frac{x^{1/i} dx}{x} = i(i-1)\mathrm{cds}_X(\mathrm{T}^*).$$

□

8.3 The Directed Weighted Path Problem

In the **Weighted Path Problem (WPP)** we are given a graph $G = (V, E)$ with edge costs $c\colon E \to \mathbb{R}_+$, a root $r \in V$, a set of terminals $S \subseteq V$ and a weight function $\omega\colon S \to \mathbb{R}_+$. The task is to construct a Steiner tree T for the set $\{r\} \cup S$ such as to minimize the total cost of the tree plus the sum of weighted distances from the terminals to the root. This problem is motivated by **critical sink problems** in VLSI design where one has to connect a set of sinks to a given root and some of the sinks are time-critical. In order to minimize signal running times and therefore achieve speedup of processor cycle running times, the critical sinks must obtain short connections to the root, and this can be modelled by giving them large weight. In the undirected case the **WPP** is a natural extension of the Steiner Tree Problem in graphs (STP): For an instance $G = (V, E), c\colon E \to \mathbb{R}_+, S \subseteq V$ of the STP define weights $\omega(s) := 0$. In this section we consider the directed variant of the problem, the so called

Directed Weighted Path Tree Problem
Instance: directed graph $G = (V.E)$ with edge costs $c\colon E \to \mathbb{R}_+$,
a root $r \in V$ and a set of sinks $S \subseteq V$ with weights
$\omega\colon S \to \mathbb{R}_+$

Solution: arborescence T in G rooted at r spanning S

Cost: $\mathrm{wpc}(T) := c(T) + \sum_{s \in S} \omega(s) \cdot \mathrm{dist}_T(r, s)$.

This problem is a generalization of the Directed Steiner Tree Problem (introduce zero weight for every terminal), hence we have the same hardness result as for the latter, obtained by reduction from Set Cover. By adapting the methods already used for the Directed Zero Zkew Tree Problem, we obtain the following result.

Theorem 47 *For each $\epsilon > 0$ there is a polynomial time approximation algorithm for the* **Directed Weighted Path Tree Problem** *with ratio $O(n^\epsilon)$, where $n = |V|$ is the size of the input graph.*

Sketch of Proof: We work with level-restricted trees as before and again follow the lines of the Charikar et al. paper [CCC+98]. It basically remains to say which generalized problem we work with:

D-Weight(k, r, X)**:** Given a root r, a set of terminals X with weights $\omega\colon X \to \mathbb{R}_+$ and a natural number $1 \leq k \leq |X|$, choose a subset $X' \subseteq X$ of size k and an arborescence T rooted at r spanning X' such as to minimize $c(T) + \omega(X')$.

We ommit the details, which will appear in a forthcomming paper on the topic.

Let us remark that two directions of future work on directed Steiner problems seem most interesting to us: On the one hand one should try to obtain progress in order to decrease the gap between upper and lower bound for approximability of directed Steiner Tree problems. An approximation algorithm with polynomial running time and sub-polynomial approximation ratio (e.g. some polylog ratio) would mean a breakthrough.

On the other hand one can look at directed versions of the Steiner Network Problem discussed in chapter ... of this thesis. One could try to work with level-restricted trees as well, but then for the general case where connection requirements might be super-polynomial, one would need an extension of the ratio level-restricted trees to unrestricted trees.

A third promising direction is to consider routing games in directed graphs, where one has to construct or approximate equilibria for a scenario with several agents having connection requirements in a directed graph. For the undirected case there is already a huge literature available, see e.g. [TR02].

Chapter 9

Dense Problems

9.1 Introduction

When dealing with NP-hard optimization problems we are interested in approximation algorithms with polynomial running time and approximation ratio as good as possible. As we have already discussed in previous chapters, polynomial time approximation schemes are of special interest. In case one has established hardness results like MAXSNP-hardness which makes it seem unlike that a polynomial time approximation scheme exists, a way to accomplish is to look for interesting subclasses of the problem and ask for polynomial time approximation schemes for such subclasses.

For various optimization problems, an important and interesting subclass are the so called *dense instances*. Typically there are two distinct notions: *average density* and *everywhere-density*. Roughly spoken, a graph is *average dense* if it contains many edges (which means the average degree of vertices is large), and *everywhere-dense* if every node has high degree. A boolean formula in conjunctive normal form is called *dense* if it contains many clauses, and *everywhere-dense* if every variable is contained in many clauses.

There are several algorithmic problems in combinatorial optimization which are easier to solve in dense graphs. Posa proves existence of Hamilton cycles in graphs with degree at least $n/2$, and such cycles can be efficiently constructed. Arora, Frieze and Kaplan [AFK96] give an approximation scheme for evaluating the Tutte polynomial in everywhere-dense graphs, which includes estimation of the reliability of a network as a special case. Arora, Karger and Karpinski [AKK95] constructed polytime approximation schemes for dense instances of a huge number of optimization problems, including Dense Maximum Cut, Dense Graph Bisection and Dense Max-3SAT. Their results are based on reformulation of the problems in terms of *smooth polynomial integer programs* and the ability to approximately solve such programs using exhaustive sampling techniques.

Since then, several further results were obtained in this area. There are several algorithmic problems for which the techniques of [AKK95] are not known to apply, including dense versions of the Set Cover Problem, the Steiner Tree Problem and

the Vertex Cover Problem. Nevertheless Karpinski and Zelikovsky [KZ97a] were able to give combinatorial algorithmic approaches for these problems based on a Greedy approach and obtained a PTAS for the dense Steiner Tree Problem and improved approximation algorithms for the dense Set Cover and Vertex Cover Problem.

This chapter deals with dense versions of the Steiner Tree Problem and some of the various Steiner problems we have considered so far, namely the Steiner Forest Problem and the Prize Collecting Steiner Tree Problem. Additionally we consider dense versions of the k-Steiner Tree problem and the Group Steiner Tree Problem. The chapter is organized as follows: In section 9.2 we give an introduction into Dense Optimization Problems and some of the main results known in this area. In section 9.3 we define the ϵ-Dense Steiner Tree Problem and describe the PTAS for this problem due to Karpinski and Zelikovsky [KZ97a]. In section 9.5 we consider the ϵ-Dense Steiner Forest Problem. We apply ideas of [KZ97a] to this problem and obtain a polynomial time approximation algorithm for this problem with ratio $1+O\left(\frac{\sum_i \log(|S_i|)}{\sum_i |S_i|}\right)$, where S_i denote the terminal sets. This provides good approximation in cases where sufficiently many terminal sets are large. In sections 9.6 and 9.7 we give polynomial time approximation schemes for the ϵ-Dense Prize Collecting Steiner Tree Problem and the ϵ-Dense $k-$Steiner Tree Problem. In section 9.8 we give an introduction to the Class Steiner Tree Problem and a PTAS for the ϵ-dense version of this problem. Interestingly, for the general Class Steiner Tree Problem there is a logarithmic lower bound for approximability, by a simple reduction from the Set Cover Problem due to Ihler and the well-known logarithmic lower bound for the latter problem by Feige [Fei98].

9.2 Dense Optimization Problems

9.2.1 Smooth Integer Programs

An *Integer Polynomial Program* (IPP) is an optimization problem of the form

$$\begin{array}{lll} \text{maximize} & p_0(x_1,\ldots,x_n) & \\ \text{subject to} & l_i \le p_i(x_1,\ldots,x_n) \le u_i & (1 \le i \le m) \\ & x_i \in \{0,1\} & (1 \le i \le n) \end{array}$$

where $p_o,\ldots,p_m$ are polynomials over $\mathbb{Z}$ (minimization instead of maximization is possible). The IPP is called a *degree d IPP* if all p_i have degree at most d. An n-variate degree-d polynomial is called *c-smooth* if the absolute value of each coefficient of each degree i monomial is at most $c \cdot n^{d-i}$. An IPP is called a *c-smooth degree-d IPP* if the objective function p_0 and constraints p_i $(1 \le i \le m)$ are c-smooth degree-d polynomials.

Theorem 48 *[AKK95] There is a randomized polynomial-time algorithm that approximately solves smooth IPPs in the following sense: Given a feasible c-smooth degree d IPP with n variables, objective function p_0 and constraints $l_i \le p_i(x_1,\ldots,x_n) \le u_i (1 \le i \le m)$, the algorithm constructs a vector $z \in \{0,1\}^n$ such that $p_0(z) \ge OPT - \epsilon \cdot n^d$ and each degree d' constraint is satisfied with an additive error of $O(\epsilon \cdot \sqrt{n \log n})$.*

9.2.2 Applications

Arora, Karger and Karpinski [AKK95] applied their result on Smooth Integer Programs to dense versions of various well-known optimization problems, including

- **Max-Cut, Max-Dicut** and **Max-Hypercut** in δ-dense graphs resp. hypergraphs, where a graph on vertices is called δ-dense iff it has at least $\delta \cdot n^2$ edges,
- **Dense Max-k-SAT**, where an instance φ of Max-k-SAT with n variables is called δ-**dense** iff the number of clauses is at least $\delta \cdot n^k$.

For many other results and proofs we refer to the original paper [AKK95].

9.3 Dense Covering Problems

The methods described in the last section are currently not known to apply directly to dense versions of covering problems like Set Cover, Vertex Cover Problem and Steiner Tree Problem. However Karpinski and Zelikovsky [KZ97a] were able to apply combinatorial algorithmic methods based on a Greedy approach to these problems and to obtain the following results:

The ϵ-Dense Set Cover Problem
Recall that in the Set Cover Problem we are given a finite set U (called *universe*) together with a family $\mathcal{S} = \{S_1, \ldots, S_m\}$ of subsets of U, the task is to find a minimum cardinality covering of U, i.e. a minimum cardinality subfamily $\mathcal{S}'$ of $\mathcal{S}$ such that $\bigcup_{S_i \in \mathcal{S}'} S_i = U$. The well known result of Feige [Fei98] states that unless $NP \subseteq P^{\log(n)\log\log(n)}$ there is no polynomial time approximation algorithm for the Set Cover Problem with ratio $(1-\epsilon) \cdot \ln(n)$. Note that the Greedy algorithm obtains ratio $\ln(n)$. It is shown in [KZ97a] that the ϵ-Dense Set Cover Problem admits a $c \cdot \log(n)$-approximation algorithm for each $c > 0$. Here an instance $(U, \mathcal{S})$ with universe $U = \{u_1, \ldots, u_n\}$ and $\mathcal{S} = \{S_1, \ldots, S_m\}$ is called ϵ-dense if every element u of the universe U is contained in at least a ϵ-fraction of the sets of $\mathcal{S}$ (formally: $\forall u \in U : |\{S_i \in \mathcal{S} : u \in S_i\}| \geq \epsilon \cdot |\mathcal{S}|$).

The ϵ-Dense Vertex Cover Problem
An instance $G = (V, E)$ of the vertex cover problem is called *strongly ϵ-dense (everywhere ϵ-dense)* if every vertex has degree at least $\epsilon \times n$, where n is the size of the vertex set V of graph G. It is called *weakly ϵ-dense (average ϵ-dense)* if $\sum_{v \in V} deg_G(v) = \epsilon \cdot n^2$. The following results were obtained in [KZ97a]:

- The Strongly ϵ-Dense Vertex Cover Problem is MAX SNP-hard, but approximable within ratio $\frac{2}{1-\epsilon}$.
- The Weakly ϵ-Dense Vertex Cover Problem is approximable within ratio $\frac{2}{2-\sqrt{1-\epsilon}}$.

The ϵ-Dense Steiner Tree Problem
An instance of the Steiner Tree Problem in Graphs consisting of graph $G = (V, E)$ and terminal set $S \subseteq V$ is called ϵ-dense if each terminal $s \in S$ has at least $\epsilon \cdot |V \setminus S|$ neighbours in $V \setminus S$. This is a variant of everywhere density considered above (indeed, it is implied by everywhere density). Karpinski and Zelikovsky showed: For every $\epsilon > 0$ there is a polynomial time approximation scheme for the ϵ-Dense Steiner Tree Problem.

In the subsequent sections of this chapter we will apply ideas from [KZ97a] and consider dense versions of the following generalizations of the Steiner Tree Problem:

Steiner Forest Problem (SFP): Given a graph $G = (V, E)$ with edge costs $c\colon E \to \mathbb{R}_+$ and pairwise disjoint nonempty terminal sets $S_1, \ldots, S_n \subseteq V$, find a forest $F = (V(F), E(F)) \subseteq G$ of minimum cost such that for all $1 \leq i \leq n$ S_i is contained in a connected component of F. Again, this is equivalent to the SFP in finite metric spaces. In the graph version (Graph-SFP), the instance consist of an unweighted graph G (i.e. every edge has cost 1 together with terminal sets $S_1, \ldots, S_n$ as above. An instance of the Graph-SFP is called ϵ-dense if for every $1 \leq i \leq n$ and every $s \in S_i$ the number of neighbours of s in $V \setminus S_i$ is at least $\epsilon \cdot |V \setminus S_i|$

Prize Collecting Steiner Tree Problem (PSTP): Given a graph $G = (V, E)$ with edge costs $c\colon E \to \mathbb{R}_+$ and a terminal set $S \subseteq V$ with prize function $p\colon S \to \mathbb{R}_+$, find a tree $T \subseteq G$ connecting a subset S' of S such as to minimize $c(T) + p(S \setminus S')$. Here we take the same density condition as for the Steiner Tree Problem: An instance is called ϵ-denes if every terminal has at least $\epsilon \cdot |V \setminus S|$ neighbours in $V \setminus S$. We will give a polynomial time approximation scheme for the ϵ-Dense PSTP.

k-Steiner Tree Problem (k-STP): Given a graph $G = (V, E)$ with edge costs $c\colon E \to \mathbb{R}_+$, a terminal set $S \subseteq V$ and a number $k \in [1, |S|]$, find a tree T in G of minimum cost $c(T)$ which connects at least k terminals from S. We will consider the same density condition as for the Steiner Tree Problem and for the Prize Collecting Steiner Tree Problem. We will give a PTAS for the ϵ-Dense k-Steiner Tree Problem.

Class Steiner Tree Problem: Given a graph $G = (V, E)$ with edge costs $c\colon E \to \mathbb{R}_+$ and a system of pairwise disjoint subsets $C_1, \ldots, C_n$ of the vertex set V, find a minimum cost tree T in G such that for each $1 \leq i \leq n$ T contains at least one vertex of C_i. The sets C_i are also called classes, hence T has to contain at least one representative for each class. This problem turns out to be at least as hard as the Set Cover Problem, hence for the general case there is a logarithmic lower bound. Nevertheless for the ϵ-dense version of the problem we are able to give a polynomial time approximation scheme. Here an instance is called ϵ-dense if it consists of a graph $G = (V, E)$ (i.e. all edge weights are 1) and classes $C_1, \ldots, C_n$ such that

In this section we consider the **SFP** restricted to dense instances. Consider an instance $G = (V, E), S_1, \ldots, S_n$ of the Graph-SFP. Let $S := \bigcup_{i=1}^{n} S_i$ the set of terminals.

For a vertex $v \in V$ let $N(v) = \{u \in V | \{u, v\} \in E\}$ be the set of neighbours of v in G. The instance is called ϵ-dense iff for every $1 \leq i \leq n$ and $s \in S_i$ $|N(s) \setminus S_i| \leq \epsilon \cdot |V \setminus S_i|$. The **$\epsilon$-Dense SFP** is the SFP restricted to ϵ-dense instances.

For the rest of the paper we will use the following notation: Let $G = (V, E)$ be a graph and $v \in V$. Then $N(v) = N_G(v) = \{u \in V : \{v, u\} \in E\}$ is the set of neighbours of v in G. Furthermore, for a set X $P(X)$ denotes the power set of X and $P_k(X), k \in \mathbb{N}$ the set of all subsets of X of size k.

9.4 The ϵ-Dense Steiner Tree Problem

In this section we will first give a brief outline of the polynomial time approximation scheme of Karpinski and Zelikovsky for the ϵ-Dense Steiner Tree Problem (recall that ϵ-density here means a version of everywhere-density). A natural question in this context is whether a polynomial time approximation scheme exist for some average-dense version of the Steiner Tree Problem. Currently we don't know the answer, nevertheless in subsection 9.4.2 we will slightly relax the density condition and give a PTAS for this generalization of the dense Steiner Tree Problem.

9.4.1 Everywhere-Density

Recall the density condition as it was defined in [KZ97a]: An instance $G = (V, E), S$ of the Graph Steiner Tree Problem is called ϵ-dense iff for every terminal $s \in S$ $|N(s) \cap (V \setminus S)| \geq \epsilon \cdot |V \setminus S|$. The idea of the Karpinski-Zelikovsky algorithm is as follows: Obviously $|S| - 1$ is a lower bound for the cost of an optimum solution (which is achieved iff there exists a spanning tree for S in G. In general the cost of a Steiner tree T for S in G equals $|S| - 1$ plus the number of Steiner points used by T (since T is a spanning tree for its vertices). Hence the task is to construct a Steiner Tree using only few Steiner points. The algorithm collects edges connecting two terminals as long as such edges exist. Each pick of such an edge is "save" in the sense that the number of Steiner points is not increased. Hence assume $E(S) = \emptyset$. Now the density condition implies that there exists a non-teminal $v \in V \setminus S$ with at least $\epsilon \cdot |S|$ neighbours in S. The algorithm picks such a star consisting of a center vertex $v \in V \setminus S$ and all its neighbours in S, takes it as part of the tree to be constructed and contracts the star, reducing the problem size. Then the whole procedure is iterated. Note that since each such pick reduces the number of terminals by at least a constant fraction, after at most a logarithmic number of picks the size of the terminal set is reduced to constant. The remaining problem is now solved to optimality by an exhaustive search.

Let us now give the description of the algorithm, taken from [KZ97a].

Algorithm DSTP[KZ97a]
(0) $S_{final} := S$
(1) **while** $|\mathcal{C}[S_{final}]| > \frac{1}{\epsilon}$ **do**
find $v \in V \setminus S_{final}$ minimizing $|\mathcal{C}[S_{final} \cup \{v\}]|$.

$S_{final} := S_{final} \cup \{v\}$

(2) find an optimum Steiner tree $T^\star(\mathcal{C}[S_{final}])$ for the contracted components $C \in \mathcal{C}[S_{final}]$ using the Tree Enumeration Algorithm

(3) **Return** $T^\star \cup \bigcup_{C \in \mathcal{C}[S_{final}]} T(C)$.

9.4.2 Towards Average-Density: Relaxing the Density Condition

We may now ask for a polynomial approximation scheme for some average-dense versions of the Steiner Tree Problem. A quite natural definition of average density is as follows: We call an instance $G = (V, E), S \subseteq V$ of the Graph Steiner Tree Problem *average ϵ-dense* if

$$|E(S, V \setminus S)| \quad \geq \quad \epsilon \cdot |S| \cdot |V \setminus S|. \tag{9.1}$$

Note that ϵ-Density as defined in [KZ97a] implies Average-ϵ-Density. Furthermore average ϵ-density implies existence of a vertex $v \in V \setminus S$ with at least $\epsilon \cdot |S|$ neighbours in S, i.e. a *good pick*. Unfortunately by no means average ϵ-density is preserved under good picks. Here is an easy example: If there exists a subset S' of S of size $\epsilon \cdot |S|$ such that every vertex $v \in V \setminus S$ is connected to all terminals from S' and to none of the terminals from $S \setminus S'$, this instance is average ϵ-dense. However after one good pick the density condition is not valid anymore.

Nevertheless we will now at least relax the density condition "towards average-density" and give a PTAS for this relaxed version. Let us first give some motivation. We observe that ϵ-density not only implies (9.1) but the following more general property:

$$|E(S', V \setminus S)| \quad \geq \quad \epsilon \cdot |S'| \cdot |V \setminus S| \quad \text{for all } S' \subseteq S \tag{9.2}$$

Indeed (9.2) is equivalent to ϵ-density. We may now ask how far we can relax (9.2). Relaxing here means to consider instances where (9.2) does not necessarily hold for all subsets S' of S but for all subsets with cardinality at least some prespecified lower bound. In (9.2) the lower bound is 1 (or 0) while in the average-density condition (9.1) it is $|S|$. The question now is: How much can we increase the lower bound (starting from 1) and still get a PTAS ? Actually we do not know the answer but at least we can relax up to logarithmic size:

Definition 42 (log**-Density)**
*An instance $G = (V, E), S$ of the Graph Steiner Tree Prolem is called $(\epsilon, c) - \log$-**dense** iff for all subsets $S' \subseteq S$ of the terminal set with $|S'| \geq c \cdot \log(|S|)$*

$$|E(S', V \setminus S)| \quad \geq \quad \epsilon \cdot |S'| \cdot |V \setminus S|. \tag{9.3}$$

Theorem 49 *For each $\epsilon > 0, c > 0$ there is a PTAS for the $(\epsilon, c) - \log$-Dense Steiner Tree Problem.*

Let us first give some ideas and then give the precise proof of Theorem 49. Our approach is quite similar to that of Karpinski and Zelikovsky [KZ97a]. The most

important difference is that when performing greedy steps and picking Steiner points, after contracting a star consisting of a vertex from $V \setminus S$ and all its neighbours in S the resulting supernode will be removed from S and hence not be considered in further greedy steps anymore. This alternative method has basically two effects: First the density condition for the actual terminal set is preserved and second afterwards we are left with a "residual" terminal set of logarithmic instead of constant size. Hence in order to solve the remaining problem we will take the Dreyfus Wagner algorithm instead of the Tree Enumeration algorithm since its running time is polynomial in the number of non-terminals (and exponential in the number of terminals, hence polynomial in the initial input size). We are now ready to describe our algorithm.

Algorithm LDSTP
Input: an instance of the (c, ϵ)-log-dense Steiner Tree Problem consisting of graph $G = (V, E)$, terminal set $S \subseteq V$
Output: Steiner tree T for S in G
(0) $\mathcal{C} := \{\{s\} : s \in S\}$ set of terminal components
$\mathcal{C}_a := \mathcal{C}$ set of active terminal components
(1) **while** $E(\mathcal{C}_a) \neq \emptyset$ **do**
Pick $e \in E(\mathcal{C}_a)$ connecting two terminal components $C_1, C_2 \in \mathcal{C}_a$.
Let $\mathcal{C}_a := (\mathcal{C}_a \setminus \{C_1, C_2\}) \cup \{C_1 \cup C_2\}$, update $\mathcal{C}$ accordingly.
(2) **while** $|\mathcal{C}_a| \geq c \cdot \log(|S|) \cdot K$ **do**
Find $v \in V \setminus S$ with the maximum number of neighbours in $\mathcal{C}_a$.
Contract the star $T(v)$ consisting of v and its neighbours $N(v, \mathcal{C}_a)$ in $\mathcal{C}_a$.
Update $\mathcal{C}$ and $\mathcal{C}_a$ accordingly:
$\mathcal{C}_a := \mathcal{C}_a \setminus N(v, \mathcal{C}_a)$, $\mathcal{C} := (\mathcal{C} \setminus N(v, \mathcal{C}_a)) \cup \{\bigcup_{C \in N(v, \mathcal{C}_a)} C\}$.
(3) Find an optimum Steiner tree $T^\star$ for $\mathcal{C}$.
(4) **Return** $T_{LD} := T^\star \cup \bigcup_{v \text{ picked in (1)}} T(v) \cup \{e |\ e \text{ picked in (1)}\}$.

Analysis. The log-density condition (9.3) directly implies that initially

$$|E(\mathcal{C}', V \setminus \mathcal{C}_a)| \quad \geq \quad \epsilon \cdot |\mathcal{C}'| \cdot |V \setminus \mathcal{C}_a| \tag{9.4}$$

for all subsets $\mathcal{C}'$ of $\mathcal{C}_a$ of size at least $c \cdot \log(|S|)$. We will now prove that (9.4) is preserved by the picks of edges in phase (1) and stars in phase (2) of algorithm LDSTP. Indeed, if an edge connecting two active components is picked, then subsets $\mathcal{C}'$ of $\mathcal{C}_a$ of size at least $c \cdot \log(|S|)$ after the pick correspond to subsets of $\mathcal{C}_a$ of size $\in [|\mathcal{C}'|, |\mathcal{C}'|+1]$ before the pick with the same neighbourhood in $V \setminus \mathcal{C}$, and since $|V \setminus \mathcal{C}|$ does not change, (9.4) still holds. On the other hand, if a vertex $v \in V \setminus \mathcal{C}$ is picked and the star consisting of v and $N(v, \mathcal{C}_a)$ is contracted, then the resulting supernode is removed from $\mathcal{C}_a$ and the cardinality of $V \setminus \mathcal{C}_a$ remains the same, hence also in this case (9.4) is preserved.

Let k be the number of picks of stars $T(v, N(v, \mathcal{C}_a))$ in phase (2), let $T_1 = T(v_1, N_1)$, $\ldots, T_k = T(v_k, N_k)$ denote these stars and let $e_1, \ldots, e_l$ denote the single edges connecting active components picked in phase (1) of algorithm LDSTP.

Now construct graph G' from G by adding edges connecting the set $N(v, \mathcal{C}_a)$ by a spanning tree for each pick v in phase (1) of algorithm LDSTP. Note that $OPT(G', S) \leq$

$OPT(G,S)$. There exists an optimum tree $T^{\star\star}$ in G' consisting of spanning trees T'_i for the sets $N_i = N(v_i, \mathcal{C}_a)$ of picks in phase (2), the set E_1 of all edges connecting two active components in phase (1) and a tree T' connecting the set of components $\mathcal{C}$ at the end of phase (2). Hence we can bound the approximation ratio of algorithm LDSTP as follows:

$$\frac{\text{cost}(T_{LD})}{\text{cost}(T^{\star\star})} \leq \frac{\text{cost}(T^{\star}) + \sum_{i=1}^{k} \text{cost}(T_i) + |E_1|}{\text{cost}(T') + \sum_{i=1}^{k} \text{cost}(T'_i) + |E_1|} \leq \frac{\sum_{i=1}^{k} |N_i|}{\sum_{i=1}^{k} (|N_i| - 1)} \leq 1 + \frac{k}{(\sum_{i=1}^{k} |N_i|) - k}$$

Hence let us assume we start with a (c,ϵ)-dense instance with no edges between teminals. Each pick of a star reduces the cardinality of $\mathcal{C}_a$ by a factor ϵ. Let $\mathcal{C}_a(i)$ denote the set $\mathcal{C}_a$ after i picks of a star, then $|\mathcal{C}_a(i)| \leq (1-\epsilon)^i |S|$. We obtain $|\mathcal{C}_a(k)| < c \cdot \log(|S|) \cdot K$ for $k \geq \frac{|S|}{c \cdot \log(|S|) \cdot K} \cdot \frac{1}{\log(1/(1-\epsilon))}$, hence we assume $k \leq \frac{|S|}{c \cdot \log(|S|) \cdot K} \cdot \frac{1}{\log(1/(1-\epsilon))} + 1$. Since

$$\sum_{i=1}^{k} |N_i| \geq \sum_{i=1}^{k} (1-\epsilon)^i \cdot \epsilon \cdot |S| = \epsilon \cdot |S| \cdot \sum_{i=1}^{k} (1-\epsilon)^i = \epsilon \cdot |S| \cdot \frac{1-(1-\epsilon)^k}{\epsilon} \quad (9.5)$$

we obtain the following bound for the approximation ratio of algorithm LDSTP:

$$\begin{aligned} \frac{\text{cost}(T_{LD})}{\text{cost}(T^{\star\star})} &\leq 1 + \frac{2k}{\sum_{i=1}^{k} |N_i|} \leq 1 + \frac{2k}{|S| \cdot (1-(1-\epsilon)^k)} \\ &\leq 1 + \frac{2 \cdot (\frac{|S|}{c \cdot \log(|S|) \cdot K} \cdot \frac{1}{\log(1/(1-\epsilon))} + 1)}{|S| \cdot (1-(1-\epsilon)^k)} \\ &\leq 1 + \frac{2 \cdot (\frac{|S|}{c \cdot \log(|S|) \cdot K} \cdot \frac{1}{\log(1/(1-\epsilon))} + 1)}{|S| \cdot \epsilon} \end{aligned} \quad (9.6)$$

since we may assume $k \geq 1$ (in case $k = 0$ algorithm LDSTP computes an optimum solution, namely a spanning tree for S). Using $\frac{1}{|S|} \leq \frac{1}{\log(|S|)}$ we obtain

$$\frac{\text{cost}(T_{LD})}{\text{cost}(T^{\star\star})} \leq 1 + \frac{1}{\epsilon} \cdot \left(\frac{1}{c \cdot K \cdot \log(\frac{1}{1-\epsilon})} + 1 \right) \cdot \frac{1}{\log(|S|)} \quad (9.7)$$

For given $\delta > 0$ we will now choose K such that the approximation ratio is bounded by $1+\delta$, i.e.

$$K \geq \left(c \cdot \log\left(\frac{1}{1-\epsilon} \right) \cdot (\epsilon \cdot \delta \cdot \log(|S|) - 1) \right)^{-1} \quad (9.8)$$

Hence choosing $K = \left(c \cdot \log\left(\frac{1}{1-\epsilon} \right) \right)^{-1}$, solving the Steiner Tree instance exactly by brute force for $|S| \leq 2^{2/(\epsilon \cdot \delta)}$ and applying LDSTP for all other instances yields a PTAS for the (c,ϵ)-log-dense Steiner Tree Problem. □

9.5 The Dense Steiner Forest Problem

In this section we will consider the ϵ-Dense Steiner Forest Problem. Currently we are not able to provide a PTAS for this problem, for the following reason: All the variants of the methods of [KZ97a] we have discussed so far (and will discuss in subsequent sections) are based on the approach of performing greedy steps until the problem size is sufficiently small and then applying some exact algorithm for the remaining instance. In the Steiner Forest Case the kind of greedy steps we have in mind reduce each single terminal set to constant size, but the number of terminal sets might not be reduced at all. On the other hand we do not know how to justify contraction steps that reduce the number of terminal sets, since melting j of them into a single terminal set might produce an additive cost of j. However we will now give an approximation algorithm for the Dense Steiner Forest Problem with approximation ratio $1+O((\sum_{i=1}^{n}\log(|S_i|))/(\sum_{i=1}^{n}|S_i|))$, where $S_1,\ldots S_n$ are the given terminal sets. Intuitively this provides good approximation in case sufficiently many terminal sets are large, and we will make this precise in this section.

Definition 43 *An instance $G=(V,E),S_1,\ldots,S_n$ of the SFP is called* **ϵ-dense'** *iff for all $1\leq i\leq n$ and $S'\subseteq S_i$ there exists a vertex $v\in V\setminus S_i$ such that $|N(v)\cap S'|\geq\epsilon\cdot|S'|$.*

Lemma 9.5.1 *For every $\epsilon>0$, every ϵ-dense instance of the SFP is ϵ-dense'.*

Proof: Let $G=(V,E),S_1,\ldots,S_n$ be ϵ-dense, let $i\in\{1,\ldots,n\}$ and $S'\subseteq S_i$. Then for all $s\in S'$ it holds $|N(s)\cap(V\setminus S_i)|\geq\epsilon\cdot|V\setminus S_i|$.
From $\sum_{v\in V\setminus S_i}|N(v)\cap S'|=\sum_{s\in S'}|N(s)\cap(V\setminus S_i)|\geq|S'|\cdot\epsilon\cdot|V\setminus S_i|$ we conclude that there exists at least on $v\in V\setminus S_i$ such that $|N(v)\cap S'|\geq\epsilon\cdot|S'|$. □

Algorithm A_k:
Input: $G=(V,E),\mathcal{S}:=\{S_1,\ldots,S_n\}\subseteq P(V)$ instance of the ϵ-Dense SFP
Output: Set of edges $F\subseteq E$ defining a Steiner Forest for $S_1,\ldots,S_n$

(0) Let $F:=\emptyset$ and $S_{i,act}:=S_i, 1\leq i\leq n$.
(1) **while** $\max_{1\leq i\leq n}|S_{i,act}|\geq k$ **do**
  Pick $i\in\{1,\ldots,n\}$ and $v\in V\setminus S_{i,act}$ such as to maximize $|N(v)\cap S_{i,act}|$.
  Let $\tilde{S}:=N(v)\cap S_{i,act}$ and $F:=F\cup\{\{v,s\}:s\in\tilde{S}\}$.
  $S_{i,act}:=S_{i,act}\setminus\tilde{S}$. Contract $\tilde{S}\cup\{v\}$.
(2) Solve the remaining instance using the Primal-Dual algorithm.

Lemma 9.5.2 *At the beginning of every call of the while-loop the sets $S_{i,act}$ are ϵ-dense'.*

Proof: The initial sets $S_{i,act}=S_i$ are ϵ-dense and therefore ϵ-dense'. Since in every iteration the removed set $\tilde{S}$ does not contain elements from $\bigcup_{s\in S_{i,act}\setminus\tilde{S}}N(s)\setminus S_{i,act}$, for every subset S' of $S_{i,act}\setminus\tilde{S}$ existence of a vertex v in $V\setminus S_{i,act}$ and hence in $V\setminus S'$

with many neighbours in S' is not disturbed. □

Analysis of A_k. We will use the following well known lower bound for the cost of an optimum solution:

$$\text{opt}(G, \mathcal{S}) \;\geq\; L \;:=\; \sum_{i=1}^{n} (|S_i| - 1).$$

Let us now give an estimate for the cost of the solution produced by algorithm A_k. For $1 \leq i \leq n$ let $j(i)$ denote the number of contractions of subsets of $S_{i,act}$ in phase (1) of the algorithm, and let $S_i^1, \ldots, S_i^{j(i)}$ be the subsets being contracted. Let $S_{i,rem} := S_i \setminus (S_i^1 \cup \ldots \cup S_i^{j(i)})$ the remaining set of S_i. Then the number of edges added to F in phase (1) is given by

$$\text{cost}_1 \;\;=\;\; \sum_{i=1}^{n} \sum_{l=1}^{j(i)} |S_i^l| \;=\; \sum_{i=1}^{n} (|S_i| - |S_{i,rem}|). \tag{9.9}$$

At the end of phase (1), for $1 \leq i \leq n$ the size of S_i is given by

$$s(i) \;\;:=\;\; j(i) + |S_i| - \sum_{l=1}^{j(i)} |S_i^l| \;=\; j(i) + |S_{i,rem}|. \tag{9.10}$$

Furthermore the size of $S_{i,act}$ after l contractions is bounded by $|S_i|(1-\epsilon)^l$, hence

$$s(i) \;\;\leq\;\; j(i) + |S_i|(1-\epsilon)^{j(i)}. \tag{9.11}$$

Hence $\text{cost}_2 :=$ the number of edges picked by the 2-Approximation Algorithm in phase (2) of A_k is bounded as follows:

$$\text{cost}_2 \;\leq\; 2 \cdot \sum_{i=1}^{n} \left(j(i) + |S_i|(1-\epsilon)^{j(i)} \right).$$

Let $x_{i,l} := |S_i^l|, 1 \leq i \leq n, 1 \leq l \leq j(i)$. An upper bound for the cost of solution generated by algorithm A_k is then given by the following optimization problem:

$$\begin{aligned}
& \max \sum_{i=1}^{n} \sum_{l=1}^{j(i)} x_{i,j} \;+\; 2 \cdot \left(|S_i| - \sum_{l=1}^{j(i)} x_{i,l} \;+ j(i) \right) \\
& \text{s.t.} \quad \sum_{l=1}^{j(i)} x_{i,l} \geq |S_i| \cdot \left(1 - (1-\epsilon)^{j(i)}\right),\; 1 \leq i \leq n \\
= \quad & \max \sum_{i=1}^{n} \left(2|S_i| - \sum_{l=1}^{j(i)} x_{i,l} \;+ 2j(i) \right) \\
& \text{s.t.} \quad \sum_{l=1}^{j(i)} x_{i,l} \geq |S_i| \cdot \left(1 - (1-\epsilon)^{j(i)}\right),\; 1 \leq i \leq n
\end{aligned}$$

Let $\text{save}_i := \sum_{l=1}^{j(i)} x_{i,l} - 2j(i).1 \leq i \leq n$. We give a lower bound for save_i as a function of $|S_i|$: Let $X = \sum_{l=1}^{j(i)} x_{i,l}$ the total number of terminals removed from S_i in phase (1) of algorithm A_k, then $X > (\epsilon|S_i| - k)/\epsilon = |S_i| - k/\epsilon$. Furthermore $k > \epsilon \cdot |S_i| \cdot (1-\epsilon)^{j(i)}$ from which we conclude

$$j(i) \quad < \quad \frac{\log\left(\frac{k}{\epsilon|S_i|}\right)}{\log(1-\epsilon)}. \tag{9.12}$$

Hence we get the following bound for the total cost of the solution generated by algorithm A_k:

$$\begin{aligned} \text{total cost} \quad &\leq \quad \sum_{i=1}^{n} \left(|S_i| + \frac{k}{\epsilon} + 2 \cdot \frac{\log\left(\frac{k}{\epsilon|S_i|}\right)}{\log(1-\epsilon)} \right) = \sum_{i=1}^{n} \left(|S_i| + \frac{k}{\epsilon} + 2 \cdot \frac{\log\left(\frac{\epsilon|S_i|}{k}\right)}{\log\left(\frac{1}{1-\epsilon}\right)} \right) \\ &=: \quad tc(\epsilon, k) \end{aligned} \tag{9.13}$$

Since $\frac{d}{dk} tc(\epsilon,k) = n \cdot \left(\frac{1}{\epsilon} - \frac{2}{\log\left(\frac{1}{1-\epsilon}\right)} \cdot \frac{1}{k} \right) = 0$ for $k^* = \frac{2\epsilon}{\log\left(\frac{1}{1-\epsilon}\right)}$ and $\frac{d^2}{dk^2} tc(\epsilon, k^*) > 0$, we choose $k = k^*$ and finally obtain

Theorem 50 *For each $\epsilon > 0$ there is a polynomial time approximation algorithm for the ϵ-Dense Steiner Forest Problem with approximation ratio $1 + O\left(\frac{\sum_{i=1}^{n} \log(|S_i|)}{\sum_{i=1}^{n} |S_i|}\right)$.*

9.6 The Dense Prize Collecting Steiner Tree Problem

In this section we describe a simple polynomial time approximation scheme for the ϵ-Dense Prize Collecing Stiener Tree Problem which uses the polynomial time approximation scheme $\mathcal{A}_{KZ}$ for the ϵ-Dense STP by Karpinski and Zelikovsky as a subroutine, based on the following fact from [KZ97a].

Theorem 51 *[KZ97a] Given an ϵ-dense instance $G = (V, E), S \subseteq V$ of the Steiner Tree Problem and some $\delta > 0$, $\mathcal{A}_{KZ}$ constructs a Steiner tree T for S in G such that $c(T) \leq (1+\delta) \cdot (|S| - 1)$.*

We use this fact by sorting terminals in decreasing order with respect to prizes and then calling algorithm $\mathcal{A}_{KZ}$ for initial segments of the sorted sequence:

Algorithm $\mathcal{A}_{Prize}$
Input: $G = (V, E), S \subseteq V, p\colon S \to \mathbb{Q}_+,\ \delta > 0$
Output: tree T connecting a subset S_T of S in G
Let $S = \{s_1, \ldots, s_n\}$ such that $p(s_1) \geq \ldots \geq p(s_n)$
For $i = 0, \ldots, n$ **do**
  Construct a $(1+\delta)$- approximative Steiner Tree T_i for $S_i := \{s_{i+1}, \ldots, s_n\}$ in G using the Karpinski-Zelikovsky algorithm.
Return the solution T_j with best value $c(T_j) + \sum_{l=1}^{j} p(s_l)$.

Lemma 9.6.1 *The algorithm $\mathcal{A}_{Prize}$ is a ptas for the ϵ-Dense Prize Collecting Steiner Tree Problem.*

Proof: Let $T^\star$ denote the optimum prize collecting solution and let $S^\star \subseteq S$ be the set it connects, then the cost of this solution is given by

$$c^\star := c(T^\star) + \sum_{s \in S \setminus S^\star} p(s) \geq |S^\star| - 1 + \sum_{s \in S \setminus S^\star} p(s) \geq |S^\star| - 1 + \sum_{l=1}^{|S \setminus S^\star|} p(s_l).$$

Note that each subset $S_i = \{s_{i+1}, \ldots, s_n\}$ of S is ϵ-dense, hence $c(T_i) \leq (1+\delta) \cdot (|S_i| - 1) = (1+\delta) \cdot (n - i - 1)$ due to theorem 51. For $i = |S \setminus S^\star|$ the prize collecting cost of T_i is

$$c(T_i) + \sum_{l=1}^{i} p(s_l) \leq (1+\delta)(|S^\star| - 1) + \sum_{l=1}^{|S \setminus S^\star|} p(s_l) \leq (1+\delta) \cdot c^\star.$$

□

9.7 The Dense k-Steiner Tree Problem

Another well known generalization of the Steiner Tree Problem is the *k-Steiner Tree Problem* where we are given an instance of the Steiner Tree Problem and a number k and are asked to construct a minimum cost tree connecting at least k elements from the terminal set. Note that k does not have to be constant or logarithmic in the number of terminals. Here we consider the ϵ-dense version of this problem:

ϵ-Dense k-Steiner Tree Problem

Instance: Graph $G = (V, E)$ with terminal set $S \subseteq V$ such that each terminal has at least $\epsilon \cdot |V \setminus S|$ neighbours in $V \setminus S$, a number $k \in \{1, \ldots, |S|\}$.

Solution: Tree T in G connecting at least k terminals from S

Cost: Minimize the number of edges in T.

Lemma 9.7.1 *There is a polynomial time approximation scheme for the ϵ-Dense k-Steiner Tree Problem.*

Proof: Let G be the graph and S the terminal set. Pick an arbitrary subset S' of S of size k. Then S' is ϵ-dense in G, hence the Karpinski-Zelikovsky algorithm $\mathcal{A}_{KZ}$ applied to G, S', δ returns a Steiner Tree for S' in G of cost at most $(1+\delta) \cdot (|S'| - 1) = (1+\delta)(k-1)$. Since $k-1$ is a lower bound for the optimum cost, this method yields a ptas for the ϵ-Dense k-STP. □

9.8 The Dense Class Steiner Tree Problem

The Class Steiner Tree Problem is an important generalization of the Steiner Tree Problem: Given a graph or finite metric space and a system of pairwise disjoint subsets $S_1, \ldots, S_n$ of the vertex set V (called *classes*), the task is to construct a min-cost tree containing at least one node from every class. Interestingly, although being defined on undirected graphs or metric spaces, this problem is at least as hard to approximate as the well-known Set Cover Problem, by a straight-forward reduction due to Ihler. Nevertheless, for an ϵ-dense version of the problem we are able to give a polynomial-time approximation scheme. Note that for other important special cases of the problem the complexity status is open. For example it is not known whether a PTAS exists for the Geometric Class Steiner Tree Problem (i.e. point sets in some $\mathbb{R}^d$ for constant d and some L_p-norm). This section is organized as follows: In subsection 9.8.1 we give a formal definition of the problem and briefly describe well-known upper and lower bounds for the general case. In subsection 9.8.2 we describe a PTAS for the ϵ-Dense Class Steiner Tree Problem.

9.8.1 Introduction

Let us start with a formal definition of the problem:

Class Steiner Tree Problem

Instance: A graph $G = (V, E)$ with edge weights $c\colon E \to \mathbb{R}_+$ and pairwise disjoint classes $S_1, \ldots, S_n \subseteq V$.

Solution: A connected subgraph T of G containing at least one vertex from every class.

Cost: the cost $c(T) = \sum_{e \in E(T)} c(e)$ of T.

The following straight-forward hardness result was first observed by Ihler and can be found as well in [GKR98].

Lemma 9.8.1 *There is an L-reduction from the Set Cover Problem to the Class Steiner Tree Problem with parameters $\alpha = \beta = 1$.*

Corollary 9.8.1 *Unless $NP \neq P^{[\log n \log\log n]}$ there is no polynomial time approximation algorithm with ratio $(1-\epsilon)\cdot\log(n)$ for the Class Steiner Tree Problem.*

Proof of Lemma 9.8.1: For a given instance $(U, \mathcal{S})$ of the Set Cover Problem with universe $U = \{u_1, \ldots, u_n\}$ and set system $\mathcal{S} = \{S_1, \ldots, S_m\}$ we define an instance of the Class Steiner Tree Problem as follows: For each set S_i we introduce a vertex s_i and for each element u_j of S_i a vertex $u_{i,j}$ being connected to s_i by an edge of cost 0 (note that for occurrences of an element $u_j \in U$ in different sets S_i and S_l the vertices $u_{i,j}$ and $u_{l,j}$ are distinct). Furthermore there is an extra vertex r and each set vertex

s_i is connected to r by an edge of cost 1. For each element u_j of U there is a class $\{u_{i,j} : u_j \in S_i\}$ and furthermore the vertex r builds a single-element class $\{r\}$. Each solution $S_i, i \in J$ for some $J \subseteq \{1, \ldots, m\}$ of cost $|J|$ directly gives us a solution of the same cost for the Class Steiner Tree instance (connect each chosen set to all its element vertices and to the root), hence $\alpha = 1$. Indeed here we have equality of the optimum values. Furthermore, let T be an abritrary solution to the Class Steiner Tree instance, then T contains r and a subset $\{s_i : i \in J\}$ of the set vertices. Then the sets $S_i, i \in J$ form a set cover of cost $|J|$ = cost of T, hence $\beta = 1$. □

Fortunately, Garg and Konjevod [GKR98] were able to obtain a randomized approximation algorithm with polylogarithmic performance ratio.

9.8.2 A PTAS for the ϵ-Dense Case

In this subsection we consider an ϵ-dense version of the Class Steiner Tree Problem which is a direct generalization of the ϵ-Dense Steiner Tree Problem considered by Karpinski and Zelikovsky in [KZ97a] (cf. section 9.3).

ϵ-Dense Class Steiner Tree Problem

Instance: Graph $G = (V, E)$, pairwise disjoint classes $S_1, \ldots, S_n \subseteq V$ such that for every $s \in S := \bigcup_{i=1}^{n} S_i$

$$|N(s) \setminus S| \geq \epsilon \cdot |V \setminus S|.$$

Solution: Subtree T of G containing at least one node from every class S_i.

Cost: $c(T) :=$ number of edges of T.

Let us call a class S_i *neighbour* of a vertex v if $S_i \cap N(v)$ is not empty. The idea of our algorithm is as follows: The density condition directly implies existence of a vertex $v \in V \setminus S$ which has many classes in its neighbourhood. Now in the first phase of our algorithm we perform the same kind of picks as in our Dense Steiner Forest algorithm (cf. section 9.5): We maintain a set of *active classes*, starting with every class S_i being active. We pick a vertex v with many classes as neighbours and for each such class S_i which is a neighbour of v an element $s_i \in S_i \cap N(v)$. We contract these vertices, declare the according classes inactive and iterate. This guarantees that in every iteration we find a vertex v with many active classes in its neighbourhood. At the end of the first phase of the algorithm we are left with a constant number of active classes S_i (i.e. classes not being involved in any contraction so far) and a logarithmic number of supervertices $\int_i$ resulting from contractions. The task is to choose representatives s_i of the active classes S_i and to construct a Steiner Tree for the terminal set consisting of the choosen representatives s_i and the supervertices $\int_j$. Note that there are only $\prod_{i:S_i \text{ active}} |S_i| = |V|^{O(1)}$ many different choices of representatives, and for each choice we are left with an instance of the Steiner Tree Problem with logarithmic number of terminals, which can be solved optimally using the primal-dual

approach (cf. the remark in section 9.5). Since we have picked only logarithmic (in the number of classes) many Steiner points and the trivial lower bound is number of classes minus 1, this algorithm yields a PTAS for the ϵ-Dense Class Steiner Tree Problem. We are now ready to give a detailed description and analysis of our algorithm.

Algorithm $\mathcal{A}_{DCL}$ (DENSE CLASS STEINER)
Input: **Graph $G = (V, E)$, classes $S_1, \ldots, S_n$, $\delta > 0$**
Output: **$(1+\delta)$-approximate Class Steiner Tree T**
Initialization:
$\mathcal{C}_a := \{S_1, \ldots, S_n\}$ (set of active classes)
Choose $k := \frac{2}{\epsilon}$
Choose $k_0 := k_0(\delta)$ (will be described in the analysis)
If $n \leq k_0$ **then**
For each system of representatives $s_i \in S_i, 1 \leq i \leq n$
Compute an optimum Steiner Tree for the instance
consisting of graph G and terminal set $\{s_1, \ldots, s_n\}$.
Return the best upon those Steiner Trees.
Phase 1:
while $|\mathcal{C}_a| \geq k$ **do**
Pick $v \in V \setminus S$ maximizing the cardinality of
$N(v, \mathcal{C}_a) = \{S_i \in \mathcal{C}_a : S_i \text{ is neighbour of } v\}$.
(Store and contract.)
Let $\mathcal{S}$ be the set of all supervertices constructed in phase 1.
Phase 2:
For each choice of representatives $s_i \in S_i$ $(S_i \in \mathcal{C}_a)$
Compute an optimum Steiner Tree T for the terminal set
$\{s_i : S_i \in \mathcal{C}_a\} \cup \mathcal{S}$.
$T_o :=$ a min-cost tree upon those constructed in the for-loop.
Return T_o.

Analysis. The density condition implies that every contraction in phase 1 of the algorithm reduces the set of active classes by an ϵ-fraction. Let $\mathcal{C}_a(i)$ be the set of active classes after i contractions and t be the total number of contractions in phase 1. Then $|\mathcal{C}_a(i)| \leq (1-\epsilon)^i \cdot n$, hence we have $|\mathcal{C}_a(i)| \leq k$ for $i \geq \log(n/k)/\log(1/(1-\epsilon))$, hence

$$t \leq 2 \cdot \log\left(\frac{n}{k}\right) / \log\left(\frac{1}{1-\epsilon}\right) =: t_u \quad \text{(upper bound for } t\text{)} \tag{9.14}$$

(we assume $n \geq k_0 \geq 2k \cdot \frac{1}{1-\epsilon}$, this implies
$2\log(n/k)/\log(1/(1-\epsilon)) \geq \log(n/k)/\log(1/(1-\epsilon)) - 1$).

Since $|\mathcal{S}| = t = O(\log(n))$ and $|\mathcal{C}_a| \leq k = O(1)$ at the end of phase 1, there are only $\prod_{S_i \in \mathcal{C}_a} |S_i| = O\left(|V|^k\right)$ many choices of systems of representatives for the classes $S_i \in \mathcal{C}_a$. Hence each terminal set considered in the for-loop in phase 2 has size bounded by $k + t = O(\log(n))$, and the according Steiner Tree can be computed in polynomial

time using the Dreyfus-Wagner algorithm. This establishes polynomial running time of the algorithm for fixed k.

We will now compute a bound on the approximation ratio of the tree T_o computed by algorithm $\mathcal{A}_{DCL}$. Let $T^\star$ denote an optimum Class Steiner Tree for $S_1 \ldots, S_n$ in G. Note that T_o consist of a set of stars $ST_1, \ldots, ST_t$ generated by contractions in phase 1 and a set of edges T_0 connecting these stars and certain representatives $s_o(j)$ for the remaining classes S_j. Now construct graph G' as follows: Start with graph G and add the following edges:

(1) for each star ST_i edges forming a spanning tree for the terminals of ST_i,

(2) for each inactive class S_l (i.e. class being involved in a contraction in phase 1) edges from the terminal $s_l \in S_l$ which was picked to all neighbours of elements of class S_l in G.

Note that $\mathrm{OPT}(G') \leq \mathrm{OPT}(G)$. An optimum class Steiner tree $T^\star_{G'}$ for classes $S_i, 1 \leq i \leq n$ in graph G' can be obtained as a set of spanning trees T'_i for the sets of terminals R_i of stars ST_i, a set of representatives f_j for the still active classes S_j and a Steiner tree T' connecting these. Note that the cost of T_0 is bounded by the cost of T'. Hence we obtain

$$\begin{aligned}\frac{\mathrm{cost}(T_o)}{\mathrm{cost}(T^\star)} &\leq \frac{\mathrm{cost}(T_o)}{\mathrm{cost}(T^\star_{G'})} = \frac{\mathrm{cost}(T_0) + \sum_{i=1}^t \mathrm{cost}(ST_i)}{\mathrm{cost}(T') + \sum_{i=1}^t \mathrm{cost}(T'_i)} \leq \frac{\sum_{i=1}^t \mathrm{cost}(ST_i)}{\sum_{i=1}^t \mathrm{cost}(T'_i)} \\ &= \frac{\sum_{i=1}^t |R_i|}{\sum_{i=1}^t (|R_i| - 1)} = \frac{\sum_{i=1}^t |R_i|}{\left(\sum_{i=1}^t |R_i|\right) - t} = \frac{n - |\mathcal{C}_a|}{n - |\mathcal{C}_a| - t} \leq \frac{n-k}{n-k-t}\end{aligned}$$

We know that for $t \geq t_u$ $|\mathcal{C}_a| \leq k$ at the end of phase 1 of the algorithm. Hence we can upperbound $\frac{n-k}{n-k-t}$ by $\frac{n-k}{n-k-t_u}$. We want to choose k and k_0 such that (a) the upper bound estimate t_u in (9.14) is valid and (b) the approximation ratio is at most $1+\delta$:

$$\begin{aligned}\frac{n-k}{n-k-t_u} \leq 1+\delta \quad &\Longleftrightarrow \quad 0 \leq \delta(n-k) - (1+\delta) \cdot \frac{2 \cdot \log(n/k)}{\log(1/(1-\epsilon))} \\ &\Longleftrightarrow \quad \delta k - \frac{2 \cdot (1+\delta)}{\log(1/(1-\epsilon))} \cdot \log(k) \leq \delta n - \frac{2 \cdot (1+\delta)}{\log(1/(1-\epsilon))} \cdot \log(n)\end{aligned}$$

for $n > k_0$. Since the function

$$f(x) := \delta \cdot x - \frac{2 \cdot (1+\delta)}{\log(1/(1-\epsilon))} \cdot \log(x)$$

is unbounded and monotone increasing for $x > \frac{2 \cdot (1+\delta)}{\delta \cdot \log(1/(1-\epsilon))}$, we also have to assure that $k \geq (2 \cdot (1+\delta))/(\delta \cdot \log(1/(1-\epsilon)))$. Hence we choose

$$k = \max\left\{\frac{2}{\epsilon}, \frac{2 \cdot (1+\delta)}{\delta \cdot \log(1/(1-\epsilon))}\right\}, \quad k_0 = 2k \cdot \frac{1}{1-\epsilon} \tag{9.15}$$

and finally obtain

Theorem 52 *Algorithm $\mathcal{A}_{DCL}$ with the choices (9.15) for k and k_0 is a PTAS for the ϵ-Dense Class Steiner Tree Problem.*

Bibliography

[AFK96] Sanjeev Arora, Alan Frieze, and Haim Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. In *37th Annual Symposium on Foundations of Computer Science*, pages 21–30, 1996.

[AKK95] Sanjeev Arora, David R. Karger, and Marek Karpinski. Polynomial time approximation schemes for dense instances of NP -hard problems. In *ACM Symposium on Theory of Computing*, pages 284–293, 1995.

[AKR91] Ajit Agrawal, Philip N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. In *ACM Symposium on Theory of Computing*, pages 134–144, 1991.

[All99] Eric Allender. Levin's lower bound theorem. Technical report, (unpublished, online available on E. Allender's homepage), 1999.

[Aro97] Arora. Nearly linear time approximation schemes for euclidean TSP and other geometric problems (one page only). In *RANDOM: International Workshop on Randomization and Approximation Techniques in Computer Science*. LNCS, 1997.

[Aro98] Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

[BBF95] V. Bafna, P. Berman, and T. Fujito. Constant ratio approximations of the weighted feedback vertex set problem for undirect graphs. In J. Staples, P. Eades, N. Katoh, and A. Moffat, editors, *ISAAC Algorithms and Computation*, pages 142–151, Berlin„ 1995. Springer.

[BD97] A. Borchers and D.-Z. Du. The k-steiner ratio in graphs. *SIAM J. Computing 26*, pages 857–869, 1997.

[Ber01] P. Berman. Personal communication, 2001.

[BF99] C. Bazgan and W. Fernandez de la Vega. A polynomial time approximation scheme for dense Min2SAT. In *Proc. 12th Int. Symp. on Fundamentals of*

Computation Theory, number 1684 in Lecture Notes in Comput. Sci., pages 91–99. Springer-Verlag, 1999.

[BfHW00] H.L. Bodlaender, M.R. fellows, M.T. Hallet, and H.T. Wareham. The hardness of perfect phylogeny, feasible register assignment and other problems on thin colored graphs. *Theoretical Computer Science*, 244:167–188, 2000.

[BG94] Richard Beigel and Judy Goldsmith. Downward separation fails catastrophically for limited nondeterminism classes. In *Structure in Complexity Theory Conference*, pages 134–138, 1994.

[BK98a] P. Berman and M. Karpinski. On some tighter inapproximability results. In *Electronic Colloquium on Computational Complexity (ECCC) 5(29)*, 1998.

[BK98b] P. Berman and M. Karpinski. On some tighter inapproximability results, further improvements. In *Electronic Colloquium on Computational Complexity (ECCC) 5(65)*, 1998.

[BK03] P. Berman and M. Karpinski. Improved approximation lower bounds on small occurrence optimization. In *Electronic Colloquium on Computational Complexity (ECCC) 10(008)*, 2003.

[Blu67] M. Blum. A machine-independent theory of the complexity of recursive functions. *Journal of the ACM*, 14(2):322–336, 1967.

[Bor69] A. Borodin. Complexity classes of recursive functions and the existence of complexity gaps. In *STOC*, 1969.

[Bor72] A. Borodin. Computational complexity and the existence of complexity gaps. *Journal of the ACM*, 19(1):158–174, 1972.

[BP89] M. Bern and P. Plassman. The steiner tree problem with edge lengths 1 and 2. *Information Processing Letters 32*, pages 171–176, 1989.

[BR94] Piotr Berman and V. Ramaiyer. Improved approximations for the steiner tree problem. *Journal of Algorithms*, pages 381–408, 1994.

[BY98] R. Bar-Yehuda. One for the price of two: A unified approach for approximating covering problems. *To appear in Algorithmica*, 1998.

[CC97] L. Cai and J. Chen. On fixed-parameter tractability and approximability of np optimization problems. *Journal of Computer and System Sciences*, 54(3):465–474, 1997.

[CC02] M. Chlebikov and J. Chlebikova. Approximation hardness of the steiner tree problem on graphs. In *Proceedings of 8th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 170–179. LNCS 2368, 2002.

[CCC+98] Moses Charikar, Chandra Chekuri, Toyat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, , and Ming Li. Approximation algorithms for directed steiner problems. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 192–200, January 1998.

[CCDF95] L. Cai, J. Chen, R.G. Downey, and M.R. Fellows. On the structure of parameterized problems in np. *Inf. Computing*, 123(1):38–49, 1995.

[CGH+88] J. Cai, T. Gundermann, J. Hartmanis, L.A. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy i: Structural properties. *SIAM Journal on Computing*, 17(6):1232–1252, 1988.

[CKK+99] M. Charikar, J. Kleinberg, R. Kumar, S. Rajagopalan, A. Sahai, and A. Tomkins. Minimizing wirelength in zero and bounded skew clock trees. In *Proc. 10th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 177–184, 1999.

[CKST96] Pierluigi Crescenzi, Viggo Kann, Riccardo Silvestri, and Luca Trevisan. Structure in approximation classes. *Electronic Colloquium on Computational Complexity (ECCC)*, 3(066), 1996.

[CP91] P. Crescenzi and A. Panconesi. Completeness in approximation classes. *Inform. and Comput.*, 93:241–262, 1991.

[CT97a] M. Cesati and L. Trevisan. On the efficiency of polynomial time approximation schemes. *Information Processing Letters*, 64(4):165–171, 1997.

[CT97b] Marco Cesati and Luca Trevisan. On the efficiency of polynomial time approximation schemes. *Electronic Colloquium on Computational Complexity (ECCC)*, 4(1), 1997.

[CT00] Pierluigi Crescenzi and Luca Trevisan. On approximation scheme preserving reducibility and its applications. *Theory of Computing Systems*, 33(1):1–16, 2000.

[DF92] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness. In *Congr. Num. 87*, pages 161–187, 1992.

[DF93] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness iii: Some structural aspects of the w-hierarchy. In *Complexity Theory: Current Research*, pages 166–191, 1993.

[DF95a] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness i: Basic theory. *SIAM Journal on Computing*, 24:873–921, 1995.

[DF95b] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness ii: On completeness for w[1]. *Theoretical Computer Science*, 141:109–131, 1995.

[dlVK00] W. Fernanadez de la Vega and Marek Karpinski. Polynomial time approximation of dense weighted instances of MAX-CUT (revised version). Technical Report 85214-CS, 2000.

[FBL96] Fernández-Baca and Lagergren. On the approximability of the steiner tree problem in phylogeny. In *ISAAC*, pages 65–74, 1996.

[Fei98] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[FK99] W. Fernandez de la Vega and M. Karpinski. On the approximation hardness of dense TSP and other path problems. *Information Processing Letters*, 70(2):53–55, 1999.

[GGP+94] M.X. Goemans, A.V. Goldberg, S.A. Plotkin, D.B. Shmoys, E. Tardos, and D.P. Williamson. Improved approximation algorithms for network design problems. In *Symposium on Discrete Algorithms*, pages 223–232, 1994.

[GKR98] Garg, Konjevod, and Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[GW92] Goemans and Williamson. A general approximation technique for constrained forest problems. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1992.

[Hak71] S.L. Hakimi. Steiner's problem in graphs and its implications. *Networks*, 1:113–133, 1971.

[Has97] J. Hastad. Some optimal inapproximability results, 1997.

[HP99] S. Hougardy and H.-J. Prömel. A 1.598 approximation algorithm for the steiner problem in graphs. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[HRS00] Refael Hassin, R. Ravi, and F. S. Salman. Approximation algorithms for a capacitated network design problem. In *APPROX*, pages 167–176, 2000.

[Jai98] Kamal Jain. Factor 2 approximation algorithm for the generalized steiner network problem. In *IEEE Symposium on Foundations of Computer Science*, pages 448–457, 1998.

[JMVW99] Jain, Mandoiu, Vazirani, and Williamson. A primal-dual schema based approximation algorithm for the element connectivity problem. In *Symposium on Discrete Algorithms*, 1999.

[Kad88] J. Kadin. The polynomial time hierarchy collapses if the booelan hierarchy collapses. *SIAM Journal on Computing*, 17(6):1263–1282, 1988.

[Kar72] R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[Kar00] Marek Karpinski. Polynomial time approximation schemes for some dense instances of NP-hard optimization problems (extended version). Technical Report 85213-CS, 2000.

[KF84] C.M.R. Kintala and P. Fischer. Refining nondeterminism in relativized complexity classes. *SIAM Journal on Computing*, 13:329–337, 1984.

[Kha79] L.G. Khachiyan. A polynomial algorithm in linear programming (in russian). *Doklady Akademii Nauk SSSR*, 244:1093–1096, 1979.

[KM96] Sanjeev Khanna and Rajeev Motwani. Towards a syntactic characterization of PTAS. pages 329–337, 1996.

[KMSV94] Sanjeev Khanna, Rajeev Motwani, Madhu Sudan, and Umesh V. Vazirani. On syntactic versus computational views of approximability. In *IEEE Symposium on Foundations of Computer Science*, pages 819–830, 1994.

[KST96] Sanjeev Khanna, Madhu Sudan, and Luca Trevisan. Constraint satisfaction: The approximability of minimization problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 3(064), 1996.

[KZ97a] Marek Karpinski and Alexander Zelikovsky. Approximating dense cases of covering problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 4(004), 1997.

[KZ97b] Marek Karpinski and Alexander Zelikovsky. New approximation algorithms for the steiner tree problems. *Journal of Combinatorial Optimization*, 1:47–65, 1997.

[KZ97c] Marek Karpinski and Alexander Zelikovsky. New approximation algorithms for the steiner tree problems. *Journal of Combinatorial Optimization*, 1:47–65, 1997.

[Lev71] A. Y. Levin. Algorithm for the shortest connection of a group of graph vertices. *Sov. Math. Dokl.*, 12:1477–1481, 1971.

[Lev74] L.A. Levin. Computational complexity of functions. *Complexity of Algorithms and Computations, Mir (Moscow)*, pages 174–185, 1974.

[Lev96] L.A. Levin. Computational complexity of functions. *Theoretical Computer Science*, 157(2):267–271, 1996.

[Mit99] Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999.

[MM69] E.M. McCreight and A.R. Meyer. Classes of computable functions defined by bounds on computation. In *STOC*, 1969.

[PY91] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[PY93] C.H. Papadimitriou and M. Yannakakis. On limited nondeterminsm and the complexity of the v-c dimension. In *Proceedings 8th Structure in Complexity Theory Conference*, pages 12–18, 1993.

[Rav94] R. Ravi. A primal-dual approximation algorithm for the steiner forest problem. *Information Processing Letters*, 50(4):185–190, 1994.

[RMR+01] R. Ravi, Madhav V. Marathe, S. S. Ravi, Daniel J. Rosenkrantz, and Harry B. Hunt III. Approximation algorithms for degree-constrained minimum-cost network-design problems. *Algorithmica*, 31(1):58–78, 2001.

[RS] S. Rao and W. Smith. Improved approximation schemes for geometrical graphs via.

[RW95] Ravi and Williamson. An approximation algorithm for minimum-cost vertex-connectivity problems. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1995.

[RZ00a] G. Robins and A. Zelikovsky. Improved steiner tree approximation in graphs. In *Proc. 11th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 770–779. ACM-SIAM, 2000.

[RZ00b] Gabriel Robins and Alexander Zelikovsky. Improved steiner tree approximation in graphs. In *Symposium on Discrete Algorithms*, pages 770–779, 2000.

[Sho77] N.Z. Shor. Cut-off method with space extension in convex programming problems. *Cybernetics*, 13:94–96, 1977.

[TFW99] Takeshita, Fujito, and Watanabe. On primal-dual approximation algorithms for several hypergraph problems. *IPSJ Math. Modeling and Problem Solving (23-3)*, 1999.

[Thi01] M. Thimm. On the approximability of the steiner tree problem. In *Proc. of the 26th Int. Symp., MFCS 2001, LNCS 2136*, pages 678–689, 2001.

[TM80] H. Takahashi and A. Matsuyama. An approximate solution for the steiner problem in graphs. *Math. Jap.*, 24:573–577, 1980.

[TR02] Eva Tardos and Tim Roughgarden. How bad is selfish routing ? *Journal of the ACM*, 49(2):236–259, 2002.

[Zel93] Alexander Zelikovsky. An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica*, 9:463–470, 1993.

[Zel95] Alexander Zelikovsky. Better approximation bounds for the network and euclidean steiner tree problems. *Manuscript*, 1995.

[Zel97] Alexander Zelikovsky. A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica*, 18(1):99–110, May 1997.

[ZM01] A. Zelikovsky and I. Mandoiu. Practical approximation algorithms for zero- and bounded-skew trees. In *Proc. 12th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 407–416, 2001.

Summary

In this thesis we study the approximation complexity of the Steiner Tree Problem and related problems as well as foundations in structural complexity theory. We give a survey on the Steiner tree Problem in chapter 5, obtaining lower bounds for approximability of the $(1, 2)$-Steiner Tree Problem by combining hardness results of [BK98a, BK98b] with reduction methods by [BP89].

We present approximation algorithms for the Steiner Forest Problem in graphs and bounded hypergraphs (section 6.4), for the Prize Collecting Steiner Tree Problem and related problems where prizes are given for pairs of terminals in sections 6.5-6.7. These results are based on the Primal-Dual method and the Local Ratio framework of Bar-Yehuda [BY98].

We study the Steiner Network Problem in chapter 7 and obtain combinatorial approximation algorithms with reasonable running time for two special cases, namely the Uniform Uncapacitated Case and the Prize Collecting Uniform Uncapacitated Case. For the general case, Jain [Jai98] gave an approximation algorithm based on the Ellipsoid Method due to [Kha79], resulting in very high running times.

We consider Directed Steiner Tree Problems in chapter 8, obtaining approximation schemes with ratio $O(n^\epsilon)$ and running time $O(n^{1/\epsilon})$ for the Directed Zero Skew Tree Problem and the Directed Weighted Path Problem. These results are based on methods developed for the Directed Steiner Tree Problem by Zelikovsky [Zel97] and Charikar et al. [CCC+98].

We consider Dense Steiner problems in chapter 8, obtaining polynomial time approximation schemes for the Dense Prize Collecting Steiner Tree Problem, Dense k-Steiner Problem and the Dense Class Steiner Tree Problem based on the methods of Karpinski and Zelikovsky for approximatong the Dense Steiner Tree Problem [KZ97b]. We also apply the methods to the log-dense Steiner Tree Problem, which can be seen as a step towards a PTAS for the Everywhere-Dense Steiner Tree Problem. For the Dense Steiner Forest Problem we obtain improved approximation ratio for the case when the number of terminals is sufficiently large compared to the number of terminal sets.

We consider Fixed Parameter Complexity in chapter 3. After giving an introduction into this field we consider structural aspects of the W-Hierarchy. We prove a Speedup Theorem for the classes FPT and SP. In section 3.4 we deal with Levin's Lower Bound Theorem [Lev96, Lev74]. We prove a version of the lower bound theorem for the class SP in section 3.4.2 and for Randomized Space Complexity in section 3.4.3.

We consider structural issues in chapter 4. We obtain a new type of PTAS-preserving reductions that provide some structure inside the class PTAS (section 4.1). We study the dependence of the running time of approximation schemes on the approximation ratio in section 4.2. We show existence of problems in PTAS for which no approximation scheme has running time recursive in ϵ. We separate PTAS from

EPTAS (section 4.3), assuming the existence of problems in NP with superpolynomial lower bound for deterministic time complexity. Separation under the weaker assumption $P \neq NP$ is unknown. Cesati and Trevisan [CT97a] obtained the same separation result under assumtion $FPT \neq W[P]$. In section 4.4 we construct a recursive oracle under which our assumption becomes true and theirs becomes false. This implies that using relativizing proof techniques one can not show that our assumption implies theirs. The oracle construction involves a counting argument in order to assure that when performing the next step of the oracle construction previously satisfied requirements do not get injured.