Background

Introduction

There is a growing number of practical problems within the areas of Discrete Optimization and Algorithmic Graph Theory that are NP-hard. This has led to an ever-increasing activity within these two intersecting areas of Discrete Mathematics and Theoretical Computer Science.

Within the Mathematics and Computer Science community it is widely believed that P ≠ NP. Under this assumption NP-hard problems cannot be solved in polynomial time. On the other hand, every problem in NP can be solved in exponential time by an exhaustive search. Although computers get faster and faster, in practical applications checking all possible solutions is too time-consuming, even if problem instances are of relatively small size. However, many NP-hard problems allow exponential time algorithms that are significantly faster than exhaustive search while still yielding an optimal solution. This explains the fast-growing interest in the design and analysis of these exact algorithms.

The current interest in exact algorithms is driven by both theoretical and practical reasons.

- Since P = NP seems very unlikely, exact algorithms that have exponential running time may be the best algorithms for obtaining optimal solutions for NP-hard problems. Whenever these problems require an optimal solution, approximation algorithms and other heuristics are not appropriate. By developing better exact algorithms (i.e., with a better running time) larger instances of NP-hard problems can be solved.

- In practice, even exponential time algorithms for problems in P may be worth studying. A “good” exponential time algorithm can run much faster than a poor polynomial time algorithm, especially if instance sizes are not too large. As an example, compare an algorithm of time complexity $O(1.1^n)$ with an algorithm of time complexity $O(n^4)$.

- From a theoretical point of view it would be worthwhile to study exact algorithms for some specific NP-hard problems. It is not known to what extent current exact algorithms can be improved. Investigating this for a number of well-known NP-hard problems is an interesting question by itself. It will also help us to gather more basic knowledge on these kinds of algorithms.

- In order to obtain a general theory that analyzes the behaviour of exact algorithms, many important theoretical questions must still be solved or further examined. How are algorithms for different problems related to each other? Some NP-hard problems currently have better exact algorithms than others; why? What is the theoretical explanation behind a number of empirical results on the performance of some exact algorithms?

Past and Current Work

Any decision problem $\Pi$ in NP can be solved by an exhaustive search: Let $x$ denote an instance of $\Pi$. Recall that membership in NP is equivalent to the existence of a polynomial time computable and polynomially bounded complexity parameter $m(x)$ together with a relation $R(x,y)$ that verifies in polynomial time whether a certificate $y$ of length $|y| \leq m(x)$ is a YES-certificate for instance $x$. Checking each certificate $y$ of length at most $m(x)$ yields an exact algorithm of time complexity $O^*(2^{m(x)}) = O(2^{m(x)}\text{poly}(|x|))$. Here, $\text{poly}(|x|)$ denotes some polynomial in $|x|$, the size of $x$. We use the notation $O^*$ to indicate that we suppress factors of order $\text{poly}(|x|)$.

In practice, we let the choice for the complexity parameter $m(x)$ depend upon the context. For graph theoretic problems, in which the input is a graph $G$ on $n$ vertices and $m$ edges, we often choose $m(x) = n$ or $m(x) = m$. As an example, consider the maximum independent set problem which asks for the largest subset $S \subseteq V$ of a given graph $G = (V, E)$ on $n$ vertices such that there is no edge between any two vertices in $S$. A brute force approach would simply check every subset of $V$ yielding an $O^*(2^n)$ time exact algorithm.

The first question is whether the trivial barrier of $2^{m(x)}$ can be brought down to a new barrier of $\alpha^{m(x)}$ with $\alpha < 2$ as small as possible. This question can be very hard. Consider for example the well-known travelling salesman problem (TSP). In this problem a travelling salesman has to make a tour through $n$ cities starting and returning to city 1 in such a way that the total travel distance is minimal. The best exact algorithm for TSP known so far [16] has $O^*(2^n)$ time complexity, and this result is from 1962!

Problems that can be solved in $O^*(\alpha^{m(x)})$ time for every $\alpha > 1$ are said to belong to the class SUBEXP of sub-exponential solveable problems (cf. [35]). As an example, we take the Euclidean travelling salesman problem. In this special case of TSP the $n$ cities are points in the Euclidean plane, and the distance between two cities is the Euclidean distance. Euclidean TSP is NP-hard [22]. Since it allows an $O^*(c^{\sqrt{n}\log n})$ time exact algorithm for some constant $c$ (see, e.g., [19]), Euclidean TSP lies in SUBEXP.

How might we prove that a problem in NP does not belong to SUBEXP? For this purpose we consider the class SNP, introduced by Papadimitriou and Yannakakis [28]. SNP is a subclass of NP and consists of many important
and well-studied problems, such as $k$-satisfiability (\textit{k-SAT}), maximum cut (\textsc{Max-Cut}), Hamiltonian circuit, vertex cover, that can be defined by means of a certain second order logical formula. For a precise definition we refer to \cite{28}. Impagliazzo, Paturi, and Zane \cite{21} introduced \textit{SERF-reductions (Sub-Exponential Reduction Family)} to describe a concept that preserves sub-exponential time complexities. They show that, analogously to classical complexity theory, there exist problems in SNP that can be considered to be the hardest: If there exists a sub-exponential algorithm that solves an SNP-complete problem under SERF-reductions, then any problem in SNP is solvable in sub-exponential time. It is widely believed that \textit{SNP $\subseteq$ SUBEXP} (cf. \cite{35}). Relative to this conjecture, it is sufficient to show that some problem is SNP-complete under SERF-reductions.

In \cite{21} it has been proven that $k$-colorability and \textit{k-SAT}, both for $k \geq 3$, are SNP-complete under SERF-reductions. By translating the corresponding NP-hard reductions into SERF-reductions Impagliazzo, Paturi, and Zane \cite{21} also show that vertex cover, independent set and several other NP-hard problems can not be solved in sub-exponential time, unless \textit{SNP $\subseteq$ SUBEXP}.

Especially for SNP-complete problems under SERF-reductions it is a natural question to ask for the lowest value of $\alpha > 1$, for which there exists an $O^*(\alpha^{m(x)})$ time exact algorithm (on the expectation that such an algorithm will not exist for all $\alpha > 1$). For \textit{3-SAT} the fastest exact algorithm so far has $O^*(1.48^n)$ time complexity \cite{5}, where $n$ is the number of variables in an instance $x$. For 3-colorability the fastest exact algorithm so far has $O^*(1.33^n)$ time complexity \cite{9}, where $n$ is the number of vertices in an input graph $G$.

We would like to point out that in general the choice for the parameter $m(x)$ plays a crucial role. However, for SNP-complete problems under SERF-reductions this is not the case \cite{21}. Consider, for example, the 3-SAT problem. For $m(x)$ we can take the number of clauses or the number of variables in an instance $x$. In \cite{21} it is shown that both options are equivalent: Whether or not 3-SAT belongs to \textsc{SUBEXP} does not depend on which of the two parameters we choose.

The state of the art for other well-known combinatorial problems includes a $O^*(2^n)$ time exact algorithm for the binary knapsack problem \cite{18}, where $n$ is the number of items, and an $O^*(1.22^n)$ time exact algorithm for the maximum independent set problem \cite{31}, where $n$ is the number of vertices. The latter algorithm uses exponential space. In \cite{4} an $O^*(1.23^n)$ time exact algorithm for the maximum independent set problem is given that uses polynomial space. Later on we will discuss results for other problems as well. For an excellent survey we refer to Woeginger \cite{35}.

Relation to Parameterized Complexity Theory

Exact algorithms are closely related to the field of parameterized complexity theory (cf. \cite{8}) that also studies the complexity of optimization problems relative to some parameter. There, one focuses on efficient solvability rather than (sub-)exponential time. However, the same algorithmic methods are applied in both areas. An increasing number of connections have been established between the two areas ranging from the design of practical algorithms to theoretical results.

Relation to Approximation Theory

The design and analysis of approximation algorithms for NP-hard problems employs different techniques and methods. However, there exist some parallels with approaches for exact algorithms. In both fields the class \textit{SNP} plays an essential role. Furthermore, there exist problems in \textsc{SUBEXP} that can be approximated arbitrarily closely. An example of such a problem is Euclidean TSP, for which a polynomial time approximation scheme exists \cite{1}. In general the relation between polynomial time approximability and sub-exponential time solvability is not clear (yet).

Relation to Extremal Combinatorics

Extremal combinatorics is essentially the study of combinatorial structures where the goal is to obtain good bounds on the maximum or minimum number of substructures of a certain type. We can use techniques from this area for designing fast exact algorithms. For example, a lower bound result by Reed \cite{30} on the size of a dominating set of any graph with minimum degree three has led to an exact algorithm for the dominating set problem with time complexity $O^*(1.94^n)$ \cite{14}. Both areas may influence each other.

Programme and Methodology

Overall Aims

\begin{enumerate}
\item Obtain more basic knowledge, tools and approaches by constructing or improving exact algorithms for a number of specific NP-hard problems.
\item Understand how progress on exact algorithms for different problems is related to each other and try to connect the results.
\end{enumerate}

Aim 1: Experimental Research

A number of concrete optimization problems can be considered. With an eye to real-life applications also exact
algorithms with polynomial space complexity are of interested, even if this implies a slower worst-case running time.

Of course, open questions are to improve the bounds for the problems already mentioned, for example the TSP problem. Improving the $O^*(2^n)$ time exact algorithm in [16] has been an open problem for more than 40 years. Even a small improvement of this time bound would already be a significant breakthrough.

**Open problem 1**

(a) Construct an exact algorithm for the TSP problem with $O^*(\alpha^n)$ time complexity for some $\alpha < 2$.

(b) Construct an exact algorithm for the TSP problem with $O^*(2^n)$ time complexity and polynomial space complexity.

Below a series of other interesting and related problems are presented.

Given a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, the MAX-CUT problem is to find a partition $V_1 \cup V_2$ of $V$ with the maximum number of edges between $V_1$ and $V_2$. In [10] an $O^*(2^n)$ time algorithm is given. However, a measurement in terms of the number of vertices seems more appropriate for this problem. Enumerating and checking all possible subsets yields a straightforward exact algorithm with time complexity $O^*(2^n)$. The best known exact algorithm in terms of the number of vertices has $O^*(1.74^n)$ time complexity [34]. This algorithm has exponential space complexity.

**Open problem 2**

(a) Construct an exact algorithm for the MAX-CUT problem with $O^*(\alpha^n)$ time complexity for some $\alpha < 1.74$.

(b) Construct an exact algorithm for the MAX-CUT problem with $O^*(\alpha^n)$ time complexity for some $\alpha < 2$ and polynomial space complexity.

Given a set of positive integers $A = \{a_1, \ldots, a_n\}$ and a goal sum $s$, the SUBSET-SUM problem is to find a subset of $A$ that sums to $s$. Enumerating and checking all possible subsets yields an exact algorithm with time complexity $O^*(2^n)$. The best known exact algorithm has $O^*(2^\sqrt{n})$ time complexity and exponential space complexity [18]. The space complexity of this algorithm has been improved considerably [32], but it has not been brought down to polynomial space.

**Open problem 3**

(a) Construct an exact algorithm for the SUBSET-SUM problem with $O^*(\alpha^n)$ time complexity for some $\alpha < \sqrt{2}$.

(b) Construct an exact algorithm for the SUBSET-SUM problem with $O^*(\alpha^n)$ time complexity for some $\alpha < 2$ and polynomial space complexity.

Due to results in [2] and [3] the problems mentioned in Open problem 4 all have an exact algorithm with $O^*(2^n)$ time complexity and polynomial space complexity.

**Open problem 4** Let $G$ be a graph on $n$ vertices. Construct $O^*(\alpha^n)$ time exact algorithms for some $\alpha < 2$ for counting the number of paths between a given pair of vertices in $G$, the number of cycles in $G$, the number of cycles through a given vertex in $G$, and the number of cycles of a given length $\ell$ in $G$.

The Hamiltonian path problem is to find a path through all the $n$ vertices of a graph $G$. In [23] an $O^*(2^n)$ time (and space) algorithm is presented. When input graphs are restricted to planar graphs the problem remains NP-hard [22]. Under this restriction the problem belongs to SUBEX: Just as for the Euclidean TSP problem (see, e.g., [19]) it allows an $O^*(e^{\sqrt{n}\log n})$ time exact algorithm [26] for some constant $c$. Recently an $O^*(e^{\sqrt{n}})$ time exact algorithm for the Hamiltonian path problem restricted to planar graphs has been developed [7].

**Open problem 5** Construct an $O^*(e^{\sqrt{n}})$ time exact algorithm for the Euclidean TSP problem.

A graph homomorphism from a graph $G = (V_G, E_G)$ to a graph $H = (V_H, E_H)$ is a vertex mapping $f : V_G \rightarrow V_H$ satisfying the property that $(f(u), f(v))$ is an edge of $H$ whenever $(u, v)$ is an edge of $G$. The H-HOMOMORPHISM problem is to decide whether an instance graph $G$ allows a homomorphism to the fixed pattern graph $H$. In [17] it has been proven that the H-HOMOMORPHISM problem is polynomially solvable if $H$ is bipartite and NP-complete otherwise. Recently, Fomin, Heggernes, and Kratsch [13] have developed exact algorithms for pattern graphs that have bounded treewidth.

Related problems, which are known to be NP-complete for an infinite number of pattern graphs (cf. [11, 12, 24]), are the H-COVER problem, the H-P-COVER problem and the H-COLOR DOMINATION problem. They all impose further conditions on the mapping $f : V_G \rightarrow V_H$ restricted to the neighborhood of any vertex $u$ in input graph $G$.

**Open problem 6** Design fast exact algorithms for the H-HOMOMORPHISM problem for various classes of pattern graphs $H$ and relate the time complexities of these algorithms to each other. Do the same for the H-COVER problem, the H-P-COVER problem and the H-COLOR DOMINATION problem.

Of course, other NP-hard problems can be studied as well. In [36] an overview is given of open questions in this field.
Aim 2: Structural Research

Questions for the second aim will be guided by the outcomes of the questions posed under the first aim. As we can not expect to lower worst-case time complexities for specific NP-hard problems beyond a certain threshold, it would be interesting to find out whether thresholds for different problems are somehow related to each other. Can results for some concrete optimization problems be connected to the widely believed conjecture $\text{SNP} \not\subseteq \text{SUBEXP}$?

This could lead to a better understanding of the worst-case time behaviour of NP-hard problems. Since $k$-$\text{SAT}$ with $k \geq 3$ is $\text{SNP}$-complete under $\text{SERF}$-reductions [21], the above conjecture can also be formulated as the Exponential Time Hypothesis (ETH) [20]: The $k$-$\text{SAT}$ problem does not have a sub-exponential time algorithm for any $k \geq 3$.

We let $s_k$ denote the infimum of all real numbers $\epsilon$ such that $k$-$\text{SAT}$ allows an $O^*(2^{\epsilon n})$ time exact algorithm. Under ETH the sequence $s_k$ converges to a number $s_\infty > 0$ [20]. In [20] the following open problem is posed.

Open problem 7 Does $s_\infty = 1$ hold under ETH?

So far little is known about lower bounds on the running time of exact algorithms for problems such as TSP, knapsack or maximum independent set. Can one find good lower bounds? Can one relate these results to the ETH conjecture or provide more evidence for Open Problem 7?

Another major open problem is the relationship between the conjectures $P \neq NP$ and $\text{SNP} \not\subseteq \text{SUBEXP}$.

Open problem 8 Does $\text{SNP} \not\subseteq \text{SUBEXP}$ imply $P = \text{NP}$?

It is interesting to study natural complexity parameters $m(x)$ for problems in NP that are not $\text{SNP}$-complete under $\text{SERF}$-reductions. (Recall that for $\text{SNP}$-complete problems under $\text{SERF}$-reductions this is of less importance [21].) This way further evidence might be obtained for the $\text{SNP} \not\subseteq \text{SUBEXP}$ conjecture. Take for example (the decision version of) TSP. For this problem, $m(x) = |V|$ seems to be more appropriate than $m(x) = |E|$. Is it possible to find a theoretical explanation for this?

Another example is the quadratic assignment problem (QAP), which asks for a permutation $\pi$ that minimizes the function $\sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j} \pi(i) \pi(j) b_{i,j}$ given two $n \times n$ matrices $A = (a_{i,j})$ and $B = (b_{i,j})$. A natural complexity parameter would be $m(x) = n$. This problem is considered to be very hard, and no exact algorithm is known that performs better than the exhaustive $O^*(n!)$ exact algorithm (cf. [29]). Solving the following problem (under reasonable complexity assumptions) could lead to further structural insights.

Open problem 9 Prove that QAP can not be solved in $O^*(\alpha^n)$ time for any fixed $\alpha$.

Methodology

During the project novel techniques to design fast exact algorithms can be developed. The techniques described below have already been proven to be useful. They can be further explored in pursuit of the research.

- Dynamic Programming is a standard approach that has led to several exact algorithms. A classical result is the $O^*(2^n)$ time exact algorithm for TSP [16].

- Instead of checking all feasible solutions we construct a search tree by branching parts of a feasible solution into several subcases. Then we speed up the search procedure analogously as for branch-and-bound algorithms (cf. [35]). This technique of pruning the search tree appears (amongst others) in exact algorithms for the maximum independent set problem (see, e.g. [31]).

- Before checking any feasible solutions we can first restructure the given data in order to subsequently answer queries to this data faster. Data preprocessing has been used to obtain faster exact algorithms for the $\text{SUBSET-SUM}$ problem [18] and the binary knapsack problem [32].

- A relatively new idea in designing exact algorithms is to use the technique of local search. Using local search methods for 3-$\text{SAT}$ has lead to an exact algorithm with time complexity $O^*(1.49^n)$ [6], later improved to $O^*(1.48^n)$ [5]. Before using this technique, the best exact algorithm for 3-$\text{SAT}$ was the $O^*(1.50^n)$ time algorithm given in [25].

- For some problems the feasible solutions can be enumerated with polynomial delay, i.e., with polynomial work per enumerated solution (cf. [15]). An example is the problem of finding a maximum size transitive subtournament in a given tournament. Enumerating all maximal transitive subtournaments can be done with polynomial delay [33]. Any $n$-vertex tournament contains at most $O(1.72^n)$ maximal transitive subtournaments [27]. This way we directly obtain an $O^*(1.72^n)$ exact algorithm for finding a maximum size transitive subtournament [36]. We still need to get a better understanding of the general type of problem that allows enumeration with polynomial delay.

- As it has been an open problem since 1962, one might start with working under the assumption that the TSP problem requires $\Omega^*(2^n)$ time. In order to obtain strong lower bounds on other NP-hard problems, such as the knapsack problem or the independent set problem, efficient reductions are needed. The field of parameterized complexity (cf. [8]) seems to offer the right tools.
References


