Open problems around exact algorithms

GERHARD J. WOEGINGER *

Abstract

We list a number of open questions around worst case time bounds and worst case space bounds for NP-hard problems. We are interested in exponential time solutions for these problems with a relatively good worst case behavior. We summarize what is known on these problems, we discuss related results, and we provide pointers to the literature.

1 Introduction

Every problem in NP can be solved in exponential time by exhaustive search: Recall that a decision problem is in NP, if and only if there exists a polynomial time decidable relation $R(x, y)$ and a polynomial $m(|x|)$ such that for every YES-instance x, there exists a YES-certificate y with $|y| \leq m(x)$ and $R(x, y)$. A trivial exact algorithm for solving instance x enumerates all possible strings y with lengths up to $m(|x|)$, and checks whether any of them yields a YES-certificate. Up to polynomial factors that depend on the evaluation time of $R(x, y)$, this yields an exponential running time of $2^{m(|x|)}$. A natural question is: Can we do better than this trivial enumerative algorithm?

Interestingly, for many of the standard combinatorial optimization problems the answer to this question is YES. Early examples include an $O^*(1.4422^n)$ algorithm for deciding 3-colorability of an n-vertex graph by Lawler [31]; an $O^*(1.2599^n)$ algorithm for finding a maximum independent set in an n-vertex graph by Tarjan & Trojanowski [42]; an $O^*(1.4142^n)$ algorithm for the SUBSET-SUM problems with n integers by Horowitz & Sahni [23]. (The notation $O^*(f(n))$ is explained at the end of this section.) The survey paper [44] by Woeginger summarizes many results in this area. Typical time complexities for optimization problems are:

- Polynomial (for instance: n^2, $n \log n$, n^3, n^{1000}).
- Quasi-polynomial (for instance: $n^{\log n}$, $n^{\log^2 n}$, $c^{\log^7 n}$).
- Sub-exponential (for instance: $2^{\sqrt{n}}$, $5^{(n^{0.98})}$).
- Exponential (for instance: 2^n, 8^n, $n!$, n^n).

*`gwoegi@win.tue.nl`. Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

NP-hard problems usually come with exponential or sub-exponential time complexities. It is highly unlikely that an NP-hard problem can be solved in quasi-polynomial time. Here is an example for a problem that can be solved in quasi-polynomial time (but probably not in polynomial time): "Given n integers, can you select $\lfloor \log_2 n \rfloor$ from them that add up to 0?" For some NP-hard problems, it is possible to reach a 'good' exponential time complexity, but it seems that we have to pay for this with an *exponential* space complexity. And algorithms with exponential space complexities are absolutely useless for real life applications. Note that the trivial enumerative algorithm discussed in the first paragraph of this section has a *polynomial* space complexity.

In this paper, we discuss a number of results and open questions around fast exponential time algorithms and algorithms with exponential space complexities for NP-hard problems. Some of these fast exponential time algorithms are closely related to certain primitive, polynomially solvable problems. Small improvements on the (polynomial) time complexities of these primitive problems yield major improvements on the corresponding (exponential) time complexities. We discuss such primitive problems and their relation to certain NP-hard problems. We present approaches, tricks, known results, and open questions.

Notation. Throughout this paper, we will use a modified big-Oh notation that suppresses polynomially bounded terms. For a positive real constant c, we write $O^*(c^n)$ for a time complexity of the form $O(c^n \cdot \text{poly}(n))$. The notations $\Omega^*(c^n)$ and $\Theta^*(c^n)$ are defined analogously.

2 Problems around paths and cycles

A number of exact algorithms in the literature attack an NP-hard problem by running through all the subsets of an underlying n-element ground set, while generating and storing useful auxiliary information. Since an n-element ground set has 2^n subsets, the time complexities of these approaches are typically $\Omega^*(2^n)$. And also the space complexities of these approaches are typically $\Omega^*(2^n)$, since they store and remember auxiliary information for every subset.

A good example for this approach is the famous dynamic programming algorithm of Held & Karp [22] for the travelling salesman problem (TSP): A travelling salesman has to visit the cities 1 to n. He starts in city 1, runs through the cities $2, 3, \ldots, n-1$ in arbitrary order, and finally stops in city n. The distance $d(i, j)$ from city i to city j is specified as part of the input. The goal is to find a path that minimizes the total travel length of the salesman. The dynamic program of Held & Karp [22] introduces for every non-empty subset $S \subseteq \{2, \ldots, n-1\}$ of the cities and for every city $i \in S$ a corresponding state $[S; i]$. By $\textsc{Length}[S; i]$ we denote the length of the shortest path that starts in city 1, then visits all cities in $S - \{i\}$ in arbitrary order, and finally stops in city i. Clearly, $\textsc{Length}[\{i\}; i] = d(1, i)$ for every $i \in \{2, \ldots, n-1\}$. And for every $S \subseteq \{2, \ldots, n-1\}$ with $|S| \geq 2$ we have

$$\textsc{Length}[S; i] \;=\; \min \{\textsc{Length}[S - \{i\}; j] + d(j, i) : \; j \in S - \{i\}\}\,.$$

By processing the subsets S in order of increasing cardinality, we can compute the value $\textsc{Length}[S; i]$ in time proportional to $|S|$. In the end, the optimal travel length is given as the minimum $\min_{2 \leq k \leq n-1} \textsc{Length}[\{2, \ldots, n-1\}; k] + d(k, n)$.

Fact 2.1 *The TSP can be solved in $O^*(2^n)$ time and $O^*(2^n)$ space.*

Open problem 2.2

(a) Construct an exact algorithm for the n-city TSP with $O^*(1.99^n)$ time complexity!

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

In the Hamiltonian path problem, we have to decide for a given graph $G = (V, E)$ with vertices $1, \ldots, n$ whether it contains a spanning path starting in vertex 1 and ending in vertex n. The Hamiltonian path problem forms a simpler special case of the TSP. Karp [29] (and independently Bax [4]) provided a cute solution for the restriction of Problem 2.2.(b) to this Hamiltonian special case. We use the following definitions. A *walk* in a graph is a sequence $v_1, \ldots, v_k$ of vertices such that every pair of consecutive vertices is connected by an edge; vertices and edges may show up repeatedly in a walk. For a subset $S \subseteq V$ we denote by $\textsc{Walk}(S)$ the set of all walks with n vertices in G that start in vertex 1, end in vertex n, and avoid all the vertices in S. Let A be the adjacency matrix of $G - S$. Recall that in the kth power A^k of A, the entry at the intersection of row i and column j counts the number of walks with $k + 1$ vertices in $G - S$ that start in vertex i and end in vertex j. Therefore, the number of walks in $\textsc{Walk}(S)$ can be read from matrix A^{n-1}:

Fact 2.3 *For every subset $S \subseteq V$, the cardinality $|\textsc{Walk}(S)|$ can be determined in polynomial time.*

If a walk through n vertices in G does not avoid any vertex k, then it must visit all the vertices, and hence must form a Hamiltonian path. Consequently, the number of Hamiltonian paths from 1 to n in G equals

$$|\textsc{Walk}(\emptyset)| - |\bigcup_{k=2}^{n-1} \textsc{Walk}(\{k\})| = \sum_{S \subseteq V} (-1)^{|S|} \cdot |\textsc{Walk}(S)|.$$

Here we have used the inclusion-exclusion principle. The sum in the right hand side of the displayed equation is straightforward to compute by applying Fact 2.3. We only need to remember the partial sum of all the terms evaluated so far, and the space used for evaluating one term can be reused in evaluating the later terms. All in all, evaluating and adding up the values of $O(2^n)$ terms yields an $O^*(2^n)$ time and polynomial space algorithm for *counting* the number of Hamiltonian paths. The following fact is a trivial consequence of this:

Fact 2.4 *The Hamiltonian path problem in an n-vertex graph can be solved in $O^*(2^n)$ time and polynomial space.*

Eppstein [15] improves on this polynomial space result for Hamiltonian paths in the special case of *cubic* graphs: He presents an algorithm that uses $O^*(1.297^n)$ time and linear space. Bax [5] and Bax & Franklin [6] have extended the inclusion-exclusion approach to a number of counting problems around paths and cycles in n-vertex graphs. For all these problems, the time complexity is $O^*(2^n)$ and the space complexity is polynomial.

Open problem 2.5 *Construct $O^*(1.99^n)$ time exact algorithms for the following counting problems in n-vertex graphs G:*

(a) Count the number of paths between a given pair of vertices in G.

(b) Count the number of cycles in G.

(c) Count the number of cycles through a given vertex in G.

(d) Count the number of cycles of a given length ℓ in G.

Alon, Yuster & Zwick [2] design a polynomial time algorithm for the following problem: Given a directed graph on n vertices, does it contain a simple directed path with $\log_2 n$ vertices? The tools developed in [2] (most probably) can not be used to settle the following question.

Open problem 2.6 *Is there a polynomial time algorithm for deciding whether a given directed graph on n vertices contains a simple directed path on $\log_2^2 n$ vertices?*

Now let us turn to some relatives of the n-city TSP. For a fixed Hamiltonian path from city 1 to city n and for a fixed city k, the length of the subpath from city 1 to city k is called the *delay* of city k. In the travelling repairman problem (TRP), the goal is to find a Hamiltonian path from city 1 to city n that minimizes the sum of delays over all cities. In the precedence constrained travelling repairman problem (prec-TRP), the input additionally specifies a partial order on the cities. A Hamiltonian path is feasible, if it obeys the partial order constraints. We refer to Afrati, Cosmadakis, Papadimitriou, Papageorgiou & Papakonstantinou [1] for more information on the TRP.

Here is a scheduling problem SCHED that forms a special case of prec-TRP: There are n jobs $1, \ldots, n$ that are specified by their processing times $p_1, \ldots, p_n$. The jobs are partially ordered (precedence constrained), and if job i precedes job j in the partial order, then i must be processed to completion before j can begin its processing. All jobs are available at time 0, and job preemption is not allowed. The goal is to schedule the jobs on a single machine such that all precedence constraints are obeyed and such that the total job completion time $\sum_{j=1}^{n} C_j$ is minimized; here C_j is the time at which job j completes in the given schedule. SCHED is the special case of prec-TRP where the distances between cities $i \neq j$ are given by $d(i,j) = p_j$. It is quite straightforward to design an $O^*(2^n)$ time and $O^*(2^n)$ space dynamic programming algorithm for prec-TRP (and for its special cases TRP and SCHED).

Open problem 2.7

(a) Construct an $O^*(1.99^n)$ time exact algorithm for TRP or SCHED or prec-TSP.

(b) Provide evidence in favor of or against the following claim: If there exists an $O^*(c^n)$ time exact algorithm with $c < 2$ for one of the four problems TSP, TRP, SCHED, prec-TSP, then there exist $O^*(c^n)$ time exact algorithms for all four problems.

The restriction of the Hamiltonian path problem to *planar* graphs is still NP-hard (Itai, Papadimitriou & Swarcfiter [27]), but it allows considerably faster exact algorithms that are based on separators: Lipton & Tarjan [32] prove that in every planar n-vertex graph $G = (V, E)$, the vertex set V can be partitioned into three sets A, B, and the so-called separator S, such that there are no edges between A and B, and such that $|A| \le 2n/3$, $|B| \le 2n/3$, and $|S| \le 3\sqrt{n}$ holds. Moreover, such a separator S can be found in linear time. A recursive approach [33] repeatedly finds a separator, enumerates a lot of cases for the vertices in the separators, and then branches into one subcase for $A \cup S$ and one subcase for $B \cup S$. All in all, this yields an algorithm for the Hamiltonian path problem in planar graphs with time complexity $O^*(c^{\sqrt{n}})$.

In the *Euclidean TSP* the n cities are points in the Euclidean plane, and the distances between the cities are the Euclidean distances. The Euclidean TSP is NP-hard (Itai, Papadimitriou & Swarcfiter [27]). By exploiting planar separator structures in a geometric setting, the Euclidean TSP on n cities can be solved in $O(n^{c\sqrt{n}})$ time. This result has been found independently, and more or less simultaneously by three groups of researchers: Warren Smith [40] has this result in his Ph.D. thesis, Viggo Kann [28] has it in his Ph.D. thesis, and Hwang, Chang & Lee [25] have in a journal article in algorithmica. These three results are all very similar, but not identical to each other. For instance Kann [28] uses circular separator structures. The time complexities of all three approaches are $O^*(c^{\sqrt{n}\log n})$. It is unclear whether we can improve the square-root term in the exponent of this time complexity. Can we at least get rid of the logarithmic term?

Open problem 2.8 *Construct an $O^*(c^{\sqrt{n}})$ time exact algorithm for the Euclidean n-city TSP.*

3 Problems around subset sums

We start this section with a couple of polynomially solvable problems: An input to the first problem "k-SUM" consists of m integers $a_1, \ldots, a_m$ and a goal sum S. The problem is to decide whether there are k of these integers that add up to S. An input to the second problem "Table-k-SUM" consists of a $k \times m$ table and a goal sum S; the entries in row i of the table are denoted by $R_i(1), \ldots, R_i(m)$. The problem is to decide whether one can choose k integers from this table, exactly one from each row, that add up to S. In both problems, the number k is a fixed integer that is not part of the input. Both problems are closely related, and they can be reduced to each other in linear time (Erickson [17]). Both problems are trivially solvable in polynomial time $O(m^k)$.

Here is how to get a better time complexity for Table-2-SUM: Sort the entries in the first row. Then for $j = 1, \ldots, m$ perform a binary search for the value $S - R_2(j)$ in this sorted first row. If the search succeeds at $R_1(i)$, then $R_1(i) = S - R_2(j)$ and the answer is YES. If all searches fail, then the answer is NO.

Fact 3.1 *Table-2-SUM can be solved in $O(m \log m)$ time and $O(m)$ space.*

The same approach also yields fast algorithms for Table-k-SUM for all $k \ge 3$: Compute the sum of every $\lceil k/2 \rceil$-tuple of integers that has one entry in each of the first $\lceil k/2 \rceil$ rows; these sums form the first row in a new table. Compute the sum of every $\lfloor k/2 \rfloor$-tuple of integers that has one entry in each of the last $\lfloor k/2 \rfloor$ rows; these sums form the second row in the new table. Apply the above algorithm to this new instance of Table-2-SUM.

Fact 3.2 *Table-k-SUM can be solved in $O(m^{\lceil k/2 \rceil} \log m)$ time and $O(m^{\lceil k/2 \rceil})$ space.*

For odd k, the time complexity can be slightly improved to $O(m^{\lceil k/2 \rceil})$; see for instance Erickson [16]. In particular, the 3-SUM problem can be solved in $O(m^2)$ time. We will not go into details, since in this paper we really do not care about logarithmic factors. The main drawback of all these algorithms is their horrible space complexity.

Schroeppel & Shamir [38] improve the space complexity for Table-4-SUM by using a data structure that enumerates the m^2 sums $R_1(i)+R_2(j)$ with $1 \le i,j \le m$ in non-decreasing order. This data structure uses only $O(m)$ space. Every time we kick it, it starts working for $O(\log m)$ time steps, and then spits out the next larger sum $R_1(i)+R_2(j)$. The data structure is based on a balanced search tree that supports deletions, insertions, and extracting the minimum with logarithmic work per operation. It is built as follows: In a preprocessing step, we bring the entries in the second row into non-decreasing order. As a consequence, we have for every fixed index i that

$$R_1(i)+R_2(1) \;\le\; R_1(i)+R_2(2) \;\le\; \cdots \;\le\; R_1(i)+R_2(m).$$

For every index i ($1 \le i \le m$), the data structure stores the pair (i,j) that corresponds to the first unvisited sum $R_1(i)+R_2(j)$ in this ordering. Whenever the data structure is kicked, it extracts and deletes the pair (i,j) with minimum sum, and inserts the pair $(i,j+1)$ instead. All in all, the enumeration of the m^2 sums costs $O(m^2 \log m)$ time.

Schroeppel & Shamir [38] use two such data structures; the first one generates the sums $x = R_1(i)+R_2(j)$ in non-decreasing order, whereas the second one generates the sums $y = R_3(s)+R_4(t)$ in non-increasing order. Whenever $x+y<S$ holds, the current value of x is too small for reaching the goal sum S; we replace it by the next larger sum $R_1(i)+R_2(j)$ from the first data structure. Whenever $x+y>S$ holds, the current value of y is too large for reaching the goal sum S; we replace it by the next smaller sum $R_3(s)+R_4(t)$ from the second data structure. These steps are repeated over and over again, until one data structure becomes empty (answer NO) or until we reach $x+y=S$ (answer YES).

Fact 3.3 *Table-4-SUM can be solved in $O(m^2 \log m)$ time and $O(m)$ space.*

Open problem 3.4

(a) Is there an $O(m^3 \log m)$ time and $O(m)$ space algorithm for Table-6-SUM?

(b) Is there an $O(m^{\lceil k/2 \rceil - \alpha})$ time algorithm for Table-k-SUM for some integer $k \ge 3$ and some real $\alpha > 0$?

Now let us turn to negative results around the k-SUM and the Table-k-SUM problem. The 3-SUM problem plays a notorious role in computational geometry. Gajentaan & Overmars [20] have put together a long list of geometric problems: All problems on this list can be solved in quadratic time, and for all of them nobody knows how to do better. All problems on this list contain 3-SUM as a special case (under linear time reductions), and for all of them this 3-SUM special case (intuitively) seems to be the main obstacle for breaking through the quadratic time barrier. One example problem on this list is: Given m (possibly overlapping) triangles in the Euclidean plane, compute the area of their union. Another one: Given m pairwise non-intersecting straight line segments in the Euclidean plane, is there a straight line that separates them into two non-empty subsets? And another one: Given m points in the Euclidean plane, are some three of them on a common line? For instance, the linear time reduction from 3-SUM to 3-POINTS-ON-A-COMMON-LINE is based on the following observation: The x-coordinates of the intersection points of the line $y = ax + b$ with the curve $y = f(x) = x^3 - Sx^2$ are the roots of $x^3 - Sx^2 - ax - b = 0$; for every line the sum of these roots equals S, the coefficient of the quadratic term. Consequently, the point set $(a_1, f(a_1)), (a_2, f(a_2)), \ldots, (a_m, f(a_m))$ contains three points $(a_x, f(a_x))$, $(a_y, f(a_y))$, $(a_z, f(a_z))$ on a common line, if and only if $a_x+a_y+a_z = S$. The bottom-line of this paragraph is that research on the 3-SUM problem is severely stuck at the threshold $O(m^2)$.

What about the general k-SUM problem with $k \geq 4$? Here research is severely around the threshold $O(m^{\lceil k/2\rceil})$. Erickson [16] proved an $\Omega(m^{\lceil k/2\rceil})$ lower bound on k-SUM in a certain restricted variant of the linear decision tree model. The additional restriction in his model is that every decision step must be based on testing the sign of some affine linear combination of at most k elements of the input. At first sight, this model seems to be strange, and the lower bound result seems to be quite weak. However, given our general failure in proving reasonable lower bounds for algorithmic problems and given the lack of tools in this area, Erickson's lower bound result in fact is a major breakthrough.

Open problem 3.5 *Prove a non-trivial lower bound for the k-SUM problem in the algebraic decision tree model or in the algebraic computation tree model (see Ben-Or [8]).*

Downey & Fellows [12, 13] have proved that the k-SUM problem with parameter k is W[1]-hard. All these negative results for k-SUM translate into analogous negative results for Table-k-SUM.

After this long polynomial time prelude, we will spend the rest of this section on NP-hard problems. In the NP-hard SUBSET-SUM problem, the input consists of n positive integers $b_1, \ldots, b_n$ and a goal sum B. The problem is to decide whether there exists some subset of the b_i that add up to B. The strongest known negative result for SUBSET-SUM is an $\Omega(n^2)$ lower bound in the algebraic computation tree model of computation [11, 8].

On the positive side, Horowitz & Sahni [23] have come up with the following approach for SUBSET-SUM: They split the instance into two parts, one part with $b_1, \ldots, b_{\lfloor n/2\rfloor}$ and another part with $b_{\lfloor n/2\rfloor+1}, \ldots, b_n$. They construct a table with two rows, where the first row consists of all the subset sums for the first part, and where the second row consists of all the subset sums for the second part. The table can be computed in $O^*(2^{n/2})$ time. The SUBSET-SUM instance has answer YES, if and only if the constructed Table-2-SUM instance with $S = B$ has answer YES. Our above discussion of Table-2-SUM yields the following result.

Fact 3.6 *SUBSET-SUM can be solved in $O^*(2^{n/2})$ time and in $O^*(2^{n/2})$ space.*

Schroeppel & Shamir [38] follow essentially the same idea, but instead of splitting the SUBSET-SUM instance into two parts, they split it into four parts of size approximately $n/4$. This leads to a corresponding instance of Table-4-SUM, and to a substantially improved space complexity.

Fact 3.7 *SUBSET-SUM can be solved in $O^*(2^{n/2})$ time and in $O^*(2^{n/4})$ space.*

Generally, if we split the SUBSET-SUM instance into $k \geq 2$ parts, then we get a corresponding table with k rows and $O(2^{n/k})$ elements per row. Applying the fastest known algorithm to the corresponding instance of Table-k-SUM gives a time complexity of $O^*(2^{f(n,k)})$ with $f(n,k) = n\lceil k/2 \rceil / k \ge n/2$. Hence, this approach will not easily lead to an improvement over the time complexity $O^*(2^{n/2})$. Schroeppel & Shamir [38] also construct $t(n)$ time and $s(n)$ space algorithms for SUBSET-SUM for all $s(n)$ and $t(n)$ with $\Omega^*(2^{n/2}) \le t(n) \le O^*(2^n)$ and $s^2(n) \cdot t(n) = \Theta^*(2^n)$.

Open problem 3.8

(a) *Construct an $O^*(1.4^n)$ time algorithm for SUBSET-SUM.*

(b) *Construct an $O^*(1.99^n)$ time and polynomial space algorithm for SUBSET-SUM.*

(c) *We have seen that positive results for Table-k-SUM yield positive results for SUBSET-SUM. Can we establish some reverse statement? Do fast (exponential time) algorithms for SUBSET-SUM yield fast (polynomial time) algorithms for Table-k-SUM?*

Another NP-hard problem in this area is the EQUAL-SUBSET-SUM problem: Given n positive integers $b_1, \ldots, b_n$, do there exist two disjoint non-empty subsets of the b_i that both have the same sum. A translation of EQUAL-SUBSET-SUM into a corresponding Table-4-SUM instance leads to an $O^*(2^n)$ algorithm for EQUAL-SUBSET-SUM. It might be interesting to design faster algorithms for EQUAL-SUBSET-SUM, and to get some understanding of the relationship between fast algorithms for SUBSET-SUM and fast algorithms for EQUAL-SUBSET-SUM.

In a weighted voting game, there are n players with integer voting weights $w_1, \ldots, w_n$ and an integer quota q. A coalition $S \subseteq \{1, \ldots, n\}$ is a winning coalition, if $\sum_{p \in S} w_p \ge q$ holds, and otherwise it is losing. A well-known example of weighted voting is the Electoral College of the United States, where the players are the fifty states plus the District of Columbia; each player casts a number of votes that is equal to the number of that state's representatives plus senators. A player p is *pivotal* for a coalition $S \cup \{p\}$, if $S \cup \{p\}$ is winning whereas S alone is losing. The *Banzhaf power index* (see Banzhaf [3]) measures the relative power of player p through the number η_p of coalitions for which player p is pivotal.

Prasad & Kelly [36] prove that computing the Banzhaf index is #P-complete. Klinz & Woeginger [30] give an $O^*(2^{n/2})$ time algorithm for computing the Banzhaf index; the approach in [30] is a slight extension of the approach of Horowitz & Sahni [23].

Open problem 3.9

(a) *Construct an $O^*(1.4^n)$ time algorithm for computing the Banzhaf index.*

(b) *Prove that there is an algorithm with time complexity $O^*(c^n)$ for SUBSET-SUM, if and only if ther is an algorithm with time complexity $O^*(c^n)$ for computing the Banzhaf index.*

4 Problems around cliques and matrix multiplication

We start this section with the polynomially solvable k-CLIQUE problem: An input consists of an undirected, simple, loopless p-vertex graph $G = (V, E)$. The problem is to decide whether G contains a clique on k vertices. We stress that k is not part of the input. On the negative side, we have that k-CLIQUE with parameter k is W[1]-hard (Downey & Fellows [12, 13]). On the positive side, k-CLIQUE is easily solved in polynomial time $O(p^k)$.

Itai & Rodeh [26] observed that fast square matrix multiplication can be used to improve this time complexity for 3-CLIQUE: Recall that the product of two $p \times p$ matrices can be computed in $O(p^\omega)$ time, where $\omega < 2.376$ denotes the so-called *square matrix multiplication exponent*; see Coppersmith & Winograd [10]. Recall that in the ℓth power A^ℓ of the adjacency matrix A of graph G, the entry at the intersection of row i and column j counts the number of walks with $\ell+1$ vertices in G that start in vertex i and end in vertex j. Furthermore, a 3-clique $\{x, y, z\}$ yields a walk $x - y - z - x$ with four vertices from x to x, and vice versa, every walk with four vertices from vertex x to x corresponds to a 3-clique. Hence, G contains a 3-clique if and only if A^3 has a non-zero entry on its main-diagonal.

Fact 4.1 *The* 3*-CLIQUE problem for a p-vertex graph can be solved in $O(p^\omega)$ time (where $\omega < 2.376$ is the square matrix multiplication exponent) and in $O(p^2)$ space.*

Nešetřil & Poljak [35] extend this idea to the $3k$-CLIQUE problem: For every k-clique C in G, create a corresponding vertex $v(C)$ in an auxiliary graph. Two vertices $v(C_1)$ and $v(C_2)$ are connected by an edge in the auxiliary graph, if and only if $C_1 \cup C_2$ forms a $2k$-clique in G. Note that the auxiliary graph has $O(p^k)$ vertices. Furthermore, graph G contains a $3k$-clique if and only if the auxiliary graph contains a 3-clique.

Fact 4.2 *The $3k$-CLIQUE problem for a p-vertex graph can be solved in $O(p^{\omega k})$ time and $O(p^{2k})$ space.*

This approach yields a time complexity of $O(p^{\omega k+1})$ for $(3k+1)$-CLIQUE, and a time complexity of $O(p^{\omega k+2})$ for $(3k+2)$-CLIQUE. Eisenbrand & Grandoni [14] slightly improve on these bounds for $(3k + 1)$-CLIQUE with $1 \leq k \leq 5$ and for $(3k + 2)$-CLIQUE with $k \geq 1$ by using fast rectangular matrix multiplication (Coppersmith [9] and Huang & Pan [24]) instead of fast square matrix multiplication. In particular, [14] improves the time complexities for 4-CLIQUE, 5-CLIQUE, 7-CLIQUE from $n^{3.376}$, $n^{4.376}$, $n^{5.751}$ down to $n^{3.334}$, $n^{4.220}$, $n^{5.714}$, respectively.

Open problem 4.3

(a) Design algorithms with better time and/or space complexities for k-CLIQUE!

(b) Is there an $O(p^{7.5})$ time algorithm for 10*-CLIQUE?*

(c) Is 3*-CLIQUE as difficult as Boolean matrix multiplication?*

Another problem that can be attacked via matrix multiplication is the problem of finding a simplicial vertex in a graph $G = (V, E)$: A vertex $x \in V$ is called *simplicial*, if the neighbors of x span a clique in G. If a simplicial vertex x has d neighbors, then G contains $d\,(d-1)$ walks $x - y - z - x$ with four vertices from x to x. Furthermore, if there are $d\,(d-1)$ such walks for a vertex x of degree d, then the neighborhood of x must contain so many edges that x is simplicial. Hence, G contains a simplicial vertex, if and only if there is a vertex of degree d for which the third power A^3 has a corresponding entry $d\,(d-1)$ on its main-diagonal. This can be checked by matrix multiplication.

Open problem 4.4

(a) *Design algorithms with better time and/or space complexities for detecting a simplicial vertex.*

(b) *Is detecting a simplicial vertex as difficult as Boolean matrix multiplication? Is detecting a simplicial vertex as difficult as* 3*-CLIQUE?*

Now let us turn to NP-hard problems. The standard CLIQUE problem asks to find a clique of maximum size in a given graph $G = (V, E)$. Feige & Kilian [19] show that a polynomial time algorithm for finding a clique of size $k \approx \log n$ in an n-vertex graph is highly unlikely to exist. Tarjan & Trojanowski [42] give an algorithm with running time $O^*(1.2599^n)$ for CLIQUE. Robson [37] gives an improved algorithm with $O^*(1.2108^n)$ time complexity and exponential space complexity. Beigel [7] presents another algorithm with a slightly weaker time complexity of $O^*(1.2227^n)$, but polynomial space complexity.

Open problem 4.5 *Construct an exact algorithm for CLIQUE with time complexity* $O^*(1.1^n)$.

Speckenmeyer [41] has shown that the problem of finding a maximum size transitive subtournament in a given tournament is NP-hard. Moon [34] has shown that every n-vertex tournament contains at most $O(1.717^n)$ *maximal* transitive subtournaments. An algorithm of Schwikowski & Speckenmeyer [39] enumerates all maximal transitive subtournaments with polynomial work per transitive subtournament. Altogether, this yields an $O^*(1.717^n)$ exact algorithm for finding a maximum size transitive subtournament. Can you do better?

In the MAX-CUT problem, the input consists of an n-vertex graph $G = (V, E)$. The problem is to find a cut of maximum cardinality, that is, a subset $X \subseteq V$ of the vertices that maximizes the number of edges between X and $V - X$. The MAX-CUT problem can be solved easily in $O^*(2^n)$ time by enumerating all possible certificates X. Fedin & Kulikov [18] present an $O^*(2^{|E|/4})$ time algorithm for MAX-CUT; however, it seems a little bit strange to measure the time complexity for this problem in terms of $|E|$ and not in terms of $n = |V|$.

Williams [43] developed the following beautiful approach for MAX-CUT: We partition the vertex set V into three parts V_0, V_1, V_2 that are of roughly equal cardinality $n/3$. We introduce a complete tri-partite auxiliary graph that contains one vertex for every subset $X_0 \subseteq V_0$, one vertex for every subset $X_1 \subseteq V_1$, and one vertex for every subset $X_2 \subseteq V_2$. For every subset $X_i \subseteq V_i$ and every $X_j \subseteq V_j$ with $j = i + 1 \pmod 3$, we introduce the directed edge from X_i to X_j. This edge receives a weight $w(X_i, X_j)$ that equals the number of edges in G between X_i and $V_i - X_i$ plus the number of edges between X_i and $V_j - X_j$ plus the number of edges between X_j and $V_i - X_i$. Note that for $X_i \subseteq V_i$ $(i = 0, 1, 2)$ the cut $X_0 \cup X_1 \cup X_2$ cuts exactly $w(X_0, X_1) + w(X_1, X_2) + w(X_2, X_0)$ edges in G. Consequently, the following three statements are equivalent:

- The graph G contains a cut with z edges.
- The auxiliary graph contains a 3-clique with total edge weight z.
- There exist non-negative integers z_{01}, z_{12}, z_{20} with $z_{01} + z_{12} + z_{20} = z$, such that the auxiliary graph contains a 3-clique on three vertices $X_i \subseteq V_i$ $(i = 0, 1, 2)$ with $w(X_0, X_1) = z_{01}$ and $w(X_1, X_2) = z_{12}$ and $w(X_2, X_0) = z_{20}$.

The condition in the third statement is easy to check: There are $O(|E|^3)$ possible triples (z_{01}, z_{12}, z_{20}) to consider. For each such triple, we compute a corresponding simplified version of the auxiliary graph that only contains the edges of weight z_{ij} between vertices $X_i \subseteq V_i$ and $X_j \subseteq V_j$. Then everything boils down to finding a 3-clique in the simplified auxiliary graph on $O(2^{n/3})$ vertices.

Fact 4.6 *The MAX-CUT problem can be solved in $O^*(2^{\omega\, n/3})$ time and $O^*(2^{\omega\, n/3})$ space. Note that $2^{\omega\, n/3} < 1.732^n$.*

Of course, William's algorithm could also be built around a partition of the vertex set V into four parts of roughly equal cardinality $n/4$, or around a partition of the vertex set V into k parts of roughly equal cardinality n/k. The problem then boils down to finding a k-clique in some simplified auxiliary graph on $O(2^{n/k})$ vertices. With the currently known k-CLIQUE algorithms, this will not give us an improved time complexity.

Open problem 4.7

(a) Design a faster exact algorithm for MAX-CUT.

(b) Construct an $O^*(1.99^n)$ time and polynomial space algorithm for MAX-CUT.

An input of the NP-hard BISECTION problem consists of an n-vertex graph $G = (V, E)$. The problem is to find a subset $X \subseteq V$ with $|X| = n/2$ that minimizes the number of edges between X and $V - X$. The approach of Williams yields an $O^*(2^{\omega\, n/3})$ time algorithm for BISECTION. Can you do better?

References

[1] F. Afrati, S. Cosmadakis, C.H. Papadimitriou, G. Papageorgiou, and N. Papakonstantinou (1986). The complexity of the travelling repairman problem. *RAIRO Informatique Théorique et Applications 20*, 79–87.

[2] N. Alon, R. Yuster, and U. Zwick (1995). Color-coding. *Journal of the ACM 42*, 844-856.

[3] J.F. Banzhaf III (1965). Weighted voting doesn't work: A mathematical analysis. *Rutgers Law Review 19*, 317–343.

[4] E.T. Bax (1993). Inclusion and exclusion algorithm for the Hamiltonian path problem. *Information Processing Letters 47*, 203–207.

[5] E.T. Bax (1994). Algorithms to count paths and cycles. *Information Processing Letters 52*, 249–252.

[6] E.T. Bax and J. Franklin (1996). A finite-difference sieve to count paths and cycles by length. *Information Processing Letters 60*, 171–176.

[7] R. Beigel (1999). Finding maximum independent sets in sparse and general graphs. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA'1999)*, 856–857.

[8] M. Ben-Or (1983). Lower bounds for algebraic computation trees. In *Proceedings of the 15th Annual ACM Symposium on the Theory of Computing (STOC'1983)*, 80–86.

[9] D. Coppersmith (1997). Rectangular matrix multiplication revisited. *Journal of Complexity 13*, 42–49.

[10] D. Coppersmith and S. Winograd (1990). Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation 9*, 251–280.

[11] D. Dobkin and R.J. Lipton (1978). A lower bound of $\frac{1}{2}n^2$ on linear search programs for the knapsack problem. *Journal of Computer and System Sciences 16*, 413–417.

[12] R.G. Downey and M.R. Fellows (1995). Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science 141*, 109–131.

[13] R.G. Downey and M.R. Fellows (1999). *Parameterized complexity.* Springer Monographs in Computer Science.

[14] F. Eisenbrand and F. Grandoni (2004). On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, to appear.

[15] D. Eppstein (2003). The traveling salesman problem for cubic graphs. In *Proceedings of the 8th International Workshop on Algorithms and Data Structures (WADS'2003)*, Springer-Verlag, LNCS 2748, 307–318.

[16] J. Erickson (1999). Lower bounds for linear satisfiability problems. *Chicago Journal of Theoretical Computer Science 1999(8).*

[17] J. Erickson (2004). Private communication.

[18] S.S. Fedin and A.S. Kulikov (2002). Solution of the maximum cut problem in time $2^{|E|/4}$. (In Russian). *Zapiski Nauchnykh Seminarov Sankt-Peterburgskoe Otdeleniya Matematicheskiĭ Institut imeni V.A. Steklova 293*, 129–138.

[19] U. Feige and J. Kilian (1997). On limited versus polynomial nondeterminism. *Chicago Journal of Theoretical Computer Science 1997.*

[20] A. Gajentaan and M.H. Overmars (1995). On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry 5*, 165–185.

[21] M.R. Garey and D.S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco.

[22] M. Held and R.M. Karp (1962). A dynamic programming approach to sequencing problems. *Journal of SIAM 10*, 196–210.

[23] E. Horowitz and S. Sahni (1974). Computing partitions with applications to the knapsack problem. *Journal of the ACM 21*, 277–292.

[24] X. Huang and V. Pan (1998). Fast rectangular matrix multiplication and applications. *Journal of Complexity 14*, 257–299.

[25] R.Z. Hwang, R.C. Chang, and R.C.T. Lee (1993). The searching over separators strategy to solve some NP-hard problems in subexponential time. *Algorithmica 9*, 398–423.

[26] A. Itai and M. Rodeh (1978). Finding a minimum circuit in a graph. *SIAM Journal on Computing 7*, 413–423.

[27] A. Itai, C.H. Papadimitriou, and J.L. Swarcfiter (1982). Hamiltonian paths in grid graphs. *SIAM Journal on Computing 11*, 676–686.

[28] V. Kann (1992). On the approximability of NP-complete optimization problems. Ph.D. thesis, Kungliga Tekniska Högskolan, Stockholm.

[29] R.M. Karp (1982). Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters 1*, 49–51.

[30] B. Klinz and G.J. Woeginger (2004). Faster algorithms for computing power indices in weighted majority games. *Mathematical Social Sciences*, to appear.

[31] E.L. Lawler (1976). A note on the complexity of the chromatic number problem. *Information Processing Letters 5*, 66–67.

[32] R.J. Lipton and R.E. Tarjan (1979). A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics 36*, 177–189.

[33] R.J. Lipton and R.E. Tarjan (1980). Applications of a planar separator theorem. *SIAM Journal on Computing 9*, 615–627.

[34] J.W. Moon (1971). On maximal transitive subtournaments. *Proceedings of the Edinburgh Mathematical Society 17*, 345–349.

[35] J. Nešetřil and S. Poljak (1985). On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae 26*, 415–419.

[36] K. Prasad and J.S. Kelly (1990). NP-completeness of some problems concerning voting games. *International Journal of Game Theory 19*, 1–9.

[37] J.M. Robson (1986). Algorithms for maximum independent sets. *Journal of Algorithms 7*, 425–440.

[38] R. Schroeppel and A. Shamir (1981). A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM Journal on Computing 10*, 456–464.

[39] B. Schwikowski and E. Speckenmeyer (2002). On enumerating all minimal solutions of feedback problems. *Discrete Applied Mathematics 117*, 253–265.

[40] W.D. Smith (1988). Studies in computational geometry motivated by mesh generation. Ph.D. Thesis, Princeton University, Princeton.

[41] Speckenmeyer (1989). On feedback problems in digraphs. In *Proceedings of the 15th Workshop on Graph-Theoretic Concepts in Computer Science (WG'1989)*, LNCS 411, Springer Verlag, 218–231.

[42] R.E. Tarjan and A.E. Trojanowski (1977). Finding a maximum independent set. *SIAM Journal on Computing 6*, 537–546.

[43] R. Williams (2004). A new algorithm for optimal constraint satisfaction and its implications. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP'2004)*, Springer Verlag, 2004.

[44] G.J. Woeginger (2003). Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization – Eureka, you shrink!"*, LNCS 2570, Springer Verlag, 185–207.