Exact Algorithms for Exact Satisfiability and Number of Perfect Matchings

Andreas Björklund and Thore Husfeldt

Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden
andreas.bjorklund@anoto.com, thore.husfeldt@cs.lu.se

Abstract. We present exact algorithms with exponential running times for variants of n-element set cover problems, based on divide-and-conquer and on inclusion–exclusion characterisations.

We show that the Exact Satisfiability problem of size l with m clauses can be solved in time $2^m l^{O(1)}$ and polynomial space. The same bounds hold for counting the number of solutions. As a special case, we can count the number of perfect matchings in an n-vertex graph in time $2^n n^{O(1)}$ and polynomial space. We also show how to count the number of perfect matchings in time $O(1.732^n)$ and exponential space.

Using the same techniques we show how to compute Chromatic Number of an n-vertex graph in time $O(2.4423^n)$ and polynomial space, or time $O(2.3236^n)$ and exponential space.

1 Introduction

We present exact algorithms with exponential running times using two simple techniques for a number of related NP-hard problems, including exact satisfiability, disjoint set covering, counting the number of perfect matchings, graph colouring, and TSP.

Our algorithms run in polynomial space and have running times of the form $O(c^n)$, where n is a natural parameter of the instance, such as 'number of vertices' or 'number of clauses,' but smaller than the instance size. For some NP problems, this can be achieved by exhaustive search: a maximum clique can be found by inspecting all $2^{|V|}$ vertex subsets. But in general, this fails: exhaustive search for a Hamiltonian cycle would require checking $\big(|V|-1\big)!$ permutations. For the problems in this paper, finding such algorithms was an open problem.

Techniques

The first idea is divide-and-conquer, where we divide the instance into an *exponential number* of sub-instances, halving n at each step. This leads to running times of the form

$$T(n) = 2^n n^{O(1)} T\big(\tfrac{1}{2}n\big),$$

which is $O(c^n)$, and the space is polynomial in n.

The second idea is inclusion–exclusion, in which we express the problem in the form

$$\sum_{S\subseteq\{1,\ldots,n\}} (-1)^{|S|} f(S),$$

M. Bugliesi et al. (Eds.): ICALP 2006, Part I, LNCS 4051, pp. 548–559, 2006.

where $f(S)$ is some easier-to-calculate predicate depending on the problem. The running time is 2^n times the time used to calculate $f(S)$, and the space is dominated by the algorithm for $f(S)$.

Both ideas obviously give rise to polynomial-space algorithms, but we will also consider exponential-space algorithms that reduce the running times. For the problems studied in this paper, the inclusion–exclusion approach always gives the better worst-case time bounds. However, the divide-and-conquer approach is more versatile and can be applied to natural variant problems, such as weighted versions; it may also run faster in practice.

Both ideas also turn out to be old. Gurevich and Shelah used exponential divide-and-conquer as a building block for an expected linear time algorithm [13], and Feige and Kilian use it to compute the bandwidth in an unpublished manuscript [12]. The idea of using inclusion-exclusion was used by Bax [2] to find Hamiltonian Cycles. Before that, a much older example of relevance to the present paper is the Ryser formula for the permanent [20], which counts the number of matchings in a bipartite graph.

Exact Satisfiability

Most of the results and techniques of this paper can be presented in terms of the Exact Satisfiability problem. It is equivalent to a set covering problem and generalises perfect matching in graphs, see Sec. 2 for the precise relationships.

Given a formula in disjunctive normal form with m clauses in n variables and total size $l = O(mn)$, the Exact Satisfiability problem asks for the existence of an assignment that satisfies exactly one literal of each clause. This can be solved easily in polynomial space and $2^n l^{O(1)}$ time by considering all assignments, and many papers have improved the 2^n factor.

Here, we analyse the problem in terms of the number m of clauses. For ordinary (non-exact) satisfiability, the seminal result is the DPLL procedure from 1962 [9], which gives a polynomial space algorithm with running time $2^m n^{O(1)}$. Again, many papers have since improved this result.

However, for *Exact* Satisfiability, no such result was known; the best bounds are exponential time and space $2^m l^{O(1)}$ using dynamic programming, or polynomial space and time $m! l^{O(1)}$ [17]. Prior to the current paper, no polynomial space algorithm with running time $c^m l^{O(1)}$ was known for any constant c, an open problem observed in [5,17].

Proposition 1 provides such an algorithm for many choices of m; the running time is bounded by $c^m l^{O(1)}$ except when $\log n < m \leq \log n \log\log n$. The algorithm works for the weighted case as well. Our inclusion–exclusion based algorithm in Proposition 2 is even simpler and faster in the worst case, achieving the desired running time of $2^m l^{O(1)}$. Both algorithms use space polynomial in l.

Number of perfect matchings

The inclusion–exclusion based algorithm actually counts the *number* of exactly satisfying assignments and therefore solves the $\#P$-complete counting variant of XSAT. A well-studied special case of this is *perfect matching*: Given a graph $G = (V, E)$ on n vertices, find a subset $M \subseteq E$ of disjoint edges that cover V.