Journal of Discrete Algorithms 26 (2014) 89–97

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Counting solutions to CSP using generating polynomials

Daniel Berend [a], Shahar Golan [b,*]

[a] Department of Computer Science and Department of Math, Ben-Gurion University of the Negev, POB 653, Beer-Sheva 84105, Israel
[b] Yahoo! Labs, Matam technology park, Haifa 31905, Israel

ARTICLE INFO

Article history:
Received 18 April 2013
Received in revised form 13 January 2014
Accepted 22 January 2014
Available online 19 February 2014

Keywords:
Constraint satisfaction problem
Generating functions
Counting problem
CNF SAT
Treewidth

ABSTRACT

Constraint Satisfaction Problems (CSPs) are ubiquitous in computer science and specifically in AI. This paper presents a method of solving the counting problem for a wide class of CSPs using generating polynomials. Analysis of our method shows that it is much more efficient than the classic dynamic programming approach. For example, in the case of #SAT, our algorithm improves a result of Samer and Szeider. The presented algorithms mostly use algebraic operations on multivariate polynomials, which allows application of known optimizations and makes it possible to use existing software to implement them easily.

1. Introduction

A Constraint Satisfaction Problem (CSP) is usually defined as a triplet $(X, \mathcal{D}, \mathcal{C})$, where $X = (X_1, X_2, \ldots, X_{n_1})$ is an n_1-tuple of variables, $\mathcal{D} = (D_1, D_2, \ldots, D_{n_1})$ an n_1-tuple of domains, and $\mathcal{C} = \{C_1, C_2, \ldots, C_{n_2}\}$ a set of constraints. Each constraint in $\mathcal{C}$ is a pair (t, R), where $t = (X_{i_1}, X_{i_2}, \ldots, X_{i_l})$ is an l-tuple of variables from X and $R \subseteq \prod_{j=1}^{l} D_{i_j}$. A *solution* of the CSP is an assignment of values $d_i \in D_i$ to the variables X_i, satisfying all the constraints. For such a CSP, the corresponding *counting problem* $\#(X, \mathcal{D}, \mathcal{C})$ is the problem of finding the number of solutions of $(X, \mathcal{D}, \mathcal{C})$.

CSPs are ubiquitous in computer science and specifically in AI. Some of the most fundamental problems, such as CNF-SAT (conjunctive normal form satisfiability) and graph colorability, may be formulated as CSPs. CSPs are known to be NP-complete in general, and hence it is interesting to find "islands of tractability" within this family of problems. One of the main approaches is to find families of CSPs that do not require backtracking. For example, the family of CSPs that are definable using datalog programs with at most k variables is tractable [2]. As another example, we mention that, if a CNF has about as many clauses as variables, then it is possible to solve the *minimal unsatisfiability* problem in polynomial time [12].

An important family of tractable CSPs comprises those with associated graphs of bounded treewidth. The notion of treewidth was introduced by Robertson and Seymour [9,10], and appears in many algorithms in graph theory. (An overview can be found in [1].) The input of problems from various fields can be represented by corresponding graphs. Thus, a large range of problems is efficiently solved by the general methods designed for problems on graphs with bounded treewidth. It is interesting to note that, in [7], several computational problems involving multivariate polynomials are solved, using optimizations for the corresponding graphs of bounded treewidth. In the current paper we solve problems on graphs with bounded treewidth, using optimizations that were developed for multivariate polynomials computations.

* Corresponding author.
E-mail addresses: berend@cs.bgu.ac.il (D. Berend), golansha@gmail.com (S. Golan).

http://dx.doi.org/10.1016/j.jda.2014.01.004
1570-8667/

Graph representations for CSPs can be constructed in many ways. The common representations are primal, dual and incidence constraint graph. By [6], a CSP of primal treewidth k has incidence treewidth at most $k+1$. Note that the opposite implication does not hold; a CSP may have a corresponding primal graph of any width, while the corresponding incidence graph is a tree (i.e., with treewidth 1). In this paper we will use the incidence graph to represent the CSP. The incidence graph is a bipartite graph and has both the variables and the constraints as its vertices. Thus it will be natural to take the quantity $n = n_1 + n_2$ as a measure of the size of a problem.

There is a vast literature on the subject of NP-hard CSPs, restricted to those whose corresponding graphs are of bounded treewidth. In [3], a general method of solving CSPs with bounded treewidth in polynomial time is described. This method is based on a relational representation of the domains of the variables and of the constraints. The main approach for this kind of CSPs is to use dynamic programming in order to compose a full solution for the problem. An application of this method is found in [11], where an algorithm for #SAT is presented. For a CNF with incidence graph of n vertices and treewidth k, and with clauses of size at most l, this algorithm solves the problem in $O(2^k(l+2^k)kn)$ time. The methods described in this paper can be applied to solve this problem in $O(3^k kn)$ time. In [5], generating polynomials are used in order to execute the dynamic programming process more efficiently. This approach is similar to the one presented here, but it is applicable only for the primal constraints graph (having larger treewidth in general) and only to 2-CSPs (where all constraints refer to at most 2 variables).

The methods for solving general CSPs with bounded treewidth do not (and cannot) take into account optimizations that can be used when restricting ourselves to certain classes of CSPs. The main challenge is to present methods that allow significant improvements for some wide enough families of CSPs by capitalizing on their special properties. In this paper we deal with the class of CSPs in which the domains D_i and constraints $C_j = (t_j, R_j)$ satisfy the following conditions:

- Each D_i is a finite subset of $\mathbb{Z}$, $1 \leqslant i \leqslant n_1$.
- Each R_j is of the form $\{\mathbf{a} \in \mathbb{Z}^{n_1} \mid P_j(\mathbf{a}) \in \mathcal{L}_j\}$, where:
  i) $P_j = \sum_{i=1}^{n_1} P_{ji}(X_i)$, the P_{ji}'s being integer polynomials in one variable.
  ii) It is easy to check whether a given $a \in \mathbb{Z}$ belongs to $\mathcal{L}_j$.

We shall refer to such CSPs as *integer CSPs*. It is interesting to compare this family of problems to that of integer linear programming (ILP) problems. In integer CSPs, we allow our constraints to be polynomial (of a special type), and not just linear. Moreover, the sets $\mathcal{L}_j$ are not necessarily sets of consecutive integers. On the other hand, unlike in ILP, we require the polynomials in the constraints to be defined over $\mathbb{Z}$ and the domains of definition of the variables to be finite.

The *incidence graph* of a CSP is a bipartite graph, where one vertex class consists of the variables, and the other – of the constraints. A variable X_i is adjacent to a constraint $C_j = (t_j, R_j)$ if it appears in t_j. It is important to note that this means that P_{ji} is non-constant, namely if X_i really appears in C_j in a non-trivial way. This point is crucial since the complexity of our algorithm depends on the treewidth of the incidence graph, which could only grow if we added unneeded edges.

An integer CSP is *non-negative* if all monomials P_{ji} assume only non-negative values on the domains D_i. Note that every integer CSP can be turned into a non-negative CSP simply by adding sufficiently large constants to the P_{ji}'s.

In this paper we provide algorithms for solving the counting problem for integer CSPs. Instead of using the relational representation of the partial solutions, we represent all the information by generating polynomials. It turns out that, by applying simple algebraic operators (such as addition and multiplication) to these polynomials, we solve the problem.

In Section 2 we present the main results. Section 3 gives the definitions relevant to the concepts of tree decomposition and treewidth. Section 4 contains the proofs of the results regarding our basic algorithm for solving the counting problem for integer CSPs. In Section 5 we present an improvement for the basic algorithm and provide the relevant proofs. We conclude in Section 6.

2. The main results

As mentioned in the introduction, any integer CSP can be translated to a non-negative CSP. Our algorithms work, and have the same complexity, for an integer CSP and for its translation. This point will be explained in Remark 4.7. Hence in this section $(X, \mathcal{D}, \mathcal{C})$ is always a non-negative CSP. This will allow us to give a clear description of the algorithm, with less complicated expressions. We denote by k the treewidth of the corresponding incidence graph. Our basic result is the following:

Theorem 2.1. *Let us assume that, for some A:*

(1) $P_j(\prod_{i=1}^{n_1} D_i) \subseteq \{0, 1, \ldots, A-1\}$, $1 \leqslant j \leqslant n_2$.
(2) $|D_i| \leqslant A$, *for any* $1 \leqslant i \leqslant n_1$.

Algorithm 1 solves $\#(X, \mathcal{D}, \mathcal{C})$ in time $O(nkA^{k+1} \log A)$.

The first assumption in the theorem is usually much stronger than the second, but we cannot always ignore the second.