An Algebraic Approach to Constraint Satisfaction Problems

Igor Rivin
Wolfram Research, Inc.
Champaign, Illinois 61826

Ramin Zabih
Computer Science Department
Stanford University
Stanford, California 94305

Abstract

A constraint satisfaction problem, or CSP, can be reformulated as an integer linear programming problem. The reformulated problem can be solved via polynomial multiplication. If the CSP has n variables whose domain size is m, and if the equivalent programming problem involves M equations, then the number of solutions can be determined in time $O(nm2^{M-n})$. This surprising link between search problems and algebraic techniques allows us to show improved bounds for several constraint satisfaction problems, including new simply exponential bounds for determining the number of solutions to the n-queens problem. We also address the problem of minimizing M for a particular CSP.

1 Introduction

We are interested in solving certain search problems that arise in combinatorics and in artificial intelligence. These problems are called Constraint Satisfaction Problems (CSP's), and include such familiar tasks as graph-coloring and the n-queens problem. Typically, such problems are solved via exhaustive search. We propose an alternative method, based on converting the CSP into an integer linear programming problem, which can be solved via polynomial multiplication.

In section 2 we describe constraint satisfaction problems and show how they can be converted into integer linear programming problems. We describe how to solve integer linear programming problems via polynomial multiplication in section 3, and analyze the time requirements of this approach. In section 4 we give an example of our method, applying it to the n-rooks problem, and show some surprising similarities with standard search techniques. In section 5 we address the problem of finding a good integer linear programming equivalent of a CSP, while in section 6 we describe some extensions to our techniques. Finally, in section 7, we derive new simply exponential bounds for the n-queens problem and the toroidal n-queens problem.

2 CSP's and integer linear programming

2.1 Constraint satisfaction problems

Formally, a CSP has a set of variables and a domain of values,

$$V = \{v_1, v_2, \ldots, v_n\} \text{ the set of variables,}$$
$$D = \{d_1, d_2, \ldots, d_m\} \text{ the set of values.}$$

Every variable x_i must be assigned a value d_j; such an assignment will be denoted $v_i \leftarrow d_j$.

A CSP also consists of some constraints saying which assignments of values to variables are compatible. We will restrict our attention to *binary* CSP's, where the constraints involve only two variables. (This is partly for simplicity, and partly because a non-binary CSP can be converted into a binary CSP by the introduction of additional variables.) A solution to the constraint satisfaction problem is an n-tuple of assignments that satisfies all the constraints.

There are several ways to formulate the task of solving a CSP.

> The *satisfiability problem* is to determine the existence of a solution to a CSP.
>
> The *counting problem* is to determine the number of solutions.
>
> The *enumeration problem* is to find all solutions.

The counting problem is a special case of the enumeration problem, and the satisfiability problem is a special case of the counting problem.

Since many $\mathcal{NP}$-complete problems, such as graph-coloring, are CSP's, the satisfiability problem is easily seen to be $\mathcal{NP}$-complete [Garey and Johnson, 1979]. We therefore cannot expect to solve the satisfiability, counting or enumeration problems in polynomial time. Typical CSP solution techniques like those surveyed in [Mackworth, 1987] perform an exhaustive search, backtracking

through all n-tuples of assignments. While these methods add some additional cleverness beyond brute-force search, their worst-case running time is still $O(m^n)$.

We will concentrate on solving the counting problem (and therefore the satisfiability problem as well). While our methods are easily extended to the enumeration problem (see section 6.1), our approach to the counting problem is surprisingly simple to analyze. In fact, we can do provably better than backtrack search for some problems, in the following sense. Backtracking counts the number of solutions by enumerating them, and hence must take at least as much time as the number of solutions. Our techniques can determine the number of solutions without actually enumerating them, so the number of solutions is not a lower bound on our running time.

2.2 Converting CSP's into integer linear programming problems

Our basic approach is to turn a CSP into an integer linear programming problem. For every possible assignment $v_i \leftarrow d_j$ we introduce a variable $x_{i,j}$. Then for every CSP variable v_i we have an equation

$$\sum_{1 \le j \le m} x_{i,j} = 1. \tag{1}$$

This expresses the fact that every CSP variable has exactly one value. In addition, for every set of assignments that the constraints prohibit, we have an inequality which states that no more than one of these assignments can hold at once. For example, if the pair of assignments $\{v_i \leftarrow d_j, v_k \leftarrow d_l\}$ is forbidden by the constraints, we will have the inequality

$$x_{i,j} + x_{k,l} \le 1. \tag{2}$$

Now consider the problem of finding all solutions to the equations of the form (1) and (2) over the non-negative integers. (It is clear that any solution will assign every variable $x_{i,j}$ either 0 or 1.) This is an integer linear programming problem, which corresponds to the original CSP. There are nm variables $x_{i,j}$, and a solution to the set of equations with $x_{i,j} = 1$ corresponds to a solution to the CSP containing the assignment $v_i \leftarrow d_j$.

3 Solving CSP's by multiplying polynomials

An integer linear programming problem is a set of linear Diophantine inequalities. We can use an approach based on generating functions to determine the number of solutions. This involves multiplying out a certain polynomial, and gives us a way of solving the original CSP.

3.1 Solving linear Diophantine equations

A set of M linear Diophantine equation in N unknowns has the form

$$\sum_{j=1}^{N} w_{i,j} x_j = s_i, \quad i = 1, \ldots, M. \tag{3}$$

The x_j's are the unknowns, and the $w_{i,j}$'s and s_i's are non-negative integers. The problem at hand is to determine the number of solutions over the non-negative integers.

Consider the generating function

$$\Phi(Y_1, \ldots, Y_M) = \prod_{j=1}^{N} \frac{1}{\left(1 - \prod_{i=1}^{M} Y_i^{w_{i,j}}\right)}. \tag{4}$$

The formal parameters of this function are the variables $Y_1, \ldots, Y_M$, and on such variable is associated with each equation. We need one non-trivial bit of mathematics: the number of solutions to equation (3) is the coefficient of

$$\prod_{i=1}^{M} Y_i^{s_i} \tag{5}$$

in Φ. (This fact was proved in [Euler, 1748], and is discussed in [Schrijver, 1986, pages 375–376].)

3.2 Applications to integer linear programming

Now consider the integer linear programming problem corresponding to a CSP. Every $w_{i,j}$ will be either 0 or 1. Assume for the moment that all the equations are of the form (1), hence $s_i = 1$. (We will relax this restriction shortly.) We can count the number of solutions by finding the coefficient of the monomial $\overline{Y} = Y_1 Y_2 \cdots Y_M$ in Φ. We apply the identity

$$\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k$$

to equation (4), giving

$$\Phi(Y_1, \ldots, Y_M) = \prod_{j=1}^{N} \left(1 + \sum_{k=1}^{\infty} \left(\prod_{i=1}^{M} Y_i^{w_{i,j}}\right)^k\right). \tag{6}$$

We can further simplify this if we remember that we are only interested in the coefficient of $\overline{Y}$. We can therefore ignore any term divisible by the square of any Y_i (terms that are not so divisible are called *square free*). By a slight abuse of notation we will refer to the square free part of the generating function as Φ. We next note that

$$\sum_{k=1}^{\infty} \left(\prod_{i=1}^{M} Y_i^{w_{i,j}}\right)^k = \prod_{i=1}^{M} Y_i^{w_{i,j}} + \text{other terms},$$

where these latter terms are not square free and so can be discarded. This gives us an expression for Φ as a product of binomials, namely

$$\Phi(Y_1, \ldots, Y_M) = \prod_{j=1}^{N} \left(1 + \prod_{i=1}^{M} Y_i^{w_{i,j}}\right). \tag{7}$$

All we need do in order to find the number of solutions to the system of equations is compute the coefficient of $\overline{Y}$ in equation (7).

The obvious way to solve this problem is to simply multiply out Φ and look at the correct term of the result. We can do this iteratively: we multiply the first two binomials of Φ and throw away the terms that are not square free, then repeat with the next binomial, etc.

We can now relax our assumption that all the equations are of the form (1). Suppose that instead we have exactly one Diophantine inequality

$$\sum_{1 \le j \le m} x_{i,j} \le 1. \tag{8}$$

This implicitly defines two equalities, one of which has 0 on the right hand side, and one of which has 1. The number of solutions to the inequality is the sum of the number of solutions to the two implicit equalities.

At first glance, it would seem that solving a system with α inequalities would require solving 2^α implicit systems of equalities. However, a closer look at the generating function shows that we only need to solve one of these implicit systems. Suppose that we replace each inequality with an equality that has a 1 on the right hand side and examine the square free part of Φ. The number of solutions to each of the 2^α implicit systems of equalities will be the coefficient of the appropriate term of Φ.

For example, suppose that there are three equations Y_0, Y_1 and Y_2, and one inequality

$$\varphi \le 1, \tag{9}$$

where φ is a sum. The inequality (9) implicitly asserts

$$\varphi = 0 \tag{9.0}$$

or

$$\varphi = 1. \tag{9.1}$$

The number of solutions to the original system (which included (9)) is the sum of the number of solutions when (9) is replaced by (9.0) plus the number of solutions when (9) is replaced by (9.1).

However, suppose that we simply replace (9) by (9.1) and calculate Φ. Let Y_3 be the generating function variable associated with (9.1). The number of solutions is the coefficient of $Y_0Y_1Y_2Y_3$ in Φ. However, the number of solutions when (9) is replaced by (9.0) is also part of Φ, namely the coefficient of $Y_0Y_1Y_2$. We thus do not need to calculate separate generating functions for both systems of equalities implicitly defined by (9). We can simply replace the inequality by an equality with 1 on the right hand side and calculate Φ. At the end, we add together coefficients for the implicit systems of equalities to determine the number of solutions to the original problem.

To summarize, we can replace the α inequalities with equalities and calculate the square free part of Φ. The total number of solutions will no longer be the coefficient of $\overline{Y}$, but rather the sum of 2^α coefficients of different terms. This justifies our earlier assumption that the integer linear programming problem consisted entirely of equalities of the form (1).

3.3 Time requirements

The time required to solve the CSP is the time to multiply out the polynomial Φ. We can do this in N iterations, where on each iteration we multiply a binomial by a polynomial that represents the product thus far. Two polynomials can be multiplied in time proportional to the product of their size, so we need to bound the size of the intermediate results.

Naively, there could be as many as N^M terms, since there are M Y's, and the degree of each one is potentially as high as N. However, remember that at each stage we will discard all the terms that are not square free. (In algebraic terminology, this is described as doing the computation over the polynomial ring

$$\mathbf{Z}[Y_1, \ldots, Y_M]/(Y_1^2, \ldots, Y_M^2)$$

instead of $\mathbf{Z}[Y_1, \ldots, Y_M]$.) This gives a better bound on the number of terms. Every Y will have degree 0 or 1, hence there are no more than 2^M terms.

Summarizing, if we compute Φ iteratively we will do N multiplications, where each multiplication computes the product of a binomial and a polynomial with no more than 2^M terms. Each multiplication will take time 2^{M+1}, which is $O(2^M)$. This puts a bound of $O(N2^M)$ on the total time complexity of our algorithm.[1] We will show in section 6.2 how to reduce the exponent to $M - n$ in the worst case.

It is worth noting that we cannot expect to find a polynomial time method for determining the value of an arbitrary coefficient of Φ. [Schrijver, 1986] proved that integer linear programming is $\mathcal{NP}$-complete even when the coefficients involved are restricted to be 0 or 1.

4 An example of our approach

The n-rooks problem consists of placing n rooks on an n-by-n chessboard so no two rooks attack each other.

[1] This analysis ignores the time required for scalar multiplication. We can take this into account by noting that the size of the coefficients will be bounded by M^N, hence their length is $O(N \log M)$. So the scalar multiplication can be done in time $O(N \log M \log(N \log M) \log\log(N \log M))$ [Aho *et al.*, 1974].

While it is clear that the solutions to this problem are the $n!$ permutations of n elements, this problem has a particularly simple encoding as an integer linear programming problem, and is therefore a useful example of our approach.

There are n^2 assignments $v_i \leftarrow d_j$ for this CSP, so we will convert it into a programming problem with n^2 variables $x_{i,j}$. There will be $2n$ equations.[2] The equations specify that there is exactly one rook in each row and each column. A typical equation is

$$x_{2,1} + x_{2,2} + \ldots + x_{2,n} = 1, \tag{10}$$

which encodes the fact that there is exactly one rook in the 2nd column. We can number these $2n$ equations so that equation i says that there is exactly one rook in the i'th column, and equation $n+i$ says that there is exactly one rook in the i'th row. The formal parameter Y_i of the generating function corresponds to the i'th constraint equation.

Applying equation (7), Φ is a product of n^2 binomials. There will be one binomial for every square on the chessboard, and the binomial for the square in row i and column j will be

$$(1 + Y_{n+i}Y_j).$$

The number of solutions is the coefficient of

$$\overline{Y} = Y_1 \cdots Y_n Y_{n+1} \cdots Y_{2n}$$

in the expansion of Φ.

Surprisingly, there is a way of multiplying out Φ that corresponds closely to backtrack search. Suppose that we attempt to find all combinations of binomial terms whose product is $\overline{Y}$. We can start by picking the second (i.e., non-constant) term of $(1 + Y_{n+i}Y_j)$. This forces us to pick the first (constant) term of every binomial whose second term includes either Y_{n+i} or Y_j, to ensure the product is square free. Now we can pick the non-constant term of some other binomial and continue, until we have produced $\overline{Y}$. The number of such combinations will be the coefficient of $\overline{Y}$ in the expansion of Φ. (It is also possible that the combination of terms selected guarantees that no selection of terms from the remaining binomials will produce $\overline{Y}$. The simplest example of this occurs when a term has been selected from all but one binomial without producing $\overline{Y}$, where the remaining binomial's non-constant term does not contain the missing Y. This phenomenon, which does not occur in the n-rooks problem, will be used to improve our performance guarantee in section 6.2.)

This whole process corresponds closely to a simple search of the original CSP. Choosing the non-constant term of the binomial $(1 + Y_{n+i}Y_j)$ places a rook on the equivalent square, while choosing the constant term means there is no rook on that square. The process described above corresponds to placing a rook on the chessboard square in row i and column j, and thus ensuring there is no rook in any other square in that row or column. At the end, we will have satisfied every constraint and produced a solution. The alternative outcome, where we cannot satisfy every constraint, corresponds to a failure in backtrack search.

Note that the expansion of Φ does not explicitly represent the solutions. In fact, which assignments constitute a particular solution is lost by this approach, as the identity of the binomials is not recorded. We will show how to explicitly represent solutions in section 6.1. However, the fact that solutions are not explicitly represented is an advantage of our approach, as the number of solutions is no longer a lower bound to our running time. Applying the analysis of section 3.3, for this problem we have $N = n^2$ and $M = 2n$, so our running time is $O(n^2 4^n)$. This is significantly smaller than $n!$, the number of solutions, which is about $4^{O(n \log n)}$.

We have just described a correspondence between a way to compute the coefficient of $\overline{Y}$ in the expansion of Φ and a simple search technique for solving the original CSP. However, searching all the combinations of binomial terms is not our intended method. We believe it would be better to multiply Φ out iteratively, as described earlier. While this does not correspond in any obvious way to applying backtrack search to the original problem, it does seem to have some similarity to breadth-first techniques. It may thus be susceptible to parallel implementation techniques like those described in [Dixon and de Kleer, 1988].

5 Finding a good encoding

A potential difficulty with our approach is that the running time is exponential in M, the number of equations needed to encode the CSP. There can be many such encodings, and it is important to be able to find a good one. Clearly M must be at least n, as there is an equation of the form (1) for each CSP variable. The remaining equations come from the constraints. The problem is to find a way of encoding the constraints as small number of equations.

In the simplest encoding, there is an equation like (9) for every pair of assignments that the constraints rule out. However, this can be quite a lot of equations, as the following example illustrates. In the case where all pairs of assignments are ruled out this will result in n^2m^2 equations, while, as we will see, there is an encoding that results in only 1 equation.

Let us define the *assignment graph* corresponding to a particular CSP. The nodes of the assignment graph represent assignments, and there is an edge between two nodes if their assignments are ruled out by the constraints. A clique C in this graph corresponds to a set of pairwise incompatible assignments. The transformation described in section 2.2 will produce a variable $x_{i,j}$ for

[2] In section 5 we will discuss how to find good encodings.

every node, and will use $\binom{n}{2}$ equations, with the equation

$$x_{i,j} + x_{k,l} \leq 1$$

expressing the fact that these two assignments are incompatible. However, we can describe the fact that all the assignments in C are pairwise incompatible with the single equation

$$\underbrace{x_{i,j} + x_{k,l} + \ldots}_{C} \leq 1.$$

So we really only need one equation for every clique. If all pairs of assignments are incompatible, the entire assignment graph is a clique, and so the constraints can be represented with 1 equation instead of needing n^2m^2.

For the n-queens problem, the assignment graph has a node for each square on the chessboard. Two nodes are connected if queens on the two corresponding squares would attack each other. The maximal cliques correspond to the columns, rows, diagonals and antidiagonals of the chessboard.

We can therefore reduce the size of M by encoding every clique as a single equation. In an optimal encoding, M can be as small as the size of the smallest covering by cliques of the assignment graph. Because the problem of covering a graph by cliques is $\mathcal{NP}$-complete [Garey and Johnson, 1979], finding an optimal encoding would take exponential time. For special cases like the chessboard problems described above, it is easy to find an optimal encoding, but in general we would have to settle for a sub-optimal encoding. It is also possible to phrase the problem as finding a small vertex cover for the assignment graph, and this can also reduce M.

5.1 Which CSP's have good encodings?

One obvious question is whether there is a better characterization of the CSP's that have good encodings. While we do not have a very good answer to this question, an observation due to Bob Floyd suggests that there may be a relationship between the number of solutions to the CSP and the smallest possible value for M. Floyd has pointed out that a solution to the CSP corresponds to an n-clique in the complement of the assignment graph (equivalently, an independent set of size n in the assignment graph). It is possible that work in combinatorics will produce a relationship between the number of independent sets of fixed size in a graph and the size of the smallest vertex cover. Such a relationship would produce a characterization of the CSP's where our algorithm can be expected to work well, in terms of the number of solutions.

6 Some extensions

The method described above relies on multiplying out the polynomial Φ defined in equation (7). A simple modification to Φ allows us to solve the enumeration problem instead of the counting problem. It is also possible to reduce the running time of our algorithm by carefully arranging the order in which the binomials comprising Φ are multiplied out.

6.1 Solving the enumeration problem

Suppose that instead of determining the number of solutions to our CSP, we are interested in enumerating these solutions. We can still convert the CSP into an integer linear programming problem, and solve it via polynomial multiplication. The only difference is that we now need to consider a slightly different function than Φ.

Recall that we start by converting our CSP into a set of equations in $N = mn$ variables, one variable per assignment $v_i \leftarrow d_j$. Let us introduce N different variables α_j, $1 \leq j \leq N$. Now consider a slight variation of Φ, namely

$$\Phi_1 = \prod_{j=1}^{N} \left(1 + \alpha_j \prod_{i=1}^{M} Y_i^{w_{i,j}}\right). \tag{11}$$

We can solve the enumeration problem by determining the coefficient of $\overline{Y}$ in the expansion of Φ_1, just as we did with Φ.

Suppose we multiply out Φ_1 by selecting terms from different binomials. The α_j's will keep track of which binomials had their non-constant side selected. When we have a solution, the coefficient of $\overline{Y}$ will the product of a number of the α_j's, which will encode a solution. When Φ_1 is multiplied out completely (iteratively or otherwise), the coefficient of $\overline{Y}$ will not be an integer, but rather the sum of a number of different products of α_j's. Every such product is a solution to the CSP, and all solutions appear as such products.

6.2 Reducing the running time

An observation due to John Lamping can be used to reduce the running time of our algorithm from $O(nm2^M)$ to $O(nm2^{M-n})$. Suppose that before we multiply out Φ we first try to identify a subset S of the binomials comprising Φ and a variable Y such that

- Y never appears outside of the binomials in S, and
- some other variable Y' does not appear in S.

If we had such an S and Y, we could effectively reduce the number of variables by 1.

We can do this by first computing the product of the binomials in S. This can be grouped into terms based on the highest power of Y that divides each term

$$S = S_0 + YS_1 + Y^2S_2 \ldots$$

where S_i is not divisible by Y, and as usual we can discard the non square free terms. If Y is contained in the coefficient(s) corresponding to solutions to the CSP, then we can replace S by S_1. We can then forget about Y, as it never appears in the remaining part of Φ. Since Y' did

not appear in S, this reduces the number of variables by 1.

Reducing the number of variables reduces the running time of our algorithm as well, which is exponential in the number of variables. It turns out that we are guaranteed that there exist at least n such pairs of S's and Y's. This enables us to reduce the number of variables from M to $M - n$, which in turn reduces our running time to $O(nm2^{M-n})$.

There are n such S and Y pairs because the encoding of a CSP will have a single equation for each CSP variable, representing the fact that the variable must be assigned exactly one value.[3] Let Y correspond to the equation which constrains the CSP variable v to have exactly one value, and consider the binomials in Φ that contain Y. These are exactly the binomials that correspond to possible values for v, so there can be only m of them. Furthermore, this set of binomials forms an appropriate S, since no Y' corresponding to another CSP variable can appear. There are n such Y's, one per CSP variable, and these Y's will always appear in the coefficient(s) that we need to calculate.

It is even possible to take advantage of such S's and Y's when Y does not necessarily appear in the required coefficients. (Recall that this is a result of the integer linear programming equation that corresponds to Y actually representing an inequality, as discussed at the end of section 3.2.) If this happens, we can replace S by $S_0 + S_1$ and still discard Y. This has the effect of summing the coefficients that stand for the solutions to the two implicit systems of equalities early in the computation, rather than waiting to multiply Φ out before calculating this sum.

We can thus reduce the number of variables by at least n. As we saw in section 4, each binomial corresponds to a possible assignment $v_i \leftarrow d_j$. The ordering that is guaranteed to reduce the number of variables by n corresponds to considering all the assignments for a fixed v_i before moving on to the next CSP variable. On the n-queens problem this is equivalent to multiplying out the binomials in row-major order.

7 Some new bounds

An additional advantage of our approach is that we can produce non-trivial upper bounds for solving certain constraint satisfaction problems. So far, we have confined out attention to CSP's based on placing pieces on a chessboard.

The classic such problem is the n-queens problem, which was known to Gauss. The n-queens problem consists of placing n queens on an n-by-n chessboard so that no two queens attack. A variation of this problem is the toroidal n-queens problem, where lines of attack between queens are considered to 'wrap' as if the board were a torus. The toroidal n-queens problem has strictly fewer solutions than the (regular) n-queens problem.

To our knowledge there is no known bound for CSP search algorithms for this problem, beyond the trivial bound of n^n. In fact, there is some evidence that the number of solutions to the toroidal problem (and hence to the regular problem) is larger than exponential [Rivin and Vardi, 1989].

The toroidal n-queens problem can be formulated as a simple set of equations. There are n^2 variables x_j. There are also $4n$ equations, one for each column, row, diagonal (top to bottom) and anti-diagonal. So we have $M = 4n$, $N = n^2$. Applying the above results, our algorithm can determine the number of solutions in time $O(n^2 8^n)$.

For the regular n-queens problem, again $N = n^2$, but this time there are $2(2n - 1)$ inequalities for the diagonals. This gives us $M = 6n - 2$, and a running time of $O(n^2 32^n)$.

However, if we do the multiplication in row-major order, it turns out that only $3n$ variables will need to be active at once. This can be seen by noting that the top row involves $3n$ constraints. Moving to the next row adds a new row constraint and removes the old row constraint. The far left square of the new row eliminates an antidiagonal constraint (all the squares involved in that constraint have been eliminated), and adds a new diagonal constraint, while the far right square eliminates a diagonal constraint and adds a new antidiagonal constraint. This gives us a much better bound of $O(n^2 8^n)$. Our preliminary experiments with this algorithm have been promising.

8 Conclusions

We have shown a surprising connection between solving search problems and multiplying certain polynomials. The performance is very simple to analyze, and we can show new bounds for several constraint satisfaction problems.

Our approach also holds our the possibility of applying sophisticated algebraic techniques to constraint satisfaction problems. For example, it is possible that the coefficient of $\overline{Y}$ in Φ can be estimated, thus providing an approximate count of the number of solutions. Also, complex mathematical results like those in [Anshel and Goldfeld, 1988] can be used to provide an upper bound on the number of solutions by studying the generating function directly, using the theory of modular forms.

Our algorithm also shares some surprising properties with Seidel's method [Seidel, 1981]. Like Seidel, we produce a method which is easy to analyze, and which gives a bound that is exponential in a certain parameter of the problem. Seidel's parameter f is a property of the topology of the constraint graph, which can be shown to be $O(\sqrt{n})$ for planar graphs. Our parameter M is a property of the way the problem gets encoded as an in-

[3] An encoding that does not have this property corresponds to a CSP whose unsatisfiability can be easily detected via arc consistency [Mackworth, 1977].

teger linear programming problem, which can be shown to be linear for variants of the n-queens problem.

Seidel faces the problem of making f small, which in general requires exponential time. Similarly, we face the problem of making M small, which requires solving an $\mathcal{NP}$-hard problem involving graph cliques. The similarities between our algorithm and Seidel's, as well as Lamping's observation described in section 6.2, suggest that our algorithm may be viewed as dynamic programming. We are currently exploring this possibility [Rivin and Zabih, 1989].

8.1 Acknowledgements

We are indebted to John Lamping and Ilan Vardi for useful suggestions. Support for this research was provided by DARPA under contract number N00039-84-C-0211. Ramin Zabih is supported by a fellowship from the Fannie and John Hertz foundation.

References

[Aho *et al.*, 1974] Alfred Aho, John Hopcroft, and Jeffrey Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[Anshel and Goldfeld, 1988] Michael Anshel and Dorian Goldfeld. Applications of the Hardy-Ramanujan partition theory to linear Diophantine equations. Unpublished pre-print, 1988.

[Dixon and de Kleer, 1988] Mike Dixon and Johan de Kleer. Massively parallel assumption-based truth maintenance. In *Proceedings of AAAI-88, St. Paul, MN*, pages 199–204. American Association for Artificial Intelligence, Morgan Kaufmann, August 1988.

[Euler, 1748] L. Euler. *Introductio Analysin Infinistorum*, vol. 1. M.–M Bousquet, Lausanne 1748; German translation by H. Maser: *Einleitung in die Analysis des Unendlichen Erster Teil,* Springer, Berlin 1983.

[Garey and Johnson, 1979] Michael Garey and David Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.

[Mackworth, 1977] Alan Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[Mackworth, 1987] Alan Mackworth. Constraint satisfaction. In Stuart Shapiro, editor, *Encylopedia of Artificial Intelligence*. Wiley-Interscience, 1987.

[Rivin and Vardi, 1989] Igor Rivin and Ilan Vardi. The n-queens problem. Forthcoming.

[Rivin and Zabih, 1989] Igor Rivin and Ramin Zabih. An algebraic approach to constraint satisfaction problems. Forthcoming extended version.

[Schrijver, 1986] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, 1986.

[Seidel, 1981] Raimund Seidel. A new method for solving constraint satisfaction problems. In *Proceedings of IJCAI-81, Vancouver, BC*, pages 338–342, August 1981.