Counting CSP Solutions Using Generalized XOR Constraints

Carla P. Gomes and **Willem-Jan van Hoeve** and **Ashish Sabharwal** and **Bart Selman**
Department of Computer Science
Cornell University, Ithaca NY 14853-7501, U.S.A.
{gomes,vanhoeve,sabhar,selman}@cs.cornell.edu

Abstract

We present a general framework for determining the number of solutions of constraint satisfaction problems (CSPs) with a high precision. Our first strategy uses additional binary variables for the CSP, and applies an XOR or parity constraint based method introduced previously for Boolean satisfiability (SAT) problems. In the CSP framework, in addition to the naive individual filtering of XOR constraints used in SAT, we are able to apply a global domain filtering algorithm by viewing these constraints as a collection of linear equalities over the field of two elements. Our most promising strategy extends this approach further to larger domains, and applies the so-called generalized XOR constraints directly to CSP variables. This allows us to reap the benefits of the compact and structured representation that CSPs offer. We demonstrate the effectiveness of our counting framework through experimental comparisons with the solution enumeration approach (which, we believe, is the current best generic solution counting method for CSPs), and with solution counting in the context of SAT and integer programming.

Introduction

In recent years there has been a significant interest in counting the number of solutions to combinatorial problems. Computationally speaking, counting the number of solutions is, both in theory and practice, considerably harder than finding a single solution or proving infeasibility. On the other hand, the ability to count solutions efficiently would open up a wide range of new applications, in particular those involving probabilistic reasoning. These promising new application areas have lead to a continued interest in the search for efficient solutions to this challenging problem.

Most of the recent work on counting the number of solutions has focused on propositional formulations of the problem domains under consideration, and builds upon the dramatic advances in Boolean satisfiability (SAT) research. While this is a promising direction, by propositionalizing domains one potentially loses a significant amount of explicit domain structure. It is therefore natural to consider more general counting techniques that apply directly to more structured constraint satisfaction problems (CSPs).

When looking at the state of the art in counting solutions for CSPs, brute-force enumeration appears the best available *generic* technique. More specialized methods have been developed for binary CSPs (Angelsmark & Jonsson 2003; Kask, Dechter, & Gogate 2004), which count blocks of solutions at a time in a manner similar to some of the complete SAT model counters, such as `Relsat` (Bayardo Jr. & Pehoushek 2000) and `Cachet` (Sang *et al.* 2004).

In this paper, we propose a new generic counting technique for CSPs, building upon on the so-called XOR streamlining technique proposed recently for counting the number of solutions of SAT instances (Gomes, Sabharwal, & Selman 2006). In this approach, a number of random XOR or parity constraints are added to the given SAT instance, and then a state-of-the-art SAT solver is used to check whether the augmented formula is still satisfiable. Depending on the outcome of this process for a few iterations, one obtains bounds on the solution count of the original SAT instance, which hold with provable high-confidence correctness guarantees. Intuitively, this method exploits the fact that the more solutions the original problem instance has, the more additional XOR constraints can be added to it before the problem becomes unsatisfiable.

To extend this framework to CSP formulations, we introduce two new approaches. In our first approach, we create a binary variable for each CSP variable-value pair, and add random XORs on these variables. We study two variations of this strategy: individual XOR filtering, using the 'watched literals' idea from SAT solvers, and global filtering. Global filtering is performed using a Gaussian elimination like method, which is particularly well-suited to handle XOR constraints, and guarantees complete global filtering.

In our second approach, we use a generalized version of XOR constraints directly on the CSP variables. We define generalized XORs as the analogue of the standard Boolean XORs for variables with arbitrary finite domain size. Let d denote the largest domain size. A generalized XOR constraint on a set of variables is satisfied when the sum modulo d of the values of the variables in the constraint equals a given "right hand side" value between 0 and $d-1$. With $d = 2$ this corresponds to the standard Boolean XOR constraint. To handle such constraints effectively, we employ a complete domain filtering algorithm based on dynamic programming, extending a technique proposed by Trick (2003).

We prove that both these approaches can be used to provide bounds on the solution counts of CSPs with similar guarantees as standard XORs have been shown to provide for SAT instances (Gomes, Sabharwal, & Selman 2006).

Our experiments reveal that our XOR framework for CSPs works quite well. We consider various hard combinatorial problems, focusing on those that have natural representations as general CSPs or integer programs, though not necessarily a natural propositional SAT formulation. For example, `games120` is a challenging counting problem based on graph coloring. Using an integer programming formulation translated into a pseudo-Boolean formula, Morgado *et al.* (2006) recently proposed a counter, which on this problem found 1.1×10^6 solutions in half an hour. They further considered a solution count preserving translation to a Boolean formula, originally due to Bailleux, Boufkhad, & Roussel (2006), on which the exact SAT model counter `Relsat` (Bayardo Jr. & Pehoushek 2000) found 1.4×10^6 solutions in half an hour. In contrast, our method using generalized XOR constraints improves this lower bound to 4.5×10^{42} (with 99% correctness confidence) in under a minute.

We also experimented with Spatially Balanced Latin Square problems. Here we are able to count solutions for squares of order 17 (our method found 1058 solutions in 14 minutes, again with 99% correctness confidence). A SAT-based approach to this problem, which involves pairwise distance computations, is arguably infeasible because the SAT encoding is too large; an integer programming based approach also does not find any solutions at all. In fact, even the CSP solver we considered did not find a single solution for this order of spatially balanced Latin squares on the original formulation, but the problem became feasible after adding XOR constraints. This shows that the XOR approach can even work as a domain-independent streamliner for some problems.

These results highlight the effectiveness of our approach, especially in structured domains naturally suited for constraint programming. The good performance of our filtering algorithms based on Gaussian elimination and dynamic programming suggest that there is potential for incorporating similar propagation techniques into SAT-based XOR counting approaches as well.

Preliminaries

Let x be a variable. The *domain* of x, denoted by $D(x)$, is a finite set of elements (also called domain values) that can be assigned to x. A *constraint* C on a finite set of variables $V = \{x_1, x_2, \ldots, x_n\}$ is defined as a subset of the Cartesian product of the domains of the variables in V, i.e. $C \subseteq D(x_1) \times \cdots \times D(x_n)$. A *solution* to C is a variable assignment $\sigma = \{x_1 = d_1, x_2 = d_2, \ldots, x_n = d_n\}$ with $d_i \in D(x_i)$ for $i = 1, \ldots, n$, such that $(d_1, d_2, \ldots, d_n) \in c$. We also say that a solution *satisfies* C. A *constraint satisfaction problem (CSP)* $\mathscr{P}$ is defined as the pair $\mathscr{P} = \langle V, C \rangle$, where V is a finite set of variables and C is a finite set of constraints defined on (subsets of) V. A solution to a CSP is a variable assignment $\sigma = \{x = d \mid d \in D(x), x \in V\}$, such that all constraints in C are satisfied.

An XOR *constraint* C over binary variables V is the logical "xor" or parity of a subset of $V \cup \{1\}$; a variable assignment σ satisfies C if it satisfies an *odd number* of elements in C. The value 1 allows us to express even parity. For instance, $C = \{x, y, z, 1\}$ represents the XOR constraint $x \oplus y \oplus z \oplus 1$, which is TRUE when an even number of x, y, z are TRUE. Note that it suffices to use only positive literals. E.g., $\neg x \oplus y \oplus \neg z$ and $\neg x \oplus y$ are equivalent to $C = \{x, y, z\}$ and $C = \{x, y, 1\}$, respectively.

When V has non-binary variables, we define a *generalized* XOR *constraint* C over V with respect to the largest domain size d in V as a pair (U, r), where $U \subseteq V$ and $r \in \{0, 1, \ldots, d-1\}$, with the following semantics: a variable assignment σ for V satisfies C iff $\sum_{x \in U} \sigma(x) = r \pmod{d}$. In words, the sum of the values of the variables in U (the "left hand side") must equal r (the "right hand side"), modulo d. For instance, for $d = 5$, $C = (\{x, y, z\}, r)$ represents the generalized XOR constraint $x + y + z = r \pmod 5$. A key property of such constraints that we will be using is that if r is chosen uniformly at random from $\{0, 1, \ldots, d-1\}$, *any* variable assignment σ satisfies the generalized XOR constraint with probability $1/d$.

We will use probabilistic arguments to compute the correctness confidence for our approach. We will be interested in the random variables that are the sum of indicator random variables: $Y = \sum_\sigma Y_\sigma$. Linearity of expectation says that $\mathbb{E}[Y] = \sum_\sigma \mathbb{E}[Y_\sigma]$. When various Y_σ are *pairwise independent*, i.e., knowing Y_{σ_2} tells us nothing about Y_{σ_1}, even variance behaves linearly: $\mathrm{Var}[Y] = \sum_\sigma \mathrm{Var}[Y_\sigma]$.

Counting Using Binary XORs

Our first counting approach uses XOR constraints based on a binary representation of the CSP. In this approach, we exploit a natural correspondence between CSP variable-value pairs and binary variables. For each variable x_i and value $v_j \in D(x_i)$ in a CSP $\mathscr{P}$, we create a new binary variable $y_{i,j}$ which is 1 iff $x_i = v_j$. In other words, we add to $\mathscr{P}$ the constraints: $y_{i,j} = (x_i == v_j)$ to obtain a new problem $\mathscr{P}'$.[1] This does not change the semantics of the CSP. In particular, $\mathscr{P}$ and $\mathscr{P}'$ have the same number of solutions. However, we are now ready to use the counting framework of Gomes, Sabharwal, & Selman (2006) essentially without any changes, as discussed next.

Let m be the number of new binary variables we have created. The counting approach will be characterized by three parameters that we fix in advance: $t \geq 1$ is the number of iterations, $s \geq 1$ is the number of XOR constraints to add, and $\alpha > 0$ is a *slack factor*. We will discuss later how to choose s, t, and α based on the desired correctness confidence and bound quality. The counting procedure is to do the following t times:

1. Add to $\mathscr{P}'$ s randomly chosen XOR constraints on the m binary variables. Call this new problem $\mathscr{P}''$.

2. Test whether $\mathscr{P}''$ is satisfiable.

If all t of the problems $\mathscr{P}''$ turn out to be satisfiable, report $2^{s-\alpha}$ as a lower bound on the solution count of $\mathscr{P}$.[2]

[1]In the actual implementation, we create $y_{i,j}$ and add its constraint only if it appears in one of the XOR constraints added later.

[2]When sufficiently long XORs are used, the method can be gen-

The probabilistic correctness guarantee associated with this algorithm, as given by the following theorem, is a direct consequence of the original XOR framework for SAT problems. We note that this guarantee holds no matter how long or short the XORs are. In practice, we choose XORs of length around 10.

Theorem 1 (Binary Approach). *When all t problems $\mathscr{P}''$ are satisfiable, the lower bound of $2^{s-\alpha}$ on the solution count of $\mathscr{P}$ is correct with probability at least $1-2^{-\alpha t}$.*

This shows that the probability of error goes down *exponentially* as the number of trials, t, or the slack factor, α, increase, making the algorithm quite robust for providing lower bounds on the model count. For instance, to achieve 99% correctness confidence, it is sufficient to have $\alpha t = 7$. Accordingly, our experiments typically used $(\alpha, t) = (1, 7)$. The value s of the number of XORs to add is chosen in practice based on a relatively fast binary search to see how far up can s be pushed while still having $\mathscr{P}''$ be satisfiable. (As s grows, $\mathscr{P}''$ starts to be unsatisfiable more and more often.)

We note that when the XOR constraints are chosen to be long enough, this framework also provides an upper bound on the solution count. This result capitalizes on the fact that if $\mathscr{P}$ has m binary variables $y_{i,j}$, randomly chosen XORs of average length $m/2$ over these variables act pairwise-independently on various assignments to these variables. We refer to Gomes, Sabharwal, & Selman (2006) for details.

Theorem 2 (Binary Approach, Upper Bound). *Let $\mathscr{P}$ have m binary variables $y_{i,j}$ as above. When random XOR constraints are chosen to be of average length $m/2$ and all t problems $\mathscr{P}''$ are* un*satisfiable, the upper bound of $2^{s+\alpha}$ on the solution count of $\mathscr{P}$ is correct with probability $\geq 1-2^{-\alpha t}$.*

Individual vs. Global Filtering

The framework of constraint programming offers various algorithmic possibilities for dealing with the new XOR constraints that we have added. We next consider two approaches for filtering the domains of the binary variables based on these constraints. The first approach – individual filtering – treats each XOR constraint individually, while the second – global filtering – treats them all together as a set of linear equations in the field of two elements.

Individual filtering for XOR constraints is straightforward: a domain value can be filtered for an XOR constraint C iff exactly one variable x_i in C is not yet bound to a value. In this case, x_i must equal the XOR or parity of the right hand side of C and the values of the other (bound) variables in C. To increase efficiency, we borrow the now-standard *watched literals* technique from SAT solvers. Namely, we maintain a *watch* on two free variables of C. We process C only when one of its watched variables is fixed to a value, in which case we attempt to find a new free variable to watch, failing which we fix the remaining watched variable to an appropriate value so as to satisfy C.

Global filtering for XOR constraints is based on the process of Gaussian elimination. Each XOR constraint can be viewed as a linear equality over the field $\mathbb{F}_2$ of two elements, 0 and 1. In this case, Gaussian elimination can be performed very efficiently, because no division is necessary (all coefficients are 1), and subtraction and addition are equivalent operations. For a system of k XOR constraints on n variables, we need to perform $O(k^2)$ row operations for the diagonalization, while each row operation takes $O(m)$ time where m is the number of elements in the row (we use a set representation with the union-find data structure). Hence, the total time complexity is $O(k^2m)$, with $m \leq n$. During the diagonalization, we check whether a row becomes empty, and whether it contains only one free variable. In the first case, we report that the system of XOR constraints is not satisfiable. In the latter case, we assign the right hand side value to the one free variable (here we actually filter the domain), update the matrix, and remove the inactive row.

When we perform filtering based on Gaussian elimination, we achieve complete filtering on the system of XOR constraints. To prove this, we show that after applying the algorithm, each remaining domain value belongs to a solution to the system, or we detect that no solution exists. First, infeasibility of the system is detected trivially by the algorithm when a row becomes empty. If the system is feasible, we argue that for every free variable x, both $x = 0$ and $x = 1$ can be extended to some solution. Observe that after applying the algorithm, each active row in the system contains at least two free variables, out of which one is part of a diagonal submatrix of the system. Choose any free variable x and assign it any value in $\{0, 1\}$. It is easy to check that if x is not part of the diagonal submatrix of the system, we can still match all right hand sides by choosing appropriate values for the variables in the diagonal submatrix. When x is an element of the diagonal submatrix, the row to which x belongs can be satisfied by the other free variable(s) in that row, while the remaining rows still have their own diagonal submatrix variables free to help match the right hand side.

Counting Using Generalized XORs

We now describe an extension of the XOR counting framework that applies more naturally to CSP variables. Instead of creating binary variables representing CSP variable-value pairs, we use generalized XORs directly on the CSP variables. Interestingly, this gives correctness guarantees for lower bounds similar to the binary approach.

Let $\mathscr{P} = \langle V, C \rangle$ be a CSP, and let d be the maximum of the domain sizes of variables in V. This counting approach will also be characterized by three parameters that we fix in advance: $t \geq 1$ is the number of iterations, $s \geq 1$ is the number of generalized XOR constraints to add, and $\alpha > 0$ is a *slack factor*. The counting procedure is to do the following t times:

1. Add to $\mathscr{P}$ s randomly chosen generalized mod-d XOR constraints on the variables V. Call this new problem $\mathscr{P}'$.
2. Test whether $\mathscr{P}'$ is satisfiable.

If all t of the problems $\mathscr{P}'$ turn out to be satisfiable, report $d^{s-\alpha}$ as a lower bound on the solution count of $\mathscr{P}$.[3]

eralized to the case where some runs are satisfiable and some are unsatisfiable (Gomes, Sabharwal, & Selman 2006).

[3]As before, this method can be generalized to the case when

The probabilistic correctness guarantee associated with this algorithm is given by the following theorem. We again note that this guarantee holds no matter how long or short the generalized XORs are. In practice, we choose generalized XORs of length around 6.

Theorem 3 (Generalized Approach). *When all t problems $\mathscr{P}'$ are satisfiable, the lower bound of $d^{s-\alpha}$ on the solution count of $\mathscr{P}$ is correct with probability at least $1-d^{-\alpha t}$.*

Proof. Let d^{s^*} be the true solution count of $\mathscr{P}$ ($s^* > 0$ is a real number). Assume for contradiction sake that we report an incorrect lower bound for $\mathscr{P}$, i.e., $d^{s-\alpha} > d^{s^*}$, or, equivalently, $s^* - s < -\alpha$. We will show that for such a choice of s and α, the probability of encountering all t $\mathscr{P}'$ problems to be satisfiable is at most $d^{-\alpha t}$, proving the desired correctness confidence result.

Let S be the set of solutions of $\mathscr{P}$. For each $\sigma \in S$, let $Y_\sigma = \sigma(\mathscr{P}')$ be a 0-1 random variable indicating whether σ is a solution of $\mathscr{P}'$ or not. Since σ already satisfies $\mathscr{P}$, the expected value of Y_σ is the probability that σ satisfies all of the s generalized XOR constraints in $\mathscr{P}'$. Note that for each generalized XOR constraint, σ induces a fixed value in $\{0,1,\ldots,d-1\}$ for the left hand side of the constraint. Since the right hand side of each generalized XOR constraint is chosen uniformly and independently at random from the set $\{0,1,\ldots,d-1\}$, the probability that σ satisfies all of them is d^{-s}, implying $\mathbb{E}[Y_\sigma] = d^{-s}$.

Let $Y = \sum_\sigma Y_\sigma$. The random variable Y equals the number of solutions of $\mathscr{P}'$, and we have $\mathbb{E}[Y] = \mathbb{E}[\sum_\sigma Y_\sigma] = \sum_\sigma \mathbb{E}[Y_\sigma] = \sum_\sigma d^{-s} = d^{s^*-s}$. Using Markov's inequality, it follows that

$$\begin{array}{rclll} \Pr[\mathscr{P}'\text{is satisfiable}] & = & \Pr[\text{num\_solutions}(\mathscr{P}') \geq 1] & & \\ = \Pr[Y \geq 1] & \leq & \mathbb{E}[Y]/1 & & \text{by Markov's ineq.} \\ = \quad d^{s^*-s} & \leq & d^{-\alpha} & & \text{by assumption.} \end{array}$$

It follows by the probabilistic independence of the t runs that if we were reporting an incorrect lower bound, the probability of encountering all t problems $\mathscr{P}''$ to be satisfiable would be less than $(d^{-\alpha})^t = d^{-\alpha t}$. This gives the desired bound on the error probability. □

Note that Theorems 1, 2, and 3 provide correctness guarantees for *every problem instance* over several runs of our (randomized) algorithms on that instance. This is in contrast with other conceivable guarantees, such as the algorithms succeeding on most instances but always failing on some.

This theoretical result, just like Theorem 1, may appear to be somewhat counter-intuitive. E.g., in the extreme case, one can imagine that the following happens: we add too many random XORs ($s \gg s^*$) but the resulting problem $\mathscr{P}'$ still always turns out to be satisfiable. In this case, we will incorrectly report a lower bound that is too high and, moreover, with the same high confidence, $1-d^{-\alpha t}$. The way to understand this apparent inconsistency is that this worst-case event, although technically feasible, is extremely unlikely to happen as a random event, and this possibility is already taken into account in the correctness confidence. In fact, the analysis used in the proof of the theorem bounds the likelihood of this and all other "bad" events with the exponentially small number $2^{-\alpha t}$, which determines our confidence.

some runs are satisfiable and some are not.

Finally, we remark that it is not clear whether randomly chosen generalized XOR constraints act pairwise independently on various variable assignments, which is a key ingredient needed for a guaranteed upper bound using large enough XORs (an analog of Theorem 2). Our experimental results will focus on relatively short generalized XORs and will provide guaranteed lower bounds.

Filtering Generalized XORs

As for the XOR constraints on binary variables, there are two approaches to filtering generalized XOR constraints: either treat each constraint individually, or treat them all together.

Filtering an individual generalized XOR constraint can be done using a technique previously introduced to filter *knapsack* constraints (Trick 2003). In this approach, we exploit a dynamic-programming graph that represents all possible combinations of variable assignments whose sum (modulo d) equals the right-hand side of the XOR constraint. More formally, for a generalized XOR constraint on the variables $x_1, x_2, \ldots, x_n$ with maximum domain size d and right hand size r, we define a graph G with 'coordinate' vertex set $V = \{(i,j) \mid i \in \{0,1,\ldots,n\}, j \in \{0,1,\ldots,d-1\}\}$ and directed edge set E, defined as

$$\{(i-1,j),(i,k)\} \in E \text{ if } (j+v) \bmod d = k, \text{ and } v \in D(x_i),$$

for $i = 1,\ldots,n$. In this graph, every directed path from vertex $(0,0)$ to (n,r) corresponds to a solution to the XOR constraint. In particular, if an edge $\{(i-1,j),(i,k)\}$ is on such a path, the corresponding domain value $((k-j) \bmod d) \in D(x_i)$ is part of the corresponding solution. To filter the domains, we remove all edges not on such paths, together with the corresponding domain values. This is done as follows. First, we remove all edges that are not on any path originating from vertex $(0,0)$. Second, we reverse the edges and remove all edges that are not on any path originating from vertex (n,r). Both steps take linear time in the size of the graph. After these two steps, all remaining edges are on a path representing a solution, or no such path exists. Then, for each variable x_i, we remove domain value $v \in D(x_i)$ if there is no edge $\{(i-1,j),(i,k)\}$ such that $(j+v) \bmod d = k$ in G. Doing so, all remaining domain values are guaranteed to be part of a solution to the constraint. Hence we achieve complete filtering in $O(nd^2)$ time.[4]

We have also implemented an alternative filtering algorithm, that simply performs a brute force enumeration of all possible value combinations of the variables. If a domain value is not supported by any solution, it is removed from the corresponding domain. Although its running time is $O(d^n)$, it can be more efficient than the above algorithm in case there are only a few (free) variables in the XOR constraint. In practice, we apply the brute-force algorithm when there are at most three free variables in the XOR constraint.

[4]The original algorithm for *knapsack* constraints runs in pseudo-polynomial time, depending on the right hand side.

Furthermore, the effectiveness of the filtering algorithm is application-dependent. We observed that in several cases few or no domain values are removed when there are more than 4 or 5 free variables left. Therefore we apply a threshold of free variables (typically 4 or 5) above which we do not apply the filtering algorithm.

As an alternative to the individual filtering of the generalized XOR constraints, one may group them together and reason upon them as a whole. A natural extension of the binary case is to apply Gaussian elimination here as well. Unfortunately, Gaussian elimination cannot be generalized in a straightforward manner to a set of linear equations modulo d for arbitrary d, because division may be ambiguous. We can overcome this problem by defining d to be the smallest prime power p^a that is at least as large as the maximum domain size of the variables, and safely apply Gaussian elimination over the larger field $\mathbb{F}_{p^a}$. However, the efficiency of the implementation of prime power fields is likely to be an issue and, more importantly, Gaussian elimination over $\mathbb{F}_{p^a}$ will not achieve complete filtering of the domains. For these practical reasons we have not implemented the corresponding filtering algorithm.

Experimental Results

We conducted experiments on a cluster of 3.8 GHz Intel Xeon machines with 2 GB memory per node running Linux 2.6.9-22.ELsmp. The constraint programming solver used to implement our filtering algorithms and evaluate the counting framework was `ILOG Solver 6.3` (ILOG, SA 2006). We also used `ILOG Solver` for exact solution counting based on enumerating all solutions systematically, which we believe is currently the best available generic solution counting method for CSPs. We call this method `pure CSP`. For integer programming domains, we compare with two of the techniques recently proposed by Morgado *et al.* (2006) based on a solution count preserving translation first to pseudo-Boolean formulas and subsequently to SAT. These comparisons are based on the results reported by these authors. Their first counting method is termed `pb-counter`. For the second method, the problem is translated into a SAT instance whose solutions are counted using `Relsat` (Bayardo Jr. & Pehoushek 2000). The solution count preserving translation used to obtain the SAT instances is due to Bailleux, Boufkhad, & Roussel (2006). We refer to this technique as `Relsat+BBR`.

In all our experiments with the binary CSP approach, α was set to $7/t$ and t to 7, giving a correctness confidence of $1-2^{-7} \approx 99\%$ (cf. Theorem 1). For all experiments with the generalized CSP approach with maximum domain size d, α was set to $(\log_d 100)/t$ and t to 7, again giving a correctness confidence of $1-d^{-\log_d 100} \approx 99\%$ (cf. Theorem 3).

First, we evaluate the performance of our different approaches: *i)* binary XOR constraints filtered individually, *ii)* binary XOR constraints filtered globally, and *iii)* generalized XOR constraints. For this evaluation we count the number of solutions to the n-`queens` problem, for which the exact number of solutions is known up to order $n = 25$.[5] We compare the approaches with each other, and with a counter based on pure enumeration using the CSP solver, with a one hour time limit. For each method, we report the number of solutions found and the corresponding running time in Table 1. For the XOR approach, the running time is the sum of all seven runs.

[5]See http://www.research.att.com/~njas/sequences/A000170.

Not surprisingly, the count obtained by the pure CSP approach is limited by the number of search nodes it can traverse in one hour, yielding a count of roughly 10^6 solutions in all cases. In contrast, our XOR-based techniques can count up to roughly 10^{15} solutions within a few minutes for order 30 (note that the exact count for this problem is unknown). The results further indicate that filtering the system of binary XOR constraints globally can be beneficial compared to individual filtering for the same set of constraints. Overall, however, for all problems in this domain, generalized XOR constraints outperform the binary approach. The behavior on n-`queens` depicts a general trend that we observed: the generalized XORs approach often works quite well in practice, although there are cases where the binary approach, with its finer granularity, outperforms generalized XORs.

We next compare our approach with the two techniques for integer programming domains mentioned earlier, `pb-solver` and `Relsat+BBR`. We use a set of graph coloring problems from the DIMACS benchmark set considered by Morgado *et al.* (2006). The results are reported in Table 2 (problems `games120` to `2_Insertions_3`).[6]. For these problems, we also report the number of solutions obtained with pure CSP in one hour. Quite surprisingly a pure CSP approach can sometimes outperform `pb-counter` and `Relsat+BBR` (for `games-120`), although it uses a longer running time. Our generalized XOR approach performs very well on these problems. In particular, for `games120`, we improve (with 99% correctness confidence) the previous lower bound of 1.4×10^6 in 30 minutes to 4.5×10^{42} solutions in under a minute.

Finally, we consider a new problem domain which, to the best of our knowledge, has not been counted before: spatially balanced Latin squares. A Latin square is an n by n matrix in which each cell is assigned a number from 1 up to n, such that each row and each column contains exactly one of the numbers 1 up to n. A spatially balanced Latin square has the additional requirement that for each pair of numbers in $\{1,2,\ldots,n\}$, the sum of their distance in each row is equal to a given constant. The largest order known for which such squares exist is 35, found by Smith, Gomes, & Fernández (2005) using a streamlined local search approach. For SAT or integer programming based methods, solving these problems is out of reach even for very small orders. Using CSP, squares have been found up to order 14 and 18 (Gomes & Sellmann 2004). Here we count spatially balanced Latin squares for order 14, 15, and 17, using generalized XOR constraints. In fact, we count *streamlined*

[6]We have scaled down the running times of `pb-solver` and `Relsat+BBR` by a factor of 2 since they were reportedly run on a 1.9 GHz machine. We also note that the counter `LattE` (De Loera *et al.* 2004) for integer programs is reportedly not able to count any of these instances.

		pure CSP		binary XORs (individual)		binary XORs (global)		generalized XORs (individual)	
order	true count	count	time	count	time	count	time	count	time
15	2.3×10^6	2.3×10^6	10 min	$\geq 6.6 \times 10^4$	41 s	$\geq 2.6 \times 10^5$	66 s	$\geq 3.9 \times 10^5$	3 s
20	3.9×10^{10}	$\geq 3.5 \times 10^6$	1 hr	$\geq 1.1 \times 10^6$	24 s	$\geq 2.1 \times 10^6$	17 s	$\geq 6.6 \times 10^8$	13 s
25	2.2×10^{15}	$\geq 2.2 \times 10^6$	1 hr	$\geq 3.4 \times 10^7$	125 s	$\geq 3.4 \times 10^7$	101 s	$\geq 2.0 \times 10^{12}$	60 s
30		$\geq 4.1 \times 10^6$	1 hr	$\geq 5.4 \times 10^8$	155 s	$\geq 5.4 \times 10^8$	155 s	$\geq 9.2 \times 10^{15}$	196 s

Table 1: *Computational results on n-Queens problems, comparing our different approaches (99% correctness confidence).*

	pb-counter		Relsat+BBR		pure CSP		CSP+XORs	
games120	$\geq 1.1 \times 10^6$	30 min	$\geq 1.4 \times 10^6$	30 min	$\geq 4.3 \times 10^8$	1 hr	$\geq \mathbf{4.5 \times 10^{42}}$	**1 min**
myciel5	$\geq 1.1 \times 10^7$	30 min	$\geq 3.6 \times 10^{11}$	30 min	$\geq 9.5 \times 10^8$	1 hr	$\geq \mathbf{4.1 \times 10^{17}}$	**12 min**
mug100_1	$\geq 2.4 \times 10^7$	30 min	$\geq 2.7 \times 10^{23}$	30 min	$\geq 7.2 \times 10^8$	1 hr	$\geq \mathbf{1.0 \times 10^{28}}$	**1 min**
2_Insertions_3	$\geq 9.0 \times 10^6$	30 min	$\geq 4.6 \times 10^8$	30 min	$\geq 1.2 \times 10^9$	1 hr	$\geq \mathbf{2.3 \times 10^{12}}$	**1 min**
					pure CSP		CSP+XORs	
sbls14					≥ 273	1 hr	$\geq$ **591**	**5 min**
sbls15					≥ 112	1 hr	$\geq$ **1,748**	**8 min**
sbls17					—	1 hr	$\geq$ **1,058**	**14 min**

Table 2: *Computational results on graph coloring problems and spatially balanced Latin square problems. The results for the* XOR *approach are with 99% correctness confidence.*

spatially balanced Latin squares, by using the streamlined model of Smith, Gomes, & Fernández (2005). The results are reported in Table 2 (`sbls 14–17`), in which we also report the solution count of pure CSP. We see, for example, that the pure CSP solver counts 112 solutions in one hour for `sbls15`, while our generalized XOR approach counts 1748 solutions in 8 minutes (again, with 99% correctness confidence). Moreover, `sbls17` cannot be solved at all by the pure CSP solver in one hour, while we count 1058 solutions in 14 minutes. This reconfirms an interesting phenomenon observed earlier by Gomes, Sabharwal, & Selman (2006): for computationally challenging problems, randomly generated XOR constraints can sometimes prove to be effective domain-independent streamliners.

Conclusion

We introduced a new generic solution counting technique for constraint satisfaction problems. This approach builds upon a method recently proposed for Boolean satisfiability problems, and combines it with the structured representation of CSPs to quickly provide lower bounds on solution counts with strong correctness guarantees. We considered both "regular" XOR constraints on an equivalent binary representation of CSPs as well as generalized XOR constraints directly on the CSP variables. For both cases, we developed efficient complete domain filtering algorithms. Our experimental evaluation on a set of challenging combinatorial problems demonstrates the effectiveness of this approach.

Acknowledgments

This work was supported by Intelligent Information Systems Institute (IISI), Cornell University (AFOSR grant FA9550-04-1-0151) and DARPA (REAL grant FA8750-04-2-0216).

References

Angelsmark, O., and Jonsson, P. 2003. Improved algorithms for counting solutions in constraint satisfaction problems. In *8th CP*, volume 2833 of *LNCS*, 81–95.

Bailleux, O.; Boufkhad, Y.; and Roussel, O. 2006. A translation of pseudo Boolean constraints to SAT. *J. on Satisfiability, Boolean Modeling and Computation* 2:191–200.

Bayardo Jr., R. J., and Pehoushek, J. D. 2000. Counting models using connected components. In *17th AAAI*, 157–162.

De Loera, J. A.; Hemmecke, R.; Tauzer, J.; and Yoshida, R. 2004. Effective lattice point counting in rational convex polytopes. *J. Symb. Comput.* 38(4):1273–1302.

Gomes, C. P., and Sellmann, M. 2004. Streamlined constraint reasoning. In *10th CP*, volume 3258 of *LNCS*, 274–289.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2006. Model counting: A new strategy for obtaining good bounds. In *21st AAAI*, 54–61.

ILOG, SA. 2006. ILOG Solver 6.3 reference manual.

Kask, K.; Dechter, R.; and Gogate, V. 2004. Counting-based look-ahead schemes for constraint satisfaction. In *10th CP*, volume 3258 of *LNCS*, 317–331.

Morgado, A.; Matos, P. J.; Manquinho, V. M.; and Marques-Silva, J. P. 2006. Counting models in integer domains. In *9th SAT*, volume 4121 of *LNCS*, 410–423.

Sang, T.; Bacchus, F.; Beame, P.; Kautz, H. A.; and Pitassi, T. 2004. Combining component caching and clause learning for effective model counting. In *7th SAT*. Online Proceedings.

Smith, C.; Gomes, C. P.; and Fernández, C. 2005. Streamlining Local Search for Spatially Balanced Latin Squares. In *IJCAI*, 1539–1540.

Trick, M. 2003. A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints. *Annals of Operations Research* 118:73–84.