An Empirical Study of Probabilistic Arc Consistency as a Variable Ordering Heuristic

Michael C. Horsch and William S. Havens

Intelligent Systems Laboratory,
School of Computing Science,
Simon Fraser University,
Burnaby, B.C., Canada V5A 1S6
{mhorsch, havens}@cs.sfu.ca

Abstract. We report on an empirical evaluation of a new probabilistic heuristic for constructive search in constraint satisfaction problems. The heuristic is based on the estimation of solution probability. We show empirically that this heuristic is more accurate than related heuristics, and reduces the number of consistency checks and backtracks in constructive search by up to several orders of magnitude. Our results also show that the time required to estimate solution probabilities is less than the time required for search using other well-known heuristics as the problem size increases.

1 Introduction

The challenge in a constraint satisfaction problem (CSP) is to determine an assignment of values to variables that satisfies each of a given set of constraints, or otherwise, to conclude correctly that no such assignment exits. In a systematic constructive search process, it is well known that the order in which the variables are assigned can have a significant impact on the cost of the search.

In this paper we present a summary of our experiments using a heuristic for constructive search based on solution probability, i.e., the frequency with which each value is used in all solutions to the CSP. Solution probabilities for a CSP can answer such questions as "Which assignment appears most often in all solutions?" Solution probabilities can therefore provide an ordering to unassigned variables, as well as an ordering for the values for each variable.

We have developed a method for approximating solution probabilities for binary CSPs. We have shown elsewhere [6] that our method is a generalization of the arc consistency algorithm AC-3 [7] for general constraint graphs. We have also shown (op. cit.) that it is a specialization of probabilistic reasoning in Bayesian networks [9]. For this reason, we call our method *probabilistic arc consistency* (pAC).

Our method can compute exact solution probabilities for binary CSPs with tree-structured constraint graphs in time linear in the number of variables. Probabilistic arc consistency can also be used iteratively to provide a good approximation for CSPs of more general structure.

Probabilistic arc consistency is an extension of the techniques expressed in several previous proposals. Work by Dechter and Pearl [2], Meisels et al.[8], and Vernooy and

Havens [13] is equivalent to pAC when the CSP's constraint graph is tree-structured. These proposals differ in their approach to more general CSPs. Peleg [10] developed a probabilistic relaxation method very similar to pAC that was intended to find a satisfying assignment to the variables in a CSP. The algorithm given by Shazeer et al. [12] on estimating posterior probability distributions over constraint problems is essentially the same as the pAC algorithm.

2 Probabilistic Arc Consistency

We briefly present the pAC algorithm which we use to compute or approximate the solution probabilities for every variable in a CSP. The formulation provided below is specific to binary CSPs with finite discrete domains. It is easily extended to deal with k-ary constraints. The claims in this section are stated without proof; proofs can be found in [5].

Let $< V, B >$ be a binary constraint satisfaction problem with variables V, and binary constraints B, such that the constraint graph $G =< V, B >$ is tree-structured. For every $X \in V$, let D_X be the domain of X, i.e., $D_X = \{x_1, \ldots, x_m\}$. For every pair of variables (X, Y) such that there is a binary constraint $C_{XY} \in B$, define the following:

$$C_{XY}(i,j) = \begin{cases} 1 \text{ if } (x_i, y_j) \in C_{XY} \\ 0 \quad \text{otherwise} \end{cases}$$

$$\mathrm{S}^{(k)}_{XY}(i) = \sum_{j=1}^{m} C_{XY}(i,j)\mathrm{M}^{(k)}_{XY}(j)$$

$$\mathrm{M}^{(0)}_{YX}(i) = 1; \quad \mathrm{M}^{(k+1)}_{YX}(i) = \begin{cases} \dfrac{\mathrm{F}^{(k)}_X(i)}{\mathrm{S}^{(k)}_{XY}(i)} \text{ if } \mathrm{S}^{(k)}_{XY}(i) > 0 \\ \\ 0 \qquad\qquad otherwise \end{cases}$$

$$\mathrm{F}^{(k)}_X(i) = \alpha \prod_{\{Y|(X,Y)\in B\}} \mathrm{S}^{(k)}_{XY}(i) \quad \text{where } \alpha \text{ is such that} \sum_{i=1}^{m} \mathrm{F}^{(k)}_X(i) = 1$$

These formulae define the pAC method. Each binary constraint in B is expressed numerically in C_{XY}. The value $\mathrm{F}^{(k)}_X(i)$ represents the kth iterative approximation to the solution probability for $x_i \in D_X$. The value $\mathrm{S}^{(k)}_{XY}(i)$ represents the kth approximation to the expected number of times value $x_i \in D_X$ is used consistently with values of neighbour Y; informally, it can understood as the support of X from Y. The value $\mathrm{M}^{(k)}_{YX}(i)$ represents the kth estimate of the solution probability for $x_i \in D_X$, factoring out all effects of neighbour Y; informally, it can be understood as a "message" sent from X to Y, from which Y computes its support from X. This message is carefully constructed so that Y does not "double count" information it previously sent to X.

Theorem 1. *Let $G =< V, B >$ be a CSP such that the constraint graph for G is tree-structured, with diameter d. For any variable $X \in V$, the relative frequency that value $x_i \in D_X$ is used in all solutions of G is $\mathrm{F}^{(d)}_X(i)$.*

For correctness, the computations described above require arbitrary precision floating point numbers. In our implementation, we use 64-bit floating point numbers, at the risk of a non-trivial loss of precision when the CSPs get very large.

When applied to CSPs with arbitrary topology, the theorem no longer applies. However, in many cases, pAC converges to a reasonable approximation of the solution probabilities. The accuracy depends on the degree to which the given CSP satisfies the assumption of independence of sub-graphs. This result is further explored in the next section. It is not too surprising in light of other recent results, wherein an algorithm for belief propagation in singly-connected Bayesian networks [9], has been shown to produce good results as an approximation method for probabilistic reasoning in networks of arbitrary topology (e.g., [3]). Elsewhere, we have shown that pAC is a special case of belief propagation in singly-connected Bayesian networks, and also a generalization of arc consistency [7].

We have no theoretical guarantees regarding convergence of our method. To limit computation, we use two parameters, ϵ and MaxIter; iteration continues while both of the following conditions are true: $\epsilon < \max_X \sum_i (\mathrm{F}_X^{(k+1)}(i) - \mathrm{F}_X^{(k)}(i))^2$ and $\mathrm{MaxIter} > k$. When the change in solution probability is less than a given ϵ, the process is declared to have converged; when the number of iterations exceeds MaxIter, the process is halted without convergence.

3 Empirical Results

We performed three sets of experiments. The first examined the quality of the approximate probabilities that pAC returns, and showed that pAC performed very well as an approximation on many problems. Second experiment showed that pAC can reduce search compared to other heuristics for counting solutions (e.g., [2, 8, 13, 10]). The third experiment compared the use of pAC as a dynamic variable ordering heuristic against the variable ordering heuristics reported in the literature. We show that pAC reduced the number of consistency checks and backtracks by several orders of magnitude.

3.1 The Accuracy of pAC as an Approximation Method

To judge the accuracy of the approximate solution probabilities, we compared the distributions pAC determines against an exact frequency distribution, for a set of relatively small problems. We used a straightforward correlation computation to compare probability distributions.

We created three sets of 1000 random binary CSPs, with 20 variables with 10 values each. The p_1 values for these problems sets were $0.2, 0.5$ and 1.0 respectively. In our three sets of problems, we create 10 CSPs each for 100 values of p_2 starting at 0.01 and increasing by 0.01. We applied pAC to each problem using $\epsilon = 0.00001$, and MaxIter $= 1001$ iterations.

The approximate solution probabilities were strongly correlated with the exact distributions; the average correlation over the problem sets was determined to be 0.833, 0.771 and 0.788 resp., with standard deviations 0.16, 0.21 and 0.18. Many of the overconstrained problems were correctly identified as such: 75%, 95% and 96%, resp.

3.2 Using pAC as a Static Value Ordering Heuristic

We applied a constructive search algorithm to three sets of 2000 problems, each set having the same properties as in the previous experiment. The approximate solution probabilities were precomputed for each problem, and this static information was used to order values by decreasing solution probability. The variables were ordered lexicographically.

We compared the results of using pAC directly to the methods and results presented by Vernooy and Havens [13]. This also provided a basis for a comparison to Dechter and Pearl's method [2], Meisel et al.'s method [8]. We also implemented Peleg's method (Peleg) [10] for comparison. Using pAC as a static variable ordering heuristic substantially reduced the search cost in terms of the number of backtracks, compared to the other methods. We observed an order of magnitude improvement or higher in 80% of the problems for each of the problem sets.

3.3 Using pAC as a Dynamic Variable and Value Ordering Heuristic

The pAC method can be used as a dynamic variable ordering heuristic. After each assignment of a value to a variable, the pAC algorithm is run again, taking into account this assignment, and new heuristic information is computed. The approximate solution probabilities determined by pAC are used to order the variables as well as the values. While several choices are available, we ordered the (unassigned) variables so that the search proceed with the most certain assignments first.

In this experiment, hundreds of random problems were created, each with 20 to 70 variables, with 10, 20 or 30 values each. Each problem was created to be close to the phase transition region for its problem size. We compared the pAC heuristic to several well studied variable ordering heuristics Brelaz, FF, FFdeg, [1, 4] as well as a random selection criterion. The constructive search procedure was a highly optimized forward-checking back-jumping search algorithm [11], and each of the heuristics was employed using this search procedure.

We observed that the pAC heuristic required one to three orders of magnitude fewer consistency checks and backtracks than any of the other heuristics. This was uniformly observed across all problem sets. Computing the approximate solution probabilities can be very expensive, and for some problem sets, constructive search using pAC requires one to two orders of magnitude more time to find solutions, as compared to the other heuristics used in our experiment. However, as the problem size increased, we observed a decrease in the difference in time required to solve the problems. For the largest of the problem sets solved in our experiment, the median solution time using pAC was actually less than the median time for the other heuristics.

4 Discussion and Future Work

Our experiments have demonstrated that pAC is a valuable heuristic for solving CSPs by search. If the process converges, it usually converges to a good approximation to the relative frequency distribution of values in solutions to the CSP. When used as a dynamic variable ordering heuristic, failure to converge is not a serious issue: at worst, a search procedure will have to backtrack over an assignment made based on unreliable information returned when pAC is halted without convergence. We have found that since pAC identifies over-constrained problems quickly, the cost of the unreliable information is small. The approximations returned by pAC need not be accurate, as inaccurate results can still provide reliable value-ordering information.

We have shown that pAC can reduce search costs by orders of magnitude, depending on the difficulty of the problem, as compared to the heuristics found in the literature. However, when the problem is small enough, the cost of computation for pAC outweighs the benefit of reducing search costs. For larger problems, pAC's runtime is competitive or superior to the other heuristics used for comparison in this experiment.

We are currently investigating methods for reducing the computation costs of pAC. We are studying the effects of reducing the ϵ and MaxIter parameters, as well as the effects of limiting propagation to a subset of the CSP. We are also exploring the use of local propagation techniques in CSPs which have an optimization component, and how it might be used as a heuristic for local search.

References

[1] D. Brelaz. New methods to color the verticies of a graph. *JACM*, 22(4):251–256, 1979.

[2] Rina Dechter and Judea Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–34, 1988.

[3] Brendan J. Frey and David J. C. MacKay. A revolution: Belief propagation in graphs with cycles. In *Advances in Neural Information Processing Systems*, pages 479–485, 1998.

[4] Richard M. Haralick and Gordon L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980.

[5] Michael C. Horsch and William S. Havens. How to count Solutions to CSPs. Technical report, School of Computing Science, Simon Fraser University, 2000.

[6] Michael C. Horsch and William S. Havens. Probabilistic Arc Consistency: A connection between constraint reasoning and probabilistic reasoning. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, 2000. To appear.

[7] Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[8] Amnon Meisels, Solomon Ehal Shimonoy, and Gadi Solotorevsky. Bayes Networks for Estimating the Number of Solutions to a CSP. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1997.

[9] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Reasoning*. Morgan Kaufmann Publishers, Los Altos, 1988.

[10] Shmuel Peleg. A new probabilistic relaxation method. *IEEE Transactions on Pattern Matching and Machine Intelligence*, 2(4):362–369, 1980.

[11] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299, 1993.

[12] Noam M. Shazeer, Michael L. Littman, and Greg A. Keim. Solving crossword puzzles as probabilistic constraint satisfaction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 156–162, 1999.

[13] Matt Vernooy and William S. Havens. An examination of probabilistic value-ordering heuristics. In *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*, 1999.