Generating Random Solutions from a Constraint Satisfaction Problem with Controlled Probability

David Larkin

University of California, Irvine

Abstract. In this paper we address the question of drawing elements from the set of solutions of a constraint satisfaction problem with well-defined probability. A distribution over solutions can be specified with the level of precision appropriate for the information on hand. This has application in functional verification, where CSPs are often used to specify the set of legal test programs that a verifier might try to run. With this technique, the probability that a randomly generated test program has any particular properties can be precisely controlled.

1 Introduction

In this paper, we are concerned with controlling the distribution of generated solutions of a constraint satisfaction problem. Commonly no constraints are placed on how this is done, and when a certain number of solutions are desired they are selected from the set of possibilities in an arbitrary way. However for some applications this is not appropriate. The requirements of functional verification in particular have inspired investigation into the task of making the distribution of generated solutions more uniform or random. This is because the solutions, corresponding to test programs, generally have to represent all programs that could be run. In [4] the bucket-elimination schema [3] was extended to an algorithm to produce uniformly-distributed solutions. An approximation method based on the mini-bucket technique [7] was also outlined. This technique is not applicable, however, when biased-random results are required. In [14], the search space of a constraint satisfaction problem is represented explicitly as a binary decision diagram. A weighted traversal of this diagram is employed to produce a solution in accordance with a user-specified distribution in which all variables must be independent. Approximation methods were not given. The authors cited a need to allow more complex probability distributions to guide the solution generation process, in which some variables would be more likely to assume certain values given certain assignments to other variables. However they did not give a method of doing this. In the present paper, we outline such a method, which has a complexity that depends on the degree of interrelationship among the variables in the probability distribution and in the CSP. Since this can be intractable for large problems, we also give approximation methods. As far as we know, we are the first to provide an algorithm for the problem of generating a random CSP solution according to an arbitrary, well-specified distribution.

Our approach allows a distribution over CSP solutions to be specified with precision appropriate for the level of information on hand. On one extreme, it can be specified exactly with a Bayesian network [12]. On the other extreme, very little information can be given (perhaps the probabilities that a solution would have a few properties) in which case many probability distributions could fit the required profile. So a crucial component of our algorithm is interpreting the information that the user supplies as a complete and well-specified distribution. This is basically the problem of probabilistic logic [11], which requires a characterization of the space of probability distributions which are consistent with some constraints. This problem is NP-hard, even when the constraints only involve two binary variables [10]. Global solution methods typically involve solving an exponentially large linear program [11]. L. van der Gaag [13] gives a solution method with complexity similar to our method's. It involves passing messages among nodes in a junction tree. Our own approach is similar, but is based upon the somewhat simpler approach of variable elimination. Also, it can be easily relaxed into an approximation technique based on the mini-bucket idea [7].

In functional verification, the task of verifying a new hardware design of substantial size is very difficult and can consume enormous resources. Fully formal verification, while offering the best assurance, is typically intractable when dealing with large systems. So the primary means of verification is testing. Both manually written and computer generated programs can be tested [2]. Care has to be taken when writing a random test generator, to ensure that the tests produced are of high quality. The vector space of potential programs is partitioned into a set of programs that will uncover a bug and a set that won't. In the later, more difficult phases of the verification process, the second set is by far the largest. Some prior knowledge of the design and the domain has to be used to improve the probability of choosing a point in the first set. There are two main means of doing this: using probabilistic constraints, and imposing deterministic constraints. They can be seen as means of concentrating and diffusing the probability mass present in the vector space to improve the odds of selecting a good program. A deterministic constraint makes a certain subspace impossible to select from, effectively deleting all probability mass from it. A probabilistic constraint sets the mass present in a region to a specific relative density. For example, a deterministic constraint may require that a STORE instruction follow every ADD instruction, in order to properly exercise the main memory. Any program that violates this should not be generated. A probabilistic constraint may specify that when a random operand for an ADD instruction is being generated, the probability that it will be zero is 25 percent and the remaining probability mass of 75 percent is distributed uniformly over all other candidates. This will increase the chance of finding specific errors that might result when 0 is used in an ADD instruction. Mathematically, if P is a distribution satisfying the probabilistic constraints, and S is the set of vectors which are deterministically allowable, the chance of generating solution X should be $P(X)/\sum_{Y \in S} P(Y)$.

This paper is divided into several parts. Following this introduction, we review basic definitions and preliminaries in Section 2. Then in Section 3 we address

the question of transforming possibly incomplete probabilistic information into a well-defined belief network. We also briefly consider an approximation based on the mini-bucket idea. In Section 4 we show how solutions to a CSP can be generated in accordance with a distribution specified by a belief network, and outline an approximation. Finally in Section 5 we conclude.

2 Preliminaries

Definition 1. *A constraint network is a set of constraints* $C = \{C_1, C_2, ..., C_m\}$ *defined on a set of variables* X. *Each* C_i *is a function mapping assignments to its scope* $S_i \subseteq X$ *to boolean values. An assignment to* X *is allowed by* C *if it is allowed by each* C_i.

Definition 2. *A belief network* $B = (G, P)$ *defined on* X *is a directed acyclic graph* G *and a set of conditional probability tables* P. *The nodes of* G *correspond to the variables in* X. *The set of nodes connected by outgoing arcs to* $X_i \in X$ *is called* X_i*'s parent set* pa_i. $P(X_i|pa_i) \in P$ *gives the probability of* X_i *assuming a specific value given an assignment to its parents. In sum the network represents a probability distribution* $\prod_i P(X_i|pa_i)$.

A distribution over solutions of a constraint network C is a function that maps every solution of C to a real number between 0 and 1. Any distribution P over X can be mapped to a distribution P' over solutions of C via a normalization constant. If $S(C)$ is the set of solutions to C, then $P'(Y) = P(Y)/\sum_{Z \in S(C)} P(Z)$.

3 Interpreting Incomplete Probabilistic Information

In this Section we consider the question of converting possibly incomplete probabilistic information (probabilistic constraints) supplied by the user into a well-defined belief network. Essentially we have to define a probability distribution P that is consistent with the constraints. In the next Section we give a method for converting P into a distribution P' over the solutions of a constraint network, and generating solutions in accordance with it.

The input at this stage is a set of probabilistic constraints. The CSP itself will be considered in the next Section. A probabilistic constraint over a set of variables X is a function mapping assignments to a subset of X (called its scope) to probability values. For example, $P(X_1|\neg X_2, X_3, \neg X_5) = 0.375$ is a probabilistic constraint. So is $P(X_2, \neg X_4) = 0.4$ and $P(X_3, X_4, \neg X_5) = 0.75$. Let $P_c = \{P_1, P_2, ..., P_m\}$ be a set of probabilistic constraints. Our task is to map P_c to a belief network B that is consistent with it. To do this we will eliminate the variables in X one by one. To eliminate a variable, we remove every constraint from P_c that mentions it and add back some new ones that don't. As we do this, we will build up a linear program. Every elimination step results in adding some variables and linear equalities and inequalities to the program. After all

variables are eliminated from P_c, the constructed linear program is solved by a standard algorithm such as Simplex. Its solution is then mapped to the required probability distribution.

To avoid overloading the term, we will use "variable" to refer to a variable in X and "linear program variable" to refer to one of the variables in the linear program. These two sets are distinct. Also, we will use "constraints" to refer to the probabilistic constraints in P_c and "linear constraints" to refer to the linear equalities and inequalities in the linear program. The original constraints in the input P_c are functions mapping assignments to their scopes to real probability values. Constraints that are added to P_c when a variable is eliminated are somewhat different, mapping assignments to their scope to linear constraints in the linear program variables.

In the base case, we set P_c equal to the input set of constraints. Let L be the linear program that we will construct, which initially contains no linear program variables or linear constraints. Now assume by induction that $P_c = \{P_1, P_2, ..., P_m\}$. We choose a variable $X_i \in X$ for elimination. Let $C_i = \{P_{t_1}, ..., P_{t_k}\}$ be the set of constraints in P_c that mention X_i in their scopes. Set Q_i to the set of all variables that appear in the scope of some constraint in C_i. For every assignment q_i to the variables in Q_i, we introduce a linear program variable p_{q_i}. We also add linear constraints $\forall q_i . p_{q_i} \geq 0$ and $\sum_{q_i} p_{q_i} = 1$. The linear program variable p_{q_i} is intended to represent the probability of Q_i assuming the value q_i in the final probability distribution. If $q_i = \{X_1 = x_1, X_2 = x_2, ..., X_i = x_i\}$, then we can also write p_{q_i} as $p_{X_1=x_1,...,X_i=x_i}$. Now, every constraint P_{t_j} in C_i can be expressed as a linear constraint on the linear program variables $\{p_{q_i}\}$. For example, if $Q_i = \{X_1, X_2, X_i\}$ and P_{t_j} defines $X_1 = 3 \wedge X_i = 1$ to have probability 0.45, then an appropriate linear constraint would be $\sum_{x_2} p_{X_1=3,X_2=x_2,X_i=1} = 0.45$. After linear constraints are introduced for every constraint in C_i, C_i is removed from P_c and deleted. Let $N_i = Q_i - \{X_i\}$. We define a new constraint P^i whose scope is N_i, which we will add back to P_c. P^i maps the assignment of n_i to N_i to the linear constraint $\sum_{x_i} p_{N_i=n_i,X_i=x_i}$. Basically P^i expresses the required distribution on N_i as a function of the distribution on Q_i, represented by the linear program variables. After P^i is added back to P_c, the procedure continues recursively to eliminate the next variable.

After all variables have been eliminated, the linear program L is solved with a standard algorithm such as Simplex. This results in assigning an explicit probability value to every linear program variable p_{q_i}. We can map this to a belief network, by associating variable X_i with the CPT $P(X_i = x_i | N_i = n_i) = p_{X_i=x_i,N_i=n_i} / \sum_x p_{X_i=x,N_i=n_i}$. The distribution represented by this belief network will conform to all of the original constraints.

Pseudo code for this algorithm, called Make-Bel, is given in Figure 1.

The complexity of this algorithm is exponential in the cardinality of the largest Q_i. This quantity is also known as the induced width of the constraint graph [1]. The induced width can be very large for problems with dense variable interrelationships, rendering the algorithm impractical. Therefore it can be desirable to generate a belief network that only approximately conforms to the

Procedure Make-Bel

Input: A set of probabilistic constraints $P_c = \{P_i\}$ defined on X.

Output: A belief network conforming to P_c.

Eliminate the variables one by one. To eliminate X_i,

Set $C_i := \{P_j | P_j \in P_c$ is defined on $X_i\}$.

Let Q_i be the variables mentioned by a $P_j \in C_i$. Let $N_i = Q_i - \{X_i\}$.

Represent each $P_j \in C_i$ with a linear constraint on Q_i's probabilities.

Remove C_i from P_c.

Add $P^i(N_i)$, defining N_i's probabilities in terms of Q_i's, to P_c.

Solve the linear program with Simplex. Make a belief network by associating X_i with

$$P(X_i = x_i | N_i) = \frac{p_{X_i = x_i, N_i}}{\sum_x p_{X_i = x, N_i}}$$

Fig. 1. Procedure Make-Bel

constraints. The basic idea is to reduce the size of the generated constraints P^i. One large constraint of this type can be approximated by a set of smaller ones via marginalization, using the basic identity $P^i_j(Y = y) = \sum_x P^i(Y = y, X = x)$, for any number of j's and Y's. By doing this any limit can be placed on the size of the generated functions, at the price of accuracy of course. This keeps the size of the Q_i's from expanding too much because of new constraints, and keeps the overall complexity within reasonable bounds.

4 Generating Well-Distributed CSP Solutions

In this Section we show how to generate solutions of a CSP C in accordance with a given probability distribution. If P is a probability function, $S(C)$ is the set of solutions of C, and Y is in $S(C)$, then solution Y should be generated with probability $P(Y)/\sum_{Z \in S(C)} P(Z)$.

We assume that P is supplied in the form of a Bayesian network, perhaps generated from incomplete information by the method of the preceding Section. P will be transformed into a distribution P' over the solutions of C.

Before we explain the algorithm, it is necessary to consider the mathematical justification. Let X be the set of n variables over which P and C are defined, ordered arbitrarily. We define the function F as follows:

$$F(X_1 = x_1, ..., X_i = x_i) = \sum_{\{x_{i+1}, ..., x_n | \{X_1 = x_1, ..., X_n = x_n\} \in S(C)\}} P(X_1 = x_1, ..., X_n = x_n)$$

In other words, $F(X_1 = x_1, ..., X_i = x_i)$ is the sum of the probabilities of all solutions of C that are consistent with the assignment $\{X_1 = x_1, ...X_i = x_i\}$.

We can also define the conditional F as

$$F(X_i = x_i | X_1 = x_1, ..., X_{i-1} = x_{i-1}) = \frac{F(X_1 = x_1, ..., X_i = x_i)}{F(X_1 = x_1, ..., X_{i-1} = x_{i-1})}$$

Now, the algorithm will work by assigning random values to the variables in order. When X_i is assigned, assuming that the previous assignment is $\{X_1 = x_1, X_2 = x_2, ..., X_{i-1} = x_{i-1}\}$, it will get value x with probability $F(X_i = x | X_1 = x_1, X_2 = x_2, ..., X_{i-1} = x_{i-1})$. So the total probability of the final assignment $\{X_1 = x_1, X_2 = x_2, ..., X_n = x_n\}$ is

$$F(X_1 = x_1|\emptyset) \cdot F(X_2 = x_2 | X_1 = x_1) \cdots F(X_n = x_n | X_1 = x_1, ..., X_{n-1} = x_{n-1}) =$$

$$\frac{F(X_1 = x_1)}{F(\emptyset)} \cdot \frac{F(X_1 = x_1, X_2 = x_2)}{F(X_1 = x_1)} \cdots \frac{F(X_1 = x_1, ..., X_n = x_n)}{F(X_1 = x_1, ..., X_{n-1} = x_{n-1})} =$$

$$\frac{F(X_1 = x_1, ..., X_n = x_n)}{F(\emptyset)}$$

Note that by definition the numerator of this fraction is $P(X_1 = x_1, ...X_n = x_n)$ if the assignment is a solution to C and zero otherwise, and the denominator is $\sum_{Y \in S(C)} P(Y)$. This shows that the sampling process does indeed conform to the required distribution. The only thing remaining is to show how to compute F.

It is easy to see that $F(X_1 = x_1, ..., X_{i-1} = x_{i-1}) = \sum_{x_i} F(X_1 = x_1, X_2 = x_2, ..., X_i = x_i)$ by the laws of probability. This allows us to apply the variable elimination procedure to compute F. In the base case, if the belief network P is given as the set of CPTs $\{P(X_i|pa_i)\}$ and the constraint network C is the set of boolean functions $\{C_j\}$, then $F(X_1 = x_1, ...X_n = x_n) = \prod_j C_j \prod_i P(X_i|pa_i)$. For induction, assume that $F(X_1 = x_1, ..., X_i = x_i) = \prod_j f_j$, where $\{f_j\}$ is a set of functions defined on subsets of $\{X_1, ..., X_i\}$, and B_i is the set of functions that mention X_i in their scopes. Then

$$F(X_1 = x_1, ...X_{i-1} = x_{i-1}) = \sum_{x_i} \prod f_j = \sum_{x_i} \prod_{f_j \in B_i} f_j \prod_{f_k \notin B_i} f_k = \prod_{f_k \notin B_i} f_k \sum_{x_i} \prod_{f_j \in B_i} f_j$$

The function $f^i = \sum_{x_i} \prod_{f_j \in B_i} f_j$ can be computed explicitly and stored like the other functions, allowing the recursion to proceed with the next variable elimination step. The algorithm will work by eliminating all of the variables in turn. The final result will be $F(\emptyset)$, the sum of the probabilities of all CSP solutions. When a variable X_i is eliminated, the set of functions B_i needs to be saved. These functions will allow us to reconstruct $F(X_i = x_i | X_1 = x_1, ..., X_{i-1} = x_{i-1})$, as can be seen by the following equalities.

$$\frac{F(X_1 = x_1, ...X_i = x_i)}{F(X_1 = x_1, ...X_{i-1} = x_{i-1})} = \frac{\prod_{f_k \notin B_i} f_k \prod_{f_j \in B_i} f_j}{\prod_{f_k \notin B_i} f_k \sum_{x_i} \prod_{f_j \in B_i} f_j} = \frac{\prod_{f_j \in B_i} f_j}{\sum_{x_i} \prod_{f_j \in B_i} f_j}$$

Pseudo-code for this algorithm, called Constrain-Bel, is given in Figure 2. Note that it is basically the same algorithm as Elim-CPE [6]. However in this

case it is the intermediate function sets $\{B_i\}$ that are of interest, and not the final answer. It is straightforward to see how the sets $\{B_i\}$ can be translated into conditional probability tables representing the conditional function F as shown above. Therefore the result of Constrain-Bel can be interpreted as a modified Bayesian network, representing the distribution P' over solutions of C.

Procedure Constrain-Bel
Input: Set of functions $F = \{f_j\}$ representing a CSP and belief network.
Output: $\sum_X \prod_j f_j$. Function sets $\{B_i\}$.
Eliminate the variables one by one. To eliminate X_i,
  Set $B_i := \{f_j | f_j \in F$ is defined on $X_i\}$.
  Let $f^i = \sum_{x_i} \prod_{f_j \in B_i} f_j$.
  Set $F = F \cup \{f^i\} - B_i$. Save B_i.
Return final set F, function sets $\{B_i\}$.

Fig. 2. Procedure Constrain-Bel

The complexity of Constrain-Bel is exponential in the cardinality of the scope of the largest computed function f^i, which again is equivalent to the induced width of the constraint graph. Since this can be prohibitive for large problems, an approximation method may be desirable. This can be done in essentially the same way as in [4]. When a variable X_i is eliminated, the functions in B_i can be partitioned into mini-buckets, and then X_i is summed out of each mini-bucket separately. This reduces the size of the computed functions. However the solution generation process has to be modified to incorporate backtracking, since the modified belief network may not satisfy certain laws of probability and may encounter a deadend during the sampling process. For more details see [4].

5 Conclusion

In this paper, we presented a method for generating solutions of a constraint satisfaction problem in accordance with possibly incomplete probabilistic information. We gave an algorithm for compiling a set of incomplete probabilistic constraints into a consistent well-defined belief network, and a method for sampling solutions from a CSP in accordance with the probability distribution specified by a belief network. Since the complexity of these algorithms is exponential in the induced width of the variable interaction graph, we also briefly considered approximation methods.

Outside of functional verification, there are other uses for the techniques presented here. Make-Bel can be used to solve probabilistic logic problems [11]. We believe it is the first variable-elimination algorithm in this area, and also it readily allows approximation methods. The problem of filling in the CPT

entries of an incompletely specified belief network can also be addressed with this technique, which would provide an alternative to the approximate stochastic methods presented in [8]. However it cannot support non-linear constraints, which includes arbitrary independence constraints, though some choice of independence constraints is available through the approximation method.

It still remains to implement and test these methods. The anytime behavior of the approximation methods, in which accuracy could be plotted as a function of time taken, should be studied on a variety of problems. It is also possible to introduce additional approximation algorithms with further variations of the mini-bucket technique, as in Iterative Join Graph propagation [5]. It would also be interesting to find the constraint-satisfying probability distribution that has maximum entropy [9], rather than choosing one arbitrarily.

References

[1] S. A. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability- a survey. *BIT*, 25:2–23, 1985.

[2] J. Bergeron. *Writing Testbenches: Functional Verification of HDL Models.* Kluwer Academic Publishers, 2000.

[3] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

[4] Rina Dechter, Kalev Kask, Eyal Bin, and Roy Emek. Generating random solutions for constraint satisfaction problems. In *Proceedings of AAAI*, 2002.

[5] Rina Dechter, Kalev Kask, and Robert Mateescu. Iterative join-graph propagation. Submitted for publication, 2002.

[6] Rina Dechter and David Larkin. Hybrid processing of beliefs and constraints. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*, 2001.

[7] Rina Dechter and Irina Rish. A scheme for approximating probabilistic inference. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*, pages 132–141, 1997.

[8] M. J. Druzdzel and L. C. van der Gaag. Elicitation of probabilities for belief networks: Combining qualitative and quantitative information. In *Proc. of the 11th Conf. on Uncertainty in Artificial Intelligence*, pages 141–8, 1995.

[9] E. T. Jaynes. Where do we stand on maximum entropy? In *The Maximum Entropy Formalism.* M. I. T. Press, 1979.

[10] Daphne Koller and Nimrod Megiddo. Constructing small sample spaces satisfying given constraints. *SIAM Journal of Discrete Mathematics*, 7(2):260–274, May 1994.

[11] N. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28:71–88, 1986.

[12] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

[13] Linda van der Gaag. Computing probability intervals under independency constraints. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*, pages 457–466, 1991.

[14] Jun Yuan, Kurt Schultz, Carl Pixley, Hiller Miller, and Adnan Aziz. Modeling design constraints and biasing using BDDs in simulation. In *Proc. of International Conference on Computer-Aided Design*, November 1999.