Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)
Journal Track

From Support Propagation to Belief Propagation in Constraint Programming (Extended Abstract) *

Gilles Pesant
Polytechnique Montréal, Montreal, Canada
gilles.pesant@polymtl.ca

Abstract

The distinctive driving force of constraint programming (CP) to solve combinatorial problems has been a privileged access to problem structure through the high-level models it uses. We investigate a richer propagation medium for CP made possible by recent work on counting solutions inside constraints. Beliefs about individual variable-value assignments are exchanged between contraints and iteratively adjusted. Its advantage over standard belief propagation is that the higher-level models do not tend to create as many cycles, which are known to be problematic for convergence. We find that it significantly improves search guidance.

1 Introduction

The distinctive driving force of *Constraint Programming* (CP) to solve combinatorial problems has been a privileged access to problem structure through the high-level models it uses. From that exposed structure in the form of so-called global constraints, powerful inference algorithms have shared information between constraints by propagating it through shared variables' domains, traditionally by removing unsupported values. The paper investigates a richer propagation medium made possible by recent work on model counting inside constraints.

Many forms of *message passing* algorithms have been investigated in AI. One of the earliest and best known, *Belief Propagation* (BP) also known as *sum-product message passing*, was proposed by Pearl [1982] to perform inference on graphical models. From a joint probability distribution it computes approximations of the marginal distributions onto individual variables (i.e. beliefs). Each message is a real-valued function over the domain of a variable that expresses a probability that the variable takes a given value in a model. BP is known to converge to the exact marginal distributions on tree topologies but may not converge in general.

In CP, *constraint propagation* can also be viewed as a message passing algorithm, announcing value deletions from domains through the constraint network and necessarily converging because some quantity, namely the size of the Cartesian product of the domains, decreases monotonically [Horsch and Havens, 2013; Werner, 2015]. From the perspective of message passing, the deleted values being propagated are messages taking the form of simpler Boolean-valued functions evaluating to false for these deleted values and to true otherwise, which is rather flat information since all non-deleted values are on an equal footing. One way to view a variable's filtered domain with respect to a constraint is as a set of variable-value pairs having non-zero frequency among its solution set [Dechter *et al.*, 2010] — if instead we share the whole frequency distribution over individual variables we can discriminate between values from the perspective of each constraint and, for example, use it for branching in the search tree. Once we consider the frequency of a variable-value pair as its likelihood, or probability, of appearing in a solution to the given constraint, we come very close to belief propagation and other message passing algorithms for probabilistic inference. Beliefs about individual variable-value assignments are exchanged between contraints and iteratively adjusted. It generalizes standard *support propagation* in CP and aims to converge to the true marginal distributions of the solutions over individual variables. Its advantage over standard belief propagation is that the higher-level models of CP featuring large-arity (global) constraints do not tend to create as many cycles, which are known to be problematic for convergence.

Since in general we don't have an explicit description of the solution set of a constraint, that frequency distribution is not readily available and would have to be approximated, perhaps very coarsely. However recent work on solution counting is bringing it within reach to compute exact (or close) distributions for several families of constraints [Pesant, 2017]. *Counting-based search* [Pesant *et al.*, 2012], a family of effective branching heuristics, drives search toward areas where there are many satisfying assignments to (some of) the constraints taken individually. These are expressed through the concept of *solution density*: given a constraint $c(x_1, \ldots, x_k)$, its number of solutions $\#c(x_1, \ldots, x_k)$, respective finite domains $D(x_j)\ _{1 \leq j \leq k}$, a variable x_i in the scope of c, and a value $v \in D(x_i)$,

$$\sigma(x_i, v, c) = \frac{\#c(x_1, \ldots, x_{i-1}, v, x_{i+1}, \ldots, x_k)}{\#c(x_1, \ldots, x_k)}$$

defines the solution density of variable-value pair (x_i, v) in

*The full version of this paper was published as [Pesant, 2019].

	1	2	3	4
θ_a	0	1/2	1/2	0
θ_b	0	1/2	1/2	0
θ_c	1	0	0	0
θ_d	1	0	0	0

Table 1: True marginals

constraint c, given as the fraction of solutions to c which include that assignment. Depending on the constraint and the combinatorial structure it encapsulates, such counting may be intractable. Nevertheless the recent work on counting-based search has provided efficient algorithms to count either exactly or approximately for several important families of constraints [Pesant *et al.*, 2012; Brockbank *et al.*, 2013; Pesant, 2015].

There is a well-established connection between probabilistic inference and weighted model counting [Chavira and Darwiche, 2008]. From the perspective of *weighted counting*, the concept of solution density has an underlying assumption that the likelihood of values in each domain follows a uniform distribution, thereby leading to all solutions having equal weight. We need to abandon this assumption and consider arbitrary distributions arising from the variable-to-constraint messages that reflect the belief acquired from the other constraints and which is iteratively adjusted.

The paper promotes a richer propagation medium in CP, made possible by extending the work on solution density to weighted counting inside constraints and close in spirit to message passing algorithms. It also contributes an instantiation of belief propagation that is less affected by cycles through the use of higher-order potentials corresponding to CP's global constraints, weighted-counting algorithms for some of the latter, and a publicly-available implementation for further research.

2 An Example

Consider the following example to illustrate our approach:

i. $\texttt{alldifferent}(a, b, c)$

ii. $a + b + c + d = 7$

iii. $c \leq d$

three constraints with variables a, b, c, $d \in \{1, 2, 3, 4\}$. There are two solutions to that CP problem: $(a = 2, b = 3, c = 1, d = 1)$ and $(a = 3, b = 2, c = 1, d = 1)$. Even if we enforce domain consistency on each constraint, no filtering occurs. To solve it we would thus be left to branch on variables having identical domains.

Variable a takes value 2 in one of the two solutions, value 3 in one solution as well, and values 1 and 4 in no solution. If we look at the set of solutions as a multivariate discrete distribution, its projection onto a yields $\langle 0, 1, 1, 0\rangle$ and under the assumption that either solution is equally likely the marginal probability distribution for a is then $\theta_a = \langle 0, 1/2, 1/2, 0\rangle$. Table 1 gives the marginal for each variable and Table 2, the marginals local to each constraint taken individually and over its own set of solutions. We see that from the point of view of the `alldifferent` constraint and for variable a (line θ^i_a) each value is equally likely since it appears in the same number of solutions to that constraint. Whereas for that same variable but from the point of view of the linear equality constraint (line θ^{ii}_a) value 1 is ten times more likely than value 4. Note also that for variable d the two local marginals give conflicting beliefs.

	1	2	3	4
θ^i_a	1/4	1/4	1/4	1/4
θ^{ii}_a	10/20	6/20	3/20	1/20
θ^i_b	1/4	1/4	1/4	1/4
θ^{ii}_b	10/20	6/20	3/20	1/20
θ^i_c	1/4	1/4	1/4	1/4
θ^{ii}_c	10/20	6/20	3/20	1/20
θ^{iii}_c	4/10	3/10	2/10	1/10
θ^{ii}_d	10/20	6/20	3/20	1/20
θ^{iii}_d	1/10	2/10	3/10	4/10

Table 2: Marginals local to each constraint

	1	2	3	4
θ_a	.01	.52	.46	.01
θ_b	.01	.52	.46	.01
θ_c	.98	.02	.00	.00
θ_d	.90	.10	.00	.00

Table 3: Computed marginals after ten iterations

Table 3 presents the computed marginals after ten iterations of belief propagation on this CP model using exact weighted counting on each constraint. Note how close these got to the true marginals in Table 1. To motivate our interest in taking advantage of the high-level modelling typically present in CP, made up of a relatively low number of high-arity constraints, consider Table 4 showing the impact of replacing the `alldifferent` constraint by its decomposition into three disequality constraints: $a \neq b$, $a \neq c$, $b \neq c$. The computed marginals are much further from the true marginals.

3 Belief Propagation in a CP Solver

Our prototype[1] is built on top of MiniCP, a recent bare-bones open-source CP solver developed for academic purposes and written in Java [Michel *et al.*, 2017]. Though few constraints and filtering algorithms are currently implemented, its small and clean architecture made it easier to implement the required architectural changes to the core of the solver without worrying about the potential impacts on a more complex system. We briefly describe the main changes.

In our setting a message corresponds to the frequency (marginal) distribution of a variable. In order to maintain the marginal distribution of a variable, each value in its domain is given a weight attribute. Variables are provided with simple methods to receive and send messages: incoming messages from constraints are combined by multiplying them and an outgoing message sent to a constraint excludes its own contribution to the distribution (this is termed the *outside belief*).

[1]Its current implementation is available at https://github.com/PesantGilles/MiniCPBP

	1	2	3	4
θ_a	.37	.40	.20	.03
θ_b	.37	.40	.20	.03
θ_c	.61	.37	.02	.00
θ_d	.40	.45	.13	.02

Table 4: Computed marginals after ten iterations but using a decomposition of `alldifferent`

Constraints are also provided with methods to receive and send messages, and another method to perform the weighted counting.

Message passing is synchronized in two phases: in the first phase constraints receive messages from the variables and then update their beliefs by performing weighted counting based on these updated marginal distributions; in the second phase marginals for the variables are reset, constraints send back messages to the variables, and marginals are normalized. This goes on for a fixed number of iterations.

We solve a problem by backtrack search as usual. At each node of the search tree we perform belief propagation and then branch based on the computed marginals.

4 Weighted Model Counting

The main technical challenge of this approach is to design efficient weighted model counting algorithms for each family of constraints. Because each has a different combinatorial structure, the kind of algorithm required likely differs as well. We present some of them here.

4.1 Table

The $\texttt{table}(x_1, x_2, \ldots, x_k, T)$ constraint restricts the sequence of variables $\langle x_1, x_2, \ldots, x_k \rangle$ to take on a combination of values appearing in T, the set of allowed tuples. The Compact Table algorithm [Demeulenaere *et al.*, 2016] implementing the `table` constraint computes bit sets $T_{xv} \subseteq T$ identifying for each variable-value pair (x, v) the tuples in which it appears. It also maintains a bit set S of currently-supported tuples (those that could be a solution to the constraint given the current domains of the variables). Counting the solutions in which $x = v$ then simply amounts to computing the cardinality of $T_{xv} \cap S$. For weighted counting we first compute the weight of each currently-supported tuple $\langle v_1, v_2, \ldots, v_k \rangle$ as the product of the outside belief of each pair (x_i, v_i) and sum the relevant weights for each T_{xv}.

4.2 Sum and Regular

The $\texttt{sum}(x_1, x_2, \ldots, x_k, y)$ constraint enforces $y = \sum_{i=1}^{k} x_i$. It can be used to state inequalities and weighted sums as well. As previously proposed to achieve domain consistency and to compute solution densities for `knapsack` constraints [Pesant and Quimper, 2008], conceptually we build a layered graph where each path from the first to the last layer represents a solution, with each component arc corresponding to an individual variable assignment. The only difference here is that now each arc carries a weight equal to the outside belief of the corresponding variable assignment and that instead of storing the number of incoming and outgoing paths at each node, we store *weighted* sums π_{in} and π_{out} of incoming and outgoing paths respectively where a path's weight is the product of the weights on its component arcs. We compute them through standard forward and backward passes and then use these weighted sums to compute the local beliefs for each variable-value pair.

The $\texttt{regular}(x_1, x_2, \ldots, x_k, \mathcal{A})$ constraint restricts the sequence of variables $\langle x_1, x_2, \ldots, x_k \rangle$ to spell out a word belonging to the regular language recognized by automaton $\mathcal{A}$. The same approach can be used for `regular` constraints since a similar layered graph has been proposed to achieve domain consistency and to count solutions [Pesant *et al.*, 2012].

4.3 Among

The $\texttt{among}(x_1, x_2, \ldots, x_k, V, c)$ constraint makes variable c correspond to the number of occurrences of values from set V in $\{x_1, x_2, \ldots, x_k\}$. By introducing binary indicator variables $y_1, y_2, \ldots, y_k$ such that $y_i = 1 \Leftrightarrow x_i \in V$ we rewrite an `among` constraint as $\texttt{sum}(y_1, y_2, \ldots, y_k, c)$ and use its weighted counting algorithm.

4.4 Alldifferent

The $\texttt{alldifferent}(x_1, x_2, \ldots, x_k)$ constraint restricts variables $x_1, x_2, \ldots, x_k$ to take on distinct values. The one-to-one correspondence between solutions to an `alldifferent` constraint and maximum matchings in the associated bipartite graph is well known. Counting such matchings in turn corresponds to computing the *permanent* of the adjacency matrix $A = (a_{ij})$ of the bipartite graph,

$$\text{per}(A) = \Sigma_{p \in \mathcal{P}} \Pi_{i=1}^{k} a_{i,p(i)}$$

where $\mathcal{P}$ denotes the set of all permutations of $\{1, 2, \ldots, k\}$. Unfortunately this is a $\#P$-complete problem (one of the so-called "hard to count – easy to decide" problems) [Valiant, 1979]. Nevertheless it is a well-studied problem, even for more general nonnegative matrices, for which several upper bounds have been proposed [Soules, 2003] and even polytime randomized approximation algorithms [Jerrum *et al.*, 2001]. We will cast our weighted counting problem as that of computing the permanent of a nonnegative matrix built from the marginals.

We first construct matrix A by setting a_{ij} to the outside belief about variable x_i taking value j. We interpret the permanent of A as a weighted counting of maximum matchings: if $a_{ij} = 1$ its participation in a matching is fully counted; the smaller a_{ij} is, the more it will be discounted; $a_{ij} = 0$ still means that no matching includes the corresponding assignment. For each variable x_i and value j, the (unnormalized) local belief is equal to the weighted counting of matchings that include assignment $x_i = j$, which we evaluate as the permanent of the sub-matrix of A obtained by removing row i and column j. We compute that permanent exactly for sub-matrices whose size does not exceed a given small threshold τ and otherwise use a quick-to-compute upper bound (bound U^3 from [Soules, 2003]) thus getting an approximate weighted counting.

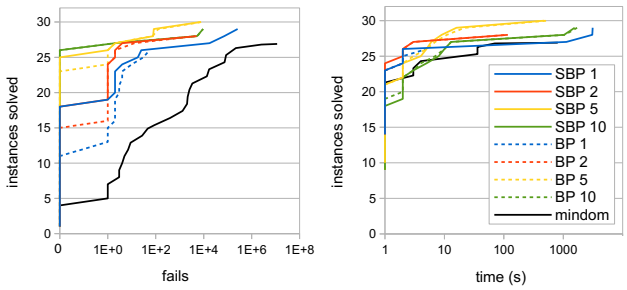

Figure 1: Number of instances solved with respect to the number of fails (left) and computation time (right) for the Primes instances.

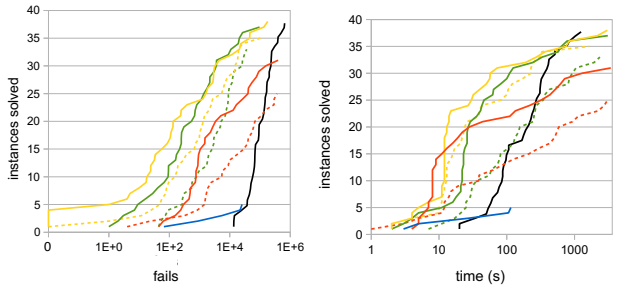

Figure 2: Number of instances solved with respect to the number of fails (left) and computation time (right) for the Magic Square instances.

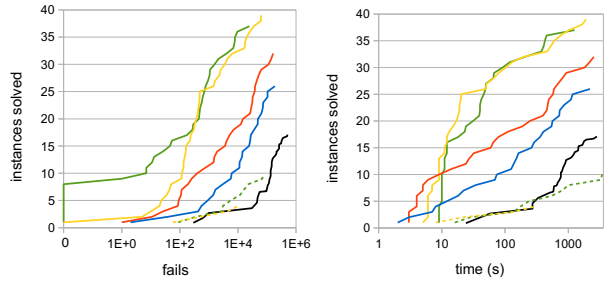

Figure 3: Number of instances solved with respect to the number of fails (left) and computation time (right) for the Latin Square instances.

5 Combinatorial Search

We investigate CP-based belief propagation's search guidance to find a solution to constraint satisfaction problems. We use the following two-way branching heuristic: max-strength assigns the variable-value pair exhibiting the largest positive difference between its marginal and the reciprocal of the domain size (i.e. its *marginal strength*) and disallows it upon backtracking. As a baseline we use minimum domain size variable ordering with random value ordering and report the average result over ten runs.

We report on three combinatorial problems from the XCSP3 repository[2] that can be modelled using the constraints from the previous section: Latin Square, Magic Square, and Primes. We stop an individual run after one hour of computation time.

5.1 Primes

The CP model for the Primes instances features 100 variables and many `sum` constraints. Figure 1 plots the number of instances solved against the number of failed search-tree nodes reached during search (left) and against the computation time (right). Each curve represents a particular configuration of the algorithm: mindom is our baseline branching heuristic (minimum domain size with random value ordering) with standard support propagation; all the others use the max-strength branching heuristic, with SBP/BP indicating that belief propagation is used with/without prior application of support propagation and the integer label (1, 2, 5, 10) representing the number of BP iterations used at each search-tree node.

We start with the number of fails. The SBP curves show that search guidance improves with the number of BP iterations, as expected. Remarkably for most instances the max-strength branching heuristic guides directly to a solution, culminating at 10 iterations with 26 out of 32 instances being solved backtrack-free. Branching heuristic mindom does not guide nearly as well and only solves 4 instances backtrack-free. In general it exhibits several orders of magnitude more fails. Turning to computation time (right plot) we see that the gap between mindom and max-strength almost closes because of the computational cost of exact counting over the many `sum` constraints but the latter nevertheless solves a few more instances within the time limit.

[2]http://www.xcsp.org/

5.2 Magic Square

The CP model for these 9×9 partially-filled instances features `sum` and `alldifferent` constraints. Figure 2 shows that heuristic mindom requires on average tens of thousands of fails to solve any of them but manages to solve almost all of them given one order of magnitude more fails. SBP with 5 and 10 iterations requires a few orders of magnitude fewer fails than mindom. Because weighted counting on the `alldifferent` constraint is not exact until few unbound variables remain, not applying support propagation (the BP curves) deteriorates performance. Turning to computation time, even though mindom eventually catches up on the last few instances solved, for the most part it is about one order of magnitude slower than max-strength.

5.3 Latin Square

The CP model for these 30×30 partially-filled instances features `alldifferent` constarints. Figure 3 shows that heuristic mindom only manages to solve 17 out of 40 instances within the time limit on average. Again max-strength's search guidance improves with the number of iterations and SBP 5 solves all but one instance. Because no constraint initially performs exact weighted counting here and therefore very little domain filtering will occur, not applying support propagation (the BP curves) is not advisable. Observing the foot of the SBP curves, where the number of fails is almost the same, gives an indication of the computational cost of each iteration of BP.

References

[Brockbank *et al.*, 2013] Simon Brockbank, Gilles Pesant, and L.-M. Rousseau. Counting Spanning Trees to Guide Search in Constrained Spanning Tree Problems. In Christian Schulte, editor, *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 175–183. Springer, 2013.

[Chavira and Darwiche, 2008] Mark Chavira and Adnan Darwiche. On Probabilistic Inference by Weighted Model Counting. *Artif. Intell.*, 172(6-7):772–799, 2008.

[Dechter *et al.*, 2010] R. Dechter, B. Bidyuk, R. Mateescu, and E. Rollon. On the Power of Belief Propagation: A Constraint Propagation Perspective. In *Festschrift book in honor of Judea Pearl*. 2010.

[Demeulenaere *et al.*, 2016] Jordan Demeulenaere, Renaud Hartert, Christophe Lecoutre, Guillaume Perez, Laurent Perron, Jean-Charles Régin, and Pierre Schaus. Compact-Table: Efficiently Filtering Table Constraints with Reversible Sparse Bit-Sets. In Michel Rueher, editor, *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, volume 9892 of *Lecture Notes in Computer Science*, pages 207–223. Springer, 2016.

[Horsch and Havens, 2013] Michael C. Horsch and William S. Havens. Probabilistic Arc Consistency: A Connection between Constraint Reasoning and Probabilistic Reasoning. *CoRR*, abs/1301.3864, 2013.

[Jerrum *et al.*, 2001] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A Polynomial-time Approximation Algorithm for the Permanent of a Matrix with Non-negative Entries. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pages 712–721, New York, NY, USA, 2001. ACM.

[Michel *et al.*, 2017] Laurent Michel, Pierre Schaus, and Pascal Van Hentenryck. Mini-CP: A Minimalist Open-Source Solver to Teach Constraint Programming. http://www.minicp.org, 2017.

[Pearl, 1982] Judea Pearl. Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach. In David L. Waltz, editor, *Proceedings of the National Conference on Artificial Intelligence. Pittsburgh, PA, August 18-20, 1982.*, pages 133–136. AAAI Press, 1982.

[Pesant and Quimper, 2008] Gilles Pesant and C.-G. Quimper. Counting Solutions of Knapsack Constraints. In Laurent Perron and Michael A. Trick, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR 2008, Paris, France, May 20-23, 2008, Proceedings*, volume 5015 of *Lecture Notes in Computer Science*, pages 203–217. Springer, 2008.

[Pesant *et al.*, 2012] Gilles Pesant, C.-G. Quimper, and Alessandro Zanarini. Counting-Based Search: Branching Heuristics for Constraint Satisfaction Problems. *J. Artif. Intell. Res.*, 43:173–210, 2012.

[Pesant, 2015] Gilles Pesant. Achieving Domain Consistency and Counting Solutions for Dispersion Constraints. *INFORMS Journal on Computing*, 27(4):690–703, 2015.

[Pesant, 2017] Gilles Pesant. Getting More Out of the Exposed Structure in Constraint Programming Models of Combinatorial Problems. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 4846–4851. AAAI Press, 2017.

[Pesant, 2019] Gilles Pesant. From Support Propagation to Belief Propagation in Constraint Programming. *J. Artif. Intell. Res.*, 66:123–150, 2019.

[Soules, 2003] G.W. Soules. New Permanental Upper Bounds for Nonnegative Matrices. *Linear and Multilinear Algebra*, 51(4):319–337, 2003.

[Valiant, 1979] L. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[Werner, 2015] Tomás Werner. Marginal Consistency: Upper-Bounding Partition Functions over Commutative Semirings. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(7):1455–1468, 2015.