Journal of Artificial Intelligence Research 66 (2019) 123–150 Submitted 05/2019; published 09/2019

From Support Propagation to Belief Propagation in Constraint Programming

Gilles Pesant GILLES.PESANT@POLYMTL.CA
Polytechnique Montréal, Montreal, Canada
CIRRELT, Université de Montréal, Montreal, Canada

Abstract

The distinctive driving force of constraint programming 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. This paper investigates a richer propagation medium made possible by recent work on counting solutions inside constraints. Beliefs about individual variable-value assignments are exchanged between contraints and iteratively adjusted. It generalizes standard support propagation 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 featuring large-arity (global) constraints do not tend to create as many cycles, which are known to be problematic for convergence. The necessary architectural changes to a constraint programming solver are described and an empirical study of the proposal is conducted on its implementation. We find that it provides close approximations to the true marginals and that it significantly improves search guidance.

1. Introduction

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.

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 & 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, Bidyuk, Mateescu, & Rollon, 2010) — if instead we share the whole frequency distribution over individual vari-

ables we can discriminate between values from the perspective of each constraint and, for example, use it for branching in the search tree. 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). 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.

Among the different types of graphical models, *factor graphs* are closest to constraint networks: they are bipartite graphs featuring variable nodes and factor nodes, the latter corresponding to constraints, with edges between variables and the constraints in which they appear. Messages are sent back and forth between variable and factor nodes, thereby solving in an iterative manner a set of mutually recursive equations defining these messages. Let $\mu_{x\to c}$ denote the message from variable x to constraint c and $\mu_{c\to x}$ the message from constraint c to variable x. We give their definition for BP:

$$\mu_{x\to c}(v) = \prod_{c'\in N(x)\setminus\{c\}} \mu_{c'\to x}(v) \qquad \forall v \in D(x) \tag{1}$$

$$\mu_{c\to x}(v) = \sum_{\boldsymbol{v}:\boldsymbol{v}[x]=v} f_c(\boldsymbol{v}) \prod_{x'\in N(c)\setminus\{x\}} \mu_{x'\to c}(\boldsymbol{v}[x']) \qquad \forall v \in D(x) \tag{2}$$

where $D(x)$ is the domain of possible values for variable x, $N(x)$ is the neighbourhood of variable x i.e. the constraints in which x appears, $N(c)$ is the neighbourhood of constraint c i.e. its scope, $\boldsymbol{v}$ is a tuple from the Cartesian product of the domains of all variables in $N(c)$, $\boldsymbol{v}[x]$ stands for the value taken by variable x in $\boldsymbol{v}$, and f_c is the function associated with c, often given as a conditional probability table but in our specific case of a CSP it is better viewed as a decision procedure returning 1 if tuple $\boldsymbol{v}$ satisfies c and 0 otherwise.

The (unnormalized) *marginal* or *bias* θ_x of variable x is computed as

$$\theta_x(v) = \prod_{c\in N(x)} \mu_{c\to x}(v) \qquad \forall v \in D(x)$$

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 been developed over the years to identify and remove unsupported values, and also more recently to explain conflicts and to count solutions. In particular the latter has led to *counting-based search* (Pesant, Quimper, & Zanarini, 2012), a family of effective branching heuristics. 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 $D(x_j)$ $_{1\le j\le k}$, 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 pair (x_i, v) in c, measuring how often a certain assignment is part of a solution to c. It is interesting to note that the numerator above corresponds

exactly to

$$\sum_{\boldsymbol{v}:\boldsymbol{v}[x]=v} f_c(\boldsymbol{v}), \tag{3}$$

the initial part of the constraint-to-variable message in Equation 2. The solution density can thus be seen as a normalized form of the latter message but *assuming a uniform distribution of values in each domain* since it does not take into account the belief acquired by each variable from the other constraints (i.e. the rest of Equation 2). Summation 3 essentially counts the models (local to c) in which $x = v$. Depending on the constraint and the combinatorial structure it encapsulates, such counting may be intractable (e.g. Valiant, 1979). Nevertheless the recent work on counting-based search has provided efficient algorithms to count either exactly or approximately for several constraints (Pesant et al., 2012; Brockbank, Pesant, & Rousseau, 2013; Pesant, 2015). An efficient implementation of belief propagation with global constraints needs to perform counting weighted by the beliefs about variables: this is close in spirit to other work on counting for optimization constraints in which individual costs are associated with each variable assignment (Pesant, 2016; Delaite & Pesant, 2017).

This paper promotes a richer propagation medium in CP, made possible by extending such previous work 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. The rest of the paper is organized as follows. Section 2 provides a motivating example. Section 3 presents the related work. Section 4 describes the architecture of our implementation in a CP solver. Section 5 gives weighted-counting algorithms for some families of constraints. Section 6 offers some empirical results to analyze and support the proposed enhanced propagation in CP. Section 7 concludes with a discussion.

2. An Example

Consider the following example to illustrate the approach:

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

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

iii. $c \leq d$

Together with these three constraints consider identical domain $\{1, 2, 3, 4\}$ for variables a, b, c, and d. There are two solutions to that CSP: $(a = 2, b = 3, c = 1, d = 1)$ and $(a = 3, b = 2, c = 1, d = 1)$. Even if we enforce domain consistency on each constraint, no filtering occurs. To solve this CSP 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

	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

	1	2	3	4
θ_a^i	1/4	1/4	1/4	1/4
θ_a^{ii}	10/20	6/20	3/20	1/20
θ_b^i	1/4	1/4	1/4	1/4
θ_b^{ii}	10/20	6/20	3/20	1/20
θ_c^i	1/4	1/4	1/4	1/4
θ_c^{ii}	10/20	6/20	3/20	1/20
θ_c^{iii}	4/10	3/10	2/10	1/10
θ_d^{ii}	10/20	6/20	3/20	1/20
θ_d^{iii}	1/10	2/10	3/10	4/10

Table 1: True marginals (left) and local marginals (right) for the running example in Section 2

	1	2	3	4
θ_a	.25	.25	.25	.25
θ_b	.25	.25	.25	.25
θ_c	.25	.25	.25	.25
θ_d	.25	.25	.25	.25

	1	2	3	4
θ_a	.50	.30	.15	.05
θ_b	.50	.30	.15	.05
θ_c	.62	.28	.09	.01
θ_d	.29	.34	.26	.11

	1	2	3	4
θ_a	.12	.41	.40	.07
θ_b	.12	.41	.40	.07
θ_c	.84	.15	.01	.00
θ_d	.65	.28	.06	.01

	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 2: Initial marginals (top left) and computed marginals after 1st (top right), 5th (bottom left), and 10th (bottom right) iteration for the running example in Section 2

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 on the left the marginal for each variable and on the right 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_a^i) 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_a^{ii}) value 1 is ten times more likely than value 4. Note also that for variable d the two local marginals give conflicting beliefs.

Successful branching heuristics based on local marginals have been proposed before, such as maxSD that branches on the variable-value pair with the highest overall solution density (i.e. local marginal) (Pesant et al., 2012). Given the information in Table 1 it would branch on one of the four variables, assigning it value 1, since overall the highest solution density (marginal) is 1/2, supplied by constraint ii indiscriminately for each variable.

	1	2	3	4
θ_a	.29	.41	.25	.05
θ_b	.29	.41	.25	.05
θ_c	.66	.31	.03	.00
θ_d	.48	.38	.12	.02

	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 3: Computed marginals after 5th (left) and 10th (right) iteration using a decomposition of `alldifferent` for the running example in Section 2

	1	2	3	4
θ_a	.01	.91	.08	.00
θ_b	.00	.10	.90	.00
θ_c	.99	.01	.00	.00
θ_d	.97	.03	.00	.00

	1	2	3	4
θ_a	.53	.40	.07	.00
θ_b	.29	.30	.37	.04
θ_c	.64	.35	.01	.00
θ_d	.41	.47	.11	.01

Table 4: Computed marginals after 10 iterations with `alldifferent` (left) vs its decomposition (right) for the running example in Section 2, having added constraint $a \leq b$

Table 2 presents the evolution of the computed marginals as the number of belief propagation iterations performed increases. Top left are the initial uniform marginals over supported values. Top right is the result of a single iteration: its effect for each variable is to take the product of the local marginals from all constraints in which it appears. Then it becomes more interesting as variables send back messages that are not necessarily uniform distributions and constraints compute local marginals weighted by these distributions. The two bottom tables show the marginals computed after five and ten iterations — note how these get closer 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 3 showing the impact of replacing the `alldifferent` constraint by its decomposition into three disequality constraints: $a \neq b$, $a \neq c$, $b \neq c$. It is much further from the true marginals and does not even appear to converge.

Finally if we add constraint $a \leq b$ there is only one solution left: $(a = 2, b = 3, c = 1, d = 1)$. Again we see in Table 4 that the marginals computed from the higher-level model get very close to that solution but not so when using the decomposition.

3. Related Work

In this section we review the related work in the area of branching heuristics and about handling cycles in message passing algorithms.

3.1 Belief-Based Branching Heuristics

Horsch and Havens (2013) proposed Probabilistic Arc Consistency for binary CSPs as a generalization of arc consistency and a specialization of BP. An empirical study on random binary CSPs is reported in which the authors use the resulting approximate marginals to branch on the strongest belief, which significantly reduced the size of the search tree but at an overall computational cost that made it less attractive than simpler branching heuristics. Building on the Expectation Maximization Belief Propagation (EMBP) framework of Hsu, Kitching, Bacchus, and McIlraith (2007) and considering general CSPs, LeBras, Zanarini, and Pesant (2009) obtain more accurate approximations of marginals inside constraints by exploiting consistency techniques. They probe for each variable-value pair whose bias we wish to estimate (i.e. making this assignment and reaching some level of consistency) and then take the product over the other variables of the sum of biases over their reduced domain. Two variants are evaluated: with (*global*) or without (*local*) propagation between constraints during the probing process. Branching as well on the strongest belief, they achieve good results for the global variant despite the heavier computational burden of probing, possibly because at the same time they gain the additional domain filtering of singleton consistency. Kask, Dechter, and Gogate (2004) investigate Iterative Join-Graph Propagation, belonging to the BP family, to estimate the number of solutions extending a partial assignment for a CSP and use it as a value ordering heuristic. Montanari, Ricci-Tersenghi, and Semerjian (2007) consider a simple randomized branching algorithm based on BP and analyze its behaviour on random k-SAT instances. To ease their analysis they choose an unbound variable to branch on uniformly at random and choose the value to assign to it randomly according to its computed marginal distribution, but mention that choosing the variable with the most strongly biased marginal would probably be more effective.

Like most of these previous efforts we aim to branch on the strongest belief according to computed variable biases. But unlike them we compute more precise distributions locally by a deeper exploitation of constraint structure made possible by recent progress on the counting problem over the corresponding combinatorial structures (Pesant, 2017).

3.2 Message Passing in the Presence of Cycles

As indicated earlier Belief Propagation may not converge when the graphical model on which it is applied contains cycles, even though there are a few documented cases for which it still performs very well. *Generalized Belief Propagation* (Yedidia, Freeman, & Weiss, 2000) and *Iterative Join-Graph Propagation* (Dechter, Kask, & Mateescu, 2002) were proposed to improve the convergence of BP in networks with cycles by clustering nodes of the network in an effort to encompass (short) cycles. More generally in probabilistic graphical models the idea of clustering manifests itself in the definition and use of *higher-order potentials*. Despite the latter generally requiring time exponential in their size to compute, in some applications they may take a special form that allows efficient inference without even relying on message passing algorithms: for example in computer vision, Kohli, Ladicky, and Torr (2009) define Robust P^n Potts potentials on which they apply a gradient descent algorithm with very good results. Previously introduced EMBP is guaranteed to converge to a local maximum but not necessarily to the exact marginals (Hsu et al., 2007). Ibrahim, Pal, and Pesant (2017) propose a variational message-passing algorithm based on Expectation Maximization in an

effort to handle cycles, combined with some local consistency to better handle determinism in graphical models. In the context of deriving lower bounds on model counting for Boolean formulas, Kroc, Sabharwal, and Selman (2011) propose `BPCount` which repeatedly uses BP to identify "balanced" variables to branch on. They address the convergence issue by applying *message damping* which helps reduce oscillation and they avoid fatal mistakes of fixing a variable to an unsupported value due to potential inaccuracies by adding safety checks to ensure satisfiability. They also write that the previous works mentioned above typically are not fast enough to be used repeatedly for branching.

Our contribution bears some similarity to generalized BP in its aim to encapsulate cycles but goes about it differently by exploiting the large arity of the global constraints naturally present in the CP model instead of building clusters. Higher-order potentials can be viewed as global constraints and, like them, they will lead to efficient inference if they can exploit special structure, as examplified by the previously mentioned computer vision application whose potentials resemble a soft `allEqual` constraint. Using the special structure of each global constraint, we aim to be just as accurate and efficient.

4. Richer Propagation in MiniCP

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, Schaus, & Hentenryck, 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 focus here on our main changes to MiniCP and refer the interested reader to its documentation for a complete description of its architecture.

4.1 `engine.core` package

This package includes classes defining domains as reversible sparse sets and various other classes implementing the `IntVar`, `Constraint`, and `Solver` interfaces. In order to maintain the marginal distribution of a variable, we created a reversible *weighted* sparse set that adds the marginal of each value in the domain as a weight attribute.

Algorithm 1: `IntVar.receiveMessage`$(v, \mu_{c\to x}(v))$

input: value v, message $\mu_{c\to x}(v)$ from some constraint c to this variable
`marginal`$[v] \leftarrow$ `marginal`$[v] \times \mu_{c\to x}(v)$;

Algorithm 2: `IntVar.sendMessage`$(v, \theta^c(v))$

input: value v, its marginal $\theta^c(v)$ local to some constraint c
output: message $\mu_{x\to c}(v)$ about v from this variable to c
return `marginal`$[v]/\theta^c(v)$;

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

4.1.1 Interface IntVar

As its name suggests, this interface describes integer finite-domain variables. We added the `marginal` distribution of the variable over its domain and simple methods to receive and send messages (Algorithm 1 and 2). Recall that incoming messages are combined by multiplying them and that the message sent to a constraint excludes its corresponding factor (Equation 1).

Algorithm 3: `Constraint.receiveMessages()`

```
foreach variable x in the scope of the constraint do
    foreach value v in the domain of x do
        outsideBelief[x][v] ← x.sendMessage(v, localBelief[x][v]);
    normalize outsideBelief[x];
```

Algorithm 4: `Constraint.sendMessages()`

```
updateBelief();
foreach variable x in the scope of the constraint do
    normalize localBelief[x];
    foreach value v in the domain of x do
        if localBelief[x][v] = 0 then
            x.remove(v);
        else if localBelief[x][v] = 1 then
            x.assign(v);
        x.receiveMessage(v, localBelief[x][v]);
```

4.1.2 Interface Constraint

We extended the class implementing this interface with a `localBelief` and `outsideBelief`, corresponding respectively to the marginal distributions for the variables in the scope of the constraint according to its local set of solutions and to the marginal distributions of the same variables according to the other constraints. We also added methods to receive and send messages (Algorithm 3 and 4), and abstract method `updateBelief()` to be defined in each family of constraints (see Section 5). Note that `sendMessages()` subsumes the domain filtering traditionally taking place in `propagate()` by acting on domains whenever an extreme belief (i.e. 0 or 1) is encountered. Therefore whenever the weighted counting performed in `updateBelief()` for some constraint is not exact, we take care not to return such extreme beliefs so that inference remains sound.

4.1.3 Interface Solver

This interface has method `fixPoint()` that propagates unsupported variable/value pairs from constraints until no more changes occur in domains. We add method `beliefPropa()` that drives the richer propagation we propose (Algorithm 5). Before initiating the BP iterations we reset to 1 all marginals and local beliefs. Note that in contrast to standard

Algorithm 5: `Solver.beliefPropa()`

```
foreach variable x do
    reset x.marginal;
foreach constraint c do
    reset c.localBelief;
repeat
    foreach constraint c do
        c.receiveMessages();
    foreach variable x do
        reset x.marginal;
    foreach constraint c do
        c.sendMessages();
    foreach variable x do
        normalize x.marginal;
until stopping condition;
```

(support) propagation, termination is not a given here since the propagation process is not limited by the number of variable/value pairs we can remove and it may not even converge. For the moment we leave the stopping condition unspecified — we will come back to it in Section 6. Message passing is synchronized in two phases: in the first phase constraints receive messages from the variables; in the second phase marginals for the variables are reset, constraints send messages to the variables, and marginals are normalized.

4.2 `engine.constraints` package

This package contains the classes for the constraints currently implemented in MiniCP. Each constraint requires a dedicated implementation of method `updateBelief()` that adjusts its `localBelief` by taking into account the current `outsideBelief`. In principe we can achieve this simply by enumerating the current solutions to a constraint (as in Equation 2) but in general it is too time-consuming. The next section describes more efficient ways of doing this for a few common constraints by exploiting their combinatorial structure.

5. Weighted Model Counting Inside Constraints

Many exact and approximate algorithms have been proposed for the difficult problem of Model Counting (Gomes, Sabharwal, & Selman, 2009), which has several applications, notably for probabilistic inference (e.g. Sang, Beame, & Kautz, 2005; Chavira & Darwiche, 2008). Recent algorithmic focus has been on fully polynomial randomized approximation schemes (FPRAS) using Monte Carlo or hashing-based techniques (see e.g. (Meel, Shrotri, & Vardi, 2018) for a recent exposition). More broadly, Weighted Counting is also of interest in complexity theory, quantum computation, and stochastic combinatorial optimization (de Campos, Stamoulis, & Weyland, 2017). Our interest here lies not in Weighted Model Counting for general models but instead for specific combinatorial structures encapsulated in constraints. Pesant (2017) discusses a few design patterns to perform (un-

weighted) model counting: adding some information to an existing compact representation of the solution set, sampling interleaved with constraint propagation, using known lower/upper bounds, or transforming finite-domain variables into discrete random variables under a uniform distribution. They can serve as a source of inspiration for weighted model counting. Recall that unless weighted counting is exact we refrain from fixing local beliefs to extreme values 0 or 1, since that would trigger value removal or variable fixing respectively.

Algorithm 6: `NotEqual.updateBelief()`

foreach *value* v *in the domain of* x **do**
  if $v-k$ *is in the domain of* y **then**
    $\texttt{localBelief}[x][v] \leftarrow 1 - \texttt{outsideBelief}[y][v-k]$;
  else
    $\texttt{localBelief}[x][v] \leftarrow 1$;
foreach *value* v *in the domain of* y **do**
  if $v+k$ *is in the domain of* x **then**
    $\texttt{localBelief}[y][v] \leftarrow 1 - \texttt{outsideBelief}[x][v+k]$;
  else
    $\texttt{localBelief}[y][v] \leftarrow 1$;

5.1 Binary Disequality

We start with a simple binary disequality constraint $x \neq y + k$, for some integer constant k, to illustrate how marginals are taken into account. For a given value v in the domain of x (respectively y), any value for y (resp. x) that is different from $v-k$ (resp. $v+k$) is a support and we could simply sum the corresponding marginals over such values. But it is sufficient and more efficient to subtract from 1 the marginal of $v-k$ for y (resp. $v+k$ for x; see Algorithm 6). Keep in mind that these computed local marginals are normalized afterwards (see Algorithm 4).

5.2 Alldifferent

For the `alldifferent` constraint, Hsu et al. (2007) essentially apply Algorithm 6 to the decomposition into disequality constraints and thus only approximate the distribution. LeBras et al. (2009) propose more accurate approximations that are not restricted to `alldifferent` by first applying some level of consistency, either locally to individual constraints or globally, but then make the simplifying assumption that every combination of values over these reduced domains constitutes a support.

As is the case for domain filtering, when the disequality relationship is extended to more than two variables then a more global view pays off for (weighted) counting. 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,

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

where $\mathcal{P}$ denotes the set of all permutations of $\{1, 2, \ldots, n\}$. 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, Sinclair, & Vigoda, 2001). We will cast our weighted counting problem as that of computing the permanent of a nonnegative matrix built from the marginals.

Algorithm 7: `AllDifferent.updateBelief()`

```
foreach unbound variable x_i in the scope of the constraint do
    foreach unassigned value j in the domain of x_i do
        a_ij ← outsideBelief[x_i][j];
add enough rows of 1's to A to make it square;
if A.order − 1 ≤ τ then
    foreach unbound variable x_i in the scope of the constraint do
        foreach unassigned value j in the domain of x_i do
            localBelief[x_i][j] ← permanentExact(i, j, A);
else
    foreach unbound variable x_i in the scope of the constraint do
        foreach unassigned value j in the domain of x_i do
            localBelief[x_i][j] ← permanentU3(i, j, A);
```

Algorithm 8: `AllDifferent.permanentU3(`i, j, A`)`

```
input: row and column indices i and j, nonnegative square matrix A
output: upper bound on the permanent of A without row i and column j
U ← 1;
for k ← 1 to A.order do
    if k ≠ i then
        sum ← 0;
        max ← 0;
        for ℓ ← 1 to A.order do
            if ℓ ≠ j then
                sum ← sum + a_kℓ;
                if a_kℓ > max then
                    max ← a_kℓ;
        if max = 0 then
            return 0;
        z ← sum/max;
        U ← U × max × (γ(⌊z⌋) + (z − ⌊z⌋) × (γ(⌈z⌉) − γ(⌊z⌋)));
return U;
```

Algorithm 7 describes our weighted counting for `alldifferent`. We first construct matrix A by setting a_{ij} to the outside belief about variable x_i taking value j. Note that we

only consider unbound variables and unassigned values in an effort to keep matrix A small. If there are more values than variables, we add fake variables (rows of A) whose belief is the same for every value. 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 threshold τ and use a fast upper bound otherwise (bound U^3 from Soules, 2003). Algorithm 8 describes the computation of bound U^3, where $\gamma(m) = \sqrt[m]{m!}$ and is precomputed.

5.3 Sum

Algorithm 9: `Sum.Sum()`

```
⋮
fwd ← 0;
bwd ← 0;
for i ← 1 to n do
    bwd ← bwd − min{v : v ∈ D(x_i)};
j_max ← 0;
for i ← 1 to n do
    j_max ← max(j_max, min(fwd, bwd));
    fwd ← fwd + max{v : v ∈ D(x_i)};
    bwd ← bwd + min{v : v ∈ D(x_i)};
fwd ← 0;
bwd ← 0;
for i ← 1 to n do
    bwd ← bwd − max{v : v ∈ D(x_i)};
j_min ← 0;
for i ← 1 to n do
    j_min ← min(j_min, max(fwd, bwd));
    fwd ← fwd + min{v : v ∈ D(x_i)};
    bwd ← bwd + max{v : v ∈ D(x_i)};
⋮
```

Next we present weighted counting for `sum` constraints. As previously proposed to achieve domain consistency and to compute solution densities for `knapsack` constraints (Pesant & 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*

Algorithm 10: `Sum.updateBelief()`

initialize π_{in} to 0;
$\pi_{in}[1][0] \leftarrow 1$;
for $i \leftarrow 1$ **to** $n-1$ **do**
  foreach *value* $v \in D(x_i)$ **do**
    for $j \leftarrow \mathtt{j}_{\min} - \min(0, v)$ **to** $\mathtt{j}_{\max} - \max(0, v)$ **do**
      if $\pi_{in}[i][j] > 0$ **then**
        $\pi_{in}[i+1][j+v] \leftarrow \pi_{in}[i+1][j+v] + \pi_{in}[i][j] \times \mathtt{outsideBelief}[x_i][v]$;
initialize π_{out} to 0;
$\pi_{out}[n][0] \leftarrow 1$;
for $i \leftarrow n$ **to** 2 **do**
  foreach *value* $v \in D(x_i)$ **do**
    $b \leftarrow 0$;
    for $j \leftarrow \mathtt{j}_{\min} - \min(0, v)$ **to** $\mathtt{j}_{\max} - \max(0, v)$ **do**
      if $\pi_{out}[i][j+v] > 0$ **then**
        $\pi_{out}[i-1][j] \leftarrow \pi_{out}[i-1][j] + \pi_{out}[i][j+v] \times \mathtt{outsideBelief}[x_i][v]$;
        $b \leftarrow b + \pi_{in}[i][j] \times \pi_{out}[i][j+v]$;
    $\mathtt{localBelief}[x_i][v] \leftarrow b$;
foreach *value* $v \in D(x_1)$ **do**
  $\mathtt{localBelief}[x_1][v] \leftarrow \pi_{out}[1][v]$;

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. Let the variables in the scope of our constraint be identified as $x_1, x_2, \ldots, x_n$. We first determine the range of prefix and suffix sums of values that can satisfy the constraint, $[j_{\min}, \ldots, j_{\max}]$, in order to define the size of our data structures in the constructor (Algorithm 9). Weighted sums π_{in} and π_{out} are two-dimensional arrays ranging over $[1, \ldots, n] \times [j_{\min}, \ldots, j_{\max}]$. Algorithm 10 computes them through standard forward and backward passes and then uses these weighted sums to compute the local beliefs for each variable-value pair.

Note that the previous algorithms can easily be adapted in order to provide the same for general linear constraints and `regular` constraints since a similar layered graph has been proposed to achieve domain consistency and to count solutions (Pesant et al., 2012). It could apply as well to other constraints such as `mdd` (Cheng & Yap, 2008).

6. Empirical Evaluation

In this section we attempt to answer several research questions empirically. All experiments were run on a cluster of dual core AMD Opteron 275 @ 2.2 GHz processors running Java SE 11 on Linux CentOS 7.6 and our prototype was built on top of MiniCP 1.0. We use four combinatorial problems that can be modelled using the constraints from the previous section: Partial Latin Square, Cost-Constrained Rostering, Partial Magic Square, and Primes. With the exception of Sections 6.3 and 6.7 we perform exact weighted counting even for `alldifferent` by using instances whose size situates us below any reasonable threshold τ in

order to evaluate our proposal given perfect counting information. We now describe these problems.

Partial Latin Square Problem An $n \times n$ grid is partially filled with integers from $\{1, 2, \ldots, n\}$ and must be completed such that each integer appears exactly once per row and column (problem 67 of the CSPLib Jefferson, Miguel, Hnich, Walsh, & Gent, 1999). This can be modeled using a matrix of integer variables and an `alldifferent` constraint for each row and each column. We randomly generated two sets of ten instances of size $n = 10$ with 50% (pls-10-50) and 55% (pls-10-55) free cells respectively, using the instance generator of (Gomes & Shmoys, 2002) with the "balanced" setting. For Section 6.7 we use the challenging set of forty larger ($n = 30$) instances *LatinSquare/m1/gp* from the XCSP3 collection of benchmark instances[2].

Cost-Constrained Rostering Problem In this simplified rostering problem m employees have to accomplish a set of tasks in a n-day schedule (Pesant & Quimper, 2008). Each daily task has a distinct duration and must be performed by exactly one employee. Moreover, there is an hourly cost for making someone work, which varies both across employees and days. For each employee, the total cost must be equal to a given value. Finally, each instance specifies some days in which a given employee cannot perform a given task (forbidden shifts). This can be modeled using a matrix of integer variables, an `alldifferent` constraint on each day, and a weighted `sum` constraint for each employee. We randomly generated ten instances with $m = 4$, $n = 10$, and 10 forbidden shifts (roster-4-10).

Partial Magic Square Problem An $n \times n$ grid is partially filled with integers from $\{1, 2, \ldots, n^2\}$ and must be completed such that each integer appears exactly once and such that the sum of the entries in each row, column, and main diagonal is the same (problem 19 of the CSPLib). This can be modeled using a matrix of integer variables, an `alldifferent` constraint over all variables, and a `sum` constraint for each row, column, and main diagonal. We use the set of forty 9×9 instances *MagicSquare/m1/gp* from XCSP3.

Primes Problem We are asked to solve a sparse system of linear equalities over a given finite subset of the integers. We use the set of thirty-two instances *Primes/m1/p10* from XCSP3, featuring 100 variables taking their value from subset $\{2, 3, \ldots, 29\}$ and between twenty and eighty linear equalities (i.e. weighted `sum` constraints) each over a relatively small subset of these variables.

6.1 Q_1: How well does it approximate marginal distributions over the solution set?

To try to answer this question we track the evolution of the marginals computed by our belief propagation and compare them to the true marginals. For each size-10 Partial Latin Square instance and each Cost-Constrained Rostering instance we compute the set of solutions a priori (these are fairly small instances), thereby allowing us to know the true frequency distributions over individual variables. We will report the Kullback-Leibler divergence of the computed marginal distributions from the true distributions for these instances.

2. http://www.xcsp.org

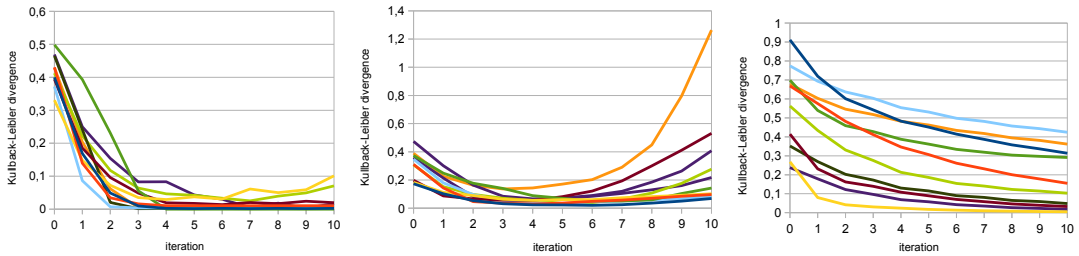

Figure 1: Average KL-divergence of computed variable marginals with respect to the true marginals as we iterate message passing between constraints and variables for sets of instances pls-10-50 (left), pls-10-55 (center), and roster-4-10 (right). Each line corresponds to an instance.

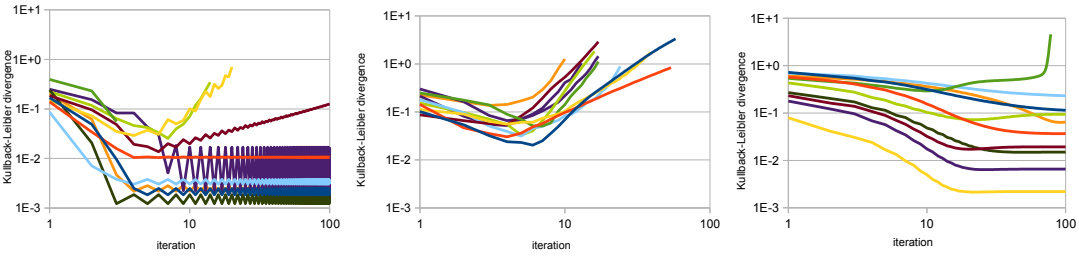

Figure 2: A presentation of the same results as in the previous figure but using a logarithmic scale and up to 100 iterations.

The *Kullback-Leibler divergence* is a measure of dissimilarity between two probability distributions. In the case of two discrete distributions P and Q it is expressed as

$$D_{KL}(P \,\|\, Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

It is asymmetric with P typically representing the true probability distribution and it measures the amount of information lost if we use Q to approximate P.

Figure 1 reports the behaviour of Algorithm 5 on the test instances. Iteration 0 corresponds to the initial domains of the variables with an implicit uniform distribution and Iteration 1, to the simple product of the marginals initially returned by the constraints. Observe that within just a few iterations the computed marginals become quite close to the true ones, though convergence is slower for about half of the roster-4-10 instances. Note also that the divergence eventually increases for some of the pls-10-$\star$ instances.

The first set of instances (left) generally has fewer solutions. The instance shown in green has a single solution, which is identified after four iterations. The two instances in yellow and light green, which start to diverge, have the largest number of solutions (respectively 20 and 9) in that set. That apparent divergence is actually a convergence to a subset of the solutions — for example for the light green instance one of the 9 solutions is dropped

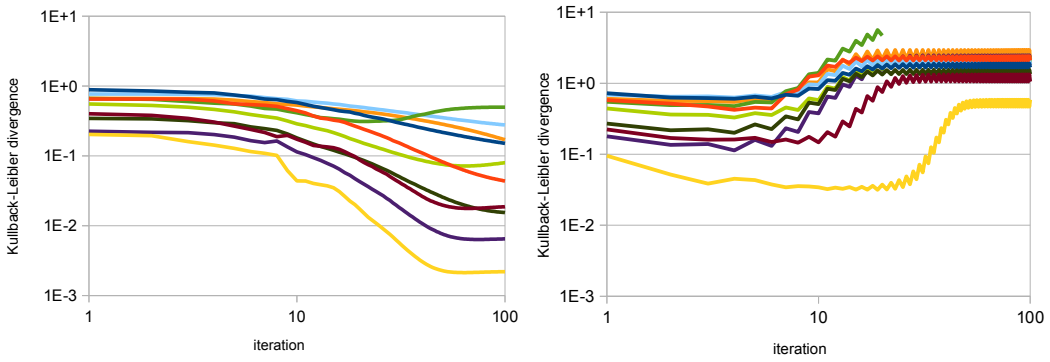

Figure 3: Average KL-divergence for the roster-4-10 instances using a decomposition of the `sum` constraints into ternary linear equalities (left) or a decomposition of the `alldifferent` constraints into binary disequality constraints (right).

at iteration 15, caused by the removal of a valid variable-value pair which translates to a corresponding zero frequency in Q that drives the KL-divergence to infinity. This appears more clearly at Figure 2 which presents the same results but over 100 iterations and using logarithmic scale. We also observe an oscillation pattern for several instances: for example on the purple instance, which has 2 solutions, it quickly identifies the 96 cells that are fixed in both solutions but then cycles through four combinations of marginal distributions for the remaining four cells (variables).

The second set of instances (middle) has more solutions (from 4 to 889). Here each instance eventually converges to a subset of the solutions (and hence yields an infinite KL-divergence, shown as an interrupted line) but in the first few iterations it approaches the true marginals, albeit not as closely as for the first set.

The set of rostering instances (right) exhibits slower convergence and only the green instance eventually drops some solutions. The lower KL-divergence seem to concide with the instances having the larger number of solutions. In particular the yellow instance (with 1471 solutions, the largest number) reaches a divergence of 0.002: as an indication the true marginal for the first variable is $\langle 0.2345, 0.3188, 0.3379, 0.1088\rangle$ whereas the final computed one is $\langle 0.2347, 0.3325, 0.3244, 0.1083\rangle$.

So a few iterations in Algorithm 5 are often sufficient here to reduce significantly the KL-divergence to the true marginal distribution. The fact that we may eventually drop some solutions is an issue if we wish to enumerate all solutions or to perform near-uniform sampling over the whole solution set, but not necessarily so if we are simply solving the instance, i.e. looking for any one solution. We come back to this in Section 7.

6.2 Q_2: Does it converge faster or better than classic BP with small factors?

With respect to classic belief propagation the advantage here should be the large factors directly obtainable from the CP model, for which we have weighted counting algorithms and which should improve convergence by reducing the number of constraints and lessening the occurrence of loops in the constraint network. Figure 3 shows what happens on the

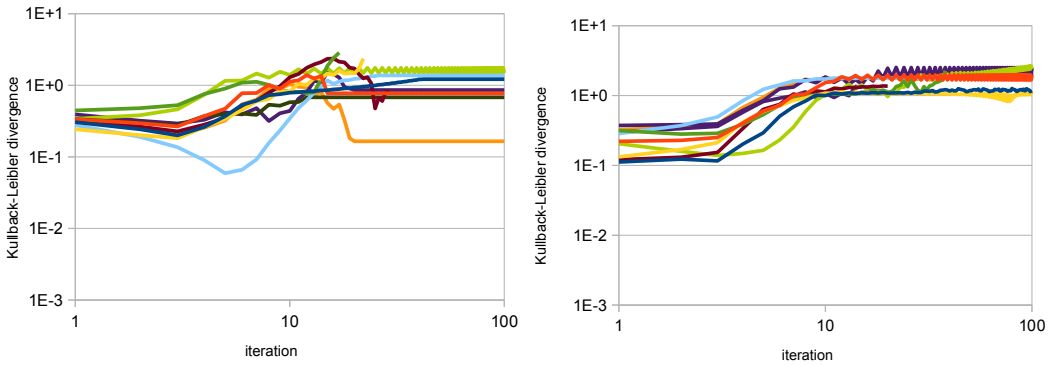

Figure 4: Average KL-divergence for the pls-10-50 (left) and pls-10-55 (right) instances using a decomposition of the `alldifferent` constraints into binary disequality constraints.

rostering instances if we replace each `sum` (left) or `alldifferent` (right) constraint by its decomposition into ternary or binary constraints respectively. Comparing the plot on the left to the one at Figure 2 right, we note a similar trend but with slower convergence. In contrast the plot on the right (`alldifferent` decomposition) is quite different: there is not much improvement over the initial KL-divergence and, even worse, it typically stabilizes to a higher value. In the former case the decomposition increases the number of constraints while decreasing their arity but these smaller constraints form a tree. In the latter, it also adds many short cycles to the constraint network and this seems to degrade belief propagation, as expected. Figure 4 confirms this behaviour on the Latin Square instances with the `alldifferent` decomposition. Worse, the green instance on the left (top line at iteration 1) — previously converging correctly to its single solution in Figure 2 — eventually becomes infeasible. The advantage appears clearly here even though the factors are not that large: each `alldifferent` constraint has arity 4 or 5 (ignoring bound variables), whose decomposition features 6 or 10 binary constraints; each `sum` constraint has an arity around 8 and therefore decomposes into about 7 ternary constraints.

6.3 Q_3: How does the accuracy of weighted counting affect the accuracy of the computed marginals?

In Section 5.2 we proposed an approximate weighted counting algorithm for `alldifferent`, to be used whenever its arity is above a given threshold τ, since the problem is intractable in general. We set $\tau = 1$, thereby forcing the approximation, and report the result on our test instances at Figure 5. Compared to Figure 1 there is a noticeable impact, with less of a reduction of the KL-divergence on most instances followed by some stagnation. The effect is less noticeable on the rostering instances, for which the `alldifferent` constraint plays a smaller role. Note that similar upper bounds on the permanent of 0-1 matrices were found to be an effective trade-off between computational effort and accuracy for counting-based search heuristics (Zanarini & Pesant, 2010)— it appears that accuracy is more critical here in order to converge to the true marginals.

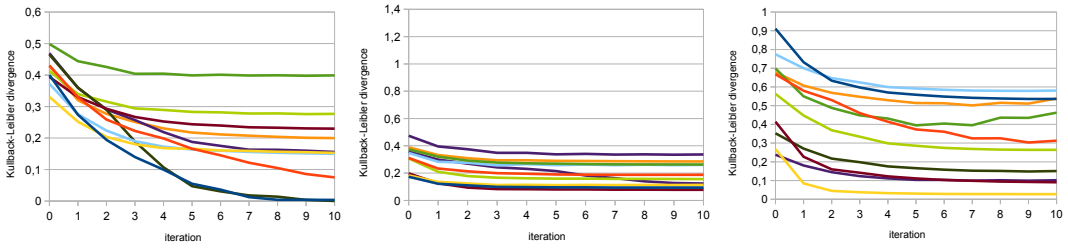

Figure 5: Effect of approximate weighted counting (using an upper bound for the permanent) on pls-10-50 (left), pls-10-55 (center), and roster-4-10 (right).

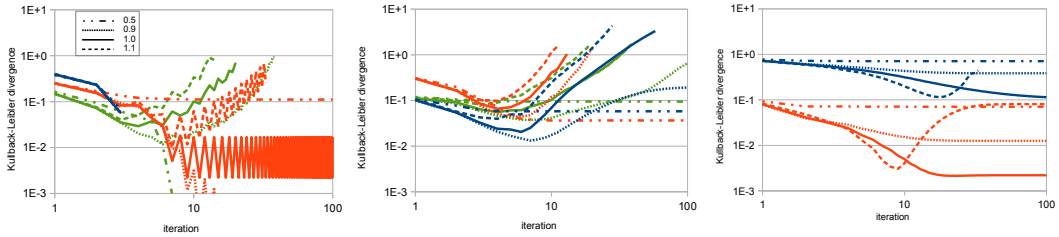

Figure 6: Effect of message damping on a few pls-10-50 (left), pls-10-55 (center), and roster-4-10 (right) instances for four values of parameter κ.

6.4 Q_4: Should we use a damping exponent on the product of constraint-to-variable messages?

As mentioned in Section 3 some authors found that message damping, which sets some parameter $0 \leq \kappa < 1$ as an exponent to the product of messages in Equation 1, was beneficial to convergence. We investigate that possibility by studying the effect of setting $\kappa \in \{0.5, 0.9, 1.0, 1.1\}$ for some representative instances. Because each variable in our test problems appears in exactly two constraints, parameter value 0.5 corresponds to taking the geometric mean; value 1.0 means no damping and the other two values are mild corrections on either side of the latter.

Figure 6 shows how the choice of κ (identified by line style) affects convergence on a few instances (grouped by colour) of varying number of solutions. On the left: the blue instance has a single solution and is not really affected by the choice of κ; the red and green instances have respectively two and twenty solutions and are affected by κ, though not so much in early iterations. On the middle and right plots a clearer trend emerges across instances of the same set: stagnation at $\kappa = 0.5$ and some discriminating effect of the other values but really only after several iterations. Overall there is no strong indication that a value other than 1.0 is beneficial, especially in the early iterations, so we do not apply message damping and thus keep the usual message passing equations.

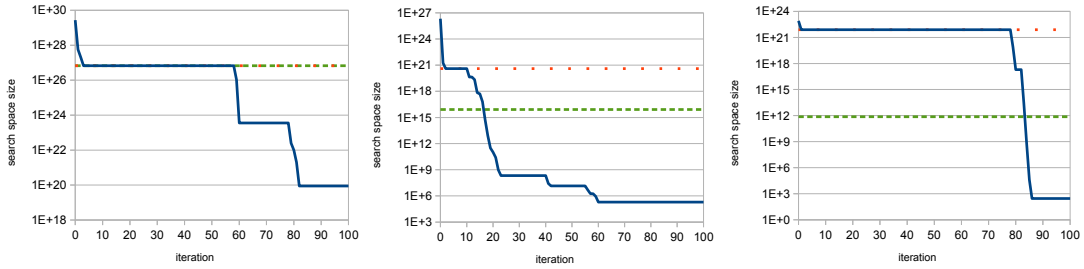

Figure 7: Search space reduction at root node for instances pls-10-55.1 (left), pls-10-55.7 (center), and roster-4-10.4 (right). The solid blue curve shows the progression for our belief propagation. The dotted red line gives the result of standard propagation achieving a fixed point with domain consistency being enforced on individual constraints. The dashed green line corresponds to every variable's domain being shrunk to values appearing in at least one solution (i.e. to the supports of the true marginals, or *global* domain consistency).

6.5 Q_5: Can it enforce a stronger level of consistency?

Whenever we perform exact weighted counting for a constraint it is clear that we achieve domain consistency for it and after some iterations of belief propagation we should reach standard propagation's fixed point (in terms of the contents of the domains). But since we are exchanging finer-grained information, could we reach a stronger level of consistency? As an indication, in an effort to unify computational approaches to inference tasks over commutative semirings Werner (2015) introduced the concept of *marginal consistency* — achieved by making to coincide the marginal distributions of variables shared by pairs of constraints — that is said to generalize some standard forms of local consistency in CP. Figure 7 reports search space size (taken as the product of domain sizes) for some representative instances. Sometimes the situation is as depicted on the left: standard propagation only leaves globally supported values (our instances are rather small) i.e. the red and green lines coincide and belief propagation reaches the same domains after just a few iterations but eventually shrinks some domains further by removing some little-supported values and thus converging to a *subset* of the solutions. Keep in mind that while search space size may plateau, the computed marginals still evolve. Other times standard propagation does not filter out all globally unsupported values (as is generally the case for more challenging instances) and so there is a gap between the red and green lines (e.g. Fig. 7 middle). Interestingly belief propagation (blue curve) removes more unsupported values (Iterations 11-16) before crossing the green line (which coincides with dropping some little-supported values). In such a case we do observe stronger consistency. Finally on the right, where there is a significant gap between the red and green lines, we immediately reach standard propagation's fixed point, stay there for many iterations but ultimately (at Iteration 79) start removing other values, though in this case these are supported. So here there is no transitional stronger consistency between the usual fixed point and the loss of some solutions.

Can we take advantage of stronger consistency without losing solutions? The varied behaviour exemplified at Figure 7 suggests that there is no simple answer to that question and we believe this warrants further investigation.

6.6 Q_6: What is a good stopping condition?

In general this iterative computation of the marginals may not reach a fixed point and so we require a stopping condition. Since in practice we do not know the true marginals it cannot be the point at which the average KL-divergence from them starts to increase. We looked at the KL-divergence of computed marginals between successive iterations but did not observe any correlation with the divergence to the true marginals. Judging from the empirical results presented so far, a few iterations generally suffice to get closer to the true marginals without starting to drop solutions. Therefore we propose to use simply some low maximum number of iterations in order for inference to remain sound.

6.7 Q_7: Does it help solve problems?

In this section we investigate the efficacy of CP-based belief propagation search guidance to find a solution to constraint satisfaction problems, measured as the number of fails and the computation time. We consider two parameters: the number of BP iterations and whether or not standard support propagation is applied before BP. 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.

In order to make the problem more challenging for these experiments we use the larger Partial Latin Square instances, the Magic Square instances, and the Primes instances. On the constraints we apply commonly-used consistency levels: domain consistency on `alldifferent` and bounds consistency on `sum`. We set $\tau = 6$, triggering approximate counting for `alldifferent` until enough variables are fixed to count exactly. We stop an individual run after one hour of computation time.

6.7.1 Primes

The CP model for the Primes instances features 100 variables whose domain ranges from 2 to 29 and many `sum` constraints on 2 to 10 variables each, for which weighted counting is exact. Figure 8 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, though the 5-iteration configuration ultimately performs a little better than the 10-iteration one and solves two more instances

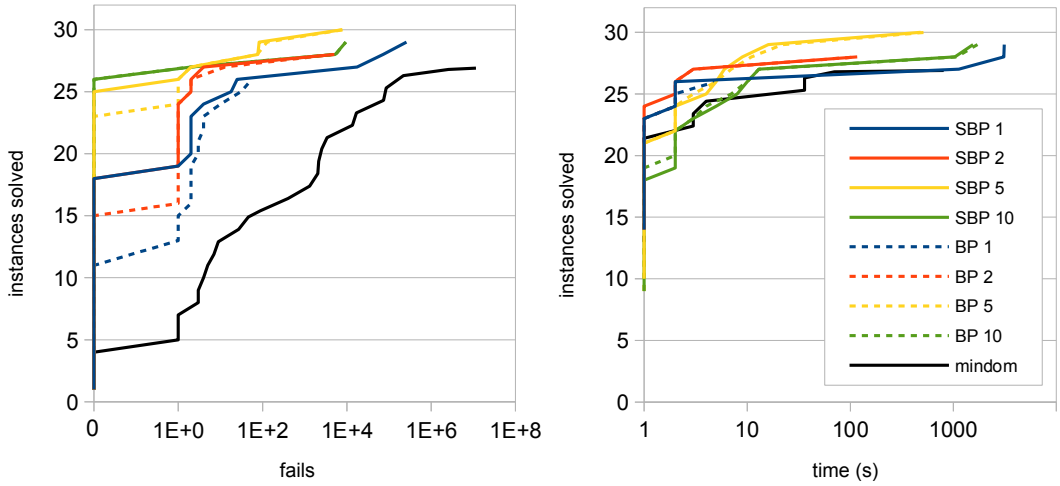

Figure 8: Number of instances solved with respect to the number of fails (left) and computation time (right) for the Primes instances.

within the time limit. Remarkably the max-strength branching heuristic solves most instances without any fails, culminating at 10 iterations with 26 out of 32 instances being solved backtrack-free. Because in this case weighted counting is exact and therefore subsumes bounds (or even domain) consistency, it is interesting to consider not applying support propagation. Comparing each SBP curve to its BP counterpart we see that they are not the same but that their difference lessens as the number of iterations increases, with that difference disappearing at 10 iterations. This can be explained by the fact that we are using a fixed number of iterations which may at times be insufficient to reach the usual propagation fixpoint. 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.

6.7.2 Magic Square

The CP model for these 9×9 partially-filled instances features 81 variables whose domain ranges from 1 to 81, but with 10 (respectively 50) of them being already fixed. There are twenty `sum` constraints, each over 8 (resp. 4) variables on average, for which weighted counting is exact, and one `alldifferent` constraint over at least (due to possible repeats in randomly-filled squares) 71 (resp. 31) variables for which weighted counting is approximate until the number of unbound variables falls below τ. Figure 9 plots the number of instances solved against the number of failed search-tree nodes reached during search (left) and the computation time (right).

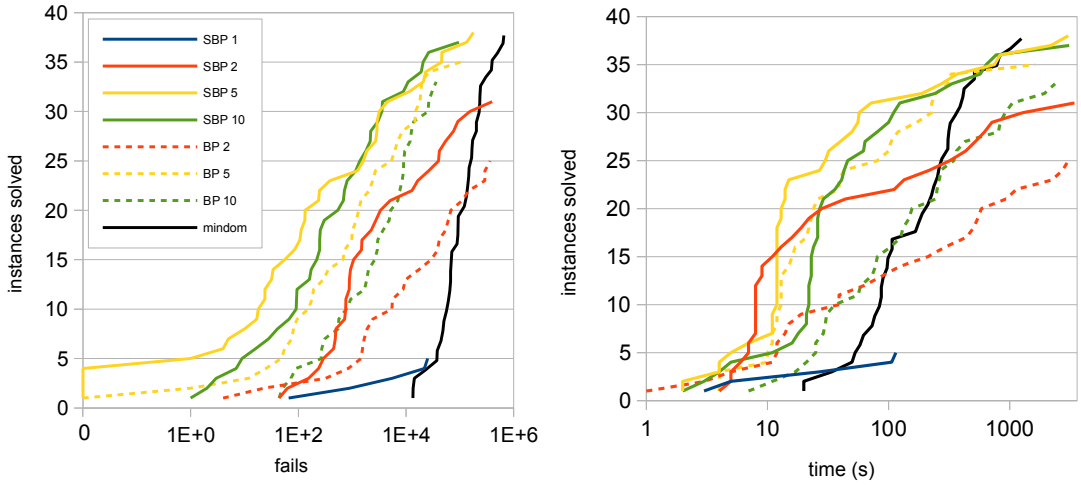

Figure 9: Number of instances solved with respect to the number of fails (left) and computation time (right) for the Magic Square instances.

These are harder to solve even though they typically admit many solutions: 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. A single iteration of BP — that is, sending initial local beliefs from constraints to variables but without receiving back the outside beliefs from which local beliefs are adjusted — performs poorly but 2 iterations already make a big difference, with 5 and 10 iterations guiding best and requiring a few orders of magnitude fewer fails than mindom. Note that SBP 5 solves 4 instances without any backtracks. 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.

6.7.3 Latin Square

The CP model here features 900 variables whose domain ranges from 1 to 30, but with 374/375 of them being already fixed. There are sixty `alldifferent` constraints over about 17 unbound variables each for which weighted counting is approximate until the number of unbound variables falls below τ. Figure 10 plots the number of instances solved against the number of failed search-tree nodes reached during search (left) and against the computation time (right).

These instances are even harder to solve: 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

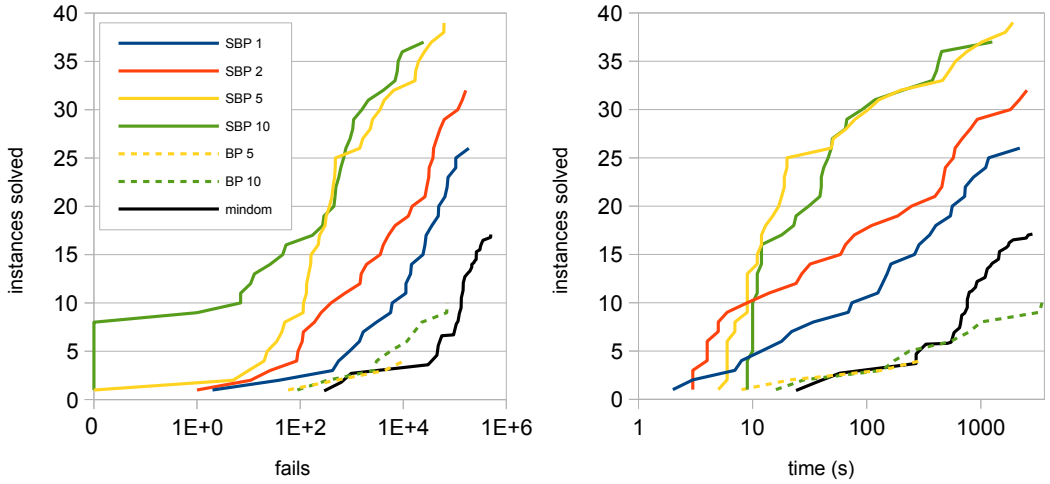

Figure 10: Number of instances solved with respect to the number of fails (left) and computation time (right) for the Latin Square instances.

will occur, not applying support propagation (the BP curves) is not advisable. The plot on the right confirms the clear superiority of max-strength over mindom for these instances in terms of computation time as well. 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.

Based on these experiments, the recommended configuration with the max-strength branching heuristic is SBP 5: applying support propagation before BP adds a negligible computation cost while significantly boosting search when weighted counting is not exact, and five iterations seem sufficient to benefit from the exchange of messages between variables and constraints.

7. Closing Discussion

CP features high-level models and solves problems through a combination of inference and search. We proposed to extend recent work on counting-based search to its complementary approach that is inference, taking advantage again of the large combinatorial substructures explicit in the models. We hope this paper laid down the foundations of a more powerful propagation of information within CP models. It also raises some issues deserving further investigation which we discuss below.

Soundness. In contrast to standard (support) propagation this iterated belief propagation could eventually remove supported values and thus exclude some solutions, as an artifact of the limitations of floating-point representation for real numbers when some marginals become very small. This was sometimes observed in Section 6.1 after many iterations but it no longer occurred once we set a small limit on the number of iterations for our investigation of search guidance in Section 6.7. Ultimately a safeguard could be put in place in the

form of a tiny threshold beyond which marginals would not be allowed to fall. But there is also an opportunity here for both inference and search. The possibility of reaching a stronger level of consistency through vanishing tiny marginals was identified in Section 6.5. If we are satisfied with finding any solution, even removing little-supported values could be acceptable as long as some of the solutions (or ultimately at least one) remain. Note that if we don't reinitialize marginals and local beliefs at each search-tree node before performing BP (see Algorithm 5), going down the left branch accumulates BP iterations and will exhibit a behaviour similar to what was observed with a large number of iterations, albeit interleaved with "branching decisions" that fix variables. This could be used as a strongly-informed greedy heuristic to quickly find a solution, but at the risk of not finding any even though there are some.

Efficiency. We showed in Section 6.7 that our proposal can significantly reduce the time required to solve combinatorial problems. But of course there is still room for improvement. The CP-based belief propagation process described in Algorithm 5 is reminiscent of AC1 and potentially wasteful since it reconsiders every constraint and every variable at each iteration. Following AC3 and subsequent constraint propagation algorithms, can we make it more efficient by reconsidering only the subset of constraints concerned by the latest modifications? Because in the present case the information exchanged is much finer — an evolving real-valued marginal for each element in a domain — every constraint may have variables whose marginals have been modified and so it will need to be reconsidered. Therefore improving that basic belief propagation algorithm may not be as simple as in the case of standard propagation. In order to avoid at least the systematic recomputation of weighted counting, recent work on lowering computation time in the unweighted case could serve as a source of inspiration here (Gagnon & Pesant, 2018). Given that exact weighted counting can be time consuming on high-arity constraints, those of them that admit an acyclic decomposition could provide an interesting trade-off (Section 6.2).

This first implementation was built from MiniCP because of its simplicity but it lacks some of the optimizations found in state-of-the-art solvers. It would be interesting to implement this new propagation framework in some of the latter. We also need to equip more constraints with weighted model counting. As mentioned in Section 5 some of the existing (unweighted) counting algorithms only require a simple adaptation and there are design patterns to try for the others.

Extensions. The proposed framework solves constraint *satisfaction* problems but combinatorial *optimization* problems abound. A natural next step would be to generalize the framework to constraint optimization problems, inspired by the work described in (Pesant, 2016).

We borrowed ideas from probabilistic inference to apply them to deterministic constraint networks. But a more general framework combining deterministic and probabilistic information makes it possible to model even more challenging problems — for example Mateescu and Dechter (2008) proposed a mixed deterministic and probabilistic network in which a constraint network (for deterministic inference) and a belief network (for probabilistic causal inference) co-exist. In our framework we could incorporate probabilistic non-causal relationships inside constraints to replace the implicit uniform distributions used to compute the initial local beliefs.

Acknowledgements

The author wishes to thank Arthur Godet for an initial implementation of this work, Alessandro Zanarini for early discussions, and the anonymous referees for their constructive criticism that helped improve this work. Financial support for this research was provided by NSERC Discovery Grant 218028/2017.

References

Brockbank, S., Pesant, G., & Rousseau, L.-M. (2013). Counting Spanning Trees to Guide Search in Constrained Spanning Tree Problems. In Schulte, C. (Ed.), *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, Vol. 8124 of *Lecture Notes in Computer Science*, pp. 175–183. Springer.

Chavira, M., & Darwiche, A. (2008). On Probabilistic Inference by Weighted Model Counting. *Artif. Intell.*, *172*(6-7), 772–799.

Cheng, K. C. K., & Yap, R. H. C. (2008). Maintaining Generalized Arc Consistency on Ad Hoc r-Ary Constraints. In Stuckey, P. J. (Ed.), *Principles and Practice of Constraint Programming, 14th International Conference, CP 2008, Sydney, Australia, September 14-18, 2008. Proceedings*, Vol. 5202 of *Lecture Notes in Computer Science*, pp. 509–523. Springer.

de Campos, C. P., Stamoulis, G., & Weyland, D. (2017). A Structured View on Weighted Counting with Relations to Counting, Quantum Computation and Applications. *CoRR*, *abs/1701.06386*.

Dechter, R., Bidyuk, B., Mateescu, R., & Rollon, E. (2010). On the Power of Belief Propagation: A Constraint Propagation Perspective. In *Festschrift book in honor of Judea Pearl*.

Dechter, R., Kask, K., & Mateescu, R. (2002). Iterative Join-Graph Propagation. In Darwiche, A., & Friedman, N. (Eds.), *UAI '02, Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence, University of Alberta, Edmonton, Alberta, Canada, August 1-4, 2002*, pp. 128–136. Morgan Kaufmann.

Delaite, A., & Pesant, G. (2017). Counting Weighted Spanning Trees to Solve Constrained Minimum Spanning Tree Problems. In Salvagnin, D., & Lombardi, M. (Eds.), *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, Vol. 10335 of *Lecture Notes in Computer Science*, pp. 176–184. Springer.

Gagnon, S., & Pesant, G. (2018). Accelerating Counting-Based Search. In van Hoeve, W. J. (Ed.), *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26-29, 2018, Proceedings*, Vol. 10848 of *Lecture Notes in Computer Science*, pp. 245–253. Springer.

Gomes, C., & Shmoys, D. (2002). Completing Quasigroups or Latin Squares: A Structured Graph Coloring Problem.. In *Proceedings of Computational Symposium on Graph Coloring and Generalizations, COLOR-02*, pp. 22–39.

Gomes, C. P., Sabharwal, A., & Selman, B. (2009). Model Counting. In Biere, A., Heule, M., van Maaren, H., & Walsh, T. (Eds.), *Handbook of Satisfiability*, Vol. 185 of *Frontiers in Artificial Intelligence and Applications*, pp. 633–654. IOS Press.

Horsch, M. C., & Havens, W. S. (2013). Probabilistic Arc Consistency: A Connection between Constraint Reasoning and Probabilistic Reasoning. *CoRR*, *abs/1301.3864*.

Hsu, E. I., Kitching, M., Bacchus, F., & McIlraith, S. A. (2007). Using Expectation Maximization to Find Likely Assignments for Solving CSP's. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pp. 224–230. AAAI Press.

Ibrahim, M. H., Pal, C. J., & Pesant, G. (2017). Improving probabilistic inference in graphical models with determinism and cycles. *Machine Learning*, *106*(1), 1–54.

Jefferson, C., Miguel, I., Hnich, B., Walsh, T., & Gent, I. P. (1999). CSPLib: A Problem Library for Constraints. `http://www.csplib.org`.

Jerrum, M., Sinclair, A., & Vigoda, E. (2001). 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, pp. 712–721, New York, NY, USA. ACM.

Kask, K., Dechter, R., & Gogate, V. (2004). Counting-Based Look-Ahead Schemes for Constraint Satisfaction. In Wallace, M. (Ed.), *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, Vol. 3258 of *Lecture Notes in Computer Science*, pp. 317–331. Springer.

Kohli, P., Ladicky, L., & Torr, P. H. S. (2009). Robust Higher Order Potentials for Enforcing Label Consistency. *International Journal of Computer Vision*, *82*(3), 302–324.

Kroc, L., Sabharwal, A., & Selman, B. (2011). Leveraging Belief Propagation, Backtrack Search, and Statistics for Model Counting. *Annals OR*, *184*(1), 209–231.

LeBras, R., Zanarini, A., & Pesant, G. (2009). Efficient Generic Search Heuristics within the EMBP Framework. In Gent, I. P. (Ed.), *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, Vol. 5732 of *Lecture Notes in Computer Science*, pp. 539–553. Springer.

Mateescu, R., & Dechter, R. (2008). Mixed Deterministic and Probabilistic Networks. *Ann. Math. Artif. Intell.*, *54*(1-3), 3–51.

Meel, K. S., Shrotri, A. A., & Vardi, M. (2018). Not all FPRASs are Equal: Demystifying FPRASs for DNF-Counting. *Constraints*.

Michel, L., Schaus, P., & Hentenryck, P. V. (2017). Mini-CP: A Minimalist Open-Source Solver to Teach Constraint Programming. `http://www.minicp.org`.

Montanari, A., Ricci-Tersenghi, F., & Semerjian, G. (2007). Solving Constraint Satisfaction Problems through Belief Propagation-Guided Decimation. *CoRR*, *abs/0709.1667*.

Pearl, J. (1982). Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach. In Waltz, D. L. (Ed.), *Proceedings of the National Conference on Artificial Intelligence. Pittsburgh, PA, August 18-20, 1982.*, pp. 133–136. AAAI Press.

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

Pesant, G. (2016). Counting-Based Search for Constraint Optimization Problems. In Schuurmans, D., & Wellman, M. P. (Eds.), *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pp. 3441–3448. AAAI Press.

Pesant, G. (2017). Getting More Out of the Exposed Structure in Constraint Programming Models of Combinatorial Problems. In Singh, S. P., & Markovitch, S. (Eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pp. 4846–4851. AAAI Press.

Pesant, G., & Quimper, C.-G. (2008). Counting Solutions of Knapsack Constraints. In Perron, L., & Trick, M. A. (Eds.), *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*, Vol. 5015 of *Lecture Notes in Computer Science*, pp. 203–217. Springer.

Pesant, G., Quimper, C.-G., & Zanarini, A. (2012). Counting-Based Search: Branching Heuristics for Constraint Satisfaction Problems. *J. Artif. Intell. Res.*, *43*, 173–210.

Sang, T., Beame, P., & Kautz, H. A. (2005). Performing Bayesian Inference by Weighted Model Counting. In Veloso, M. M., & Kambhampati, S. (Eds.), *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pp. 475–482. AAAI Press / The MIT Press.

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

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

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

Yedidia, J. S., Freeman, W. T., & Weiss, Y. (2000). Generalized Belief Propagation. In Leen, T. K., Dietterich, T. G., & Tresp, V. (Eds.), *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pp. 689–695. MIT Press.

Zanarini, A., & Pesant, G. (2010). More Robust Counting-Based Search Heuristics with Alldifferent Constraints. In Lodi, A., Milano, M., & Toth, P. (Eds.), *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18,*

2010. Proceedings, Vol. 6140 of *Lecture Notes in Computer Science*, pp. 354–368. Springer.