Efficient Generic Search Heuristics within the EMBP framework

Ronan Le Bras[1,2], Alessandro Zanarini[1,2], and Gilles Pesant[1,2]

[1] École Polytechnique de Montréal, Montreal, Canada
[2] CIRRELT, Université de Montréal, Montreal, Canada
{Ronan.LeBras,Alessandro.Zanarini,Gilles.Pesant}@cirrelt.ca

Abstract. Accurately estimating the distribution of solutions to a problem, should such solutions exist, provides efficient search heuristics. The purpose of this paper is to propose new ways of computing such estimates, with different degrees of accuracy and complexity. We build on the Expectation-Maximization Belief-Propagation (EMPB) framework proposed by Hsu et al. to solve Constraint Satisfaction Problems (CSPs). We propose two general approaches within the EMBP framework: we firstly derive update rules at the constraint level while enforcing domain consistency and then derive update rules globally, at the problem level. The contribution of this paper is two-fold: first, we derive new generic update rules suited to tackle any CSP; second, we propose an efficient EMBP-inspired approach, thereby improving this method and making it competitive with the state of the art. We evaluate these approaches experimentally and demonstrate their effectiveness.

Key words. Constraint Satisfaction; Search Heuristics; Probabilistic Reasoning; Expectation Maximization; Belief Propagation

1 Introduction

In this paper we address Constraint Satisfaction Problems (CSPs). To guide the backtrack search, we estimate the percentages of solutions that have a given variable assigned a particular value. Accurately estimating this distribution of the solutions, should such solutions exist, provide efficient search heuristics. Previous work [4] already shows a positive correlation between accuracy of the estimates and efficiency of the heuristic. The more accurate the estimations are, the more efficient the heuristic is. On one side of this accuracy spectrum, a distribution in which every variable-value assignment is equally probable represents a totally random heuristic. On the other side lie exact marginal probabilities for a randomly sampled solution of the problem. The first approach is trivial whereas the second remains intractable for hard problems. Indeed, finding the proportion of solutions that assign a variable a particular value is a generalization of the problem of detecting backbone variables. (A *backbone variable* is defined as a variable that takes the same value in every solution of a given problem; thus, the marginal distribution of such a variable corresponds to a probability of 1

for this particular value, and 0 for the other values in its domain.) Since even an approximation of a backbone is intractable in general [6], the problem of estimating the distribution of solutions is *a fortiori* also intractable in general for hard problems. The purpose of this paper is therefore not to present exact estimation methods for these distributions but to specify approximate methods, with different degrees of accuracy and of complexity.

In this probabilistic environment, inference methods such as message-passing algorithms were proven to be very effective. Even if they are more traditionally applied in probabilistic inference, their aptitude to estimate marginal probability makes them particularly suitable for a backtrack search framework. For instance, Belief Propagation (BP) [10, 8] (and later on Generalized Belief Propagation [13]) was developed to tackle inference problems such as problems arising in computer vision or error-correcting coding theory. Kask et al. [5] then demonstrated that BP was especially efficient when applied to CSPs and used as a value-ordering heuristic. Mezard et al. [9] invented Survey Propagation, which eventually turned out to be nothing less than the state of the art to solve large random Boolean satisfiability (SAT) problems [1, 7]. More recently, Hsu et al. [3, 4] suggested Expectation Maximization (EM) variants of these two major message-passing algorithms.

Here we exploit Expectation-Maximization Belief Propagation (EMBP) which was proposed by [3] to address the Quasigroup with Holes (QWH) problem. The contribution of this paper is two-fold: first, we derive new generic update rules suited to tackle any CSP; second, we propose an efficient EMBP-inspired approach, thereby improving this method and making it competitive with the state of the art. We evaluate these approaches experimentally and demonstrate their effectiveness.

The following section presents the EMBP algorithm. Section 3 then introduces the local consistency extensions of this algorithm, while Section 4 describes the global consistency approach. Section 5 reports the experimental results on three problems. Section 6 discusses connections between the EMBP framework and search heuristics from the literature. Final comments are given in Section 7.

2 Expectation-Maximization Belief Propagation

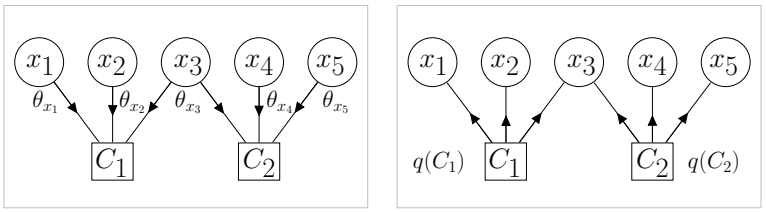
A *Constraint Satisfaction Problem* (CSP) consists of a finite set of variables $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ with finite domains $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ such that $x_i \in D_i$ for all i, together with a finite set of constraints $\mathcal{C}$, each on a subset of $\mathcal{X}$. A constraint $C_i \in \mathcal{C}$ is a subset $T(C_i)$ of the Cartesian product of the domains of the variables that are in C_i. We write $X(C_i)$ to denote the set of variables involved in C_i and we call tuple $\tau \in T(C_i)$ an allowed combination of values of $X(C_i)$.

This problem can be modeled indifferently as a Factor Graph, a Pairwise Markov random field or a Bayesian Network [13]. These models then define a joint probability function $P(x_1, x_2, \ldots, x_n)$. Thus, estimating the distribution of solutions

boils down to approximating marginal probabilities $P(x_i), \forall i \in 1..n$. Here lies the purpose of inference methods, and particularly of message-passing algorithms.

2.1 EMBP Framework

Like other message-passing algorithms, EMBP [3] iteratively adjusts the probability for a variable x_i to be assigned a particular value v in a randomly chosen solution. We will denote $\theta_{x_i}(v)$ and call *bias* such a probability. Hence, θ_{x_i} represents a probability distribution over the values in $D(x_i)$ which approximates the exact marginal distribution $P(x_i)$. The EMBP framework also introduces a binary-vector random variable y that indicates whether each constraint is satisfied. EMBP proceeds by sending two kind of messages (thus falling into the "message-passing" category), as illustrated on Figure 1. First, a variable sends its probability distribution to the constraints (Figure 1(a)) so that the latter hypothetically compute the probability of satisfying tuples. Then, every variable retrieves this information from its relevant constraints (Figure 1(b)) in order to update its probability distribution.



(a) E-Step (b) M-Step

Fig. 1. Illustrating Expectation-Maximization messages for the following example: $\mathcal{X} = \{x_1, x_2, x_3, x_4, x_5\}$, $\mathcal{C} = \{C_1, C_2\}$, with $X(C_1) = \{x_1, x_2, x_3\}$ and $X(C_2) = \{x_3, x_4, x_5\}\}$.

2.2 EMBP General Update Rule

Let Θ be the vector of variable biases θ_{x_i}. In essence, the basic goal of the EM algorithm consists of finding the variable biases Θ that maximize the probability $P(y|\Theta)$ that the constraints are satisfied. However, the EMBP methodology assumes that the vector of variables y was originally generated by using not only the parameters Θ, but also hidden variables that we did not observe. In a CSP environment, these latent variables (that we will call z) are actually the satisfying configurations, i.e. tuples, of the constraints. In other words, the variable z ranges over all valid solutions of the problem. We denote by S_z the support of the distribution of z.

Hence, we now want to maximize $P(y, z|\Theta)$. The difficulty lies in the fact that we cannot marginalize on z since it would assume that we can observe the

solutions of the problem. Instead, we divide this computation into two iterative steps. In the first step of the EM algorithm (the E-Step), we hypothesize the distribution $Q(z) = P(z|y, \Theta)$. The distribution function $Q(z)$ represents each solution probability given the biases Θ and given the observation y that the constraints are satisfied. Figure 1(a) illustrates this step, in which every constraint C receives the probability distribution θ_{x_i} (where $x_i \in X(C)$) and computes the probability of the satisfying configurations given these distributions. In the second step (the M-Step), we revise the variable biases Θ. As illustrated in Figure 1(b), the variables adjust their distribution taking into account the probability of the valid tuples of the constraints.

Within the Expectation-Maximization (EM) methodology, the E-step firstly assumes the independence between constraints to compute $Q(z)$ as we hypothetically consider only satisfying configurations. Thus, $Q(z)$ becomes the product of the probabilities of the constraints, given a particular set of biases Θ. Therefore, $Q(z) = \prod_{i=1}^{m}(q(C_i))$, where $q(C_i)$ is the probability of a given configuration for the constraint C_i.

The M-step then enforces the dependence between constraints, and yields the following general formula:

$$\theta_{x_i}(v) = \frac{1}{\eta} \sum_{C_k \in \mathcal{C}: x_i \in X(C_k)} \left(\sum_{z \in S_z : x_i = v} Q(z) \right) \tag{1}$$

Here η is a normalizing constant and equals the summation of the numerator over all values of v. For more details about how to derive Equation (1), please refer to [3].

As a result, EMBP provides a general algorithm to compute variable biases. Within this framework, the definition of $Q(z)$ remains however unspecified. The methods presented in the following section exploit this degree of freedom.

2.3 Computing EMBP: a tradeoff between accuracy and complexity

One might consider determining S_z, the solution space of the problem, and then computing $Q(z)$ exactly. This approach is however intractable since it implies expressing each solution of the problem to estimate the biases in order to find a solution. Hence, an approximation of $Q(z)$ is required. The methods described in this paper (summarized in Table 1) gradually improve the accuracy of the estimation of $Q(z)$ and, by doing so, increase the complexity of its computation. In that regard, the methods need to find the supports either locally (i.e. at the constraint level - *Lsup*) or globally (i.e. after propagating the whole constraint network - *Gsup*) enforcing a level of consistency denoted by X.[3]

[3] Note that the constraints involved can even implement different levels of consistency, as is common in practice.

Table 1. EMBP-based search heuristics

Heuristics	Consistency
EMBP-a	local Arc-Consistency
EMBP-Lsup	local X-Consistency
EMBP-Gsup	global X-Consistency

3 EMBP and local consistency

Whereas Equation (1) sums over the probabilities of all satisfying tuples, the following local X-consistency methods (EMBP-a and EMBP-Lsup) are designed to approximate this summation by simplifying it.

3.1 EMBP-a for the `alldifferent` Constraint

Originally, Hsu et al. [3] suggest such an approximation for the `alldifferent` constraint. The probability that variable x_i is assigned the value v can be approximated by the probability that no other variable in the constraint takes the value v. The approach is therefore ensuring pairwise consistency between x_i and the other variables in the `alldifferent` constraint. Thus, the authors obtain the following update rule for a set $\mathcal{C}_a$ of alldifferent constraints:

$$\begin{aligned}\theta_{x_i}(v) &= \frac{1}{\eta} \sum_{C_k \in \mathcal{C}_a : x_i \in X(C_k)} \left(\prod_{x_j \in X(C_k) \setminus x_i} \sum_{v' \in D(x_j) \setminus v} \theta_{x_j}(v') \right) \\ &= \frac{1}{\eta} \sum_{C_k \in \mathcal{C}_a : x_i \in X(C_k)} \left(\prod_{x_j \in X(C_k) \setminus x_i} \big(1 - \theta_{x_j}(v)\big) \right) \end{aligned} \tag{2}$$

where η is again a normalizing constant.

3.2 EMBP-Lsup

We propose to derive local X-consistency EMBP methods, which are a fairly natural extension of EMBP-a. The reasoning behind these methods to compute $\theta_{x_i}(v)$ for a given constraint is to consider all assignments $x_j = v'$ that are X-consistent with the assignment $x_i = v$ within this constraint. In other words, these methods take into account domain reductions at the constraint level when enforcing X-consistency. Essentially, the method processes one constraint at a time, and for each variable-value assignment looks for the supports that are X-consistent and updates the biases using the following formula:

$$\theta_{x_i}(v) = \frac{1}{\eta} \sum_{C_k \in \mathcal{C} : x_i \in X(C_k)} \left(\prod_{x_j \in X(C_k) \setminus x_i} \sum_{v' \in \tilde{D}_{x_i = v}(x_j)} \theta_{x_j}(v') \right) \tag{3}$$

where $\tilde{D}_{x_i=v}(x_j)$ represents the reduced domain of the variable x_j after assigning $x_i = v$ and enforcing X-consistency on C_i.

Considering that ensuring pairwise consistency between one variable and the others is reminiscent of arc consistency, EMBP-a indeed falls into the set of local X-consistency EMBP methods, with X standing for arc consistency. Nonetheless if we consider stronger consistency, such as domain consistency, the improvement is twofold: it provides better accuracy since the variable biases rely only on supports that are domain consistent (rather than arc consistent for EMBP-a) and it is easily implementable for any constraint for which we can enforce domain consistency. The method can be easily extended to consider any form of consistency. As in the general EMBP update rule, EMBP-Lsup still assumes independent constraints and only the M-Step enforces the dependence between the constraints.

Algorithm 1 shows the pseudo-code to update $\theta_{x_i}(v)$ at iteration t. P_{C_i} represents the contribution of C_i to $\theta_{x_i}(v)$ and S_{x_j} the summation for the variable x_j. Firstly, the algorithm picks up a constraint C_i whose variable set contains x_i, then propagates the assignment $x_i = v$ (line 3). For every variable x_j in the constraint's scope, we add up the value biases of the reduced domain (line 8) and multiply the result with P_{C_i} (line 9). Once the algorithm is done with processing constraint C_i, it adds the contribution of this constraint to $\theta_{x_i}(v)$, rolls back the effect of the propagation (lines 10–11), and continues to iterate over the remaining constraints. Note that when all the $\theta_{x_i}(\cdot)$ have been computed, they have to be normalized.

```
1  θ^{t+1}_{x_i}(v) = 0;
2  for each constraint C_i ∈ C : x_j ∈ X(C_i) do
3      enforce X-consistency on C_i with x_i = v;
4      P_{C_i} = 1;
5      for each variable x_j ∈ X(C_i) \ x_i do
6          S_{x_j} = 0;
7          for each value v' ∈ D̃_{x_i=v}(x_j) do
8              S_{x_j} = S_{x_j} + θ^t_{x_j}(v');
9          P_{C_i} = P_{C_i} × S_{x_j};
10     θ^{t+1}_{x_i}(v) = θ^{t+1}_{x_i}(v) + P_{C_i};
11     rollback propagation C_i;
12 normalize θ^{t+1}_{x_i}(v);
```

Algorithm 1: EMBP-Lsup update algorithm for $\theta_{x_i}(v)$ at iteration t

Given k the number of constraints in which x_i is involved, n their maximum arity, d the maximum cardinality of the variable domains and P the worst-case complexity of the constraints' propagation algorithms, the worst-case time complexity of Algorithm 1 is $\mathcal{O}(kP+knd)$. To reach convergence, the code shown in Algorithm 1 should be run for every variable-value pair at each iteration of the EMBP. In order to improve the efficiency of the procedure, the propagation

of the constraints is performed once and the information related to $\tilde{D}_{x_i=v}$ is cached since the reduced domains remain constant over the iterations.

This method aims to be a tradeoff between accuracy and performance: at any given time no propagation other than the one of the processed constraint is performed, which is less expensive than propagating the whole model, but also less accurate.

4 EMBP and global consistency

Introducing a new method that we call EMBP-Gsup, we suggest to go one step further in terms of accuracy for the computation of $Q(z)$. Indeed, in EMBP-Gsup, the problem is considered as a whole and the method directly exploits the dependence between constraints when computing $Q(z)$. EMBP-Lsup considers supports that are X-consistent for one constraint at a time whereas EMBP-Gsup will improve the quality of the approximation by taking into account supports that are X-consistent after propagating each problem's constraint. In the new representation underlying the approximation of $Q(z)$, instead of having m constraints (see Figure 1), we now consider only one, in which every variable in $\mathcal{X}$ is involved.

The update formula is then:

$$\begin{aligned}\theta_{x_i}(v) &= \frac{1}{\eta} \prod_{x_j \in \mathcal{X} \setminus x_i} \sum_{v' \in \hat{D}_{x_i=v}(x_j)} \theta_{x_j}(v') \\ &= \frac{1}{\eta} \prod_{x_j \in \mathcal{X} \setminus x_i} \left(1 - \sum_{v' \in D(x_j) \setminus \hat{D}_{x_i=v}(x_j)} \theta_{x_j}(v') \right) \end{aligned} \tag{4}$$

where $D(x_j)$ is the domain of x_j before assigning $x_i = v$ and $\hat{D}_{x_i=v}(x_j)$ stands for the reduced domain of the variable x_j after assigning $x_i = v$ and enforcing X-consistency on each problem constraint.

The method indirectly depends on the problem modelling since it depends on the overall inference power underlying the specific modelling. However, this approach also offers the possibility of using different levels of consistency during the probing phase (when we compute $\hat{D}_{x_i=v}$) and during the effective propagation phase of the search.

Algorithm 2 shows the pseudo-code to update $\theta_{x_i}(v)$ at iteration t. The main difference with Algorithm 1 lies in the fact that there is no summation over the constraints since the contribution of each constraint is implicitly reflected in $\hat{D}_{x_i=v}(x_j)$ due to the initial propagation over the whole problem (line 2). In practice the experiments revealed that it is faster to iterate over the delta domain of the variables (which is the complementary set of $\hat{D}_{x_i=v}(x_j)$) rather than over $\hat{D}_{x_i=v}(x_j)$ itself (line 5).

$\theta_{x_i}^{t+1}(v) = 1$;
enforce X-consistency on the whole problem with $x_i = v$;
for *each variable* $x_j \in \mathcal{X} \setminus x_i$ **do**
    $S_{x_j} = 1$;
    for *each value* $v' \in D(x_j) \setminus \hat{D}_{x_i=v}(x_j)$ **do**
        $S_{x_j} = S_{x_j} - \theta_{x_j}^{t}(v')$;
    $\theta_{x_i}^{t+1}(v) = \theta_{x_i}^{t+1}(v) \times S_{x_j}$;
rollback propagation of $x_i = v$;
normalize $\theta_{x_i}^{t+1}(v)$;

Algorithm 2: EMBP-Gsup update algorithm for $\theta_{x_i}(v)$ at iteration t

Given K the total number of constraints, N the total number of variables, d the maximum cardinality of the variable domains and P the worst-case complexity of the constraints' propagation algorithms, the worst-case time complexity of Algorithm 2 is $\mathcal{O}(KP + Nd)$. Even though not obvious from the worst-case complexities, Algorithm 2 happens to be more time consuming than Algorithm 1 since we usually have $K \gg k$ and $N \gg n$. In order to avoid propagation at each iteration of the EMBP, the information related to $D(x_j) \setminus \hat{D}_{x_i=v}$ is also computed once and then cached, as in Algorithm 1.

5 Experiments

In this section we evaluate search heuristics based on the methods we propose. Fundamentally, at each node of the search tree, after computing the variable biases according to EMBP-a, EMBP-Lsup, or EMBP-Gsup, the search heuristic branches on the variable-value pair that presents the highest bias. We evaluate the proposed search heuristics on three benchmark problems - the Nonogram problem, the Quasigroup With Holes problem and the Magic Square problem. We then compare our results with six other heuristics, rndMinDom, MaxSD, IlogIBS, IlogAdvIBS, RSC-LA, and RSC2-LA. Before presenting the results of the experiments, we detail these heuristics and explain the rationale behind the selection of these six other search heuristics:

- rndMinDom randomly picks up a variable with the smallest domain size and then randomly selects a value from its domain. We present the results for this heuristic as a benchmark for fairly uninformed yet very common heuristics.

- MaxSD stands for maximum Solution Density and belongs to the family of solution counting based heuristics. This heuristic exploits counting information provided by the constraints in order to branch on the part of the search tree where it is likely to find a higher number of solutions [14]. MaxSD is considered the state-of-the-art for solving the hard QWH problems.

- Impact Based Search [11] methods first choose the variable whose instantiation triggers the largest search space reduction (highest impact) that is approxi-

mated as the reduction of the Cartesian product of the domains of the variables. The impact is either approximated as the average reduction observed during the search or computed exactly at a given node of the search (the exact computation provides better information but is more time consuming). IlogIBS represents Impact Based Search where the impacts are approximated. IlogAdvIBS chooses a subset of 5 variables with the best approximated impacts and then it breaks ties based on the exact impacts while further ties are broken randomly [11].

- RSC-LA stands for restricted singleton consistency look-ahead heuristic [2]. RSC-LA maintains restricted singleton consistency during the search while RSC2-LA maintains this level of consistency for a subset of variables whose domain size equals 2. While enforcing singleton consistency, the method collects look-ahead information under the form of impacts and uses it in a manner similar to [11]. RSC-LA is similar to EMBP-Gsup since they both perform a complete look-ahead procedure at every choice point. Hence, it is definitely worth comparing the results for these two approaches, as they only differ on how to aggregate the look-ahead information.

All tests were performed with Ilog Solver 6.5 on a AMD Opteron 2.4GHz with 1GB of RAM; for the heuristics that have some sort of randomization we present the arithmetic averages over 10 runs. In the heuristics based on EMBP, the bias computation is performed at every node. At each node, we randomly initialize the biases and iterate until convergence or until a fixed number of iterations is reached. Even though EMBP methods do guarantee convergence [3], they converge to a *local* optimum, which might differ from the exact marginalizations. Nonetheless, convergence is relatively fast and every iteration is quite time consuming so we decided to limit to 5 the number of iterations during these experiments.

Nonogram A Nonogram (problem 12 of CSPLib) is built on a rectangular $n \times m$ grid and requires filling in some of the squares in the unique feasible way according to some clues given on each row and column. As a reward, one gets a pretty monochromatic picture. Each individual clue indicates how many sequences of consecutive filled-in squares there are in the row (column), with their respective size in order of appearance. Each sequence is separated from the others by at least one blank square but we know little about their actual position in the row (column). Such clues can be modeled with `regular` constraints (the actual automata $\mathcal{A}_i^r, \mathcal{A}_j^c$ are not difficult to derive but lie outside the scope of this paper):

$$\begin{array}{ll} \texttt{regular}((x_{ij})_{1\leq j\leq m}, \mathcal{A}_i^r) & 1 \leq i \leq n \\ \texttt{regular}((x_{ij})_{1\leq i\leq n}, \mathcal{A}_j^c) & 1 \leq j \leq m \\ x_{ij} \in \{0,1\} & 1 \leq i \leq n,\ 1 \leq j \leq m \end{array}$$

We experimented with 180 instances[4] of sizes ranging from 16×16 to 32×32. We enforced domain consistency on the constraints and set a timeout of 10 minutes for each run.

[4] Instances taken from http://www.blindchicken.com/∼ali/games/puzzles.html

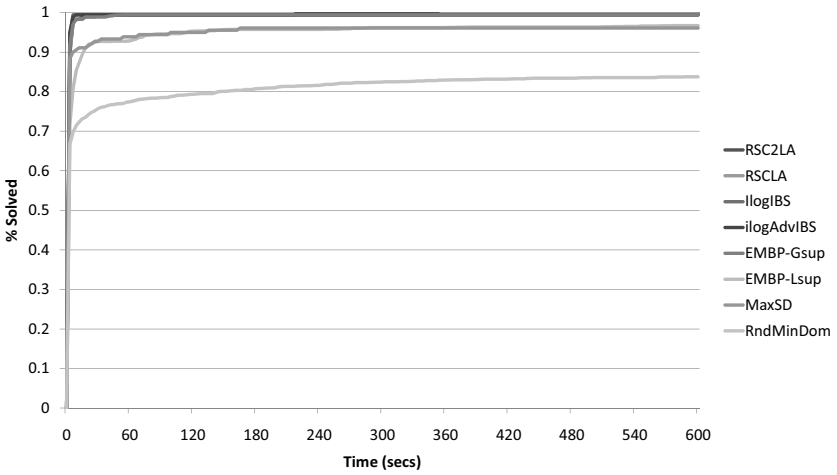


Fig. 2. Percentage of solved instances vs time for 180 Nonogram instances

Figure 2 shows the percentage of instances solved within a given time. In Nonograms the additional constraint inference performed by EMBP does not bring a clear advantage over simply using information such as impacts. In fact, as shown in the plot, this problem is a fairly easy one, most of the instances are solved within few seconds; yet EMBP-Gsup, despite its inherent overhead, performs basically the same as RSC-based and IBS-based heuristics. MaxSD and EMBP-Lsup are slightly behind in this test, being able to solve fewer instances. Finally, our baseline RndMinDom is significantly slower compared to the rest of the heuristics. What the figure doesn't show is that our proposed heuristics dramatically improve the total number of backtracks (by three orders of magnitude) over any other heuristic tested except *RSC-LA* methods, even if it does not translate to the best overall total running time (mainly due to a single instance that timed out).

Quasigroup with Holes A Latin Square of order n is defined on a $n \times n$ grid whose cells each contain an integer from 1 to n such that each integer appears exactly once per row and column. The Quasigroup with Holes (QWH) problem gives a partially-filled Latin Square instance and asks to complete it. It can be modeled easily as follows:

$$\begin{array}{ll} \texttt{alldifferent}((x_{ij})_{1\leq j\leq n}) & 1 \leq i \leq n \\ \texttt{alldifferent}((x_{ij})_{1\leq i\leq n}) & 1 \leq j \leq n \\ x_{ij} = d & (i,j,d) \in S \\ x_{ij} \in \{1,2,\ldots,n\} & 1 \leq i,j \leq n \end{array}$$

where S represents the set of pre-assigned cells.

The set of instances is the same as in [14]: 40 instances with $n = 30$ and a percentage of holes around 42% (near the phase transition). We enforced domain consistency on the constraints and set a timeout of 20 minutes for each run.

For this test we also added the heuristic dom/ddeg; min conflicts that chooses the variable with the lowest ratio of domain size over dynamic degree and then selects the value with the minimum number of conflicts (this has been considered for years one of the best custom heuristics for QWH). For the heuristic MaxSD, the counting algorithm has been set as in [14].

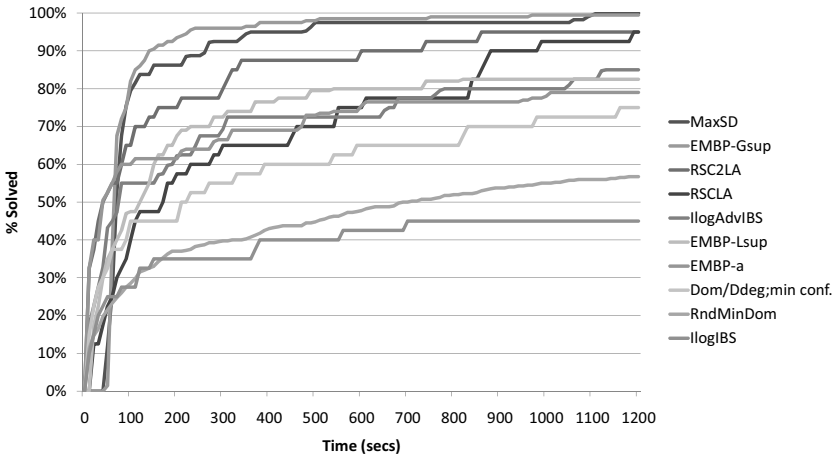


Fig. 3. Percentage of solved instances vs time for 40 hard QWH instances of order 30

Figure 3 shows the results (presented as in the Nonogram problem). Despite the fact that EMBP-Lsup is significantly more lightweight than EMBP-Gsup, its performance does not match that of EMBP-Gsup because of poorer heuristic guidance. EMBP-Gsup performs very well in terms of number of backtracks (at least one order of magnitude better than any other heuristic) and scores the best total time. Again most of the computation time is spent at each node of the search tree in propagating the whole problem for each variable-value pair but the accuracy of the variables' biases definitely pays off. This overhead can be seen in Figure 3: the heuristic takes some time to get close to the leaves of the search tree to find solutions. Therefore the heuristic hardly solves any instance within the first minute. However it is able to solve about 75% of the instances in 90 seconds. MaxSD presents the same behavior, due to the sampling algorithm used to count the number of solutions of the `alldifferent` constraints which causes a significant overhead. Nonetheless, the total running time of EMBP-Gsup is significantly lower. The heuristics that were scoring the best running times on the Nonogram problem (RSC-based and IBS-based heuristics) struggle more here and their running time goes from almost two to seven times the running time of EMBP-Gsup to solve the same number of instances. The total number of instances solved is also significantly lower compared to EMBP-Gsup. Finally, it is interesting to see how Hsu et al.'s approach behaves with respect to EMBP-Lsup: the two methods are similar (although Hsu et al.'s only applies to

`alldifferent` constraints), the only difference being that ours enforces domain consistency on the constraints whereas EMBP-a keeps a weak form of arc consistency. Hence, EMBP-a is able to perform more backtracks w.r.t. EMBP-Lsup in the same amount of time but has poorer heuristic guidance.

Magic Square This very old puzzle is built on a $n \times n$ grid. The task is to place the first n^2 integers in the grid so that each row, column and main diagonal sums up to the same value. A partially filled Magic Square Problem asks for a solution, if one exists. It can be made harder to solve than the traditional version starting from a blank grid. More formally, here is a model for the Magic Square Problem:

$$
\begin{array}{ll}
\texttt{alldifferent}((x_{i,j})_{1 \leq i,j \leq n}) & \\
\sum_{1 \leq j \leq n} x_{i,j} = sum & 1 \leq i \leq n \\
\sum_{1 \leq i \leq n} x_{i,j} = sum & 1 \leq j \leq n \\
\sum_{1 \leq i \leq n} x_{i,i} = sum & \\
\sum_{1 \leq i \leq n} x_{(n+1-i),i} = sum & \\
x_{i,j} \in \{1, n^2\} & 1 \leq i \leq n,\ 1 \leq j \leq n
\end{array}
$$

where $sum = \frac{n(n^2+1)}{2}$. On the `alldifferent` constraint, domain consistency is enforced, while for the equality `knapsack` constraint, bound consistency is enforced. The set of instances is the same as in [12]: 40 instances with prefilled cells (in order to avoid trivial solutions) — half of the instances have 10 preset variables and the other half, 50. A timeout of one hour was set for each run.

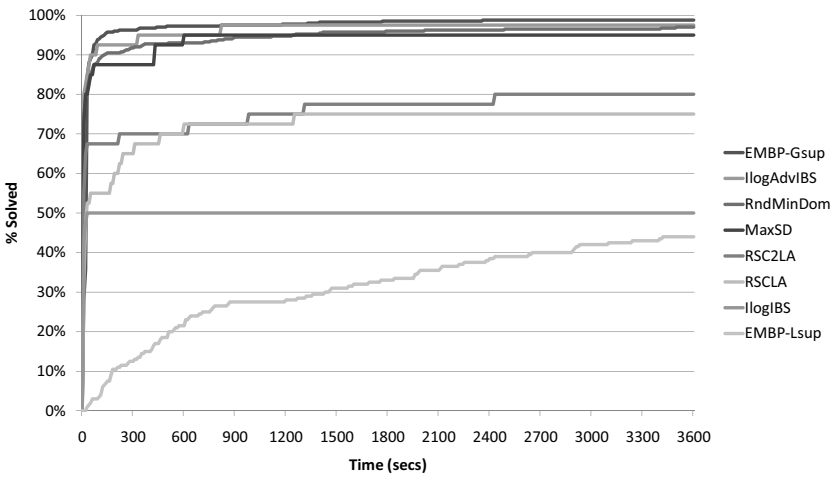


Fig. 4. Percentage of solved instances vs time for 40 Magic Square instances

Figure 4 shows the results as presented on the previous two problems. EMBP-Gsup is the best performing heuristic, outperforming by about 35% the second best heuristic (ilogAdvIBS) in terms of total time (including timeouts). As shown

in the plot, in this problem the heuristic is well suited both for hard and easy instances. Again, the number of backtracks is the lowest among the group of heuristics by at least one order of magnitude. Despite being fairly good on the QWH problem, MaxSD falls far behind in this test taking more than double the time of EMBP-Gsup.

As before, EMBP-Lsup was not able to provide an interesting performance with respect to EMBP-Gsup. However, it is worth mentioning that the EMBP-Gsup underlying algorithm provides at no extra cost a reduced form of singleton consistency whereas EMBP-Lsup does not enforce implicitely such form of consistency.

Overall, compared to some state-of-the-art heuristics, EMBP-Gsup turns out to be the most consistent heuristic on the problems considered. RSC2-LA performs better on Nonogram instances but not on QWH and it fails to solve 20% of the Magic Square instances. Impact-Based Search heuristics achieve comparable performance on Nonogram but finish far behind on QWH instances. Finally, MaxSD performs very well on QWH but not as well as EMBP-Gsup when considering Nonogram and Magic Square.

6 Discussion

In this section we draw connections between some of the heuristics used in this paper.

Solution counting algorithms [14] propose an approach which is closely related to the EMBP method. Indeed, constraint-centered solution counting also offers marginals of the variables for a specific constraint. This method however considers constraints separately. In the EMBP framework, this would be equivalent to considering one independent factor graph for each constraint. Hence, constraint-level solution counting estimates marginal distributions of *a priori* independent constraints. Within a backtrack search tree, this approach then suggests to consider a basic aggregation of these marginals but misses a more global reasoning that considers the dependence between the constraints. Also, the first iteration of the local X-consistency EMBP method would give the exact computation of the solution densities as defined in [14] (again assuming variable biases are uniformly initialized) if we were able to exactly compute $Q(z)$.

The `RSC-LA` methods proposed by Correia and Barahona in [2] also present strong similarities with EMBP-Gsup. In our case, we are building search heuristics using look-ahead information which in return provides restricted singleton consistency. Conversely, in [2], the authors ensure restricted singleton consistencies and take advantage of the information provided during look-ahead procedures to derive search heuristics. Compared to [2], when computing EMBP-Gsup, we thus also benefit from restricted singleton consistency that allows us to shave the search tree with every variable-value pair that is inconsistent. As a result, both methods present a similar overhead of computation time at each node.

The difference in the search heuristics lies in the fact that [2] exploits impact information whereas we are performing inference reasoning.

There are also some interesting connections between `IBS` and EMBP methods. Were we to uniformly initialize the variable biases, the first iteration of EMBP-Gsup would compute the impact of every variable-value pair as the reduction of the Cartesian product of the domain, as [11, 2] would do. Subsequent iterations further refine this impact, thereby generating a sort of "weighted" impact.

7 Conclusion and Open Issues

This paper provided generic and efficient heuristics built upon the EMBP framework. Whereas previous EMBP proposals in [3] addressed problems involving only `alldifferent` constraints, we lifted that restriction with our contribution. Furthermore, we provided a more efficient formulation that achieves very promising performance and is competitive with the state of the art — it was the most consistent on the problems considered. The number of backtracks for our proposed heuristics was also consistently much lower, thus indicating excellent heuristic guidance and some potential for runtime improvement if parts of the computation are optimized or approximated (e.g. see the next paragraph). An important step ahead has been achieved compared to the approaches presented in [3] and [14]. While these previous approaches respectively require constraint-specific update rules and constraint-specific solution counting algorithms, EMBP-Gsup and EMBP-Lsup are completely general and easily pluggable into any model.

Several important open issues still remain. First, even though accurate biases certainly provide useful information for an underlying search framework, the question remains of determining the most efficient way to use it. Indeed, when used within a variable-ordering heuristic, a method providing a set of biases still needs to define a branching strategy. Here the possibilities are numerous. For example, we can choose to branch on the highest bias as we have done during these experiments, but we could also choose the highest strength (difference between a bias and the reciprocal of the domain size) or even remove the value with the lowest bias from a variable domain. Second, instead of computing a survey at each node of the search tree, we could use previous information. For instance, we could set more than one variable at once, or also compute partial surveys and keep track of the domain reduction information, as an IBS heuristic would do. Finally, in future work, we would like to derive equivalent update rules for the Expectation-Maximization Survey Propagation (EMSP) [4] and exploit the promising potential of Survey Propagation.

Acknowledgements

The authors would like to thank Eric Hsu for useful comments about the EMBP framework and the anonymous referees for their constructive comments. The

authors also thank the FCI-Relève program for financing the equipment on which the experiments were partially conducted. This research was supported in part by an NSERC discovery grant and an FQRNT doctoral scholarship.

References

1. Braunstein, A., Mézard, M., and Zecchina, R. Survey propagation: An algorithm for satisfiability. *Random Structures Algorithms 27*, 2 (2005), 201–226.
2. Correia, M., and Barahona, P. On the integration of singleton consistencies and look-ahead heuristics. In *CSCLP* (2007), pp. 62–75.
3. Hsu, E. I., Kitching, M., Bacchus, F., and McIlraith, S. A. Using expectation maximization to find likely assignments for solving csp's. In *AAAI* (2007), pp. 224–230.
4. Hsu, E. I., Muise, C. J., Beck, J. C., and McIlraith, S. A. Applying probabilistic inference to heuristic search by estimating variable bias. In *Proceedings of the 1st International Symposium on Search Techniques in Artificial Intelligence and Robotics (at AAAI08)* (Chicago, IL, USA, July 13–14 2008), vol. WS-08-10, pp. 68 – 75.
5. Kask, K., Dechter, R., and Gogate, V. Counting-based look-ahead schemes for constraint satisfaction. In *Proc. of 10th International Conference on Constraint Programming (CP 04)* (2004), pp. 317–331.
6. Kilby, P., Slaney, J. K., Thibaux, S., and Walsh, T. Backbones and backdoors in satisfiability. In *AAAI* (2005), M. M. Veloso and S. Kambhampati, Eds., AAAI Press / The MIT Press, pp. 1368–1373.
7. Kroc, L., Sabharwal, A., and Selman, B. Survey propagation revisited. In *23rd UAI* (Vancouver, BC, July 2007), pp. 217–226.
8. Kschischang, F. R., Frey, B. J., and Loeliger, H. A. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on 47*, 2 (2001), 498–519.
9. Mezard, M., Parisi, G., and Zecchina, R. Analytic and algorithmic solution of random satisfiability problems. *Science 297*, 5582 (August 2002), 812–815.
10. Pearl, J. *Probabilistic Reasoning in Intelligent Systems:Networks of Plausible Inference*. Morgan Kaufmann, 1988.
11. Refalo, P. Impact-based search strategies for constraint programming. In *CP* (2004), pp. 557–571.
12. Trick, M. A. A dynamic programming approach for consistency and propagation for knapsack constraints. In *Annals of Operations Research* (2003), vol. 118, pp. 73–84.
13. Yedidia, J. S., Freeman, W. T., and Weiss, Y. *Understanding belief propagation and its generalizations*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
14. Zanarini, A., and Pesant, G. Solution counting algorithms for constraint-centered search heuristics. *Constraints 14*, 3 (2009), 392–413.