Exploiting Entropy in Constraint Programming

Auguste Burlats[1](✉) and Gilles Pesant[2](✉)

[1] UCLouvain, Ottignies-Louvain-la-Neuve, Belgium
auguste.burlats@uclouvain.be
[2] Polytechnique Montréal, Montreal, Canada
gilles.pesant@polymtl.ca

Abstract. The introduction of Belief Propagation in Constraint Programming through the CP-BP framework makes possible the computation of an estimation of the probability that a given variable-value combination belongs to a solution. The availability of such marginal probability distributions, effectively ranking domain values, allows us to develop branching heuristics but also more generally to apply the concept of entropy to Constraint Programming. We explore how variable and problem entropy can improve how we solve combinatorial problems in the CP-BP framework. We evaluate our proposal on an extensive set of benchmark instances.

1 Introduction

Constraint Programming (CP) is a powerful approach to solve combinatorial problems. It can significantly reduce the search space by using constraints and their powerful inference algorithms to filter out infeasible variable-value combinations at each node of the search tree. The order in which variables are branched on has a significant impact on the shape of the tree and thus on search efficiency. This is why finding robust and generic variable ordering heuristics is crucial. The introduction of Belief Propagation (BP) in CP [6] makes possible the computation of an estimation of the probability that a given variable-value combination belongs to a solution. The availability of such marginal probabilities, effectively ranking domain values, allows us to develop variable ordering heuristics [1] but also more generally to apply the concept of entropy to CP, which is the subject of this paper.

A *Constraint Satisfaction Problem* (*CSP*) $P = \langle X, D, C\rangle$ is a combinatorial problem defined by a triplet where:

- $X = \{x_1, x_2, \dots, x_n\}$ is a finite set of variables,
- $D = \{D(x_1), D(x_2), \dots, D(x_n)\}$ is a finite set of finite domains,
- $C = \{c_1, c_2, \dots, c_m\}$ is a finite set of constraints.

A. Burlats—Most of this work was carried out while the first author was at Polytechnique Montréal.

A. A. Cire (Ed.): CPAIOR 2023, LNCS 13884, pp. 320–335, 2023.
https://doi.org/10.1007/978-3-031-33271-5_21

A *solution* $\mathbf{s} = (v_1, v_2, \ldots, v_n)$ to P assigns to each variable $x_i \in X$ a value v_i from its corresponding domain $D(x_i)$ such that all constraints in C are satisfied. Let S^P denote the set of all solutions to P and $\mathbf{s}[x]$ the value assigned to variable x in solution $\mathbf{s}$. Define

$$\theta_x^P(v) = \frac{|\{\mathbf{s} \in S^P : \mathbf{s}[x] = v\}|}{|S^P|}$$

as the proportion of solutions in which variable x takes value v.[1] We will call this quantity the *marginal* of variable-value pair (x, v) in reference to the marginal probability of x taking value v in a solution chosen uniformly at random from S. Note that we assume for the moment that S in nonempty i.e. that P is *satisfiable*: otherwise we will consider all marginals to be null. We define $H(x)$, the *entropy* of variable x using Shannon entropy [8]:

$$H(x) = \; -\textstyle\sum_{v \in D(x)} \theta_x(v) \log(\theta_x(v)).$$

This nonnegative quantity can be interpreted as the uncertainty about which value x should take in a solution: a null entropy corresponds to $\theta_x(v) = 1$ for some v, and so $\theta_x(v') = 0 \;\forall v' \neq v$, i.e. $x = v$ in every solution (and x is thus a *backbone* variable); maximum entropy $\log(|D(x)|)$ is reached whenever $\theta_x(v) = \frac{1}{|D(x)|} \;\forall v \in D(x)$ i.e. its values are uniformly distributed among solutions. The normalized entropy (also called efficiency) of x divides its entropy by the logarithm of the cardinality of its domain, unless its domain is a singleton in which case the (normalized) entropy is null. We derive the entropy of problem P as

$$H(P) = \frac{\sum_{x \in X \,:\, |D(x)|>1} \frac{H(x)}{\log(|D(x)|)}}{|X|}.$$

It corresponds to the average normalized entropy of its variables and thus lies between 0 and 1 inclusive. We make two general observations about CSP entropy:

Observation 1. *A null CSP entropy only occurs when either it admits a single solution (including the special case where all variables are bound in a consistent assignment) or it has no solution. In the former case each variable has some unique value in its domain with a unit marginal whereas in the latter, each variable has all null marginals.*

Observation 2. *A CSP for which every assignment is a solution (or with uninformed marginals) will exhibit uniformly distributed marginals for each variable and an entropy equal to the proportion of its variables with non-singleton domains. In particular if all variables have non-singleton domains the CSP entropy is one.*

Of course our notion of entropy relies on marginals of which we typically do not know the exact value. This is where BP comes in to provide estimates of

[1] We will generally omit superscript P for ease of notation.

these marginals. In this paper we investigate several uses of entropy to help solve CSPs, particularly for branching heuristics.

We follow with a review of the CP-BP framework in Sect. 2 and then evaluate the accuracy of the marginals computed in this framework in Sect. 3. Section 4 presents different uses of entropy and follows with comparative experiments in Sect. 5. We then conclude in Sect. 6.

2 Belief Propagation for CSPs

Belief Propagation (BP) is an algorithm introduced by Pearl [5]. It is able to compute the marginal distribution for each non-observed node in a graphical model (e.g. a factor graph), conditioned by the value of the observed nodes.

Pesant [6] introduced a framework combining CP and BP in which beliefs about variable-value pairs are propagated as messages between variables and constraints, thus generalizing the simpler propagation of unsupported pairs. A CSP can be viewed as a factor graph where the constraints are the factor nodes and the variables are the variable nodes. We note $\mu_{c \rightarrow x}$ the message from constraint c to variable x, and $\mu_{x \rightarrow c}$ the message from variable x to constraint c. Their definition is

$$\begin{cases} \mu_{x\rightarrow c}(v) = \prod_{c' \in N(x) \setminus \{c\}} \mu_{c' \rightarrow x}(v) \\ \mu_{c \rightarrow x}(v) = \sum_{\mathbf{v}:\mathbf{v}[x]=v} f_c(\mathbf{v}) \prod_{x' \in N(c)\setminus\{x\}} \mu_{x' \rightarrow c}(\mathbf{v}[x']) \end{cases}$$

where $N(x)$ is the neighbourhood of variable x, i.e. the constraints applied to this variable, $N(c)$ is the neighbourhood of constraint c, i.e. its scope, $\mathbf{v}$ is a tuple from the Cartesian product of all the variables in the scope of c, $\mathbf{v}[x]$ is the value taken by x in $\mathbf{v}$ and $f_c(\mathbf{v})$ is a function that returns 1 if tuple $\mathbf{v}$ satisfies c and 0 otherwise. We are thus able to estimate the marginal of a variable x as

$$\hat{\theta}_x(v) = \prod_{c \in N(x)} \mu_{c \rightarrow x}(v) \qquad \forall v \in D(x).$$

Messages are sent iteratively: first, all variables send their messages (initially, uniform distributions over their domain); then all constraints send their messages. This cycle is repeated for a fixed number of iterations. Among other things, in this paper we offer a way to decide this number dynamically at each node of the search tree. Computing $\sum_{\mathbf{v}:\mathbf{v}[x]=v} f_c(\mathbf{v})$ is equivalent to counting solutions (local to c) where $\mathbf{v}[x] = v$. Therefore messages from constraints report the number of such solutions, each being weighted by the product of corresponding messages from variables. Pesant [6] provided efficient dedicated algorithms for weighted counting on several constraints. If there is no such algorithm for a given constraint it simply sends back a uniform distribution. Let's examine a small example from [6] to illustrate the behaviour of marginals.

Example 1. Consider variables a, b, c, and d with identical domains $\{1, 2, 3, 4\}$, and the following constraints:

$$\texttt{alldifferent}(a, b, c), \qquad a + b + c + d = 7, \qquad c \leq d.$$

Table 1. True marginals (a), initial estimated marginals (b), marginals after 1st iteration of BP (c) and after 10th iteration (d) for Example 1.

	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

(a) true marginals

	1	2	3	4
$\hat{\theta}_a$	.25	.25	.25	.25
$\hat{\theta}_b$	.25	.25	.25	.25
$\hat{\theta}_c$	.25	.25	.25	.25
$\hat{\theta}_d$	.25	.25	.25	.25

(b) initial marginals

	1	2	3	4
$\hat{\theta}_a$	.50	.30	.15	.05
$\hat{\theta}_b$	.50	.30	.15	.05
$\hat{\theta}_c$	.62	.28	.09	.01
$\hat{\theta}_d$	.29	.34	.26	.11

(c) 1st iteration

	1	2	3	4
$\hat{\theta}_a$	.01	.52	.46	.01
$\hat{\theta}_b$	.01	.52	.46	.01
$\hat{\theta}_c$	.98	.02	.00	.00
$\hat{\theta}_d$	.90	.10	.00	.00

(d) 10th iteration

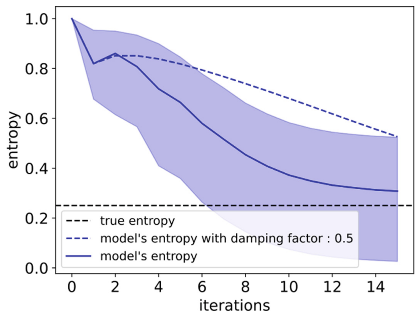


Fig. 1. Evolution of entropy during Belief Propagation for the CSP in Example 1.

This CSP has two solutions: $\langle a = 2, b = 3, c = 1, d = 1\rangle$ and $\langle a = 3, b = 2, c = 1, d = 1\rangle$. If we examine variable a, we observe that assignment $a = 2$ is present in one solution and that assignment $a = 3$ is present in the other one. There is no valid solution containing $a = 1$ or $a = 4$. Therefore its true marginal distribution is $\theta_a(1) = 0, \theta_a(2) = 1/2, \theta_a(3) = 1/2, \theta_a(4) = 0$. If we examine variable c, we can observe that only assignment $c = 1$ can be in a valid solution. Thus, its marginal distribution is $\theta_c(1) = 1, \theta_c(2) = 0, \theta_c(3) = 0, \theta_c(4) = 0$. BP starts from a uniform distribution for each variable: $\hat{\theta}_{x_i}(v) = 1/|D(x_i)|, \forall v \in D(x_i), \forall x_i \in X$. And, as we can see in Table 1, BP tends to converge to the true marginal distributions after a few iterations. Figure 1 also traces the evolution of the CSP entropy $H(P)$ (solid curve) with a lighter band showing the range of entropy for individual variables (normalized $H(x)$). For comparison the uninformed CSP entropy, i.e. considering domain values to be equally likely, corresponds to the initial value of the curve (1.0) and the true entropy is 0.25.

BP is assured to converge when there is no cycle in the graph [5] but the graphical representation of a CSP typically contains such cycles. However the large arity of global constraints, in addition to performing efficient inference, allows us to encapsulate some of those cycles and prevent the marginals from oscillating [6]. For instance in Example 1 marginals converge despite the remaining cycles in the model. In case marginals still oscillate, *message damping* has been known to help. Babaki et al. [1] propose using the weighted average of the current and previous messages from variables to constraints:

$$\mu_{x\to c}(v) = \lambda\mu_{x\to c}^{\text{current}}(v) + (1-\lambda)\mu_{x\to c}^{\text{previous}}(v)$$

where the *damping factor* ($0 \leq \lambda \leq 1$) balances the old and the new. Observe that for Example 1 damping is not needed and even slows down convergence (dashed curve in Fig. 1).

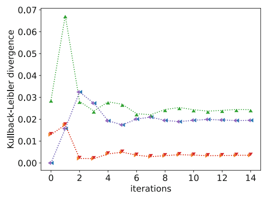
The estimated marginals $\hat{\theta}_x(v)$ can serve to inform dynamic search heuristics. For example *max-marginal* [1] branches on variable $\text{argmax}_{x\in X} \max_{v\in D(x)} (\hat{\theta}_x(v))$, assigning it its domain value with the strongest marginal.

3 Accuracy of BP-Estimated Marginals and Entropy

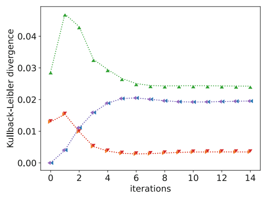
In this section we evaluate empirically the accuracy of the marginals (and ultimately of the entropy) computed by BP on a CP model. After the initial constraint propagation, we track these estimated marginals as BP iterations proceed (and before any branching occurs). Whenever we activate message damping we use a damping factor $\lambda = 0.5$ (the default value in MiniCPBP, the prototype solver implementing the CP-BP framework). We use several instances of combinatorial problems: some with a single solution (Sudoku and Nonogram) and others with a moderate number of solutions (n-queens and Feature model [9]). We enumerate the solutions to these instances and use them to compute the true marginals, against which we compare the estimated marginals using the *Kullback-Leibler divergence*, a measure of dissimilarity between two probability distributions:

$$\textstyle\sum_{v\in D(x)} \theta_x(v) \cdot \log(\theta_x(v)/\hat{\theta}_x(v)).$$

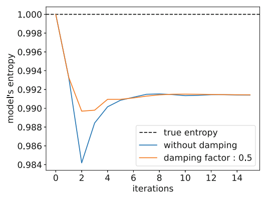
For n-queens we use the instances ranging from $n = 5$ to 9 that have respectively 10, 4, 40, 92, and 352 solutions (we do not break symmetries). We observe at Fig. 2 that the KL-divergence stabilizes after a few iterations and to a low value. The exception is for 6-queens where the divergence is about one order of magnitude greater. That small instance has the fewest solutions: each variable has four domain values with identical true marginal (0.25) and the remaining two with a null true marginal. Upon inspection, no estimated marginal is null but the four outstanding values in each domain do have larger estimated marginals for all variables, meaning they are ranked correctly, though with less of a distinction for *q1* and *q4*. There are also a few misses: for example with 7-queens $\hat{\theta}_{q3}(3)$ is the lowest in the domain whereas it should be the highest. Damping does not make much of a difference here. Even though we typically cannot compute the KL-divergence because we do not know the true marginals, the observed entropy of the estimated marginals (Fig. 2, right column), which we can compute, will align very well with the unobserved divergence in terms of the iteration when they become stable, and this could be used to decide when to stop iterating BP. The difference between the estimated and true problem entropy is often under 1%.



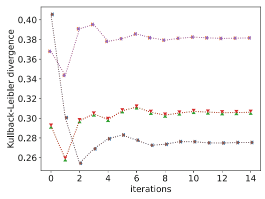
5-queens, no damping



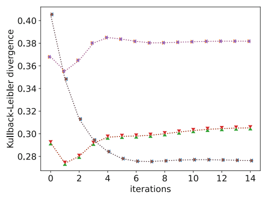
5-queens, $\lambda = 0.5$



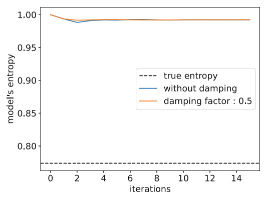
5-queens



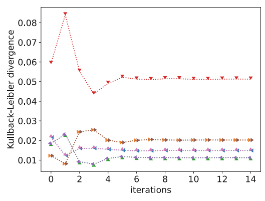
6-queens, no damping



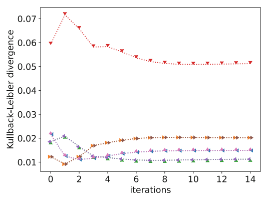
6-queens, $\lambda = 0.5$



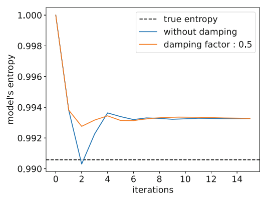
6-queens



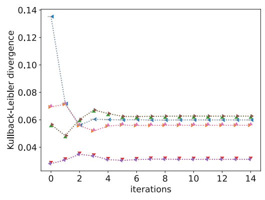
7-queens, no damping



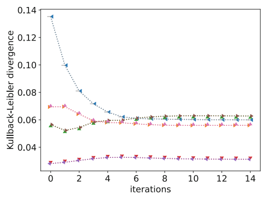
7-queens, $\lambda = 0.5$



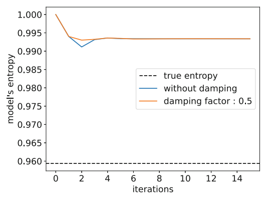
7-queens



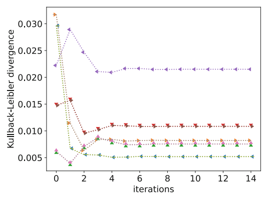
8-queens, no damping



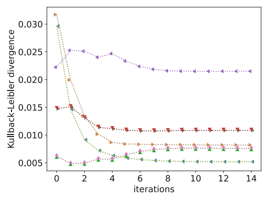
8-queens, $\lambda = 0.5$



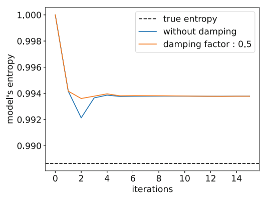
8-queens



9-queens, no damping



9-queens, $\lambda = 0.5$



9-queens

Fig. 2. KL-divergence of variable marginals and entropy as we iterate belief propagation for instances of n-queens.

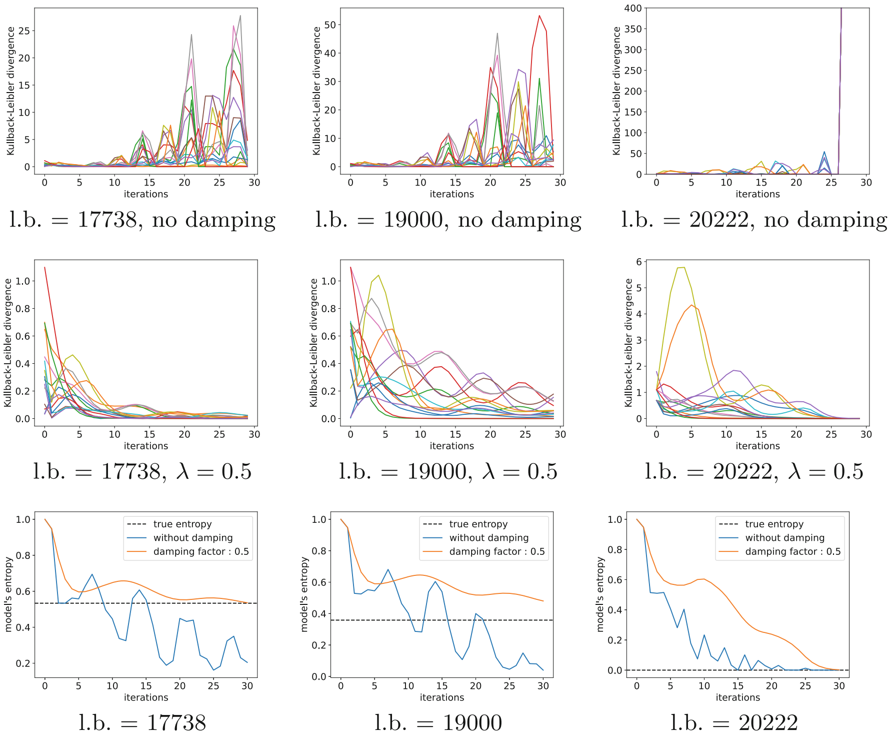


Fig. 3. KL-divergence of variable marginals and entropy as we iterate belief propagation for instances of Feature Model.

For Feature Model, which is a maximization problem, we derive three instances of a CSP by bounding the objective: lower bounds 17738, 19000, and 20222 (its optimal value) respectively admit 95, 15, and 1 solutions. Each instance has 15 unbound variables. The results are shown at Fig. 3. Here damping has a dramatic effect: without it the divergence of many variable marginals oscillates with increasing amplitude whereas with damping the divergence may still oscillate but with much smaller amplitude and tends to converge to a low value. Note also that for a few variables the divergence quickly stabilizes to a near null value. And in the case of the single-solution instance (right column) iterated BP actually identifies that solution. The divergent behaviour without damping appears as an oscillation in the observed problem entropy whereas the latter is smoother and even sometimes stable for the better-performing damping. Observe also how the estimated entropy with damping appears to converge to the true entropy.

Lastly we turn to instances with a larger number of variables and a single solution. The Sudoku instance we use at Fig. 4 features 33 unbound variables after constraint propagation. Without damping, severe oscillation occurs for many of

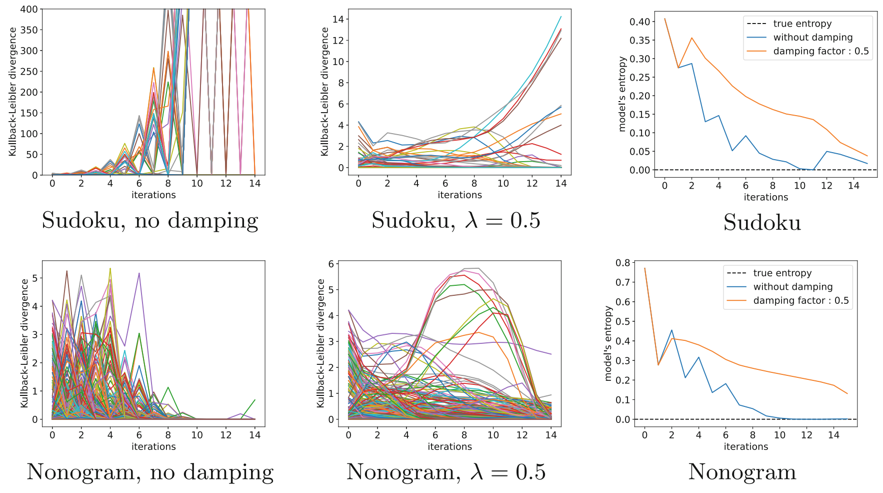


Fig. 4. KL-divergence of variable marginals and entropy as we iterate belief propagation for instances of Sudoku and Nonogram.

the variable marginals. It is again accompanied by an oscillation of the entropy, though less pronounced. Damping is very useful to keep the marginals under control except for a few which start to diverge around Iteration 10. The Nonogram instance has 444 unbound variables out of 576. In contrast with the previous instance no damping behaves better: it even momentarily stabilizes to the solution around Iteration 12. Another difference is that entropy without damping oscillates until Iteration 6, a behaviour that had coincided with increased divergence in the previous instances.

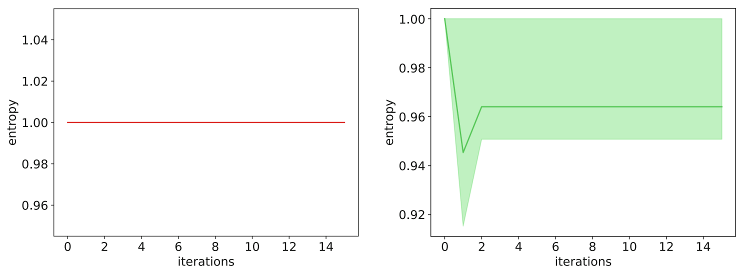
So, damping is not always better and entropy oscillation does not necessarily signal that we should use damping. But according to this limited empirical investigation damping generally helps much more than it hurts: for Feature Model and Sudoku it avoids very large (50) or even infinite divergence; for Nonogram the divergence with damping never strays above 6.

4 Exploiting Entropy

Now that we have empirical evidence for the accuracy of the computed marginals and entropy, we propose in this section several uses for such information.

4.1 Deciding When to Use BP

In their empirical evaluation of search based on BP for CSPs, Babaki et al. [1] reported two problems on which the approach performed particularly badly:



(a) Dubois instance (b) PigeonPlus instance

Fig. 5. Evolution of problem entropy during BP for two problematic instances.

Dubois and PigeonsPlus[2]. Both feature unsatisfiable instances but more importantly it was noticed at the time that the computed marginals were close to being uniform. Figure 5 confirms that in both cases the computed entropy stagnates at a value close to 1. We can turn this observation into a criterion to decide when problem entropy should be used to help solve an instance and when computationally expensive BP should be interrupted instead and replaced by a cheaper variable ordering heuristic, at least until useful information can be inferred again to guide search.

4.2 Deciding When to Stop BP Iterations

Variations of entropy give us information about the variations of marginals. If the entropy of a variable undergoes important variations along BP iterations, the marginals of this variable are varying too. It may mean that we shouldn't stop BP just yet. Based on this idea, we design a dynamic criterion to decide at each search-tree node when we should stop BP iterations. This criterion is based on the variations of the problem entropy $H(P)$. After iteration t, we compare the current entropy $H_t(P)$ to the entropy at the previous iteration $H_{t-1}(P)$. If $0 \leq H_{t-1}(P) - H_t(P) \leq \alpha$, for some threshold α, BP iterations are stopped and a branching decision is taken. This difference must be positive: otherwise it means that the entropy is increasing and that we shouldn't stop BP.

Another potential criterion is to look at the value of the smallest variable entropy. If this entropy becomes very low, we can consider this variable as almost decided, because we have strong knowledge about the value it should take. Thus, additional BP iterations are unnecessary and the variable should be branched on. Another incentive to do so is that in the next few iterations this variable will likely have all its marginals at zero except for the one value, which we observed will have a cascading effect on several other variables, dropping their entropy close to zero as well, which will make it harder to discriminate between the variable at the origin of this phenomenon and the other variables when deciding which one to branch on.

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

4.3 Deciding When to Activate Damping

But perhaps the problem entropy never stabilizes and so does not meet our first stopping criterion. We saw in Sect. 3 that marginals may sometimes oscillate with increasing amplitude — which can be signaled by an oscillating entropy — and that damping can alleviate this issue. However damping is not necessarily desirable and can in some cases slow down convergence to the true marginals, as in the case of Example 1. As an alternative to activating damping by default, we will investigate starting BP without damping and switching it on whenever such entropy oscillation is observed.

4.4 Branching to Search for a Solution

The lower a variable's entropy, the stronger the information about which value the variable should take in a solution. Hence entropy is a powerful tool that we can exploit to make better branching decisions. We introduce variable ordering heuristic *min-entropy* that selects the variable with the lowest entropy and first tries fixing it to its domain value with the strongest marginal. Notice that, if the marginal distributions are uniform (i.e. we have no discriminating information between domain values), the variable with the lowest entropy will be the one with the smallest domain. Therefore, we can consider that *min-entropy* is a generalization of standard *smallest-domain* where we can discriminate between domain values based on the CSP.

5 Experimental Evaluation

In this section, we evaluate the quality of our resulting search strategy. In order to position our work with respect to the state of the art, we compare its performance to the *dom/wdeg* [2] and *IBS* [7] heuristics, and to another heuristic based on marginals and BP, *max-marginal*. Our metrics are the number of fails, which shows the accuracy of a heuristic, i.e. how good are the branching decisions, and the runtime, which indicates if the extra cost induced by our heuristics still makes them worthwhile.

5.1 Experimental Protocol

We ran our experiment on a set of 1319 instances from XCSP3[3] and the Minizinc Challenge[4]. One limitation of MiniCPBP is that it needs to store each value in the domain of each variable, and each corresponding marginal. Therefore when variables have very large domains this can be very space-consuming. We selected problems where variables have reasonable-size domains: summing over the variables, our largest instance has about 810 000 domain values. Our other criterion for problem selection was the constraints in the model. We chose problems where,

[3] Availables at http://www.xcsp.org/instances/.
[4] Available at https://www.minizinc.org/challenge2022/mznc2022_probs.tar.gz.

for the majority of the constraints present, our solver provides a weighted counting algorithm, in order to have a meaningful observation of the contribution of BP. The experiments were performed on a server with two Intel E5-2683 v4 Broadwell @ 2.1 GHz. We used the solver MiniCPBP[5], which is implemented over MiniCP [4] and is able to perform BP. Each run had a 20-min timeout and up to 12 GB of memory available.

Our results are presented as performance profiles: each point of a graph shows the proportion of instances (given on the y axis) that are solved with a number of failures or runtime less than or equal to the value on the x axis. We compare *min-entropy* with *max-marginal* during depth-first search (DFS), to see if entropy is a better exploitation of the marginals. Before each branching decision, unless indicated otherwise, five iterations of belief propagation are performed (the current default number in MiniCPBP). To avoid the oscillation of marginals, we apply damping on the messages sent during BP with a damping factor $\lambda = 0.5$.

As an attempt to improve basic *min-entropy* and as described in Sect. 4, we consider a dynamic configuration where the use of damping and the number of BP iterations are dynamically decided during the search. BP is stopped when the variation of the problem entropy is lower than threshold $\alpha = 0.1$ (see Sect. 4.2): experiments showed a strong variance on the best value for α depending on the problem but that parameter value generally performs well on our benchmark problems. In Sect. 4.2, we describe another stopping criterion which stops BP iterations when the smallest entropy among variables falls under a threshold. We tested it with different threshold values and it showed better performance than the criterion based on the entropy's variations on some problems, but the latter remained the best choice overall. At each search-tree node, BP is performed at first without damping and if oscillations in the problem entropy are detected, damping is activated (with $\lambda = 0.5$ as before). To detect oscillations, we count how many times we observe a decreasing entropy starting to increase: if it switches 3 times from a negative variation of the entropy to a positive variation, we activate damping for the rest of the propagation. After the branching decision, damping is deactivated again.

As a state-of-the-art reference, we use *dom/wdeg* with restarts (initial restart: $3n$ failures, where n is the number of variables in the problem; increased by a factor 1.4 after each restart) and Impact-Based-Search (*IBS*) [7], also with restarts (initial restart: $2n$ failures; increased by a factor 1.2 after each restart). For *IBS*, to initialize impacts before the search, we try each possible assignment and register its impact.

5.2 Evaluation

According to Fig. 6 *min-entropy* shows better performance than *max-marginal* on opt-cryptoanalysis, MagicSquare and MagicHexagon. For the other problems,

[5] Solver and used instances are available at https://github.com/PesantGilles/MiniCPBP.

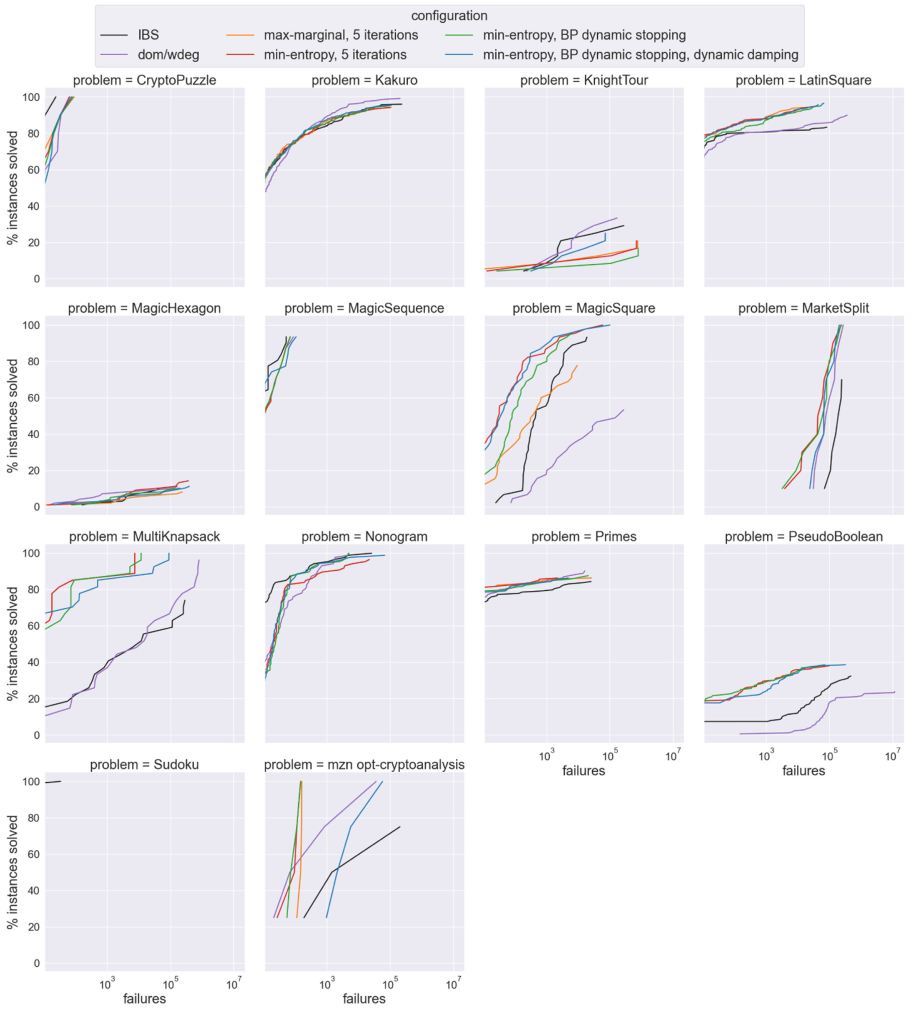


Fig. 6. % instances solved vs #fails for several branching heuristics

we observe similar performance for these two heuristics. The binary domains in Nonogram, PseudoBoolean, MultiKnapsack and MarketSplit explain that we observe identical performance between *max-marginal* and *min-entropy* (i.e. the orange and red curves coincide): with such domains, the variable that presents the strongest marginal is also the one with the lowest entropy. We can therefore conclude that *min-entropy* is a good improvement of marginal usage.

If we now compare our heuristics to *dom/wdeg* and *IBS*, we see that we outperform them on five problems in our dataset (LatinSquare, MagicSquare, MultiKnapsack, PseudoBoolean, and opt-cryptoanalysis). For three problems some of the state of the art is showing better performance: for KnightTour

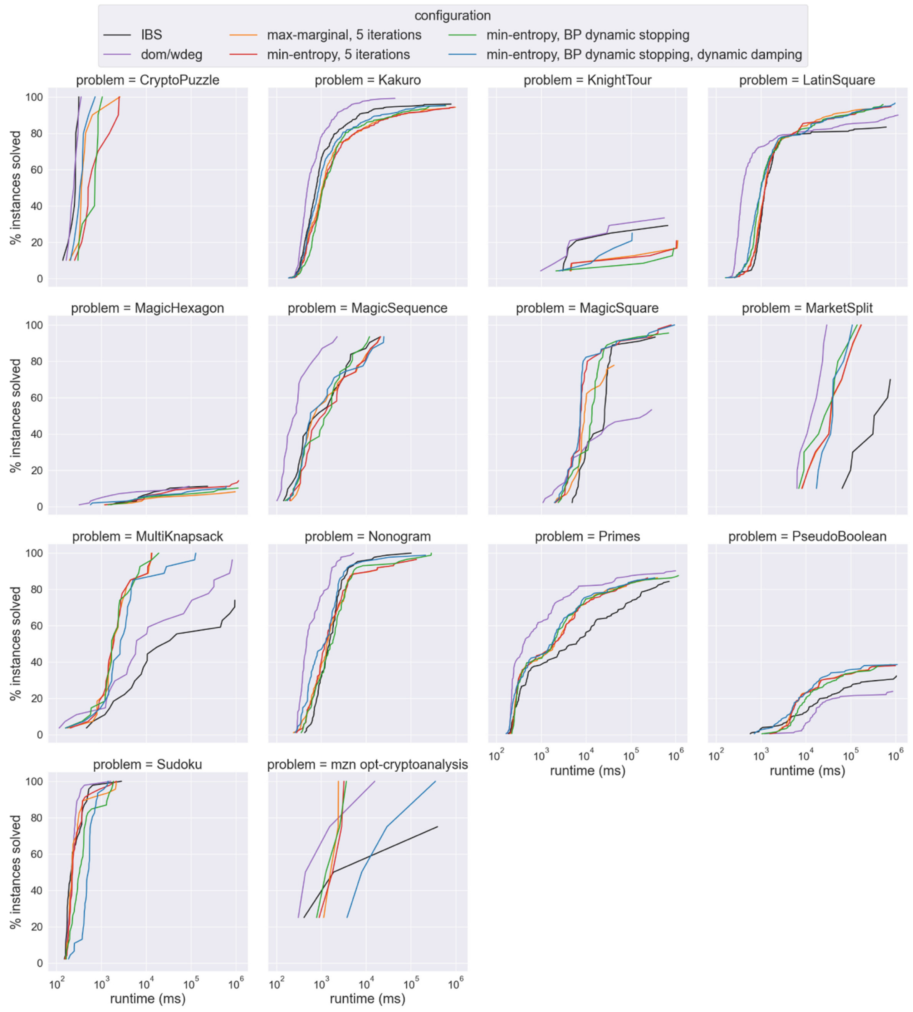


Fig. 7. % instances solved vs runtime (ms) for several branching heuristics

both *dom/wdeg* and *IBS* outperform *min-entropy*, for *Kakuro* only *dom/wdeg* is better, and for *CryptoPuzzle* only *IBS* is better. On the remaining six problems heuristics perform similarly. Based on these results, exploiting entropy to make branching decisions is a competitive approach.

Let's take a closer look at MarketSplit and MultiKnapsack. They are interesting because they present similar structure, i.e. their variables are binary and they only contains `sum` constraints applied on all variables. However, we observe a strong difference in the performance of our heuristic between these two problems. On MultiKnapsack, our heuristics clearly outperform *dom/wdeg* and *IBS*, whereas on MarketSplit the results are more mixed. If we compare the entropy of the variable in each problem, the reason is clear: if in MultiKnapsack they quickly decrease, in MarketSplit they often stay above 0.9. MarketSplit is thus a good example of a problem where BP is not as informed, as we discussed in Sect. 4.1, and typically we would detect this and then decide to use another branching heuristic.

We now turn to runtime to evaluate the cost of adding a second kind of propagation at each node of the search tree. If we look at Fig. 7 and compare with Fig. 6 we can observe this additional cost. If we encountered fewer failures with *min-entropy* and *max-marginal* than *dom/wdeg* for problems like MarketSplit and MagicSquare, we observe similar runtimes for these problems. For Kakuro, Nonogram and MagicSequence, the performance in terms of failures was similar, but *dom/wdeg* outperforms the other heuristics in terms of runtime. Sudoku is a particular problem because it is the only one for which we don't observe any additional cost. This is because the majority of instances are solved during the first constraint propagation, before the use of belief propagation. Despite this additional cost, our heuristics still outperform *dom/wdeg* and *IBS* on PseudoBoolean and MultiKnapsack. And for LatinSquare, *max-marginal* and *min-entropy* are able to solve more instances. With some optimization, this approach could be a good option for a wide spectrum of problems.

Speaking of BP optimization, we should now look at the performance of our *dynamic* strategies. From Fig. 6 we observe that the heuristic quality is not strongly impacted by the use of the dynamic parameters. We observe a small degradation of performance for PseudoBoolean. This deterioration is due to dynamic damping, because the configuration that is only using dynamic stopping for BP shows performance as good as static min-entropy. This degradation is stronger for MultiKnapsack and opt-cryptoanalysis and is once again mainly due to dynamic damping. On the contrary for KnightTour and Nonogram we observe a significant reduction of the number of failures. But the primary goal of the dynamic stopping criterion is to improve runtime by choosing the best moment to stop BP and thus spare useless iterations. At Fig. 7 we observe an improvement for CryptoPuzzle, Kakuro, PseudoBoolean, and KnightTour. Concerning the latter, the improvement is connected to the reduction of failures. For the other problems, the improvement is not as significant. By using this criterion, our goal is to spare a few iterations at each node of the search tree when it is adequate. Therefore, the reduction of runtime will be linear, which is less noticeable on a logarithmic scale. Finally we observe a small deterioration for Sudoku, for MultiKnapsack, which is linked to the increase of failures, and for MarketSplit. A limitation of our dynamic stopping criterion is the variability of the best value for α, as we mentioned in Sect. 4.2. We chose the value that showed the most stability in its performance, but it was not the best configuration for all problems, and for some problems, like MagicSquare, we observed significantly better performance by using our other stopping criterion, which is stopping BP when the smallest entropy falls under a threshold. In conclusion, using a dynamic number of iterations shows good potential to reduce the runtime, but it requires more work to find a stopping criterion that would show improvement consistently on a diverse set of problems. Concerning dynamic damping, we observe a slight improvement for *KnightTour* in Fig. 6, but otherwise using it shows similar or worse performance. Further work is required to find a better criterion for detecting the need of damping.

6 Conclusion

We investigated the entropy of CSPs and of their finite-domain variables, made possible by estimating marginal distributions using the CP-BP framework on constraint programming models. Our study showed that these estimated distributions can get quite close to the true distributions, and that message damping during BP may be necessary to obtain convergence. We proposed several ways to exploit entropy in order to help solve combinatorial problems. Our experiments on 1319 instances from 14 different problems showed that branching on the variable with the lowest entropy is an insightful variable-ordering strategy.

Because performing belief propagation does come with a computational cost, we considered two stopping criteria to decide dynamically when BP iterations should stop in an effort to avoid unproductive work. The experiments showed that such stopping criteria have potential but require further refinements in order to be robust across problems.

We also considered when message damping should be activated. Instead of turning it on with a fixed damping factor, we tried to adjust the use of damping dynamically according to the observed effect on entropy oscillation. Experimental results show that further work is required to find a more accurate criterion.

A possible pragmatic improvement could be not to use BP at each node of the search tree. If a branching decision has a very small impact on domains, we can consider that it would have a very small impact on the marginals. Therefore we could reuse the marginals computed at the previous node and spare a mostly redundant phase of belief propagation. A similar approach gave significant acceleration when applied to *maxSD* [3], a branching heuristic based on solution counting. Thus, it is a promising idea that could be explored in future work.

References

1. Babaki, B., Omrani, B., Pesant, G.: Combinatorial search in CP-based iterated belief propagation. In: Simonis, H. (ed.) CP 2020. LNCS, vol. 12333, pp. 21–36. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58475-7_2
2. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: de Mántaras, R.L., Saitta, L. (eds.) Proceedings of the 16th European Conference on Artificial Intelligence, ECAI 2004, Including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, 22–27 August 2004, pp. 146–150. IOS Press (2004)
3. Gagnon, S., Pesant, G.: Accelerating counting-based search. In: van Hoeve, W.-J. (ed.) CPAIOR 2018. LNCS, vol. 10848, pp. 245–253. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93031-2_17

4. Michel, L., Schaus, P., Van Hentenryck, P.: MiniCP: a lightweight solver for constraint programming. Math. Program. Comput. **13**(1), 133–184 (2021). https://doi.org/10.1007/s12532-020-00190-7
5. Pearl, J.: Reverend Bayes on inference engines: a distributed hierarchical approach. In: Proceedings of the National Conference on Artificial Intelligence, Pittsburgh, PA, 18–20 August 1982, pp. 133–136 (1982). http://www.aaai.org/Library/AAAI/1982/aaai82-032.php
6. Pesant, G.: From support propagation to belief propagation in constraint programming. J. Artif. Intell. Res. **66**, 123–150 (2019). https://doi.org/10.1613/jair.1.11487
7. Refalo, P.: Impact-based search strategies for constraint programming. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 557–571. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30201-8_41
8. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**(3), 379–423 (1948). https://doi.org/10.1002/j.1538-7305.1948.tb01338.x
9. Vavrille, M., Truchet, C., Prud'homme, C.: Solution sampling with random table constraints. In: Michel, L.D. (ed.) 27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), 25–29 October 2021. LIPIcs, vol. 210, pp. 56:1–56:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.CP.2021.56