Exploiting Model Entropy to Make Branching Decisions in Constraint Programming

Auguste Burlats
Polytechnique Montréal, Canada

Gilles Pesant
Polytechnique Montréal, Canada

Abstract

Branching decisions have a strong impact on performance in Constraint Programming (CP). Therefore robust generic variable ordering heuristics are an important area of research in the CP community. By allowing us to compute marginal probability distributions over the domain of each variable in the model, CP-based Belief Propagation also allows us to compute the entropy of each variable. We present two new branching heuristics exploiting entropy: one chooses the variable with the smallest entropy; the other is inspired by impact-based search and chooses the variable whose fixing is believed to have the strongest impact on the entropy of the model. We also propose a dynamic stopping criterion for iterated belief propagation based on variations in model entropy. We test our heuristics on various constraint satisfaction problems from XCSP benchmarks.

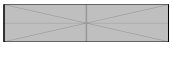
2012 ACM Subject Classification Author: Please fill in 1 or more \ccsdesc macro

Keywords and phrases Constraint Programming, Variable Ordering heuristic, Belief Propagation, Entropy

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Constraint Programming (CP) shows good performance to solve combinatorial problems. Indeed it strongly reduces 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. But the order in which variables are fixed also has a significant impact on the search space. This is why finding robust and generic variable ordering heuristics is crucial. The smallest-domain (*dom*) variable-ordering heuristic is the classic example of a generic heuristic showing good results [4]. Some heuristics observe the consequence of an assignment and make branching decisions based on those observations. *dom/wdeg* [2] observes which constraints cause failures. It improves *dom* by adding a weight to favor variables in the scope of those constraints. *Impact-Based Search* [10] fixes the variable that is susceptible to lead to the largest reduction of the search space. *Activity-Based Search* [6] first assigns the variables whose domain is the most frequently reduced by other assignments. *Conflict-History-Based Search* [3] favors variables that are the most often involved in recents failures. *crbs-max* [12] observes the correlations between variables, i.e. the possibility of conflict between those variables, and fixes the variable that presents the strongest correlation with another variable. *MaxSD* [9] computes for every possible assignment its *solution density* in each constraint, i.e. the proportion of valid local solutions in which it appears, and branches on the assignment of maximum solution density. Finally, Belief Propagation makes possible the computation of an estimation of a global solution density for each assignment by computing the marginal distribution over the variables' domains [8]. *Max-marginal* branches on the assignment with the largest marginal. But computing marginal probability distributions also allows us to compute the entropy of each variable. In this paper we present two new variable ordering heuristics that exploit this entropy. We also show that it is possible to determine when BP


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

should stop by observing the variation of a model's entropy.

2 Preliminaries

2.1 Constraint Satisfaction Problem

A *Constraint Satisfaction Problem* (*CSP*) is a problem defined by a triplet $< X, D, C >$ where :

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

The goal is to find for each variable $x_i \in X$, an assignment from $D(x_i)$ that doesn't break any constraint from C.

2.2 Belief Propagation for CSPs

Belief Propagation (BP) is an algorithm introduced by Pearl [7]. It is able to compute the marginal distribution for each non-observed node in a Markov Random Field, conditioned by the value of the observed nodes.

Pesant [8] introduced a framework combining CP and BP. 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 $\mu_{c \to x}$ the message from constraint c to variable x, and $\mu_{x \to x}$ the message from variable x to constraint c. Their definition is :

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

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 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 compute the *marginal* of a variable x :

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

Messages are sent iteratively. First, all variables send their messages. Then it is for all constraints to send their messages. This cycle is repeated for a fixed number of iterations. This paper offers a solution to make this number dynamically decided at each node of the search tree. Computing $\sum_{\mathbf{v}:\mathbf{v}[x]=v} f_c(\mathbf{v})$ is equivalent to counting solutions (locals to c) where $\mathbf{v}[x] = v$. Therefore computing messages from constraints is similar to performing weighted counting. Pesant [8] provided efficient dedicated algorithms for weighted counting on several constraints.

Let's examine a small example from [8] to illustrate the behavior of marginals :

▶ **Example 1.** Consider variables a, b, c and d with $D(a) = D(b) = D(c) = D(d) = \{1, 2, 3, 4\}$, and the following constraints :

(1) $alldifferent(a, b, c)$

(2) $a + b + c + d = 7$

(3) $c \leq d$

	1	2	3	4
θ_a	0	.50	.50	0
θ_b	0	.50	.50	0
θ_c	1	0	0	0
S θ_d	1	0	0	0

	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	.01	.52	.46	.01
θ_b	.01	.52	.46	.01
θ_c	.98	.02	.00	.00
θ_d	.90	.10	.00	.00

Table 1 True marginals (top left), initial marginals (top right), marginals after 1st iteration of BP (bottom left) and after 10th iteration (bottom right) for Example 1

This CSP has two solutions : $(a = 2, b = 3, c = 1, d = 1)$ and $(a = 3, b = 2, c = 1, d = 1)$. 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: $\theta_{x_i}(v) = 1/|D(x_i)|, \forall v \in D(x_i), \forall x_i \in X$. And, as we can see in Table 1, BP tends to converge to the true marginal distributions after a few iterations.

BP is assured to converge when there is no cycle in the graph [7]. But the graphical representation of a CSP often contain such cycles. However, the large arity of global constraints allows us to encapsulate those cycles and perform efficient inference [8]. The *Max-marginal* [1] heuristic exploits those marginals by branching on the variable x_i that maximizes $max_{v \in D(x_i)}(\theta_{x_i}(v)))$, assigning it the value with the strongest marginal.

3 Exploiting Entropy

Because Belief Propagation allows us to compute marginal distributions for each domain, we are also able to compute the entropy of each variable. Entropy is a powerful estimation of the knowledge that we have on a variable. The lower its entropy, the stronger the information about which value the variable should take. Thus, entropy is a powerful tool that we can exploit to make better branching decisions.

3.1 Min-Entropy

We define the entropy $H(x)$ of variable x using Shannon's entropy [11] :

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

Branching heuristic *Min-entropy* selects the variable with the lowest entropy, and the 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 would be the one

with the smallest domain. Therefore, we can consider that *min-marginal* is a generalization of *dom* where we can discriminate between domain values based on the CSP.

3.2 Using Entropy as a Dynamic Stopping Criterion for BP

Variations of entropy gives 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 means that we shouldn't stop BP.

Based on this idea, we designed 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 entropy of the model $\overline{H}$. We compute the model's entropy as the mean of variables' entropies :

$$\overline{H} = \frac{\sum_{x \in X} \frac{H(x)}{log(|D(x)|)}}{|X|}$$

Notice that the entropy of each variable is divided by the logarithm of the size of the domain. Thus we are assured that $\overline{H} \in [0, 1]$ regardless of the domains' size. Otherwise, $\overline{H}$ would be very different at the beginning and at the end of the search. Therefore the threshold α that we will introduce could become less relevant as search progresses.

After each iteration of BP, we compare the current entropy $\overline{H}$ to the entropy at the previous iteration $\overline{H}'$. If $0 \leq \overline{H}' - \overline{H} \leq \alpha$, 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. The value of α can be fixed by the user. Experiments showed that $\alpha = 0.01$ gives good results on a large spectrum of problems.

3.3 Impact-entropy

Impact-Based Search[10] (*IBS*) shows good results by basing its branching decisions on the expected impact of these decisions. But this heuristic only looks at impacts on the size of the search space, approximated by the size of the Cartesian product of the domains. When a variable is fixed, other variables' entropy is also impacted. Therefore exploiting this impact could lead to good branching decisions.

When a solution is found, $\overline{H}$ is null. We can thus consider that the objective is to minimize $\overline{H}$. And that it is preferable to first fix variables that are the most susceptible to reduce this entropy.

Each time a variable x is fixed to a value a, we measure the impact on $\overline{H}$:

$$I(x = a) = 1 - \frac{\overline{H}_{after}}{\overline{H}_{before}}$$

where $\overline{H}_{after}$ is the entropy after the assignment and $\overline{H}_{before}$ is the entropy before.

The impact of an assignment is the mean of the observed impacts :

$$\overline{I}(x = a) = \frac{\sum_{k \in K} I^k(x = a)}{|K|}$$

where K is the set of observed impacts for assignment $x = a$.

To compute the impact of a variable, we compute the mean of the impact of each possible assignment for this variable :

$$I(x) = \frac{\sum_{v \in D'(x)} \overline{I}(x = v)}{|D'(x)|}$$

where $D'(x) \subset D(x)$ is the set of values that are still possible assignments at the current node.

According to the *Fail-first Principle*, *Impact-entropy* chooses the variable with the strongest impact. It chooses the value with the strongest marginal.

4 Experimental Evaluation

In this section, we evaluate the quality of our search strategies. In order to position our work in the state-of-the-art, we compare the performances of our heuristics with the *dom/wdeg* and *IBS* heuristics. Our metrics are the number of fails, which shows the quality of the heuristics, i.e. how good are the branching decisions, and the runtimes, which indicates if the extra cost induced by our heuristics still makes them viable.

4.1 Experimental Protocol

We ran our experiment on a set of 1033 instances of various problems from XCSP3[1]. We track the performance of *min-entropy* and *impact-entropy* performing a Depth-First Search. For *impact-entropy* and *IBS* we use restarts, with a cutoff set at 100 failures. After each restart, the cutoff is multiplied by 1.5. For each branching strategy, we compare a configuration with a number of BP iterations fixed at 5 and a configuration where the number of iterations is dynamically selected by our stopping criterion with $\alpha = 0.01$. The experiments are performed on a server with two Intel E5-2683 v4 Broadwell @ 2.1Ghz. We use the solver MiniCPBP [2], which is implemented over MiniCP[5] and is able to perform *Belief Propagation*. Each experiment has a 20-minute timeout and up to 12GB of memory available.

4.2 Results

Figure 1 shows the proportion of instances solved within a number of failed search tree nodes by each configuration. Globally, *min-entropy* seems to be the best strategy : it shows best performances for *LatinSquare*, *MagicSquare* and *Primes*, very closely followed by *max-marginal*. And *min-entropy* clearly outperforms the other heuristics for *MagicSquare*. It shows that exploiting entropy is a promising approach for search strategies.

Impact-entropy does not perform as well as expected, except for *Nonogram*. Moreover, as shown in Figure 2, the initialisation of impacts has an important computational cost. Therefore it is not an interesting strategy yet. Finding another estimation for the impact of a variable could improve performance. We can notice that, because our search strategies are better for *MagicSquare*, *LatinSquare* and *MultiKnapsack*, we are competitive in runtime with *dom/wdeg* and *IBS* despite the additional computational cost of the marginals. The dynamic stopping criterion for BP iterations often gives similar search guidance to the configuration with a fixed number of iterations. The exceptions are *MagicSquare* and *Primes*, where using the stopping criterion with *min-entropy* and *max-marginal* results in an increase of the number of failures. And it is also able to reduce the runtime, as we can observe it in the performances for *Kakuro*, *MultiKnapsack* and *LatinSquare*. Because of the bad performances on *MagicSquare* and *Primes* we cannot conclude that it is a reliable stopping criterion yet. But it shows a good potential for reducing the computational cost of *BP*.

[1] Availables at http://www.xcsp.org/instances/
[2] Available at https://github.com/PesantGilles/MiniCPBP

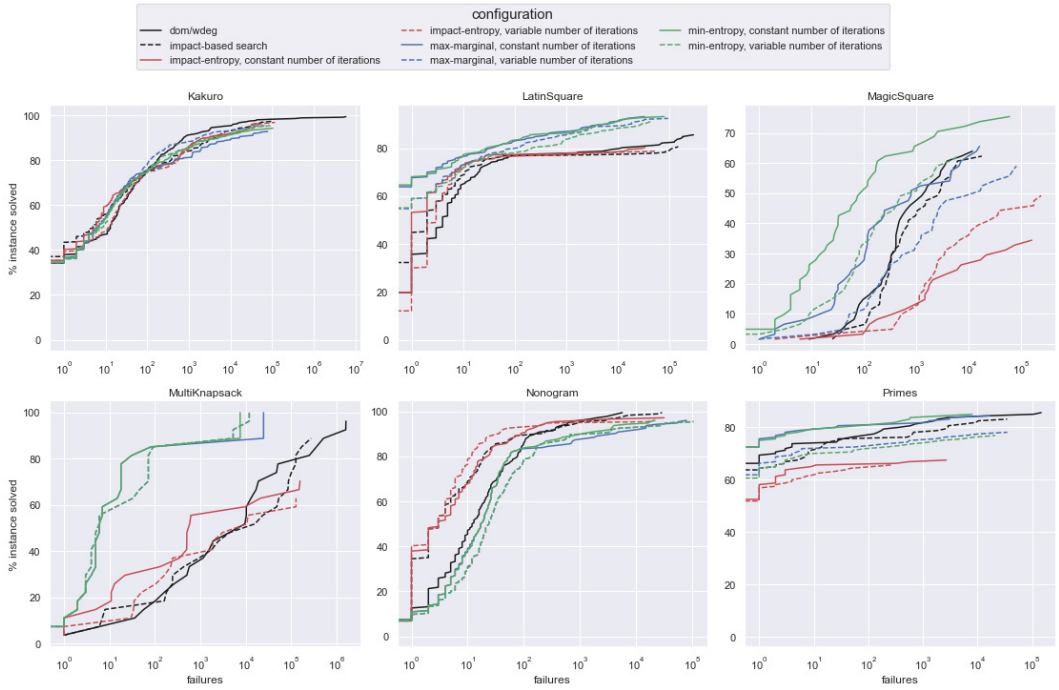


Figure 1 % instances solved vs #fails for our heuristics against *dom/wdeg*

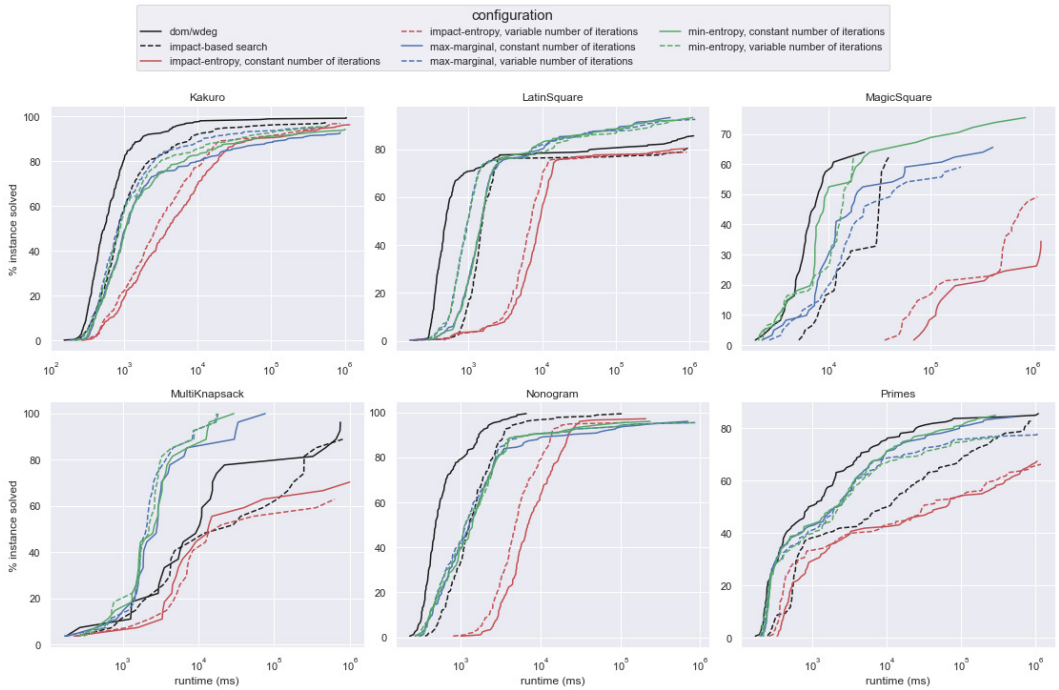


Figure 2 % instances solved vs runtimes (ms) for our heuristics against *dom/wdeg*

5 Conclusion

We proposed two search strategies that exploit entropy. The first one, *min-entropy*, fixes the variables with the lowest entropy first. The second one, *impact-entropy*, fixes the variable that is supposed to induce the strongest reduction on the entropy of the model. We also propose a dynamic stopping criterion for Belief Propagation based on the variation of the model's entropy. We compare our heuristics with *max-marginal*, *dom/wdeg* and *Impact-Based Search*. Results show that *min-entropy* is better to guide search than state-of-the-art *dom/wdeg* and *IBS* on the instances considered. Our stopping criterion shows good potential for reducing the overall cost of *BP* but still needs some refinement to be able to preserve good performances on all problems. Finally *impact-entropy* seems to be a less promising strategy.

References

1. Behrouz Babaki, Bilel Omrani, and Gilles Pesant. Combinatorial search in cp-based iterated belief propagation. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming*, pages 21–36, Cham, 2020. Springer International Publishing.
2. Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. pages 146–150, 01 2004.
3. Djamal Habet and Cyril Terrioux. Conflict history based search for constraint satisfaction problem. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, page 1117–1122, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3297280.3297389.
4. Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980. URL: https://www.sciencedirect.com/science/article/pii/000437028090051X, doi:https://doi.org/10.1016/0004-3702(80)90051-X.
5. L. Michel, P. Schaus, and P. Van Hentenryck. Minicp: a lightweight solver for constraint programming. *Mathematical Programming Computation*, 13(1):133–184, 2021. doi:10.1007/s12532-020-00190-7.
6. Laurent Michel and Pascal Van Hentenryck. Activity-based search for black-box contraint-programming solvers. volume abs/1105.6314, 05 2011. doi:10.1007/978-3-642-29828-8_15.
7. Judea Pearl. *Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach*, page 129–138. Association for Computing Machinery, New York, NY, USA, 1 edition, 2022. URL: https://doi.org/10.1145/3501714.3501727.
8. Gilles Pesant. From support propagation to belief propagation in constraint programming. volume 66, pages 123–150, 2019. doi:10.1613/jair.1.11487.
9. Gilles Pesant, Claude-Guy Quimper, and Alessandro Zanarini. Counting-based search: Branching heuristics for constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 43:173–210, 2012.
10. Philippe Refalo. Impact-based search strategies for constraint programming. In Mark Wallace, editor, *Principles and Practice of Constraint Programming – CP 2004*, pages 557–571, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
11. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948. doi:10.1002/j.1538-7305.1948.tb01338.x.
12. Ruiwei Wang, Wei Xia, and Roland H. C. Yap. Correlation heuristics for constraint programming. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1037–1041, 2017. doi:10.1109/ICTAI.2017.00159.