Combinatorial Search in CP-Based Iterated Belief Propagation

Behrouz Babaki[1], Bilel Omrani[2], and Gilles Pesant[2(✉)]

[1] HEC Montréal, Montreal, Canada
behrouz.babaki@hec.ca
[2] Polytechnique Montréal, Montreal, Canada
{bilel.omrani,gilles.pesant}@polymtl.ca

Abstract. Compared to most other computational approaches to solving combinatorial problems, Constraint Programming's distinctive feature has been its very high-level modeling primitives which expose much of the combinatorial substructures of a problem. Weighted counting on these substructures (i.e. constraints) can be used to compute beliefs about certain variable-value assignments occurring in a solution to the given constraint. A recent proposal generalizes the propagation mechanism of constraint programming to one sharing such beliefs between constraints. These beliefs, even if not computed exactly, can be very revealing for search. In this paper we investigate how best to guide combinatorial search in this CP-based belief propagation framework. We empirically evaluate branching heuristics on a wide set of benchmark constraint satisfaction problems.

1 Introduction

Compared to most other computational approaches to solving combinatorial problems, Constraint Programming's (CP) distinctive feature has been its very high-level modeling primitives which bring out much of the combinatorial structure of a problem explicitly in the statement of the model. These primitives take the form of so-called global constraints [3] which encapsulate powerful algorithms for both inference and search. On the side of inference, filtering algorithms remove a value from the domain of possible values for a variable if it does not appear in any tuple satisfying a given constraint (i.e. is unsupported) and then share that information between constraints by propagating it through common variables. On the side of search, explaining and recording the sequence of decisions that led to a failure avoids wasting search effort by repeatedly rediscovering that same dead-end [4].[1] As well, work on model counting at the level of individual constraints led to counting-based search [10], a family of effective branching heuristics. Generalizing this to weighted counting allowed one to handle combinatorial *optimization* problems in which individual costs are associated with each variable assignment [8]. The close relationship between weighted counting

[1] One could argue that this is also a manifestation of inferring redundant constraints.

H. Simonis (Ed.): CP 2020, LNCS 12333, pp. 21–36, 2020.
https://doi.org/10.1007/978-3-030-58475-7_2

and probabilistic inference had already been pointed out in Chavira and Darwiche [2]. From a CP perspective it can be interpreted for a given constraint as the belief in a certain variable-value assignment computed by summing, over satisfying tuples of the constraint that feature that assignment, the likelihood of each tuple expressed as the combined beliefs of its component assignments. Pesant [9] investigates a richer propagation medium for CP which does not simply share unsupported variable-value assignments but shares instead for each variable-value assignment the corresponding quantitative belief that it occurs in some solution. The resulting process is akin to iterated belief propagation [7].

These computed beliefs can be viewed as approximate marginals onto individual variables of the joint probability distribution defined by the set of solutions. Such information, even if not exact, can be very revealing for search. In this paper we investigate how best to guide combinatorial search given the approximate marginals computed by CP-based belief propagation. We empirically evaluate generic branching heuristics on a wide set of benchmark constraint satisfaction problems.

The rest of the paper is organized as follows. Section 2 recalls the recently proposed CP-BP framework and presents the search heuristics we investigate. Section 3 reports the results of our initial experiment and analyzes them for further improvement. Section 4 investigates two benchmark problems on which we did not perform well. Section 5 investigates an oscillation phenomenon observed on the computed marginals. Section 6 concludes with final recommendations about combinatorial search in the CP-BP framework and points to future directions for research.

2 The CP-BP Framework

In this section we briefly introduce the CP-BP framework and refer the reader to Pesant [9] for details. A *factor graph* is a bipartite graph with variable nodes, factor nodes, and edges between some variable-factor pairs. A *message-passing* algorithm on a factor graph iteratively exchanges messages between variable nodes and factor nodes. The message from a factor node is a real-valued function of the variables it is connected to. The message from a variable node is a real-valued function of that variable. The *sum-product algorithm*, also known as belief propagation, is one instantiation of message passing where the message from a variable to a factor is the product of the messages received from the other factors, and the message from a factor to a variable is essentially a sum of products of messages received from the other variables. It has been used to compute the *marginals* of individual variables for the factored function. That computation is exact for trees but not for general graphs with cycles.

A constraint graph can be viewed as a special case of a factor graph with factors (constraints) issuing 0/1 messages. In that sense standard constraint propagation in CP is a form of message passing, exchanging information about which variable-value assignments are supported by each constraint. The CP-BP framework proposes to go back to real-valued messages providing information about

the likelihood of a given variable-value assignment in a solution to the problem. It does so by performing weighted counting of solutions within each constraint. The approximate marginals resulting from message passing can inform branching heuristics.

To illustrate this CP-BP framework consider the following example taken from [9]: constraints ALLDIFFERENT(a, b, c), $a + b + c + d = 7$, and $c \leq d$, with variables $a, b, c, d \in \{1, 2, 3, 4\}$. This CSP is domain consistent but only admits two solutions: $\langle 2, 3, 1, 1\rangle$ and $\langle 3, 2, 1, 1\rangle$.

In CP-BP message passing is synchronized in two phases: in the first phase constraints receive messages from the variables; in the second phase constraints send messages to the variables, which are multiplied and then normalized. This process is repeated for some number of iterations. Initially variables send identical (i.e. uniform) messages in the first phase. For the second phase, consider variable a that appears in the ALLDIFFERENT and the sum constraints: if we count the proportion of solutions to the former constraint in which a takes value 1, it is $\frac{1}{4}$ and the same is true for the other values in its domain; the proportion from the latter constraint is $\frac{10}{20}$ for value 1 and $\frac{6}{20}$, $\frac{3}{20}$, $\frac{1}{20}$ respectively for the other values $2, 3, 4$. The combined information, obtained by multiplying these "messages" from each of the two constraints to variable a, indicates a strong positive bias (or marginal) toward assigning value 1 to a. The situation is identical for variable b. Variable c appears in the same two constraints and the messages it receives from them are the same as those for a and b, but it also appears in the binary inequality: from that one the proportions are $\langle \frac{4}{10}, \frac{3}{10}, \frac{2}{10}, \frac{1}{10}\rangle$. So there is an even stronger bias toward assigning value 1 to c. Variable d gets conflicting information: $\langle \frac{10}{20}, \frac{6}{20}, \frac{3}{20}, \frac{1}{20}\rangle$ from the sum constraint and $\langle \frac{1}{10}, \frac{2}{10}, \frac{3}{10}, \frac{4}{10}\rangle$ from the binary inequality.

Table 1. Biases (marginals) after ten iterations of iterated belief propagation on our running example (left) and true biases (right).

	1	2	3	4
a	.01	.52	.46	.01
b	.01	.52	.46	.01
c	.98	.02	.00	.00
d	.90	.10	.00	.00

	1	2	3	4
a	0	1/2	1/2	0
b	0	1/2	1/2	0
c	2/2	0	0	0
d	2/2	0	0	0

For the next iteration of message passing and all others that follow, the messages sent by each variable (first phase) reflect its current biases. And from now on, each solution to a constraint that we count (second phase) is weighted by the product of the biases of the individual assignments that make up that solution. Note that typically we do not explicitly enumerate the solutions to a constraint and compute their weight: efficient algorithms specialized for each type of constraint have been designed, much like what has been done for domain filtering.

After ten iterations we have converged to the biases on the left in Table 1, which are quite close to the true ones (Table 1, on the right) derived from the two solutions we previously identified. Because from a factor graph perspective, in a typical CP model each constraint represents a bigger chunk of the problem involving more variables, there tends to be fewer cycles in the graph and it was observed in [9] that iterated belief propagation converges better. Initial experiments suggested that applying constraint propagation followed by a fixed number (five) of BP iterations works well.

2.1 Candidate Search Heuristics

As is commonplace in CP, we consider two-way branching heuristics that build a search tree. We can choose to use the computed marginals in two complementary ways: either we first explore subtrees defined by making highly likely assignments or subtrees defined by excluding highly unlikely assignments. Heuristic max-marginal first tries (left branch) to assign the variable-value pair exhibiting the largest marginal and disallows that assignment upon backtracking (right branch). Heuristic max-strength first tries (left branch) to assign the variable-value pair exhibiting the largest (positive) difference between its marginal and the reciprocal of the domain size (i.e. its *marginal strength*) and disallows that assignment upon backtracking (right branch). To illustrate how these two heuristics may differ, consider variable $x \in \{a, b\}$ with respective marginals $\langle .8, .2\rangle$ and variable $y \in \{a, b, c, d\}$ with marginals $\langle .7, .1, .1, .1\rangle$. Heuristic max-marginal will first branch on $x = a$ whereas max-strength will branch on $y = a$ since $.7 - 1/4 = .45 > .8 - 1/2 = .3$. Heuristic min-marginal identifies the variable-value pair exhibiting the smallest marginal, first removes that value from the domain of that variable and then assigns it upon backtracking. Heuristic min-strength identifies the variable-value pair exhibiting the smallest marginal strength, first removes that value from the domain of that variable and then assigns it upon backtracking.

3 Initial Experiments

The recommendations about search in [9], namely branching in a depth-first search tree according to maximum marginal strength (max-strength), were based on a small set of problems and instances. We first wish to revisit this on a larger and more diverse set of benchmark problems. The current implementation of MiniCPBP[2] dictates that these should be constraint satisfaction problems[3] on integer variables and with constraints among allDifferent, sum, (positive) table, regular, among, and cardinality. The `xcsp.org` website features a selection engine that enables one to select instances from a large set according to such requirements. We selected the following problems (excluding some

[2] Available at https://github.com/PesantGilles/MiniCPBP.

[3] The search guidance provided by its branching heuristics currently ignores solution cost and so it is unlikely to perform well on constraint optimization problems.

series of instances that were trivially solved by all heuristics or that caused an out-of-memory error): CarSequencing, Dubois, Kakuro (sumdiff/medium), LatinSquare (m1, xcsp2/bqwh⋆), MagicSquare (sum/s1), Partially-Filled MagicSquare (m1/gp), MultiKnapsack, Nonogram (regular), Ortholatin, PigeonPlus, Primes, and PseudoBoolean (dec). In all, 1336 instances. The experiments were executed on Intel E5-2683 2.1 Ghz machines running CentOS Linux 7. The memory budget was set to 10 GB and the timeout to 10 h.

3.1 Comparison Between Our Candidate Branching Heuristics

Figure 1 compares the search guidance of the four heuristics presented in Sect. 2.1 using depth-first search (DFS) and limited-discrepancy search (LDS). Each plot presents for a benchmark problem the percentage of instances solved within a number of failed search tree nodes by each heuristic. One interesting observation is that max-marginal is performing better overall than what was previously thought best, max-strength. That recommendation was based on three problems (LatinSquare, Partially-Filled MagicSquare, Primes) for which we see that their performance is comparable. This broader experiment points us in a slightly different direction. One possible advantage of max-marginal is that it may be biased toward smaller domains, as illustrated in Sect. 2.1.

Generally LDS performs better than DFS, which is usually the case with a good search heuristic since it prioritizes search tree nodes reached by branching against the heuristic as little as possible. Figure 2 shows the proportion of instances solved using at most a given number of such discrepancies. Interestingly for some problems a large proportion are solved in very few or even no discrepancies, directly reaching a solution.

Based on these results we continue with combination max-marginal / LDS and now turn to comparing with other heuristics and solvers.

3.2 Comparison with Default Search of State-of-the-Art Solvers

We compare to the best CP solvers in the latest XCSP competition, `Choco-4.10.2`[4] and `Abscon-19.06`.[5] We executed these solvers using the same command line options as those used in the competition. As a result, solvers use their default search strategies. The default heuristics for variable and value selection are dom-wdeg and lexicographic for both solvers. Such a comparison will not allow us to isolate perfectly the effects of search heuristics on guidance and runtime because the solvers are different and the filtering algorithms implemented for each constraint may be different as well. For example an inspection of Choco's source code indicates that for the ALLDIFFERENT constraint, the domain consistency algorithm is called probabilistically according to how much/often it performs more filtering than the bounds consistency algorithm on

[4] Available at https://github.com/chocoteam/choco-solver/releases/tag/4.10.2.
[5] Available at https://www.cril.univ-artois.fr/~lecoutre/#/softwares.

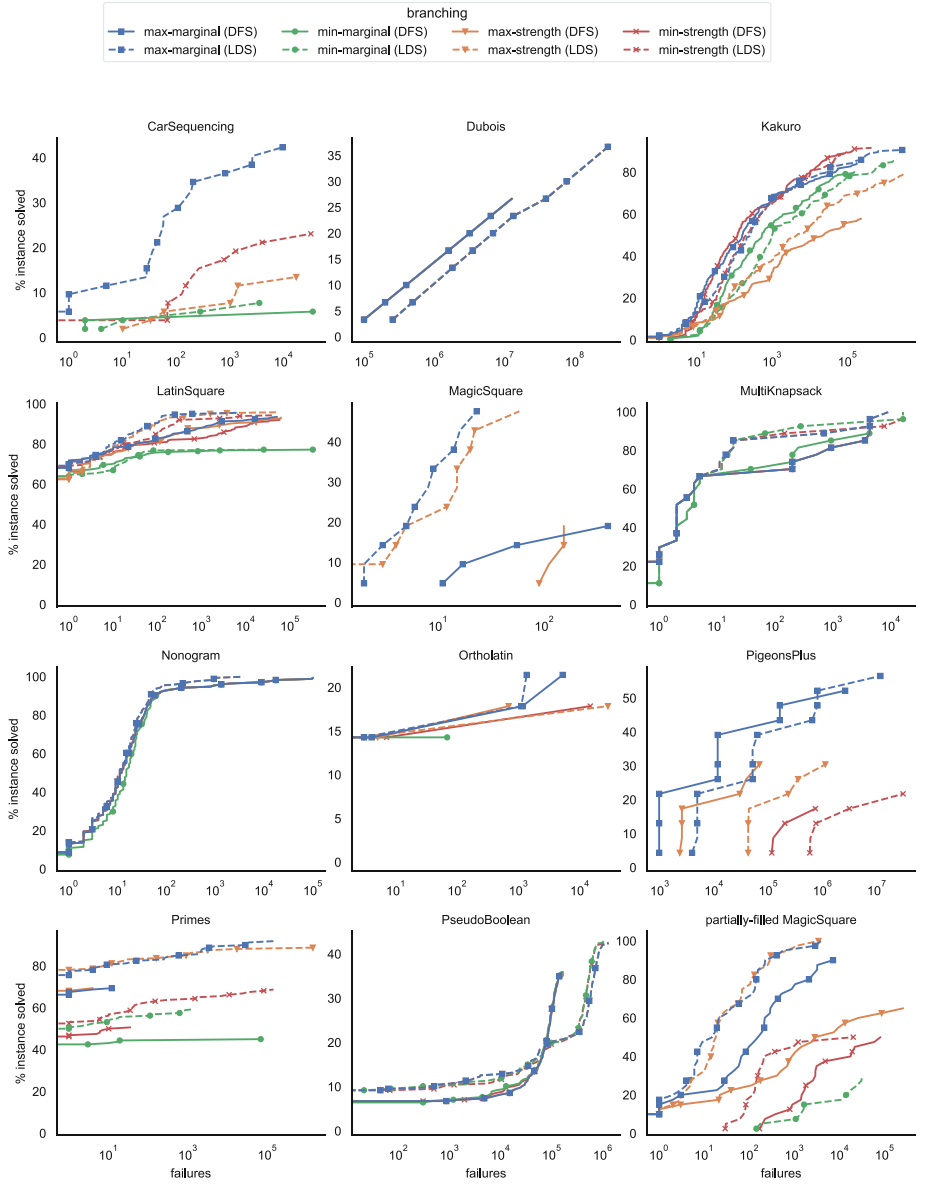

Fig. 1. %instances solved vs #fails for CP-BP branching heuristics on 12 benchmark problems

a given instance—in contrast MiniCPBP exclusively uses the domain consistency algorithm, which may provide a bit more filtering but also may put us at a disadvantage for computation time. Nevertheless it provides some indication of where we stand compared to the implementation choices of state-of-the-art solvers.

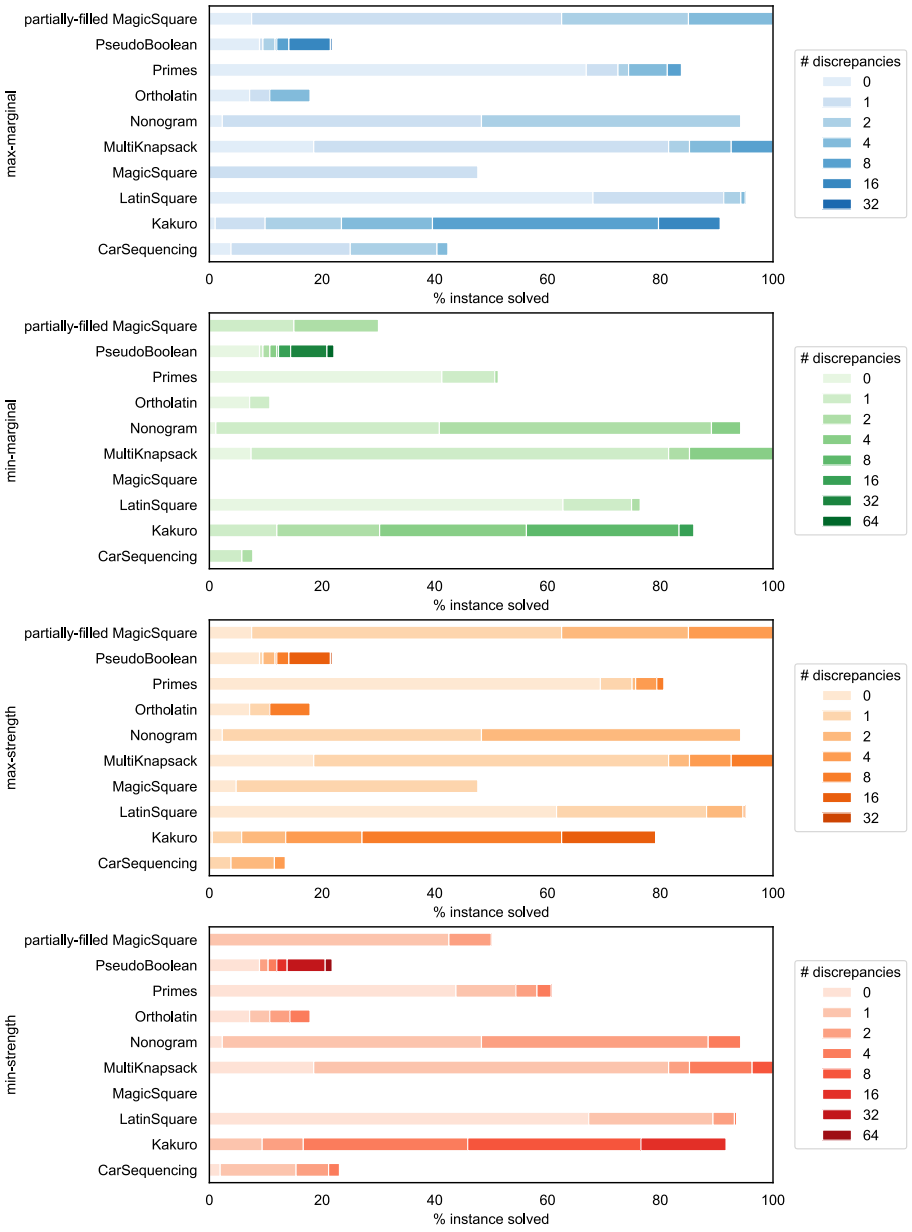

Fig. 2. %instances solved given #discrepancies allowed in the search tree

As a more easily comparable baseline we also use MiniCP[6] with min-domain variable ordering and random value ordering and report the median result over ten runs. Here the underlying solver is essentially the same.

[6] Available at http://www.minicp.org/.

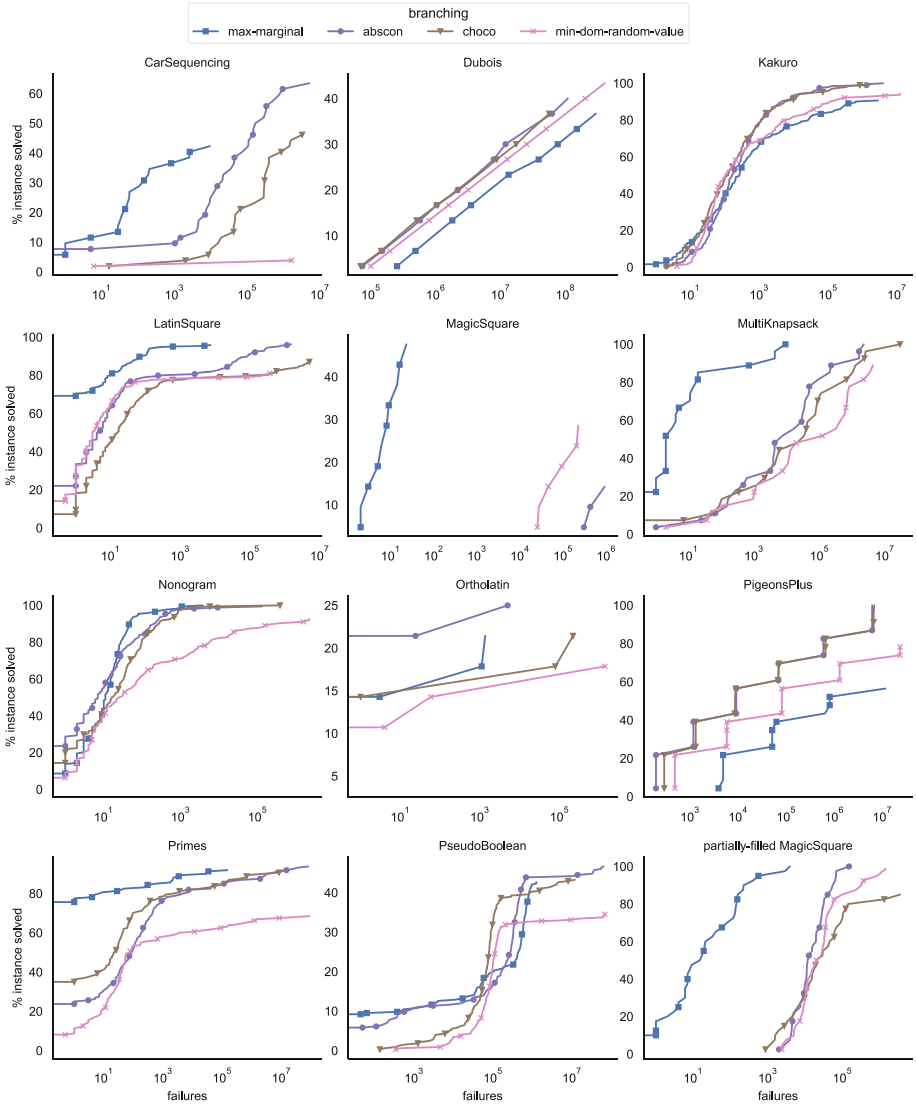

Fig. 3. %instances solved vs #fails for our best-performing search against Abscon and Choco

Figure 3 compares search guidance: we observe that we perform better than these state-of-the-art solvers on six problems (LatinSquare, MagicSquare, MultiKnapsack, Nonogram, Primes, Partially-Filled MagicSquare) and about as well on three of them (CarSequencing, OrthoLatin, PseudoBoolean). But there are three problems on which we do not perform as well: Dubois, Kakuro, and PigeonPlus.

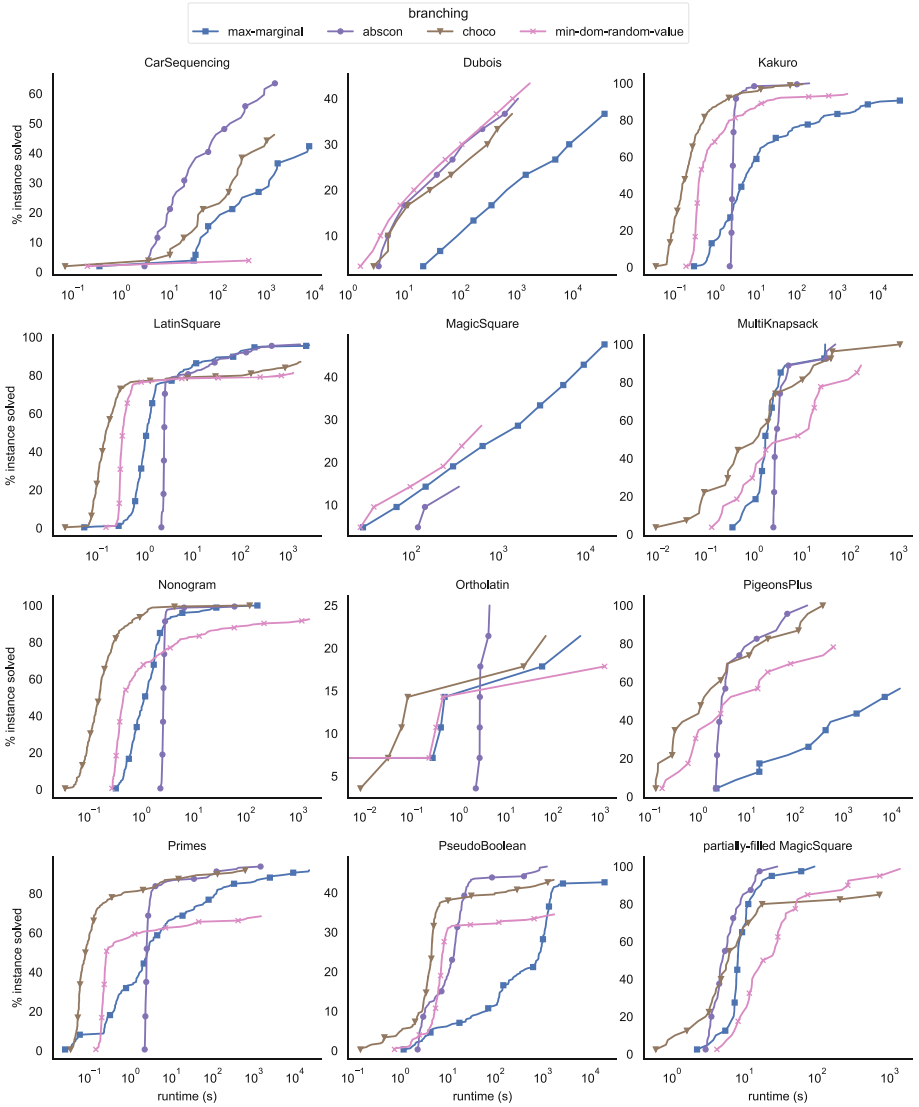

Fig. 4. %instances solved vs runtime for our best-performing search against Abscon and Choco

Comparing runtimes is difficult here even when using the same computers for the reasons cited above. In addition MiniCPBP is implemented on top of MiniCP, an academic solver not as optimized as the ones we are comparing to. Nevertheless Fig. 4 presents such a comparison. We find that our search heuristic remains competitive for the problems on which guidance was superior. It is difficult to give a general estimation of the additional computational effort of belief propagation compared to standard support propagation since it depends

on the number, arity, and type of constraints in the model (the weighted counting algorithms are different for each type). If we take a problem for which search effort is similar for max-marginal and min-domain (both basically running MiniCP), such as Kakuro (see Fig. 3), there is in this case approximately a one order of magnitude difference (see Fig. 4).

In the following sections we look in detail at some of the problems on which we do not perform well in an effort to improve our combinatorial search heuristics further.

4 Dubois and PigeonPlus—Uniform Marginals

Dubois are crafted instances with binary variables and ternary constraints. They have a special structure, essentially hiding in an increasingly long chain of ternary constraints a tiny unsatisfiable instance: one constraint says (x, y, z) should exhibit an even number of 1's whereas another says the opposite. Even on such a tiny instance, we essentially enumerate potential solutions. That behaviour is confirmed by the straight lines we observe for all solvers/heuristics when we plot the number of instances solved against the number of failed search tree nodes in log scale (the number of variables in these instances increases smoothly). The marginals are uniform (all at 0.5) and remain so even once we start branching. Therefore there is no heuristic information to allow us to discriminate between values. We likely perform worse than the rest because we are using LDS so some search tree nodes are visited multiple times (see also Fig. 1).

The same is true for PigeonPlus: instances are unsatisfiable, so we need to exhaust the search tree, and marginals are either uniform (x variables) or almost uniform (y variables), remaining the same throughout BP iterations. The branching for max-marginal tends to alternate between x and y variables, which may not be optimal for the size of the refutation tree because x variables have a slightly smaller domain—hence min-domain would be expected to perform better here. The staircase appearance of search effort is linked to the size of the instances partly increasing in discrete steps.

The lesson here is that when marginals are (almost) uniform and remain so even after we branch, there is no discriminative information to exploit in the marginals so we should use some other branching heuristic and spare the effort of computing marginals. Fortunately this is easy to check at the beginning of search.

4.1 An Opportunity for Higher Forms of Consistency

Note that Dubois is a problem for which the generalization to belief propagation of marginals on *subsets* of variables would be worthwhile. Consider marginals on pairs of variables: as soon as we set one of the three variables, say x to 1, unsatisfiability is uncovered: the marginals on $yz \in \{00, 01, 10, 11\}$ from the "even" constraint are $\langle 0, .5, .5, 0\rangle$ and from the "odd" one are $\langle .5, 0, 0, .5\rangle$, whose combination makes every pair of values vanish. The current CP-BP framework could

be extended to offer such a consistency level reminiscent of path consistency, though at some computational expense.

5 CarSequencing—Handling Dramatic Marginal Oscillation

We noticed that for CarSequencing we were initially branching on null marginals, a disconcerting behaviour. We studied the progression of marginals during BP iterations on a small instance (CarSequencing-m1-jcr/CarSequencing-dingbas.xml) and observed that marginals of the variables start an extreme oscillation between two values in their domain after a few iterations that eventually drives all marginals to zero. Even if we stop after fewer iterations, that oscillation makes the information we use for branching very unstable. Marginal oscillation (and, more generally, failure to converge) is a documented phenomenon [6] that is linked to the presence of cycles in the constraint graph but as we wrote in Sect. 2 CP models with large-arity constraints tend to have marginals converge more. Still, some oscillation was observed in [9] but attributed to an alternation between values taken by a given variable in the multiple solutions to an instance. This cannot be the case here: there are six solutions to that small instance but some variables are backbone (i.e. are assigned the same value in all solutions) and yet their marginals oscillate as well.

5.1 Studying a Tiny Instance with a CARDINALITY Constraint

One reason for this oscillation seems to be the way MiniCPBP decomposes CARDINALITY constraints into a set of AMONG constraints, one per counted value, and how the indicator variables are linked to the original variables. That decomposition is used in MiniCPBP because of the availability of an efficient exact weighted counting algorithm for SUM constraints whereas (unweighted) counting on CARDINALITY as a whole is already quite challenging and time consuming [1].

Consider this tiny instance:

$$x_1, x_2, x_3 \in \{0, 1, 2\}$$

$$\text{CARDINALITY}(\{x_1, x_2, x_3\}, \langle 0, 1, 2\rangle, \langle c_0, c_1, c_2\rangle)$$

for some fixed integer parameters c_i prescribing the number of occurrences of value i in $\{x_1, x_2, x_3\}$. This CARDINALITY constraint is decomposed into

$$\text{AMONG}(\{x_1, x_2, x_3\}, \langle i\rangle, c_i) \qquad 0 \leq i \leq 2$$

each AMONG itself decomposed into

$$y_j^i = 1 \iff x_j = i \qquad 1 \leq j \leq 3$$
$$\sum_{j=1}^{3} y_j^i = c_i$$

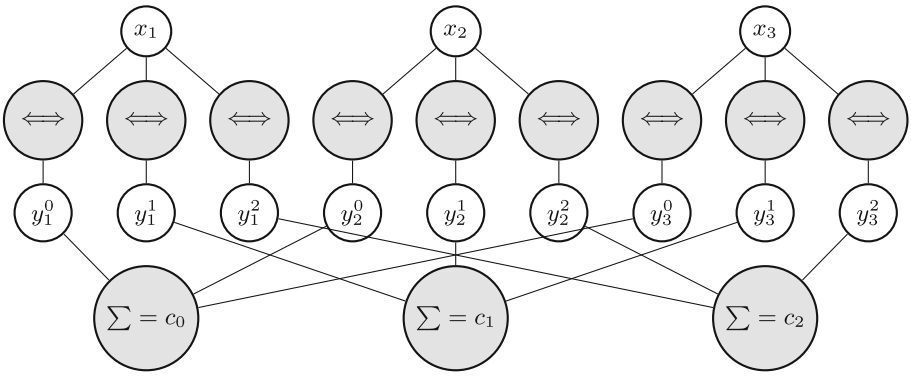

Fig. 5. Constraint graph for decomposed CARDINALITY. White vertices represent variables and shaded ones, constraints.

introducing an indicator variable y_j^i for each original variable x_j. The decomposition of a single AMONG does not introduce cycles in the constraint graph but the combination of all of them for CARDINALITY does: Fig. 5 shows the corresponding constraint graph for the whole CARDINALITY constraint using that decomposition. It contains many long cycles (length 12). One drawback of the decomposition is the many binary bijective constraints it contains. We can aggregate them into a single TABLE constraint per x_j variable linking it to its indicator variables as shown below and in Fig. 6: shorter cycles (length 8), fewer constraints, larger-arity constraints. In general if there are d values in a domain, we replace d binary constraints by a single arity-$(d+1)$ constraint.

$$\texttt{table}(\langle x_j, \langle y_j^i \rangle_{0 \le i \le 2} \rangle, \mathcal{T}) \qquad 1 \le j \le 3$$
$$\sum_{j=1}^{3} y_j^i = c_i \qquad 0 \le i \le 2$$

At Fig. 7 the dotted curves in the top and middle plots show the behaviour of the old and new decompositions, respectively, for $\langle c_0, c_1, c_2 \rangle = \langle 1, 1, 1 \rangle$ and $\langle 1, 2, 0 \rangle$. For the former, the marginals computed by each decomposition for the x_j's are identical and immediately stabilize to $1/3$, which is the true value. For the latter there is oscillation with both decompositions and their amplitude is the same but the new decomposition features a shorter period, which can translate into faster convergence.

Observe that there are some redundant constraints in the CARDINALITY decomposition: the number of occurrences of a given value, say c_0, is equal to the number of x_j variables minus the sum of the other c_i's. So we could leave out one of the SUM constraints from the decomposition, thereby introducing fewer cycles. We see at Fig. 7 that doing this in the uniform case, $\langle 1, 1, x \rangle$ meaning that the sum for value 2 is left out, does not degrade much the accuracy of the computed marginal which soon moves very close to the true value, while significantly improving accuracy and showing convergence in the non-uniform case

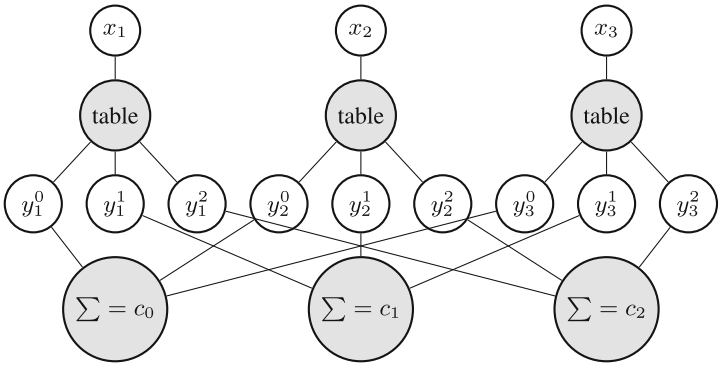

Fig. 6. Constraint graph for improved decomposition of CARDINALITY.

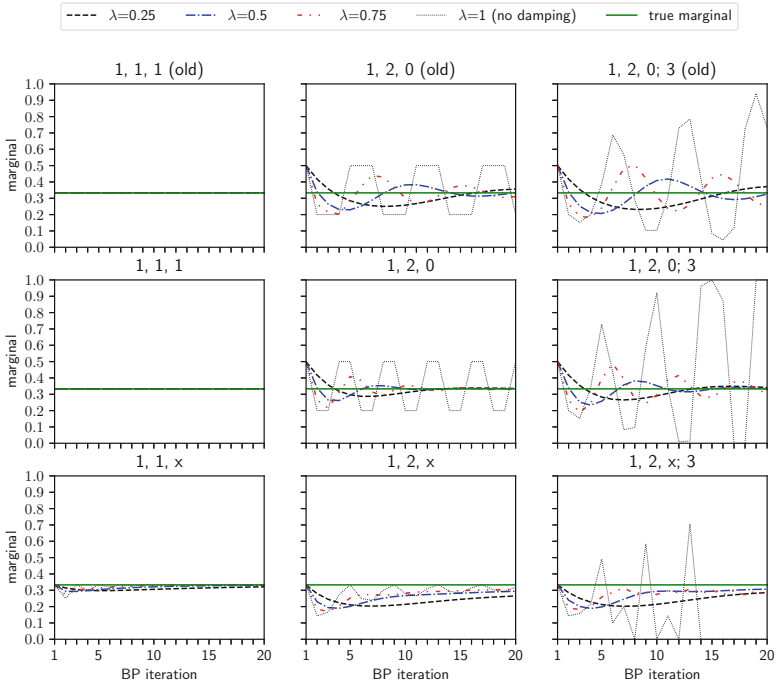

Fig. 7. The effect of message damping with different values of parameter λ

(see $\langle 1, 2, x \rangle$). Surprisingly, adding the redundant constraint that the sum over all y_j^i's is equal to 3 (thus introducing more cycles) makes things much worse, as exemplified by plots $\langle 1, 2, 0; 3 \rangle$ and $\langle 1, 2, x; 3 \rangle$ that increasingly oscillate until eventually stabilizing to 0 or 1. So it may be the case that adding redundant constraints to a model hurts belief propagation and should be avoided. Note that the CarSequencing instances include redundant constraints.

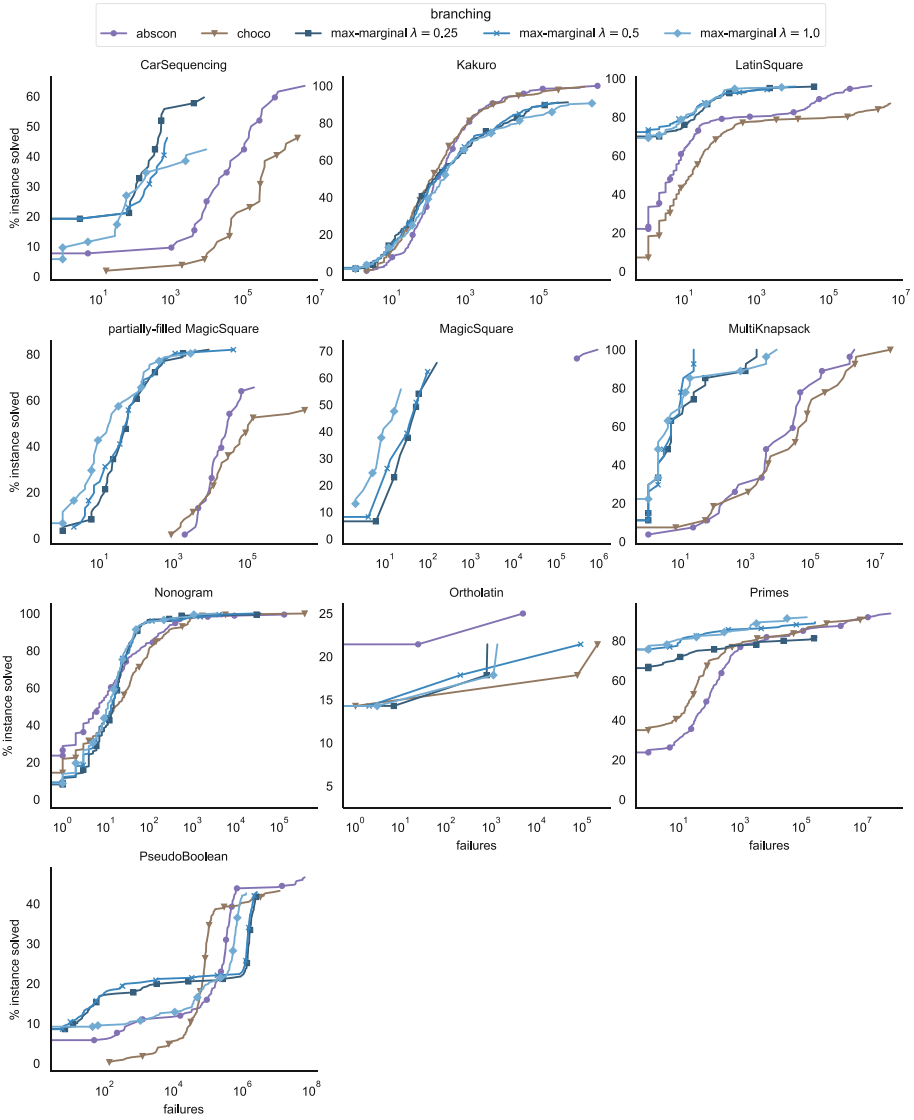

Fig. 8. Final comparison between Abscon, Choco, and max-marginal with LDS and damping

The fact that redundant constraints, which typically improve inference, may be deteriorating our search is unsettling. Fortunately there are known remedies to oscillation.

5.2 Incorporating Message Damping

Convergence of belief propagation can only be theoretically guaranteed for simple graphs. In practice, it is possible that the beliefs keep getting updated without converging to a value. This results in oscillating messages and beliefs. *Damping* is a common method for reducing the chance of such oscillations [5]. It involves taking a weighted average of the new and old beliefs and using that to update the belief. We apply damping to the messages sent from variables to constraints. At iteration t, those messages are calculated as:

$$\mu^{(t)}_{x\to c}(v) = \lambda\mu_{x\to c}(v) + (1-\lambda)\mu^{(t-1)}_{x\to c}(v)$$

where the *damping factor* ($0 \leq \lambda \leq 1$) balances the old and new beliefs. The solid curves at Fig. 7 show the effect of damping for three values of λ: all eventually cancel oscillation. Note that another form of message damping was investigated in [9] without any significant benefit being observed.

Figure 8 presents a final comparison of search guidance over the benchmark problems[7] in which message damping has been added to max-marginal / LDS. For several problems, notably those on which we already performed better, damping does not change the outcome of the comparison and even helps further in the case of MultiKnapsack. For those where the comparison was close it shows a greater impact, in particular for CarSequencing which provided the original motivation to look into message damping and whose solving is improved.

6 Conclusion

In this paper we undertook an empirical investigation of combinatorial search heuristics for the CP-BP framework using a wide set of benchmark problems and deepened our understanding of their behaviour, which led to an improved search: LDS with max-marginal as branching heuristic and message damping. It appears to guide search better than state-of-the-art search heuristic dom-wdeg, enough to remain often competitive in terms of computing time. In Sect. 4 we identified a situation under which no benefit can come from computing marginals and a simple way to identify it.

There are other opportunities for improved search that we would like to pursue. There are sometimes a large number of (quasi-)ties for branching: should we proceed differently in that case? Other times several domain values are tied for maximum marginal but one stands out for minimum marginal: instead of choosing a priori to branch according to max-marginal or min-marginal, should we decide between them at each search tree node depending on the context? The number of BP iterations to compute the marginals is currently fixed to five: should that number be adapted according to how marginals progress while iterating?

[7] We no longer report on Dubois and PigeonPlus now that they have been settled.

Acknowledgements. The authors wish to thank the anonymous referees for their constructive criticism that helped improve this work. Financial support for this research was provided by IVADO through the Canada First Research Excellence Fund (CFREF) grant, the *Fonds de recherche du Québec–Nature et technologies* (FRQNT), and NSERC Discovery Grant 218028/2017. This research was enabled in part by support provided by Calcul Québec and Compute Canada.

References

1. Bianco, G.L., Lorca, X., Truchet, C., Pesant, G.: Revisiting counting solutions for the global cardinality constraint. J. Artif. Intell. Res. **66**, 411–441 (2019). https://doi.org/10.1613/jair.1.11325
2. Chavira, M., Darwiche, A.: On probabilistic inference by weighted model counting. Artif. Intell. **172**(6–7), 772–799 (2008). https://doi.org/10.1016/j.artint.2007.11.002
3. van Hoeve, W., Katriel, I.: Global constraints. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2, pp. 169–208. Elsevier (2006). https://doi.org/10.1016/S1574-6526(06)80010-6
4. Katsirelos, G., Bacchus, F.: Generalized NoGoods in CSPs. In: Veloso, M.M., Kambhampati, S. (eds.) Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, 9–13 July 2005, Pittsburgh, Pennsylvania, USA, pp. 390–396. AAAI Press/The MIT Press (2005). http://www.aaai.org/Library/AAAI/2005/aaai05-062.php
5. Murphy, K.P.: Machine Learning - A Probabilistic Perspective. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge (2012)
6. Murphy, K.P., Weiss, Y., Jordan, M.I.: Loopy belief propagation for approximate inference: an empirical study. In: Laskey, K.B., Prade, H. (eds.) Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, UAI 1999, Stockholm, Sweden, July 30–August 1, pp. 467–475. Morgan Kaufmann (1999)
7. Pearl, J.: Probabilistic Reasoning in Intelligent Systems - Networks of Plausible Inference. Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann (1989)
8. Pesant, G.: Counting-based search for constraint optimization problems. In: Schuurmans, D., Wellman, M.P. (eds.) Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, 12–17 February 2016, Phoenix, Arizona, USA, pp. 3441–3448. AAAI Press (2016). http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12065
9. 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
10. Pesant, G., Quimper, C.G., Zanarini, A.: Counting-based search: branching heuristics for constraint satisfaction problems. J. Artif. Intell. Res. **43**, 173–210 (2012). https://doi.org/10.1613/jair.3463