Evaluating Simple Fully Automated Heuristics for Adaptive Constraint Propagation

Anastasia Paparrizou and Kostas Stergiou
Department of Informatics and Telecommunications Engineering
University of Western Macedonia, Kozani, Greece
{apaparrizou, kstergiou}@uowm.gr

***Abstract*—Despite the advancements in constraint propagation methods, most CP solvers still apply fixed predetermined propagators on each constraint of the problem. However, selecting the appropriate propagator for a constraint can be a difficult task that requires expertise. One way to overcome this is through the use of machine learning. A different approach uses heuristics to dynamically adapt the propagation method during search. The heuristics of this category proposed in [1] displayed promising results, but their evaluation and application suffered from two important drawbacks: They were only defined and tested on binary constraints and they required calibration of their input parameters. In this paper we follow this line of work by describing and evaluating simple, fully automated heuristics that are applicable on constraints of any arity. Experimental results from various problems show that the proposed heuristics can outperform a standard approach that applies a preselected propagator on each constraint resulting in an efficient and robust solver.**

***Index Terms*—constraint programming; search; constraint propagation;**

I. Introduction

Despite the advances in Constraint Programming (CP), there are still some important obstacles that prevent it from becoming even more widely known and applied. One significant such obstacle is the rigidness of CP solvers, in the sense that decisions about algorithms and heuristics to be used on a specific problem are taken prior to search during the modeling process and cannot change during search.

Concerning constraint propagation in particular, which is at the core of CP's strength and the focus of this paper, the decision on which algorithm to select for the different constraints of the CP model is either predetermined or placed on the shoulders of the user/modeler. For instance, the modeler may select to propagate the alldifferent constraints in a problem using a domain consistency algorithm. However, during search it may turn out that domain consistency achieves little extra pruning compared to bounds consistency. Unfortunately, standard CP solvers do not allow to change the decisions taken prior to search "on the fly". Hence, it will not be possible to automatically switch to a bounds consistency propagator during search.

This research has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.

To overcome these difficulties the CP research community has followed various different approaches. The most common one is the use of information such as the arity of the constraints and the complexity of propagation algorithms to evaluate the cost of the available propagators. Then these are typically ordered in increasing cost (although other factors may also play a part). This approach is analyzed, for example, in [2] and it is followed by many state-of-the art solvers such as Gecode and Choco. A drawback of this approach is that it does not take into account the actual effects of propagation during search.

Another approach concerns the use of machine learning (ML) methods for the automatic selection of constraint propagation methods (as well as tuning other parameters of a CP solver). Although some ML-based methods for dynamic parameter tuning during search do exist [3], most approaches focus on the easier problem of static tuning prior to search (e.g. [4], [5]). Hence, the use of ML may alleviate the burden of user involvement in parameter tuning, and the selection of propagation methods (propagator) in particular, but the rigidness displayed by existing solvers still remains.

As an alternative to both of the above approaches, heuristic methods for the automatic tuning of constraint propagation have also been recently proposed [6], [1]. Their advantage is twofold: they are inexpensive to apply, and they are perfectly suited to a dynamic application because they exploit information concerning the actual effects of propagation during search. In this paper we are concerned with the heuristics proposed in [1] for dynamically adapting the propagation method used on the constraints of the given problem. Although this approach displayed quite promising results, it suffered by important limitations. First, the description as well as the evaluation of the heuristics was limited to binary constraints. And second, their successful application depended on user interference for careful parameter tuning. The former limits the applicability of the heuristics while the latter severely compromises their autonomicity and puts burden on the shoulders of the users.

In this paper we confront and remedy both these problems. First, we evaluate two simple heuristics for constraints of any arity that allow to dynamically switch between two different propagators on individual constraints in a fully automated way. The first (resp. second) heuristic applies a standard propagator on a constraint (e.g. domain consistency) until the constraint causes a domain wipeout - DWO (resp. at least one value deletion). Then, in the immediately following revision of the constraint, a stronger local consistency (e.g. SAC) is applied. For the following revision we revert back to the standard propagator and this is repeated throughout search. These heuristics allow to exploit the filtering power offered by strong propagation methods without incurring severe cpu time penalties since they invoke the strong propagator very sparsely. And importantly, this is achieved without requiring any user involvement.

We also propose and evaluate refinements of the above heuristics that, while still being fully automated, achieve better performance by targetting the use of the strong propagator on variables that are more likely to be filtered. Also, we evaluate the heuristics using different methods as the strong propagator. Overall, our experimental results demonstrate that the simple heuristics we employ outperform the rigid method that applies a standard propagator throughout search, resulting in most robust solvers.

II. Background

A *Constraint Satisfaction Problem* (CSP) is defined as a tuple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ where: $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of n variables, $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ is a set of finite domains, one for each variable, with maximum cardinality d, and $\mathcal{C} = \{c_1, \ldots, c_e\}$ is a set of e constraints. Each constraint c is a pair $(var(c), rel(c))$, where $var(c) = \{x_1, \ldots, x_m\}$ is an ordered subset of $\mathcal{X}$, and the relation $rel(c)$ is a subset of the *Cartesian* product $D(x_1) \times \ldots \times D(x_m)$ that specifies the allowed combinations of values for the variables in $var(c)$.

Local consistencies are properties that are enforced on the constraints of a problem so that infeasible values are located and pruned. The most commonly used local consistency is *generalized arc consistency* (GAC) or *domain consistency*. A value $a_i \in D(x_i)$ is GAC iff for every constraint c s.t. $x_i \in var(c)$ there exists a valid tuple $\tau \in rel(c)$ that includes the assignment of a_i to x_i. In this case τ is called a *support* of a_i. A variable is GAC iff all its values are GAC. A problem is GAC iff there is no empty domain in $\mathcal{D}$ and all the variables in $\mathcal{X}$ are GAC.

Numerous local consistencies that are stronger than GAC have been proposed. Some of these have shown promise (e.g. SAC and maxRPWC) but in general they are all too expensive to apply throughout search. Therefore, an important question is how to best exploit their filtering power without paying a high cpu time penalty. This paper tries to answer this question.

A value $a_i \in D(x_i)$ is *singleton arc consistent* (SAC) iff the problem derived from assigning a_i to x_i is GAC [7]. A problem is SAC iff every value of every variable is SAC.

When applied, GAC and its weaker variants, such as Bounds Consistency (BC), focus on one constraint at a time. In contrast, some strong local consistencies exploit the fact that very often constraints have two or more variables in common, to achieve stronger pruning. One of the most promising consistencies of this type is *max Restricted PairWise Consistency* (maxRPWC) [8].

A value $a_i \in D(x_i)$ is maxRPWC iff $\forall c_j \in C$, where $x_i \in var(c_j)$, a has a support $\tau \in rel(c_j)$ s.t. $\forall c_l \in C$ $(c_l \neq c_j)$, s.t. $var(c_j) \cap var(c_l) \neq \emptyset, \exists \tau' \in rel(c_l)$, s.t. $\tau[var(c_j) \cap var(c_l)] = \tau'[var(c_j) \cap var(c_l)]$ and τ' is valid. In this case we say that τ' is a PW-support of τ. A variable is maxRPWC iff all values in its domain are maxRPWC. A problem is maxRPWC iff there is no empty domain in $\mathcal{D}$ and all variables are maxRPWC.

III. Adaptive Propagator Selection

Modern CP solvers offer an impressive array of specialized constraint propagation algorithms that typically achieve GAC or BC on specific types of constraints. However, typically solvers follow one of the following patterns:

1) The choice of propagation algorithm for a specific constraint is made during the modeling process and cannot change during search.
2) All the available propagators for a constraint are used, in increasing order of cost, unless there is a theoretical guarantee that a propagator cannot achieve extra pruning (as discussed in [9], [10]).

A drawback of the second approach, which is more sophisticated, is that even if a propagator's cost can be accurately predicted (which is not always true), the prediction of a propagator's impact is not nearly as straightforward. Schulte and Stuckey concluded that an obvious way to further speed up constraint propagation is to consider the estimated impact for a propagator and not only its cost [2].

Exploring ways to achieve this, [1] proposed heuristics for dynamically switching between a weak (W) and a strong (S) propagator for individual constraints during search. The motivation for these heuristics was based on the observation that in structured problems propagation events (DWOs and value deletions) caused by individual constraints are often highly clustered. That is, they occur during consecutive or very close revisions of the constraints. Hence, the intuition behind the proposed heuristics is twofold. First to target the application of the strong consistency on areas in the search space where a constraint is highly active so that domain pruning is maximized and dead-ends are encountered faster. And second, to avoid using an expensive propagation method when pruning is unlikely.

The following two heuristics generalize the main heuristics of [1] to non-binary constraints in a straightforward and fully automated way.

- Heuristic H$_{dwo}$ monitors the revisions and DWOs caused by the constraints in the problem. For any constraint c and any variable $x_i \in var(c)$, each $v_i \in D(x_i)$ is made W unless the immediately preceding revision of c resulted in the DWO of a variable in $var(c)$. In this case the values of $D(x_i)$ are made S.
- Heuristic H$_{del}$ monitors revisions and value deletions. For any constraint c and any variable $x_i \in var(c)$, each $v_i \in D(x_i)$ is made W unless the immediately preceding revision of c resulted in at least one value deletion from the domain of a variable in $var(c)$. In this case the values of $D(x_i)$ are made S.

A significant difference between H$_{dwo}$ and H$_{del}$ and their corresponding versions for binary constraints, called H$_1$ and H$_2$ in [1], is that the latter required the manual setting of a parameter l to optimize their performance. For any constraint c this parameter determined the number of revisions after the latest revision of c that caused a DWO (resp. value deletion) during which S will be applied. In contrast, H$_{dwo}$ and H$_{del}$ do not use this parameter and as a result they are fully automated.

As reported in [1], the disjunctive combination of the two basic heuristics that applies S whenever the conditions of either of the heuristics is met, achieves particularly good performance being more robust than individual heuristics. However, given the definitions of H_{dwo} and H_{del} here, their disjunctive combination is pointless since it is equivalent to applying H_{del}. Hence, we do not consider it.

IV. Experiments

In our experimental evaluation of the heuristics we have considered GAC as the standard propagator W, given that it is the most commonly used local consisteny. Since we are interested in non-binary problems, we have considered two strong local consistencies as the S propagators. Namely, maxRPWC and SAC. All methods used the *dom/wdeg* heuristic for variable ordering and *lexicographic* value ordering under a binary branching scheme. The propagation queue was variable-oriented (i.e. the elements of the queue are variables) and was ordered in a FIFO manner. A cpu time limit of 6 hours was set for all instances. All the evaluated heuristic methods used the S propagator on all constraints for preprocessing.

The classes of problems we have considered include both structured and random problems, some of which are specified extensionally and others intensionally. These classes, which are taken from C.Lecoutre's XCSP repository and are commonly used in the CSP Solver Competition, are: *random* and *forced random*, *positive table*, *BDD*, *aim*, *pret*, *dubois*, *chessboard coloration*, *Schurr's lemma*, *modified Renault*.

In the case of extensionally specified constraints we have used the efficient algorithm of [11] for the implementation of GAC. This is also the basis for the implementation of SAC and maxRPWC. For the former, the implementation is straightforward. For the latter, we have used a simplified version of the algorithm presented in [12]. In the case of intensionally specified constraints we have used the generic algorithms GAC2001/3.1 [13] and maxRPWC1 [8]. GAC2001/3.1 was also the basis for the implementation of SAC.

In the following, we first evaluate H_{dwo} and H_{del} using maxRPWC as the strong propagator. Then we analyze the performance of the heuristics (H_{dwo} in particular) to explain their success. Finally, we propose and evaluate refinements of the heuristics and give results from the use of SAC as the strong propagator.

A. Evaluating the heuristics

In Table I we show the mean performance of H$_{dwo}$ and H$_{del}$ on all tested classes, measured in cpu time and nodes explored. To put these results into perspective, we also give results from: 1) an algorithm that propagates all constraints using GAC throughout search, 2) an algorithm that propagates all constraints using maxRPWC throughout search, and 3) the H$_{del}$ heuristic implemented as in [1], with parameter l set to 10 (i.e. maxRWPC is applied for the 10 revisions following a revision that deleted at least one value). We also report the mean percentage (%) of constraint revisions where the strong consistency (maxRPWC) was applied.

TABLE I
MEAN CPU TIMES (T) IN SECS, NODES (N), AND THE PERCENTAGE OF CONSTRAINT REVISIONS (S) CARRIED OUT USING MAXRWPC. CPU TIMES IN BOLD DEMONSTRATE THE FASTEST METHOD. A DASH (-) INDICATES THAT THE METHOD WAS UNABLE TO SOLVE ALL INSTANCES WITHIN THE TIME LIMIT.

Class		GAC	maxRPWC	H_{dwo}	H_{del}	H_{del} 10
Rand-fcd	t	**182**	233	229	202	195
	n	131,745	59,245	161,247	95,576	54,316
	s	0	100	1.1	24.8	73.3
Random	t	**220**	333	221	236	270
	n	151,039	79,771	154,657	105,944	72,353
	s	0	100	1.1	24.9	73
Positive table-8	t	**1,629**	3,947	2,233	1,984	3,109
	n	47,073	15,142	45,108	26,425	14,747
	s	0	100	3	26.5	77.5
Positive table-10	t	-	643	647	667	691
	n	-	0	0	0	0
	s	-	100	100	100	100
Aim	t	9.5	**2.4**	3.9	2.8	1.6
	n	1,324,118	217,459	468,262	302,870	127,723
	s	0	100	2.6	20.3	53.2
BDD	t	7,771	6.4	**3.9**	4.2	5.1
	n	36,804	10	10	10	10
	s	0	100	24.5	56.9	69.2
Chess-board	t	**4.6**	37.7	5.5	8.2	12.8
	n	57,024	43,644	66,177	65,609	59,826
	s	0	100	2.7	6	26.2
Schurr's lemma	t	63	100	**62**	73	87.2
	n	559,971	524,909	549,868	552,197	562,221
	s	0	100	1.4	17.1	59.8
Dubois	t	934	**878**	925	1,282	912
	n	175,325,461	144,632,439	161,619,009	225,836,708	163,285,042
	s	0	100	1.9	41.7	98.35
Pret	t	**46**	**46**	48	50	47
	n	37,017,710	37,017,710	37,017,710	37,017,710	37,017,710
	s	0	100	3.2	42.4	98.7
Renault	t	**118**	181	126	143	167
	n	801	334	521	413	328
	s	0	100	12	25.5	83

The results given in Table I demonstrate the efficacy of the studied fully automated heuristics. Although they do not achieve the best mean results on any class (with the exception of *BDD*), one or both of the heuristics achieve the best performance on several individual instances. But more importantly, the heuristics succeed in striking a balance between the performance of GAC and maxRPWC. Specifically, in problems where GAC thrases (*positive table-10* and *BDD*), the heuristics follow maxRPWC in solving the problems with little or no search. In problems where GAC is clearly better than maxRPWC (*chessboard coloration*, *positive table-8*, and *random*) the performance of the heuristics is closer to GAC making them clearly superior to maxRPWC. In a case where the opposite occurs, i.e. maxRWPC is better than GAC (*aim*), the heuristics follow maxRPWC making them superior to GAC. In other cases, where GAC and maxRWPC are closely matched, the performance of the heuristics typically lies in between GAC and maxRPWC.

Comparing H$_{dwo}$ to H$_{del}$ we can note that there are no significant differences in their performance. This occured not only with respect to their mean performance but, largely, with respect to individual instances as well. What is interesting is that H$_{dwo}$, which is slightly better overall, achieves its results with only few invocations of the strong propagator as the percentages s show, with *positive table-10* and *BDD* being exceptions to this.

Finally, comparing H$_{del}$ to its parameterized version with l set to 10, we can note that the fully automated version is generally preferable. It achieves better mean performance on 7 out of the 11 classes and it is not significantly outperformed in the other 4. This hints at a particular importance of the revisions that immediately follow a propagation event in terms of the likelihood of another propagation event occuring.

B. Are revisions after DWOs important?

In this section we investigate the reason for the success of H$_{dwo}$. In Table II we record ratios concerning value deletions to demonstrate the effects of the calls to S in revisions immediately following a revision that caused a DWO. We have picked an indicative instance from each class. D^{dwo} is the number of revisions that caused value deletions and immediately follow a revision that caused a DWO. D is the number of all revisions that caused deletions. R^{dwo} is the number of revisions that immediately follow a revision that caused a DWO. Table II gives the ratios D^{dwo}/D and D^{dwo}/R^{dwo} for GAC, maxRPWC, and H_{dwo}.

TABLE II
PERCENTAGES OF REVISIONS THAT CAUSED VALUE DELETIONS AFTER A PREVIOUS DWO TO ALL REVISIONS THAT CAUSED DELETIONS (D^{dwo}/D) AND REVISIONS THAT CAUSED VALUE DELETIONS AFTER A PREVIOUS DWO TO ALL REVISIONS EXECUTED AFTER A PREVIOUS DWO (D^{dwo}/R^{dwo}) FROM REPRESENTATIVE INSTANCES.

Class Instance		GAC	$maxRPWC$	H_{dwo}
	D^{dwo}/D	0.55	0.59	1.01
Rand-fcd	D^{dwo}/R^{dwo}	9.8	12.15	17.98
	D^{dwo}/D	0.5	0.58	1.01
Random	D^{dwo}/R^{dwo}	8.95	11.7	19.01
	D^{dwo}/D	1.48	2.95	4.92
Positive table-8	D^{dwo}/R^{dwo}	3.54	6.24	12.88
	D^{dwo}/D	1.73	1.08	2.01
Aim	D^{dwo}/R^{dwo}	14.61	2.27	10.24
	D^{dwo}/D	1.37	2.49	2.75
Chessboard	D^{dwo}/R^{dwo}	3.46	5.85	7.53
	D^{dwo}/D	0.02	0.92	0.01
Schurr's lemma	D^{dwo}/R^{dwo}	0.28	6.3	0.21
	D^{dwo}/D	0.22	0.62	0.11
Dubois	D^{dwo}/R^{dwo}	6.65	8.93	7.28
	D^{dwo}/D	0.77	0.77	0.77
Pret	D^{dwo}/R^{dwo}	13.58	13.58	13.58
	D^{dwo}/D	2.26	2.31	2.31
Renault	D^{dwo}/R^{dwo}	3.58	4.18	4.14

H$_{dwo}$ has the highest percentages, compared to GAC and maxRPWC, for both ratios shown in Table II. Especially on *Random*, *Random-fcd* and *Positive table* we observe that the numbers for H$_{dwo}$ are more than two times higher, showing that **applying a strong consistency after a DWO can increase the likelihood of value pruning**. For the rest of the classes the advantage is less obvious for two reasons: either because the strong consistency cannot offer extra pruning (i.e. *pret*) or because it is applied very few times (i.e. *Chessboard coloration*). Note that no instance from the *BDD* class is included. This is because in these problems very few constraints give non-zero results for D when maxRPWC or H$_{dwo}$ is applied (in contrast to GAC). That is, very few constraints are active during the (very short) search process with these methods.

V. REFINING THE HEURISTICS

Heuristics H$_{dwo}$ and H$_{del}$ apply the strong propagator S on all variables involved in a constraint if one of these variables suffered a DWO (resp. value deletion) in the previous revision of the constraint. This may incur unnecessary invocations of S that only increase the cpu time overhead without offering any filtering. The following heuristics are refinements of H$_{dwo}$ and H$_{del}$ that try to improve on this by targetting the use of the strong propagator on variables that are more likely to be filtered.

- Heuristic H$^{v}_{dwo}$ monitors the revisions of constraints and the DWOs of the **variables' domains**. For any constraint c and any variable $x_i \in var(c)$, each $v_i \in D(x_i)$ is made W unless the immediately preceding revision of c resulted in the DWO of $D(x_i)$. In this case the values of $D(x_i)$ are made S.
- Heuristic H$^{v}_{del}$ monitors the revisions of constraints and the value deletions from the **variables' domains**. For any constraint c and any variable $x_i \in var(c)$, each $v_i \in D(x_i)$ is made W unless the immediately preceding revision of c resulted in at least one value deletion from $D(x_i)$. In this case the values of $D(x_i)$ are made S.

H$^{v}_{dwo}$ and H$^{v}_{del}$ restrict the application of the strong propagator on variables that suffered a propagation event (DWO or value deletion) in the immediately preceding constraint revision as opposed to all variables in the constraint's scope. The intuition behind this is that such variables are more likely to suffer a DWO or value deletion(s) again, especially in hard parts of the search space. The experimental results given below indicate that this is true since the effects of restricting the invocations of S on the search effort are not significant while cpu times improve.

Table III presents mean results from all tested instances. Columns H$^{v}_{del}$ and H$^{v}_{dwo}$ give results from the use of maxRPWC as the strong propagator, while column S-H$^{v}_{dwo}$ gives results from the use of SAC. The last column, called Hybrid, gives results from a simple heuristic method that applies SAC and maxRPWC alternatively. Specifically, maxRPWC is selected as the S propagator when a constraint intersects with another constraint on more than one variable and SAC otherwise. Note that maxRPWC cannot achieve any extra filtering compared to GAC when constraints intersect on exactly one variable [8], while SAC can. Results from Table III are similar to those from Table I in the sense that again the heuristic methods H$^{v}_{dwo}$ and H$^{v}_{del}$ achieve a balance between GAC and maxRWPC.

On the other hand, heuristic S-H$^{v}_{dwo}$ is not as successful. Although it often manages to cut down the number of node visits considerably (the two random classes and *aim*), this is not reflected to cpu times (with the exception of *aim*) meaning that singleton checks are quite expensive. In addition, there are many classes where S-H^v_{dwo} does not manage to save search effort compared to GAC. However, the performance of S-H^v_{dwo} is still close to that of GAC, being sometimes better, and it is by far superior to the performance of an algorithm that applies SAC on all variables throughout search[1].

TABLE III
AVERAGE CPU TIMES (T) IN SECS, NODES (N) AND THE PERCENTAGE OF THE STRONG CONSISTENCY (S) FROM ALL CLASSES.

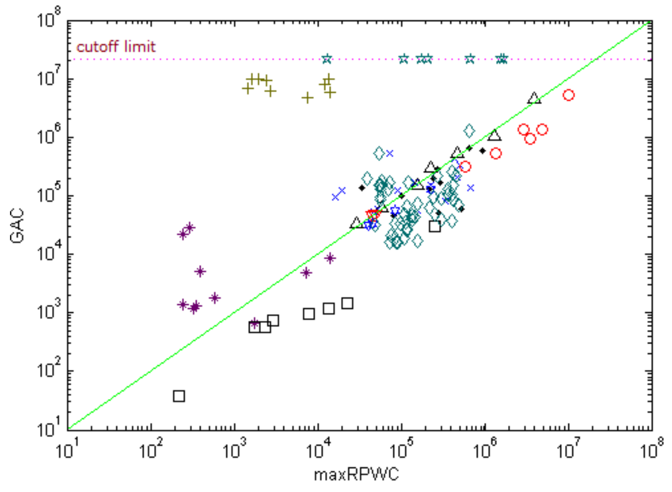
Class		GAC	H^v_{del}	H^v_{dwo}	S-H^v_{dwo}	Hybrid
Rand-fcd	t	182	179	165	192	**133**
	n	131,745	87,271	125,447	44,346	113,984
	s	0	10.7	0.3	0.4	0.2
Random	t	220	237	195	325	**176**
	n	151,039	111,768	138,985	67,690	150,706
	s	0	12	0.3	0.4	0.2
Positive table-8	t	1,629	1,609	1,746	**1,594**	1,693
	n	47,073	27,740	45,108	42,330	47,101
	s	0	4.5	0.3	0.3	0.3
Positive table-10	t	-	640	**625**	-	664
	n	-	0	0	-	0
	s	-	100	100	-	100
Aim	t	9.5	3.5	4.3	**2.2**	2.5
	n	1,324,118	391,493	547,469	186,262	250,618
	s	0	8.6	1.4	2.2	0.5
BDD	t	7,771	3.9	**3.2**	10,768	4
	n	36,804	10	10	36,896	10
	s	0	56.8	56.8	0.4	56.8
Chess-board	t	**4.6**	6.2	5.3	5.1	5.4
	n	57,024	61,374	59,390	58,491	65,640
	s	0	2.4	1.4	3.5	1.4
Schurr's lemma	t	**63**	73	**63**	67	65
	n	559,971	571,976	549,335	492,630	482,396
	s	0	8.5	0.6	0.2	0.2
Dubois	t	**934**	1,282	936	1,287	1,357
	n	175,325,461	225,836,708	172,724,047	189,160,406	215,484,904
	s	0	41.7	1.9	1.7	0.8
Pret	t	**46**	49	48	53	50
	n	37,017,710	37,017,710	37,017,710	33,190,315	34,392,941
	s	0	18.1	1.7	1.7	0.6
Renault	t	**118**	122	122	-	430
	n	801	417	544	-	580
	s	0	12.4	7.3	-	8.8

Comparing heuristics H^v_{dwo} and H^v_{del} to H_{dwo} and H_{del}, we can note that the former are more efficient. Although they restrict the application of the strong consistency by 50% up to more than 80%, as the percentages s show, this does not incur any significant increase in node visits while at the same time cpu effort is saved. In contrast, there are many cases where the number of node visits is cut down (e.g. *random* class). These results show that H^v_{dwo} and H^v_{del} achieve a better focus in the application of the strong consistency.

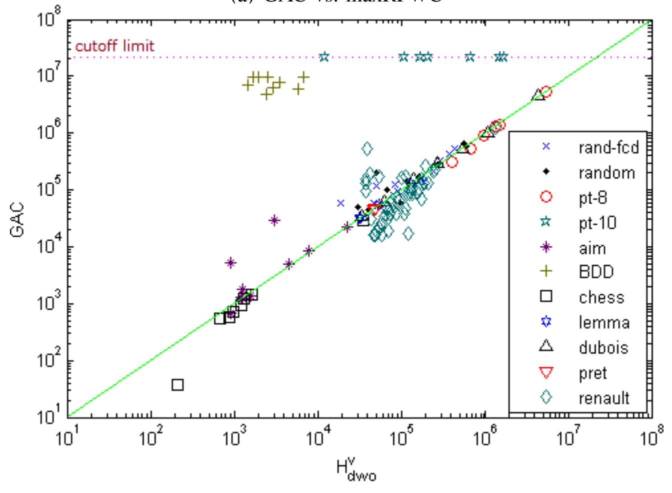
Finally, the Hybrid method is very competitive on all classes, except *modified Renault*, being faster than all other methods on the *random-fcd* and *random* classes. Again it is interesting that this method ahieves a good performance with very few invocations of the strong propagator.

Figure 1 summarizes our results by presenting pairwise comparisons on all tested instances. Figure 1(a) compares the cpu times of GAC to those of maxRPWC in a logarithmic scale. Points above (resp. below) the diagonal correspond to instances that were solved faster by maxRPWC (resp. GAC). This figure clearly demostrates the performance gap between

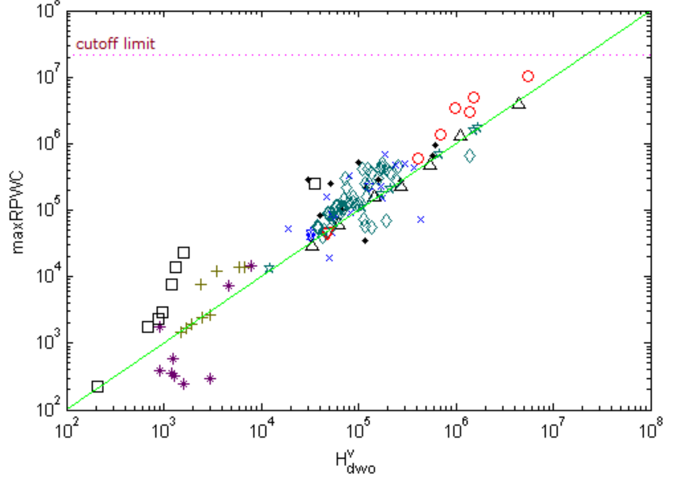
[1]Results of this algorithm are not given because it is not competitive in cpu times in most cases.



(a) GAC vs. maxRPWC



(b) GAC vs. H^v_{dwo}



(c) maxRPWC vs. H^v_{dwo}

Fig. 1. Cpu times of H^v_{dwo} compared to *GAC* and *maxRPWC*, for all evaluated instances.

GAC and maxRPWC. GAC is faster on the majority of the instances, often by large margins, but since it is a weaker consistency level, it sometimes thrashes, while the stronger maxRPWC does not. These results justify the need for a robust method that can achieve a balance between the two.

Figure 1(b) (resp. Figure 1(c)) compares the cpu times of H^v_{dwo} to those of GAC (resp. maxRPWC). These figures clearly demonstrate the benefits of the adaptive heuristics. Although the majority of the instances is still below the diagonal in Figure 1(b), they are much closer to it, indicating small differences between the two methods on those instances. These are instances where the application of maxRPWC does not offer any notable reductions in search tree size. By keeping the number of calls to the maxRPWC propagator low, the adaptive heuristic manages to avoid slowing down search considerably. On the other hand, there are still instances where GAC thrashes while H^v_{dwo}, following maxRPWC, does not.

In Figure 1(c) most instances are above the diagonal demonstrating that H^v_{dwo}, following GAC, is faster than maxRPWC. On the other hand, there are no instances where H^v_{dwo} thrashes.

VI. Related Work

As discussed earlier, selecting the appropriate propagator for a constraint is a problem that is essential to CP and therefore has attracted a lot of interest. Standard solvers do not use adaptive methods to tackle this problem. They either preselect the propagator or use costs and other measures to order the various propagators. Regarding the second approach, Schulte and Stuckey describe some state-of-the-art methods which are used to order propagators by many well known solvers (e.g. Gecode, Choco) [2].

Automatic CP solver tuning has attracted a lot of interest recently. Several researchers have approched this problem through the use of ML methods (e.g. [14], [15]). [3] proposed the use of reinforcement learning for the dynamic selection of a variable ordering heuristic at each point of search for CSPs. Another recent work uses ML to decide prior to search whether lazy learning will be switched on or off [5]. Closer to the focus of this paper, there has been little research on learning strategies for constraint propagation. [4] uses ML methods for the automatic selection of constraint propagation techniques. In particular, a static method for the pre-selection between Forward Checking and Arc Consistency is proposed. [16] evaluates ensemble classification for selecting an appropriate propagator for the alldifferent constraint. Again this is done in a static way prior to search.

Following a different line of work, but with a similar goal, there are some works proposing heuristic methods to automatically adapt contraint propagation. Apart from [1], we can mention the following: El Sakkout et al. proposed a scheme called *adaptive arc propagation* for dynamically deciding whether to process individual constraints using AC or forward checking [17]. Freuder and Wallace proposed a technique, called *selective relaxation* which can be used to restrict AC propagation based on two local criteria; the distance in the constraint graph of any variable from the currently instantiated one, and the proportion of values deleted [18]. *Probabilistic arc consistency* is a scheme that can dynamically adapt the level of local consistency applied avoidis some constraint checks and revisions that are unlikely to cause pruning [6].

VII. Conclusion

In this paper we described and evaluated simple heuristics for the dynamic adaptation of constraint propagation methods. These are based on the heuristics proposed in [1], but overcoming the limitations of that work, they are applicable on constraints of any arity and, importantly, they are fully automated. Experimental results show that refinements of the basic heuristics that target the use of strong propagators on variables that are more likely to be filtered achieve the best results and outperform the standard method that applies a fixed propagator throughout search, resulting in most robust solvers. We believe that this work is a step towards the efficient exploitation of the filtering power offered by strong propagators in a fully automated way. In the future we will examine the applicability of adaptive propagation heuristics on global constraints with efficient specialized propagators for domain and bounds consistency.

References

[1] K. Stergiou, "Heuristics for Dynamically Adapting Propagation," in *ECAI-2008*, 2008, pp. 485–489.

[2] C. Schulte and P. Stuckey, "Efficient Constraint Propagation Engines," *ACM Trans. Program. Lang. Syst.*, vol. 31, no. 1, pp. 1–43, 2008.

[3] Y. Xu, D. Stern, and H. Samulowitz, "Learning Adaptation to solve Constraint Satisfaction Problems," in *Proceedings of Learning and Intelligent Optimization (LION)*, 2009.

[4] S. Epstein, . Freuder, R. Wallace, and X. Li, "Learning propagation policies," in *Proceedings of the 2nd International Workshop on Constraint Propagation and Implementation*, 2005, pp. 1–15.

[5] I. P. Gent, C. Jefferson, L. Kotthoff, I. Miguel, N. C. A. Moore, P. Nightingale, and K. E. Petrie, "Learning when to use lazy learning in constraint solving," in *Proceedings of ECAI-2010*, 2010, pp. 873–878.

[6] D. Mehta and M. van Dongen, "Probabilistic Consistency Boosts MAC and SAC," in *Proceedings of IJCAI-2007*, 2007, pp. 143–148.

[7] R. Debruyne and C. Bessière, "Domain Filtering Consistencies," *JAIR*, vol. 14, pp. 205–230, 2001.

[8] C. Bessiere, K. Stergiou, and T. Walsh, "Domain filtering consistencies for non-binary constraints," *Artificial Intelligence*, vol. 172, no. 6-7, pp. 800–822, 2008.

[9] C. Schulte and P. J. Stuckey, "When do bounds and domain propagation lead to the same search space?" *ACM Trans. Program. Lang. Syst.*, vol. 27, no. 3, pp. 388–425, 2005.

[10] C. Schulte and P. Stuckey, "Dynamic analysis of bounds versus domain propagation," in *Proceedings of ICLP '08*, 2008, pp. 332–346.

[11] C. Lecoutre and R. Szymanek, "Generalized arc consistency for positive table constraints," in *Proceedings of CP'06*, 2006, pp. 284–298.

[12] A. Paparrizou and K. Stergiou, "An Efficient Higher-Order Consistency Algorithm for Table Constraints," in *Proceedings of AAAI-2012*, 2012.

[13] C. Bessiére, J. Régin, R. Yap, and Y. Zhang, "An Optimal Coarse-grained Arc Consistency Algorithm," *Artificial Intelligence*, vol. 165, no. 2, pp. 165–185, 2005.

[14] S. Minton, "Automatically Configuring Constraint Satisfaction Programs: A Case Study," *Constraints*, vol. 1, no. 1/2, pp. 7–43, 1996.

[15] S. Epstein and S. Petrovic, "Learning to Solve Constraint Problems," in *ICAPS-07 Workshop on Planning and Learning*, 2007.

[16] L. Kotthoff, I. Miguel, and P. Nightingale, "Ensemble Classification for Constraint Solver Configuration," in *Proceedings of CP'2010*, 2010, pp. 321–329.

[17] H. El Sakkout, M. Wallace, and B. Richards, "An Instance of Adaptive Constraint Propagation," in *Proceedings of CP-1996*, 1996, pp. 164–178.

[18] E. Freuder and R. Wallace, "Selective relaxation for constraint satisfaction problems," in *Proceedings of ICTAI-1996*, 1996.