Weight-Based Variable Ordering in the Context of High-Level Consistencies

Robert J. Woodward
Berthe Y. Choueiry

Constraint Systems Laboratory
University of Nebraska-Lincoln, USA
{rwoodwar|choueiry}@cse.unl.edu

September 11, 2018

Abstract

Dom/wdeg is one of the best performing heuristics for dynamic variable ordering in backtrack search [Boussemart *et al.*, 2004]. As originally defined, this heuristic increments the weight of the constraint that causes a domain wipeout (i.e., a dead-end) when enforcing arc consistency during search. "The process of weighting constraints with dom/wdeg is not defined when more than one constraint lead to a domain wipeout [Vion *et al.*, 2011]." In this paper, we investigate how weights should be updated in the context of two high-level consistencies, namely, singleton (POAC) and relational consistencies (RNIC). We propose, analyze, and empirically evaluate several strategies for updating the weights. We statistically compare the proposed strategies and conclude with our recommendations.

1   Introduction

Variable-ordering heuristics are critical for the effectiveness of backtrack search to solve Constraint Satisfaction Problems (CSPs). Common heuristics implement the fail-first principal, choosing the most constrained variable as

1

the next variable to assign. One such heuristic is dom/ddeg, which selects the variable with the smallest ratio of its current domain to its future degree. A more recent heuristic, dom/wdeg, uses the weighted degree of a variable by assigning a weight, initially set to one, to each constraint, and incrementing this weight whenever the constraint causes a domain wipeout [Boussemart *et al.*, 2004]. Recently, higher-level consistencies (HLC) have shown promise as lookahead for solving difficult CSPs [Bennaceur and Affane, 2001; Woodward *et al.*, 2011; Woodward *et al.*, 2012; Balafrej *et al.*, 2014].

Because HLC algorithms typically consider more than one constraint at the same time, updating the weights of the constraints in dom/wdeg is currently an open question [Vion *et al.*, 2011]. This paper focuses on answering this question in the context of two high-level consistencies, namely, Partition-One Arc-Consistency (POAC) [Bennaceur and Affane, 2001] and Relational Neighborhood Inverse Consistency (RNIC) [Woodward *et al.*, 2011]. Our study focuses on these two consistencies because they have both been shown to be beneficial when used for lookahead during search.

For POAC and RNIC we introduce four and three strategies, respectively, to increment the weights of the constraints. For both consistencies we find that a baseline strategy corresponding to the original dom/wdeg proposal is statistically the worst of the proposed strategies. We conclude the high-level consistency should influence the weights. For POAC we find that the proposed strategy ALLS is statistically the best. For RNIC the two non-baseline strategies are statistically equivalent.

Other popular variable-ordering heuristics include Impact-Based Search [Refalo, 2004] and Activity-Based Search [Michel and Van Hentenryck, 2012]. These heuristics rely on information about the domain filtering resulting from enforcing a given consistency. Because they ignore the operations of the consistency algorithm, it is not clear how these heuristics could be used to order the propagation queue of the consistency algorithm [Wallace and Freuder, 1992; Balafrej *et al.*, 2014]. Further, it is also not clear how to apply them in the context of consistency algorithms that filter the relations [Woodward *et al.*, 2011; Woodward *et al.*, 2012].

The paper is structured as follows. Section 2 summarizes relevant background information. Section 3 introduces our weighting schemes for POAC and RNIC and Section 4 empirically evaluates them. Finally, Section 5 concludes the paper.

2 Background

A Constraint Satisfaction Problem (CSP) is defined by $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$. $\mathcal{X}$ is a set of variables where a variable $x_i \in \mathcal{X}$ has a finite domain $dom(x_i) \in \mathcal{D}$. A constraint $c_i \in \mathcal{C}$ is specified by its scope $scp(c_i)$ and its relation $rel(c_i)$. $scp(c_i)$ is the set of variables to which c_i applies and $rel(c_i)$ is the set of allowed tuples. A tuple on $scp(c_i)$ is consistent with c_i if it belongs to $rel(c_i) \cap \prod_{x_i \in scp(c_i)} dom(x_i)$. A solution to the CSP assigns, to each variable, a value taken from its domain such that all the constraints are satisfied. The problem is to determine the existence of a solution and is known to be NP-complete. To this day, backtrack search remains the only known sound and complete algorithm for solving CSPs [Bitner and Reingold, 1975]. Search operates by assigning a value to a variable and backtracks when a dead-end is encountered. The variable-ordering heuristic determines the order that variables are assigned in search, which can be dynamic (i.e., change during search). Boussemart *et al.* [2004] introduced dom/wdeg, a popular dynamic variable-ordering heuristic. This heuristic associates to each constraint $c \in \mathcal{C}$ a weight $w_c(c)$, initialized to one, that is incremented by one whenever the constraint causes a domain wipeout when enforcing arc consistency. The next variable x_i chosen by dom/wdeg is the one with the smallest ratio of current domain size to the weighted degree, $\alpha_{wdeg}(x_i)$, given by

$$\alpha_{wdeg}(x_i) = \sum_{(c \in \mathcal{C}_f) \wedge (x_i \in scp(C))} w_c(c) \tag{1}$$

where $\mathcal{C}_f \subseteq \mathcal{C}$ is the set of constraints with at least two future variables (i.e., variables who have not been assigned by search).

Modern solvers enforce a given consistency property on the CSP after each variable assignment. This lookahead removes from the domains of the unassigned variables values that cannot participate in a solution. Such filtering prunes from the search space fruitless subtrees, reducing thrashing and the size of the search space. The higher the consistency level enforced during lookahead, the stronger the pruning and the smaller the search space.

The standard property for lookahead is Generalized Arc Consistency (GAC) [Mackworth, 1977]. A CSP is GAC iff, for every constraint c_i, and every variable $x \in scp(c_i)$, every value $v \in dom(x)$ is consistent with c_i (i.e., appears in some consistent tuple of rel_i). Singleton Arc-Consistency (SAC) ensures that no domain becomes empty when enforcing GAC after assigning a value to a variable [Debruyne and Bessière, 1997]. This operation is called

3

a *singleton test*. Algorithms for enforcing SAC remove all domain values that fail the singleton test. Partition-One Arc-Consistency (POAC) adds an additional condition to SAC [Bennaceur and Affane, 2001]. Let (x_i, v_i) denotes a variable-value pair, $(x_i, v_i) \in \mathcal{P}$ iff $v_i \in dom(x_i)$. A constraint network $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is Partition-One Arc-Consistent (POAC) iff $\mathcal{P}$ is SAC and for all $x_i \in \mathcal{X}$, for all $v_i \in dom(x_i)$, for all $x_j \in X$, there exists $v_j \in dom(x_j)$ such that $(x_i, v_i) \in \text{GAC}(\mathcal{P} \cup \{x_j \leftarrow v_j\})$, where $\text{GAC}(\mathcal{P} \cup \{x_j \leftarrow v_j\})$ is the CSP after assigning $x_j \leftarrow v_j$ and running GAC [Bennaceur and Affane, 2001].

Using the terminology of Debruyne and Bessière [1997], we say that a consistency property p is *stronger* than p' if in any CSP where p holds p' also holds. Further, we say that p is *strictly stronger* than p' if p is stronger than p', and there exists at least one CSP in which p' holds but p does not. We say that p and p' are equivalent if p is stronger than p', and vice versa. Finally, we say that p and p' are incomparable when there exists at least one CSP in which p holds but p' does not, and vice versa. In practice, when a consistency property p is stronger than another p', enforcing p never yields less pruning than enforcing p' on the same problem. POAC is strictly stronger than SAC and SAC than GAC.

Balafrej *et al.* [2014] introduced two algorithms for enforcing POAC: POAC-1 and its adaptive version APOAC. POAC-1 operates by enforcing SAC. When running a singleton test on each of the values in the domain of a given variable, POAC-1 maintains a counter for each value in the domain of the remaining variables to determine whether or not the corresponding value was removed by any of the singleton tests. Values that are removed by each of those singleton tests are identified as not POAC and removed from their respective domains. POAC-1 was found to reach quiescence faster than SAC. In POAC-1, all the CSP variables are singleton tested and the process is repeated over all the variables until a fixpoint is reached. In APOAC, the adaptive version of POAC-1, the process is interrupted as soon as a given number of variables are singleton tested. This number depends on input parameters and is updated by learning during search.

Neighborhood Inverse Consistency (NIC) [Freuder and Elfe, 1996] ensures that every value in the domain of a variable x_i can be extended to a solution of the subproblem induced by x_i and the variables in its neighborhood. In the dual graph of a CSP, the vertices represent the CSP constraints and the edges connect vertices representing constraints whose scopes overlap. Relational Neighborhood Inverse Consistency (RNIC) [Woodward *et al.*, 2011]

enforces NIC on the dual graph of the CSP. That is, it ensures that any tuple in any relation can be extended in a consistent assignment to all the relations in its neighborhood in the dual graph. NIC and RNIC are theoretically incomparable [Woodward *et al.*, 2012], but RNIC has two main advantages over NIC. First, NIC was originally proposed for binary CSPs and the neighborhoods in NIC likely grow too large on non-binary CSPs; second, RNIC can operate on different dual graph structures to save time. Three variations of RNIC were introduced, wRNIC, triRNIC, and wtriRNIC, which operate on modified dual graphs. Given an instance, selRNIC uses a decision tree to automatically select the dual graph for RNIC to operate on.

3 Weighting Schemes

We introduce weighting schemes first in the context of singleton consistencies, namely Partition-One Arc-Consistency (POAC), and then in that of relational consistencies, namely Relational Neighborhood Inverse Consistency (RNIC).

Enforcing a high-level consistency (HLC) property is typically costlier than enforcing GAC, but typically yields more powerful pruning. Further, it is often more effective, in terms of CPU time, to run a GAC before an HLC algorithm [Debruyne and Bessière, 1997], as we choose to do in this paper.

3.1 Partition-One Arc-Consistency

We first investigate the case of POAC, which operates by initially running a GAC algorithm then applying the following operation to each variable until no change occurs. For a given variable, it applies a singleton test to each value in the domain of the variable. A singleton test assigns the value to the variable and enforces GAC on the problem. We propose four strategies to increment weights during POAC:

OLD: We allow only the GAC call before POAC to increment the weight of the constraint that causes a domain wipeout. That is, POAC is not allowed to alter the weights. This strategy is the simplest and it is a direct application of the original proposal [Boussemart *et al.*, 2004]. In our experiments we use this strategy as a baseline and show it does not perform well in practice.

ALLS: In addition to incrementing the weights according the above strategy (i.e., OLD), we allow every singleton test to increment the weight of a constraint whenever enforcing GAC on this constraint during the singleton test directly wipes out the domain of a variable. This update is made at most once for each singleton test. Under this strategy, all constraints that caused domain wipeouts are affected, thus, we call it ALLS. Notice that the weight of more than one constraint may be updated even though search does not have to backtrack. This behavior differs from the original proposal [Boussemart *et al.*, 2004].

LASTS: In addition to incrementing the weights according to OLD, we increment the weight of the constraint causing a domain wipeout at the *last* singleton test on a given variable if and only if all previous singleton tests on the values of this variable have failed. Thus, we only increment the weight of a single constraint and do so only when search has to backtrack, which conforms to the spirit of the original heuristic. Notice, the order of values singleton tested affects this strategy.

VAR: This strategy encapsulates OLD as a first step and increments the weight of the *variable* on which all singleton tests have failed (thus forcing search to backtrack). In order to implement this strategy we add a counter for the weight of each variable w_v, initially zero. When a variable fails all of its singleton tests during propagation the counter w_v for that variable is incremented by one. We propose to integrate w_v with the weighted degree function of dom/wdeg as follows:

$$\alpha_{wdeg}^{\text{VAR}}(x_i) = w_v(x_i) + \sum_{(c \in \mathcal{C}_f) \wedge (x_i \in scp(c))} w_c(c) \tag{2}$$

where $\mathcal{C}_f \subseteq \mathcal{C}$ is the set of constraints with at least two future variables. The rationale behind this strategy is the following. The goal of the heuristic dom/wdeg is to identify the conflicts in the problem and address them earlier, rather than later, in the search. VAR puts the blame on the variable that first caused the failure of POAC.

3.2 Relational Neighborhood Inverse Consistency

The relational consistency property RNIC is equivalent to enforcing Neighborhood Inverse Consistency (NIC) on the dual graph of the CSP [Freuder

and Elfe, 1996; Woodward *et al.*, 2011]. The RNIC property ensures that every tuple in every relation can be extended to a solution in the subproblem induced on the dual graph of the CSP by the relation and its neighboring relations. The RNIC algorithm operates on table constraints and removes, from a given relation, all the tuples that do not appear in a solution in the induced (dual) CSP of its neighborhood [Woodward *et al.*, 2011]. We propose three strategies to increment weights when RNIC is used for lookahead during search:

OLD: As in POAC in Section 3.1, we allow only the GAC call (preceding the call to RNIC) to increment the weight of the constraint that causes domain wipeout.

ALLC: This strategy encapsulates OLD as a first step. During lookahead, RNIC is called on each constraint with two or more future variables. When the RNIC algorithm removes all the tuples of a given relation, ALLC increments the weights of all the relations in the induced (dual) CSP. The rationale being that this considered combination of relations (which is the relation and its neighborhood in the dual graph) is 'collectively' responsible for the 'relation' wipeout.

HEAD: This strategy is similar to ALLC, except that we increment only the weight of the constraint whose relation was emptied by the RNIC algorithm and do not increment the weights of its neighborhood in the dual graph.

4   Experimental Evaluation

We evaluate the effectiveness of the strategies proposed for POAC and RNIC in Sections 4.2 and 4.3, respectively.

4.1   Experimental Setup

We consider the problem of finding a single solution to a CSP using backtrack search with some lookahead, d-way branching, dom/wdeg dynamic variable-ordering heuristic [Boussemart *et al.*, 2004], and lexicographic value ordering. We use STR2+ for enforcing GAC [Lecoutre, 2011], APOAC for enforcing

POAC [Balafrej *et al.*, 2014],[1] and selRNIC for enforcing RNIC [Woodward *et al.*, 2011]. We use the benchmark problems available from Lecoutre's website.[2] Benchmarks are selected separately for POAC and RNIC. For a given consistency level, if any instance is solved by any of the weighing schemas of the considered consistency within the time limit of 60 minutes and memory limit of 8GB, then the entire benchmark is included in the experiment. For benchmarks in intension we convert the instance to extension prior to solving and do not include the time for conversion.[3] From the 254 benchmark problems (total 8,549 instances) available on Lecoutre's website, our results are reported on 144 benchmarks (total 4,233 instances) for POAC and 132 (total 3,869 instances) for RNIC.

We summarize the results of these experiments in Tables 2–7 and Figures 1 and 2. For each strategy, we report in Tables 2–7:

- The number of completions (# Completions) with the total number of instances in parenthesis.

- The sum of the CPU time in seconds ($\sum$CPU sec.) computed over instances where at least one algorithm terminated (given in parenthesis). When an algorithm does not terminate within 60 minutes, we add 3,600 seconds to the CPU time and indicate with a > sign that the time reported is a lower bound. We boldface the smallest CPU time.

- The average number of node visits (Average NV) computed over the instances where all strategies completed (given in parenthesis).

Figures 1 and 2 plot the number of instances solved by each strategy (Y-axis) as the CPU time increases (X-axis).

In addition to the above experiment, we also conduct a statistical analysis of the relative performance of the proposed strategies. We compare pairwise

---

[1]Using the terminology of Balafrej et al. [Balafrej *et al.*, 2014], we use the following parameters and their recommended values for APOAC $maxK = n$, last drop with $\beta = 0.05$, and 70%-PER. Where $maxK$ indicates the number of processed items in the propagation queue, β is the threshold of search-space reduction during the learning phase and 70%-PER is the percentile for learning the value of $maxK$.

[2]`www.cril.univ-artois.fr/~lecoutre/benchmarks.html`

[3]In a study not reported we found that STR2+ is faster at solving CSP instances than running GAC on the original intension constraints because STR explores the satisfying tuples instead of valid tuples. As STR and RNIC algorithms require table constraints we pre-convert the instances. The conversion time is the same for each algorithm and can safely be ignored.

the strategies corresponding to each higher-level consistency (i.e., POAC and RNIC) in order to determine whether or not a statistical difference exists between the strategies. Because search may fail to complete within the time limit, we consider our results to be right-censored and analyze them using a nonparameterized Wilcoxon signed-rank test [Wilcoxon, 1945]. The test operates by comparing the rank of the differences of the paired data. Differences of zero have no effect on the test and are safely discarded before ranking. Further, given the clock precision, we discard data points where the CPU difference is less than one second. We assume a one-tailed distribution and significance level of $p = 0.05$.[4] In the presence of censored data, we adopt the following procedure to generate the data for each pairwise test. First, we run each strategy on each instance for the time limit (i.e., 60 minutes). If both strategies solve the instance, the data is included in the analysis. If neither strategy solves the instance, the instance is excluded from the analysis (i.e., the difference is zero and discarded). If one strategy completes within the time threshold and the other does not, we re-run the second strategy with double the time limit (i.e., 120 minutes), recording this limit as the completion time in case search does not terminate earlier. By allowing the additional time, the censored data no longer affects the significance of the analysis [Palmieri *et al.*, 2016].[5] The results obtained with the doubled time limit are used only for the statistical analysis ranking the relative performance of the strategies (Table 1 and Expression (3)), but not used for the results reported in Tables 2–7.

4.2 Partition-One Arc-Consistency

Based on the statistical analysis comparing the relative performance for OLD, ALLS, LASTS, and VAR for POAC, we conclude that *overall* (Table 1):

- ALLS outperforms all others strategies

- LASTS and VAR are equivalent

- OLD exhibits the worst performance of the four strategies, showing that it is important for dom/wdeg to increment the weights with POAC,

---

[4]Check Palmieri et al. [Palmieri *et al.*, 2016] for an overview of the Wilcoxon signed-rank test and the adopted methodology.

[5]Our approach is similar to that of Palmieri et al. [Palmieri *et al.*, 2016] except that we exclude instances that neither strategy completes with the original time limit.

Table 1: Statistical analysis of weighting schemes for POAC

Benchmark	Ranking						
All benchmarks, put together	ALLS	>	LASTS	≡	VAR	>	OLD
'QCP/QWH,' 'BQWH' (quasi-group completion)	LASTS	>	OLD	>	ALLS	≡	VAR
'Graph Coloring'	VAR	>	ALLS	>	LASTS	>	OLD
'RAND' (random)	VAR	>	ALLS	≡	LASTS	≡	OLD
'Crossword'	VAR	>	ALLS	≡	LASTS	≡	OLD

which justifies our investigations.

However, a careful study of the individual benchmarks shows that LASTS on many quasi-group completion benchmarks and VAR are competitive on many, but *not* all, graph coloring, random, and crossword benchmarks.[6] Rerunning the statistical analysis on each group of those benchmarks yields the results shown in the last four rows of Table 1. Again, we insist that even when considering individual benchmarks, the performance of ALLS remains *globally* the most robust and consistent of all four strategies.

Table 2 summarizes the experiments' results on the 144 tested benchmarks. In terms of the number of completed instances and the CPU time,

Table 2: Overall results of experiments for POAC

	OLD	ALLS	LASTS	VAR
Completion (4,233)	2,804	2,822	2,814	2,811
∑CPU sec. (2,846)	>1,139,552	**>1,033,699**	>1,075,640	>1,065,547
Average NV (2,775)	19,181	16,712	16,503	21,875

ALLS is the best (with 2,822 instances and >1,033,699 seconds) and OLD is the worst (with 2,804 instances and >1,139,552 seconds) of the four proposed strategies. In terms of the average number of nodes visited (i.e., reduction

---

[6]Using the categories identified on Lecoutre's website.

of the search space), LASTS visits the least amount of nodes on average (16,503), followed by ALLS (16,712), OLD (19,181), and VAR (21,875).[7]

Table 3 summarizes individual benchmark results for the quasi-group completion category. Compared to the quasi-group completion analysis in Table 1, the benchmarks typically follow the statistical trend with LASTS performing the best on the QCP-15 and QWH-20 benchmarks. However, although LASTS was statistically the best, on bqwh-15-106, ALLS was the fastest.

Table 3: Examples of quasi-group completion benchmark for POAC

Benchmark		OLD	ALLS	LASTS	VAR
Where LASTS *performs best*					
QCP-15	Completion (15)	15	15	15	15
	$\sum$CPU sec. (15)	3,920	5,480	**3,214**	6,083
	Average NV (15)	30,488	38,641	23,963	33,589
QWH-20	Completion (10)	9	9	9	9
	$\sum$CPU sec. (9)	6,625	7,329	**5,631**	12,337
	Average NV (9)	57,453	58,623	45,095	63,225
... but ALLS *can still win on such benchmarks*					
bqwh-15-106	Completion (100)	100	100	100	100
	$\sum$CPU sec. (100)	196	**167**	189	211
	Average NV (100)	599	433	531	507

Table 4 summarizes individual benchmarks for graph coloring, random, and crossword benchmarks. For these categories of benchmarks the statistical analysis of Table 1 shows that VAR performs the best. Indeed, for full-insertion, tightness0.8, and wordsVg VAR has the smallest CPU time of the strategies. However, individual benchmarks may vary despite the identified statistical groupings. For example, ALLS performs best on the tightness0.1, sgb-book, and ukVg benchmark, respectively.

We conclude that, unless we know enough about the problem instance

---

[7]We offer the following hypothesis as to why VAR has the largest average of nodes visited. The heuristic dom/wdeg is a 'conflict-directed' heuristic in that it attempts to select the variable that participates in the largest number of 'wipeouts.' By incrementing the weight of the variable being singleton-tested, VAR perhaps increases the importance of a variable that 'sees' the conflict rather than those variables that 'cause' the conflict. This hypothesis deserves a more thorough investigation.

Table 4: Examples of graph coloring, random, crossword benchmarks for POAC

Benchmark		OLD	ALLS	LASTS	VAR
Where VAR *performs best*					
full-insertion	Completion (41)	28	28	28	29
	∑CPU sec. (29)	>12,720	>10,055	>10,182	**7,238**
	Average NV (28)	16,725	12,676	13,312	8,749
tightness0.8	Completion (100)	98	97	97	99
	∑CPU sec. (99)	>59,907	>53,042	>56,945	**41,848**
	Average NV (97)	1,213	1,085	1,196	1,315
wordsVg	Completion (65)	55	56	54	59
	∑CPU sec. (59)	>24,376	>24,190	>28,533	**17,913**
	Average NV (54)	298	391	411	250
... but ALLS *can still win on such benchmarks*					
sgb-book	Completion (26)	20	20	20	20
	∑CPU sec. (20)	9,677	**8,315**	8,455	8,565
	Average NV (20)	143,653	148,055	148,985	134,099
tightness0.1	Completion (100)	100	100	100	100
	∑CPU sec. (100)	46,926	**43,766**	44,971	69,974
	Average NV (100)	10,347	9,762	9,948	12,457
ukVg	Completion (65)	29	31	28	30
	∑CPU sec. (31)	>19,466	**19,040**	>20,961	>19,119
	Average NV (28)	141	411	133	139

under consideration, we should use ALLS in conjunction with POAC, as the overall analysis shows us.

Figure 1 shows the cumulative number of instances completed by each strategy as CPU time increases. For easy instances (< 100 seconds), the completions of the strategies are similar. As the time limit increases OLD becomes dominated by the other three strategies. To better compare ALLS, LASTS, and VAR we examine the hard instances, zooming the chart on the cumulative CPU time solved between 1,000 and 3,600 seconds. Although VAR performs well on smaller CPU time (VAR contends with ALLS for the most completed instances between 1,000 and 1,700 seconds) it becomes dominated by ALLS and LASTS on the harder instances. ALLS clearly dominates all other strategies. These curves confirm the results of the statistical analysis given in Table 1.
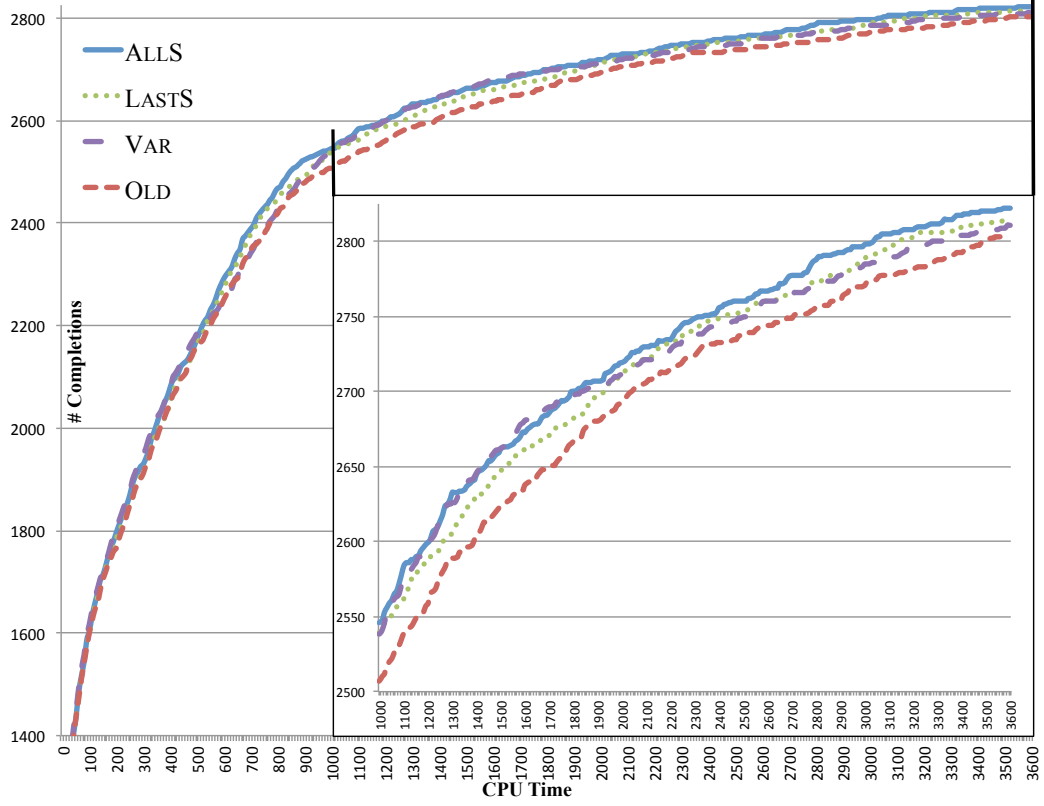
Figure 1: Cumulative number of instances completed by CPU time for POAC

4.3 Relational Neighborhood Inverse Consistency

The statistical analysis compares the relative performance for OLD, ALLC, and HEAD for RNIC. It shows that, *overall*, ALLC and HEAD are equivalent and OLD has the worst performance. The following holds in general for all benchmarks:

$$\text{ALLC} \equiv \text{HEAD} > \text{OLD} \tag{3}$$

The fact that OLD is the worst demonstrates that RNIC's contribution to the weights of dom/wdeg should not be ignored, thus justifying our investigations.

Table 5 summarizes the experiments' results on all the 132 tested benchmarks. ALLC is the best strategy on all measures while OLD is the worst.

13

Table 5: Results of experiments for RNIC

	OLD	ALLC	HEAD
# Completion (3,869)	2,420	2,427	2,423
$\sum$CPU sec. (2,416)	>1,032,130	>**1,010,221**	>1,014,635
Average NV (2,432)	77,067	45,696	45,803

We were not able to uncover meaningful categories of benchmarks to distinguish between ALLC and HEAD. Table 6 summarizes individual benchmark results for the Dimacs category. Within the category, either ALLC or HEAD perform the best by all measures on different benchmarks. Similar results are obtained on the graph coloring category, shown in Table 7. Having such different results between ALLC and HEAD explains why the statistical analysis found them to be equivalent. Regardless, either ALLC or HEAD performs better than OLD in a statistically significant manner.

Figure 2 shows the cumulative number of instances completed by each strategy as CPU time increases. As was the case for POAC, on easy instances (< 100 seconds), the completions of the strategies are similar. Focusing on harder instances, solved between 2,300 and 3,600 seconds, OLD becomes dominated by ALLC and HEAD. The curves of ALLC and HEAD remain close to one another. These curves confirm the ranking in Equation 3.

Table 6: Examples of Dimacs benchmarks where ALLC and HEAD perform best

Benchmark		OLD	ALLC	HEAD
	Completion (8)	4	4	4
pret	ΣCPU (4)	196	**28**	61
	Average NV (4)	1,285,234	125,793	273,736
	Completion (13)	6	9	11
dubois	ΣCPU (6)	>22,041	>10,088	**1,348**
	Average NV (11)	11,222,349	1,522,902	382,329

Table 7: Two graph coloring benchmarks where AllC and Head perform best

Benchmark		Old	AllC	Head
mug	Completion (8)	8	8	8
	ΣCPU (8)	5,098	**548**	2,819
	Average NV (8)	1,501,379	189,595	883,130
leighton-15	Completion (26)	5	5	5
	ΣCPU (5)	2,219	1,493	**1,222**
	Average NV (5)	25,014	12,461	4,972
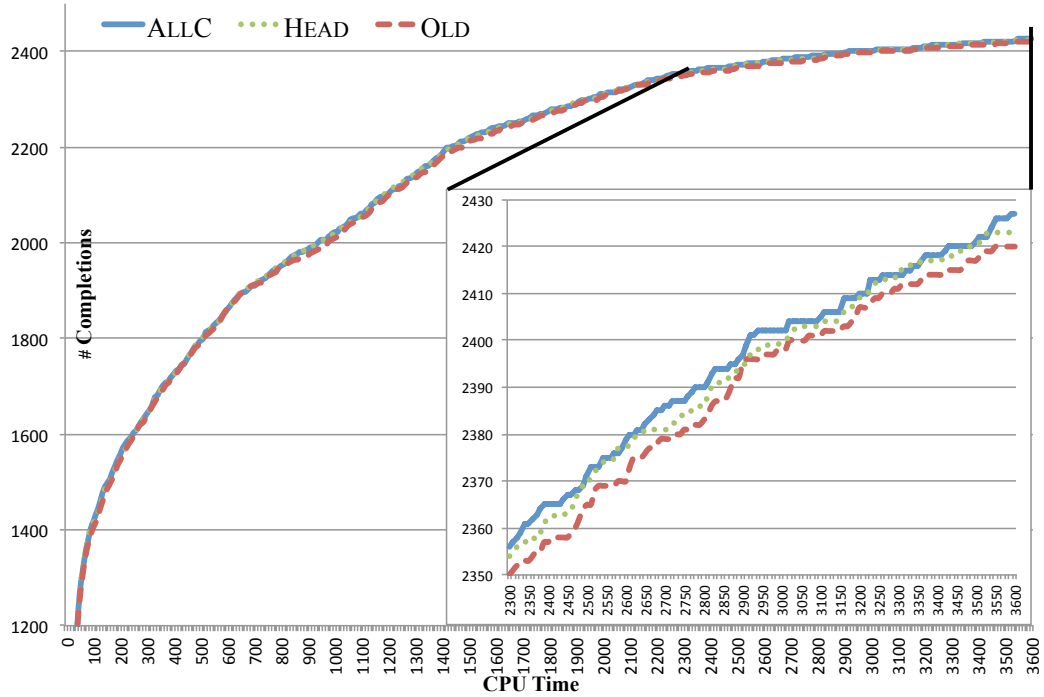
Figure 2: Cumulative number of instances completed by CPU time for RNIC

5 Conclusion

This paper introduces four strategies for incrementing the weight in dom/wdeg for singleton consistencies (POAC) and three strategies for relational consistencies (RNIC). For both consistencies, OLD is the worst strategy and a weighting schema involving the higher-level consistency is necessary. We show that for POAC the best method is ALLS, which increments the weights

15

at every singleton test. For RNIC, we show ALLC and HEAD are statistically equivalent. Our work is a first step in the right direction, especially given the importance of higher-level consistencies in solving difficult CSPs. Future work may need to investigate more complex strategies for these and other consistencies.

Acknowledgments

References

[Balafrej *et al.*, 2014] Amine Balafrej, Christian Bessiere, El-Houssine Bouyakhf, and Gilles Trombettoni. Adaptive Singleton-Based Consistencies. In *Proc. of AAAI 2014*, pages 2601–2607, 2014.

[Bennaceur and Affane, 2001] Hachemi Bennaceur and Mohamed-Salah Affane. Partition-k-AC: An Efficient Filtering Technique Combining Domain Partition and Arc Consistency. In *Proc. of CP 2001*, volume 2239 of *LNCS*, pages 560–564, 2001.

[Bitner and Reingold, 1975] James R. Bitner and Edward M. Reingold. Backtrack Programming Techniques. *Communications of the ACM*, 18(11):651–656, November 1975.

[Boussemart *et al.*, 2004] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting Systematic Search by Weighting Constraints. In *Proc. of ECAI 2004*, pages 146–150, 2004.

[Debruyne and Bessière, 1997] Romuald Debruyne and Christian Bessière. Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In *IJCAI 1997*, pages 412–417, 1997.

[Freuder and Elfe, 1996] Eugene C. Freuder and Charles D. Elfe. Neighborhood Inverse Consistency Preprocessing. In *Proc. of AAAI 1996*, pages 202–208, 1996.

[Lecoutre, 2011] Christophe Lecoutre. STR2: Optimized Simple Tabular Reduction for Table Constraints. *Constraints*, 16(4):341–371, 2011.

[Mackworth, 1977] Alan K. Mackworth. On Reading Sketch Maps. In *Proc. of IJCAI 77*, pages 598–606, 1977.

[Michel and Van Hentenryck, 2012] Laurent Michel and Pascal Van Hentenryck. Activity-Based Search for Black-Box Constraint Programming Solvers. In *Proc. of CPAIOR 2012*, volume 7298, pages 228–243. Spring, 2012.

[Palmieri *et al.*, 2016] Anthony Palmieri, Jean-Charles Régin, and Pierre Schaus. Parallel Strategies Selection. In *Proc. of CP 2016*, volume 9892 of *LNCS*, pages 388–404. Springer, 2016.

[Refalo, 2004] Philippe Refalo. Impact-Based Search Strategies for Constraint Programming. In *Proc. of CP 2004*, volume 3258 of *LNCS*, pages 557–571. Springer, 2004.

[Vion *et al.*, 2011] Julien Vion, Thierry Petit, and Narendra Jussien. Integrating Strong Local Consistencies into Constraint Solvers. In *14th Annual ERCIM International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2009*, volume 6080 of *LNAI*, pages 90–104. Springer, 2011.

[Wallace and Freuder, 1992] Richard J. Wallace and Eugene C. Freuder. Ordering Heuristics for Arc Consistency Algorithms. In *AI/GI/VI 92*, pages 163–169, 1992.

[Wilcoxon, 1945] Frank Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[Woodward *et al.*, 2011] Robert Woodward, Shant Karakashian, Berthe Y. Choueiry, and Christian Bessiere. Solving Difficult CSPs with Relational Neighborhood Inverse Consistency. In *Proc. of AAAI 11*, pages 112–119, 2011.

[Woodward *et al.*, 2012] Robert J. Woodward, Shant Karakashian, Berthe Y. Choueiry, and Christian Bessiere. Revisiting Neighborhood Inverse Consistency on Binary CSPs. In *Proc. of CP 2012*, volume 7514 of *LNCS*, pages 688–703. Springer, 2012.