Experimental evaluation of modern variable selection strategies in Constraint Satisfaction Problems

Thanasis Balafoutis[1] and Kostas Stergiou[1]

Department of Information & Communication Systems Engineering
University of the Aegean
Samos, Greece
{abalafoutis,konsterg}@aegean.gr

Abstract

Constraint programming is a powerful technique for solving combinatorial search problems that draws on a wide range of methods from artificial intelligence and computer science. Constraint solvers search the solution space either systematically, as with backtracking or branch and bound algorithms, or use forms of local search which may be incomplete. Systematic methods typically interleave search and inference. A key factor that can dramatically reduce the search space is the criterion under which we decide which variable will be the next to be instantiated. Numerous heuristics have been proposed for this purpose in the literature. Recent years have seen the emergence of new and powerful methods for choosing variables during CSP search. Some of these methods exploit information about failures gathered throughout search and recorded in the form of constraint weights, while others measure the importance/impact of variable assignments for reducing the search space. In this paper we experimentally evaluate the most recent and powerful variable ordering heuristics, and new variants of them, over a wide range of academic, random and real world problems. Results demonstrate that heuristics based on failures are in general faster. To be precise, heuristic dom/wdeg and its variants are the dominant heuristics in most instances tried.

1 Introduction

Constraint programming is a powerful technique for solving combinatorial search problems that draws on a wide range of methods from artificial intelligence and computer science. The basic idea in constraint programming is that the user states the constraints and a general purpose constraint solver is used to solve them. Constraints are just relations and a constraint satisfaction problem (CSP) states which relations hold among the given decision variables.

CSPs can be solved either systematically, as with backtracking or branch and bound algorithms, or using forms of local search which may be incomplete. When solving a CSP using backtracking search, a sequence of decisions must be made as to which variable to instantiate next. These decisions

Proceedings of the 15th International RCRA workshop (RCRA 2008):
Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion
Udine, Italy, 12–13 December 2008

are referred to as the variable ordering decisions. It has been shown that for many problems, the choice of variable ordering can have a dramatic effect on the performance of the backtracking algorithm with huge variances even on a single instance.

A variable ordering can be either static, where the ordering is fixed and determined prior to search, or dynamic, where the ordering is determined as the search progresses. Dynamic variable orderings have received much attention in the literature. One common dynamic variable ordering (DVO) strategy, known as "fail-first", is to select as the next variable the one likely to constrain the remainder of the search space the most. All other factors being equal, the variable with the smallest number of viable values in its (current) domain will have the fewest subtrees rooted at those values, and therefore the smallest search space below it.

Recent years have seen the emergence of numerous modern heuristics for choosing variables during CSP search. Most of them are quite successful and the choice of best general purpose heuristic is not an easy procedure. All these new heuristics, in their original papers, have been tested over a narrow set of problems and they have been compared mainly with older heuristics. Hence, there is no accurate view of the strengths and weaknesses of these heuristics. The aim of this paper is to experimentally compare and evaluate the efficiency of the modern variable ordering heuristics, and new variants of them, over a wide range of academic, random and real world problems.

The rest of the paper is organized as follows. Section 2 gives the necessary background material. In Section 3 we briefly overview the existing variable ordering heuristics, while in Section 4 we give additional details on the heuristics which we have selected for the evaluation. In Section 5 we present experimental results from a wide variety of real, academic and random problems. Finally, a general discussion and conclusions are presented in Section 6.

2 Background

A *Constraint Satisfaction Problem* (CSP) is a tuple (X, D, C), where X is a set containing n variables $\{x_1, x_2, ..., x_n\}$; D is a set of domains $\{D(x_1), D(x_2),..., D(x_n)\}$ for those variables, with each $D(x_i)$ consisting of the possible values which x_i may take; and C is a set of constraints $\{c_1, c_2, ..., c_k\}$ between variables in subsets of X. Each $c_i \in C$ expresses a relation defining which variable assignment combinations are allowed for the variables *vars(c_i)* in the scope of the constraint. Two variables are said to be *neighbors* if they share a constraint. The *arity* of a constraint is the number of variables in the scope of the constraint. The *degree* of a variable x_i, denoted by $\Gamma(x_i)$, is the number of constraints in which x_i participates. A binary constraint between variables x_i and x_j will be denoted by c_{ij}.

An arc (x_i,x_j) is *arc consistent* (AC) iff for every value $a \in D(x_i)$ there exists at least one value $b \in D(x_j)$ such that the pair (a,b) satisfies c_{ij}. In this case we say that b is a *support* of a on arc (x_i,x_j). Accordingly, a is a support of b on arc (x_j,x_i). A problem is AC iff there are no empty domains and all arcs are AC. The application of AC on a problem results in the removal of all non-supported values from the domains of the variables. The definition of arc consistency for non binary constraints (also called *generalized arc consistency*, or GAC) is a direct extension of the binary one [14, 13].

A *support check* (consistency check) is a test to find out if two values support each other. The *revision* of an arc (x_i,x_j) using AC verifies if all values in $D(x_i)$ have supports in $D(x_j)$. A *DWO-revision* is one that causes a *DWO*. That is, it results in an empty domain.

A partial assignment is a set of tuple pairs, each tuple consisting of an instantiated variable and the value that is assigned to it in the current search state. A full assignment is one containing all n variables. A solution to a CSP is a full assignment such that no constraint is violated.

3 Overview of existing variable ordering heuristics

The order in which variables are assigned by a backtracking search algorithm has been understood for a long time to be of prime importance. The first category of heuristics used for ordering variables was based on the initial structure of the network. These are called static or fixed variable ordering heuristics (SVOs) as they keep the same ordering of the variables during search. Examples of such heuristics are *lexico* where variables are ordered lexicographically, *min width* which chooses an ordering that minimizes the width of the constraint network [9], *min bandwidth* which minimizes the bandwidth of the constraint graph [20], and *min degree* (*deg*), where variables are ordered according to the initial size of their neighborhood [8].

A second category of heuristics includes dynamic variable ordering heuristics (DVOs) which take into account information about the current state of the problem at each point in search. The first well known dynamic heuristic, introduced by Haralick and Elliott, was *dom* [12]. This heuristic chooses the variable with the smallest remaining domain. The dynamic variation of *deg*, called *ddeg* selects the variable with largest dynamic degree. That is, the variable that is constrained with the largest number of unassigned variables. By combining *dom* and *deg* (or *ddeg*), the heuristics called *dom/deg* and *dom/ddeg* [3, 19] were derived. These heuristics select the variable that minimizes the ratio of current domain size to static degree (dynamic degree) and can significantly improve the search performance.

When using variable ordering heuristics, it is a common phenomenon that ties can occur. A tie is a situation where a number of variables are considered equivalent by a heuristic. Especially at the beginning of search,

where it is more likely that the domains of the variables are of equal size, ties are frequently noticed. A common tie breaker for the *dom* heuristic is *lexico*, (*dom+lexico* composed heuristic) which selects among the variables with smallest domain size the lexicographically first. Other known composed heuristics are *dom+deg* [10], *dom+ddeg* [5, 18] and *BZ3* [18].

Bessière et al. [2], have proposed a general formulation of DVOs which integrates in the selection function a measure of the constrainedness of the given variable. These heuristics (denoted as *mDVO*) take into account the variable's neighborhood and they can be considered as neighborhood generalizations of the *dom* and *dom/ddeg* heuristics. For instance, the selection function for variable X_i is described as follows:

$$H_a^{\circledcirc}(x_i) = \frac{\sum_{x_j \in \Gamma(x_i)} (\alpha(x_i) \circledcirc \alpha(x_j))}{|\Gamma(x_i)|^2} \quad (1)$$

where $\alpha(x_i)$ can be any simple syntactical property of the variable such as $|D(x_i)|$ or $\frac{|D(x_i)|}{|\Gamma(x_i)|}$ and $\circledcirc \in \{+, \times\}$. Neighborhood based heuristics have shown to be quite promising.

Boussemart et al. [4] proposed conflict-directed variable ordering heuristics. In these heuristics, every time a constraint causes a failure (i.e. a domain wipeout) during search, its weight is incremented by one. Each variable has a *weighted degree*, which is the sum of the weights over all constraints in which this variable participates. The weighted degree heuristic (*wdeg*) selects the variable with the largest weighted degree. The current domain of the variable can also be incorporated to give the domain-over-weighted-degree heuristic (*dom/wdeg*) which selects the variable with minimum ratio between current domain size and weighted degree. Both of these heuristics (especially *dom/wdeg*) have been shown to be extremely effective on a wide range of problems.

Grimes and Wallace [11] proposed alternative conflict-driven heuristics that consider value deletions as the basic propagation events associated with constraint weights. That is, the weight of a constraint is incremented each time the constraint causes one or more value deletions. They also used a sampling technique called *random probing* with which they can uncover cases of *global contention*, i.e. contention that holds across the entire search space.

Inspired from integer programming, Refalo introduced an *impact* measure with the aim of detecting choices which result in the strongest search space reduction [16]. An impact is an estimation of the importance of a value assignment for reducing the search space. He proposes to characterize the impact of a decision by computing the Cartesian product of the domains before and after the considered decision. The impacts of assignments for every value can be approximated by the use of averaged values at the current level of observation. If K is the index set of impacts observed so far for

assignment $x_i = \alpha$, $\overline{I}$ is the averaged impact:

$$\overline{I}(x_i = \alpha) = \frac{\sum_{k \in K} I^k(x_i = \alpha)}{|K|} \quad (2)$$

The impact of a variable x_i can be computed by the following equation:

$$I(x_i) = \sum_{\alpha \in D(x_i)} 1 - \overline{I}(x_i = \alpha) \quad (3)$$

An interesting extension of the above heuristic is the use of "node impacts" to break ties in a subset of variables that have equivalent impacts. Node impacts are the accurate impact values which can be computed for any variable by trying all possible assignments.

Correia and Barahona [7] propose variable orderings, by integrating Singleton Consistency propagation procedures with look-ahead heuristics. This heuristic is similar to "node impacts", but instead of computing the accurate impacts, it computes the reduction in the search space after the application of Restricted Singleton Consistency (RSC) [15], for every value of the current variable. Although this heuristic was firstly introduced to break ties in variables with current domain size equal to 2, it can also be used as a tie breaker for any other variable ordering heuristic.

Cambazard and Jussien [6] go a step further by analyzing where the reduction of the search space occurs and how past choices are involved in this reduction. This is implemented through the use of *explanations*. An explanation consists of a set of constraints C' (a subset of the set C of the original constraints of the problem) and a set of decisions $dc_1, ..., dc_n$ taken during search. An explanation of the removal of value a from variable v can be written as:

$$C' \wedge dc_1 \wedge dc_2 \wedge ... \wedge dc_n \Rightarrow v \neq a$$

Finally, Balafoutis and Stergiou [1], propose new variants of conflict-driven heuristics. These variants differ from *wdeg* in the way they assign weights. They propose heuristics that record the constraint that is responsible for any value deletion during search, heuristics that give greater importance to recent conflicts, and finally heuristics that try to identify contentious constraints by detecting all possible conflicts after a failure. The last heuristic, called "fully assigned", increases the weights of constraints that are responsible for a DWO by one (as *wdeg* heuristic does) and also, only for revision lists that lead to a DWO, increases by one the weights of constraints that participate in fruitful revisions (revisions that delete at least one value). Hence, this heuristic records all variables that delete at least one value during constraint propagation and if a DWO is detected, it increases the weight of all these variables by one.

4 Details on the evaluated heuristics

For the evaluation we have selected heuristics from 5 recent papers mentioned above. These are: i) *dom/wdeg* from Boussemart et al. [4], ii) the random probing technique and "alldel by #del" heuristic where constraint weights are increased by the size of the domain reduction (Grimes and Wallace [11]), iii) Impacts and Node Impacts from Refalo [16], iv) the "RSC" heuristic from Correia and Barahona [7] and finally v) the "fully assigned" heuristic from Balafoutis and Stergiou [1].

We have also included in our experiments some combinations of the above heuristics. For example, *dom/wdeg* can be combined with RSC (in this case RSC is used only to break ties). Random probing can be applied to any conflict-driven DVO, hence it can be used with the *dom/wdeg* and "fully assigned" heuristics. Moreover, the impact heuristic can be combined with RSC for breaking ties.

The full list of the heuristics that we have tried in our experiments includes 15 variations. These are the following: 1) dom/wdeg, 2) dom/wdeg + RSC (the second heuristic is used only for breaking ties), 3) dom/wdeg with random probing, 4) dom/wdeg with random probing + RSC, 5) Impacts, 6) Node Impacts, 7) Impacts + RSC, 8) alldel by #del, 9) alldel by #del + RSC, 10) alldel by #del with random probing, 11) alldel by #del with random probing + RSC, 12) fully assigned, 13) fully assigned + RSC, 14) fully assigned with random probing, and 15) fully assigned with random probing + RSC. In all these variations the RSC heuristic is used only for breaking ties.

5 Experiments and results

We now report results from the experimental evaluation of the selected DVOs described above on several classes of problems. In Section 5.1 we present results from the radio link frequency assignment problem (RLFAP). In Section 5.2 we present results from structured problems. These instances are taken from some academic (langford), real world (driver) and patterned (graph coloring) problems. In Section 5.3 we consider instances from quasi-random and random problems. The last experiments presented in Section 5.4 include Boolean instances. Finally in Section 5.5 we make a general discussion where we summarize our results.

In the experiments we used MGAC (maintaining generalized arc consistency) [17, 3] as our search algorithm. In MGAC a problem is made generalized arc consistent after every assignment, i.e. all values which are generalized arc inconsistent given that assignment, are removed from the current domains of their variables. If during this process a domain wipeout (DWO) occurs, then the last value selected is removed from the current do-

main of its variable and a new value is assigned to the variable. If no new value exists then the algorithm backtracks.

All benchmarks are taken from C. Lecoutre's web page[1], where the reader can find additional details on how the benchmarks are constructed. In our experiments we included both satisfiable and unsatisfiable instances. Each selected instance involves constraints defined either in intension or in extension (we have not included instances expressed with global constraints).

We have used two measures of performance for the DVOs tested: cpu time in seconds (t) and number of visited nodes (n). The solver we used applies d-way branching, and lexicographic value ordering. It also employs restarts. Concerning the restart policy, the initial number of allowed backtracks for the first run has been set to 10 and at each new run the number of allowed backtracks increases by a factor of 1.5.

In our experiments the random probing technique is run to a fixed cutoff C = 40, and for a fixed number of restarts R = 50 (these are the optimal values from [11]). Cpu time and nodes for random probing are averaged values for 10 runs. For this heuristic, we have measured the total cpu time and the total nodes visited (from both random probing initialization and final search). However, in the next tables (except Table 1) we have also put in parenthesis results from the final search only (random probing initialization is excluded).

Concerning impacts, we have approximated their values at the initialization phase by dividing the domains of the variables into (at maximum) four sub-domains. In all the experiments, a time out limit has been set to 1 hour.

5.1 RLFAP instances

The Radio Link Frequency Assignment Problem (RLFAP) is the task of assigning frequencies to a number of radio links so that a large number of constraints are simultaneously satisfied and as few distinct frequencies as possible are used. A number of modified RLFAP instances have been produced from the original set of problems by removing some frequencies (denoted by f followed by a value). For example, scen11-f8 corresponds to the instance scen11 for which the 8 highest frequencies have been removed.

Results from Table 1 show that conflict-driven heuristics (dom/wdeg, alldel and fully assigned) have the best performance. The Impact heuristic seems to make a better exploration of the search tree on some easy instances (like s2-f25, s11, s11-f12, s11-f11), but it is slower compared to conflict-driven heuristics. This is mainly because the process of impact initialization is time consuming. However, on hard instances, the Impact heuristic has worse performance and in some cases it cannot solve the problem within the time

[1] http://www.cril.univ-artois.fr/~lecoutre/research/benchmarks/

limit. In general we observed that impact based heuristics cannot handle efficiently problems which include variables with relatively large domains. RLFA problems, for example, have 680 variables with maximum 44 values in their domains.

Node Impact and its variation, "Impact RSC", are strongly related, and this similarity is depicted in the results. As mentioned in Section 3, Node Impact computes the accurate impacts and "RSC" heuristic computes the reduction in the search space, after the application of Restricted Singleton Consistency. Since node impact computation also uses Restricted Singleton Consistency (it subsumes it), these heuristics differ only in the measurement function that assigns impacts to variables. Hence, when they are used to break ties on the Impact heuristic, they usually make similar decisions.

When "RSC" is used as a tie breaker for conflict-driven heuristics, results show that it does not offer significant changes in the performance. So we have excluded it from the experiments that follow in the next sections, except for the dom/wdeg + RSC combination.

Concerning "random probing", although experiments in [11] show that it has often better performance when compared to simple *dom/wdeg*, our results show that this is not the case when *dom/wdeg* is combined with a geometric restart strategy. Even on hard instances, where the computation cost of random probes is smaller (compared to the total search cost) results show that *dom/wdeg* and its variations are dominant. Moreover, the combination of "random probing" with any other conflict-driven variation heuristic ("alldel" or "fully assigned") does not offer significant changes. Thus, for the next experiments we have kept only the "random probing" and *dom/wdeg* combination.

Finally, among the three conflict-driven variations, in this set of benchmarks, *dom/wdeg* seems to have a slightly better performance.

5.2 Structured instances

This set of experiments contains instances from academic problems (langford), some real world instances from the "driver" problem and 6 graph coloring instances.

Since some of the variations presented in the previous paragraph (Table 1) were shown to be less interesting, we have omitted their results from the next tables.

Results in Table 2 show that the behavior of the selected heuristics is close to the behavior that we observed in RLFA problems. Conflict-driven variations are again dominant here. The *dom/wdeg* heuristic has in most cases the best performance, followed by "alldel" and "fully assigned".

Although random probing has in most experiments worse cpu times compared to dom/wdeg, we have to notice that in the langford2-10 instance it is faster. Impact based heuristics have by far the worst performance.

Table 1: Cpu times (t), and nodes (n) from frequency allocation problems. Best cpu time is in bold. The s and g prefixes stand for scen and graph respectively.

Instance		*d/wdeg*	*d/wdeg r.probe*	*d/wdeg RSC*	*d/wdeg r.probe RSC*	*Impact*	*Node Impact*	*Impact RSC*	*alldel*	*alldel r.probe*	*alldel RSC*	*alldel r.probe RSC*	*fully*	*fully r.probe*	*fully RSC*	*fully r.probe RSC*
s2-f25	t	**7,5**	36,6	11,5	25,7	9,9	15	14,9	9,6	36,1	11,5	26,1	11,5	37,2	11,7	35,8
(unsat)	n	1815	17572	2109	16087	1507	1491	1491	2250	17052	2185	15831	2155	17721	1907	17754
s3-f10	t	2	44,4	18,7	43,5	13	32,2	33,2	1,9	48,8	9,6	35,5	**1,8**	44,1	14,7	45,5
(sat)	n	610	18520	697	15861	1292	1252	1252	645	17990	683	16572	544	18605	458	18653
s3-f11	t	7,4	50,6	9,9	44,3	>1h	>1h	>1h	16,8	69,8	16	49,5	**6,7**	58,7	10,2	62,5
(unsat)	n	969	19149	977	17043	-	-	-	2215	20656	2190	18784	774	20488	1098	20511
g8-f10	t	**21,9**	71	53,8	88,5	>1h	>1h	>1h	22,7	83,5	30	69,7	76,3	91,5	38,2	78,8
(sat)	n	6312	24655	4589	23698	-	-	-	6628	25990	4513	22948	9424	25296	3408	25987
g8-f11	t	4	75,8	10,9	67,3	91,4	100,6	99,8	2,7	87,1	1,3	67,7	**1**	76	1,5	77,1
(unsat)	n	936	23515	695	22119	12832	12832	12832	586	24004	245	21686	134	23576	132	24209
g14-f27	t	**22,9**	63,9	151,2	103,1	189,4	> 1h	337,6	30,6	85,5	77,6	77,7	58,3	52,5	103,9	73,2
(sat)	n	14103	30334	16986	22309	34862	-	27927	18635	39653	34849	42919	26582	26149	14626	38573
g14-f28	t	16,8	48,8	103,4	45,1	> 1h	> 1h	> 1h	**2,3**	274	37,9	46,8	17,7	58,4	32,5	60,9
(unsat)	n	7386	21175	23211	20825	-	-	-	1117	133031	15902	21601	6085	26156	8626	26224
s11	t	**4,8**	146	277,7	202	29,9	227,8	235,6	5,8	143	116,75	109,6	5,9	140,4	272,5	241
(sat)	n	836	35404	1020	28591	832	832	832	964	33582	1632	28155	949	34927	962	33927
s11-f12	t	**2,9**	75,1	5,7	58	23,5	31,1	26,2	2,9	80,1	7	58,9	3,5	69,5	4,6	78,5
(unsat)	n	448	24174	648	21551	327	327	327	612	23989	1079	21853	623	24644	630	24106
s11-f11	t	2,8	72,9	5,6	58,7	22,9	31,2	26,3	**2,6**	78,9	11,9	61,2	3,5	66,5	4,6	77,2
(unsat)	n	448	24379	648	21988	327	327	327	441	24270	1890	22280	623	24207	630	24279
s11-f10	t	4,3	68,1	5,4	60,8	> 1h	2153,7	1887,6	**2,8**	86,9	4,4	60	4,4	76,6	6,7	85,5
(unsat)	n	674	23398	482	20900	-	0,2M	0,2M	445	23496	469	21559	602	23489	640	23499
s11-f9	t	17,2	85,4	26,2	78,5	32,4	49,4	50	17,6	95,7	23	80,1	**14,2**	86,7	26	98
(unsat)	n	1767	24692	1716	22126	1610	1610	1610	1979	24572	1581	22056	1442	24950	1482	24892
s11-f8	t	34,5	107,6	48,7	101,2	55,1	80,7	81,7	26,2	120	40,4	96	**25,9**	109,8	36,8	125,9
(unsat)	n	2879	28399	2892	25484	3045	3126	3126	2717	28535	3090	25091	2375	28294	2317	28173
s11-f7	t	189,3	272,6	200,3	245,5	169,8	250,9	252,3	**137,7**	252	193,7	257	138	217,9	169,5	262,5
(unsat)	n	26139	53034	22341	49484	14536	19715	19715	17937	45247	22462	48078	16599	44669	15605	45012
s11-f6	t	**291,3**	473,5	555,7	505,7	> 1h	2216,5	2183,4	421,9	628	388,1	408,8	459,5	441,4	590,1	644
(unsat)	n	36331	78404	69463	71087	-	0,2M	0,2M	53998	86380	51657	68227	56190	73123	59274	89006

5.3 Random instances

In this set of experiments we have selected some quasi-random instances which contain some structure ("ehi" and "geo" problems) and also some purely random instances, generated following Model RB and Model D.

Model RB instances (frb30-15-1 and frb30-15-2), are random instances forced to be satisfiable. Model D instances are described by four numbers $<n,d,e,t>$. The first number n corresponds to the number of variables. The second number d is the domain size and e is the number of constraints. t is the tightness, which denotes the probability that a pair of values is allowed by a relation.

Results are presented in Table 3. All the conflict-driven heuristics (*dom /wdeg*, "alldel" and "fully assigned") have very similar node visits and much better cpu times compared to impact based heuristics. In pure random problems the "alldel" heuristic has the best cpu times, while in quasi-random instances the three conflict-driven heuristics share a win.

5.4 Boolean instances

This set of experiments contains instances involving only Boolean variables. We have selected a representative subset from Dimacs problems. All the selected instances have constraint arity equal to 3, except for the "jnhSat" instances which have maximum arity of 14.

Results from these experiments can be found in Table 4. The behavior of the evaluated heuristics in this data set is slightly different from the behavior that we observed in previous problems. Although conflict-driven heuristics again display here the best overall performance, impact based heuristics are in some cases faster.

One of the main bottlenecks that impact based heuristics have, is the time consuming initialization process. On binary instances, where the variables have binary domains, the cost for the initialization of impacts is small. And this can significantly increase the performance of these heuristics.

We observed also that in 'jnhSat" instances (which include constraints with maximum arity of 14)the Impact heuristic is in all cases faster than *dom/wdeg*. This result, have lead us to an empirical presumption that on binary problems with high constraint arity, the Impact heuristic is quite strong.

Among the conflict-driven heuristics, the "alldel" heuristic is always better than its competitors. We recall here that in this heuristic, constraint weights are increased by the size of the domain reduction. Hence, on binary instances constraint weights can be increased at minimum by one and at maximum by two (in each DWO). This way of incrementing weights seems to work better on binary problems.

However, all these empirical remarks are quite preliminary and further

Table 2: Cpu times (t), and nodes (n) from structured problems. Best cpu time is in bold.

Instance		*d/wdeg*	*d/wdeg r.probe*	*d/wdeg RSC*	*Impact*	*Node Impact*	*Impact RSC*	*alldel by #del*	*fully assigned*
langford-	t	**51,7**	54,8 (49,7)	59,4	73,4	85,2	83,2	54	54,5
2-9(unsat)	n	71056	76395 (70820)	70767	83492	89727	91147	74041	74126
langford-	t	446,3	**429,1** (422,3)	464,7	486,6	528,5	543,6	431,2	433,4
2-10(unsat)	n	494523	499595 (494060)	494685	480986	480975	485481	515053	512515
langford-	t	**634**	754,4 (700,4)	639,9	2839	3178	2172,5	783,2	732,5
3-11(unsat)	n	145687	155471 (148504)	148067	427142	445954	300355	165552	159894
langford-	t	**74,2**	264 (59,9)	108	157	220,1	221,8	85,1	76,1
4-10(unsat)	n	5640	13411 (3910)	5257	8834	9011	9001	6250	5703
driver-	t	17,7	56,6 (0,8)	36,8	31,3	35,5	34,7	**1,7**	1,8
8c (sat)	n	4613	9466 (411)	3556	426	424	424	669	638
driver-9	t	**159,1**	498,6 (388,9)	249,2	$> 1h$	593,4	2071	182,4	164,4
(sat)	n	31067	73437 (61871)	19110	-	11232	64639	16136	16447
will199-5	t	**1,6**	22,8 (4,2)	4,6	36,8	42,5	264,8	2	2,4
(unsat)	n	700	14076 (1786)	740	4267	4251	49619	741	730
will199-6	t	20,3	46,1 (20,2)	31,9	$> 1h$	42	42,1	15,5	**15,4**
(unsat)	n	5941	22912 (5849)	5827	-	3982	3982	3986	3976
ash608-4	t	4,1	25,1 (2,9)	79,6	34,3	343,2	115,2	3	**1,5**
(sat)	n	3168	21381 (2185)	2536	3342	4622	2329	2605	1269
ash958-4	t	15,2	45,3 (5,8)	289,7	110,8	650,2	497,5	**13,5**	14,8
(sat)	n	8418	27618 (3050)	3926	6902	4578	4578	7462	7551
ash313-5	t	**22,4**	163,6 (23,5)	49,3	190,8	272	298,1	23,1	22,8
(unsat)	n	723	10428 (723)	723	723	723	723	723	723
ash313-7	t	**980**	1157 (926)	1366	1192	1791	$> 1h$	1137	1217
(unsat)	n	28789	43233 (27956)	28709	28710	28712	-	28725	28724

work must be done is this and other classes of non-binary problems in order to achieve a better understanding of why these differentiations in performance exist.

5.5 A general summary of the results

In order to get a summarized view of the evaluated heuristics, we present two figures. In these figures we have computed averaged values for runtime and node visits for the three major conflict-driven variants (*dom/wdeg*, "alldell" and "fully assigned") and we have compared them graphically to the Impact heuristic (which has the best performance among the impact based heuristics).

Results are collected in Figure 1. The left plot in this figure corresponds to run times and the right plot to visited nodes. Each point in these plots, shows the runtime (or nodes visited) for one instance from all the presented benchmarks. The y-axis represent the solving time (or nodes visited) for the Impact heuristic and the x-axis the averaged values for the conflict-driven heuristics. Therefore, a point above line $y = x$ represents an instance which is solved faster (or with less node visits) using the conflict-driven heuristics.

Table 3: Cpu times (t), and nodes (n) from random problems. Best cpu time is in bold. The r prefix stand for rand.

Instance		$d/wdeg$	$d/wdeg$ $r.probe$	$d/wdeg$ RSC	$Impact$	$Node$ $Impact$	$Impact$ RSC	$alldel$ $by\ \#del$	$fully$ $assigned$
ehi-85-0	t	2,6	117,8 (0,18)	3,1	13,4	13,8	13,5	**0,18**	1,5
(unsat)	n	742	7951 (6)	67	7	7	7	6	180
ehi-85-2	t	**1.2**	118,3 (0,15)	2,6	13,1	13,6	13,7	2,4	1,3
(unsat)	n	258	7834 (7)	13	6	6	6	386	157
geo50-d4-	t	206	581 (535)	288	$> 1h$	$> 1h$	$> 1h$	254	**86,5**
75-2(sat)	n	29383	89437 (77187)	41201	-	-	-	38426	11119
frb30-15-1	t	20,2	49,3 (17,9)	19,4	774,3	775	257,7	**12,8**	50,6
(sat)	n	5655	16966 (5401)	5341	365152	320598	98629	3853	14481
frb30-15-2	t	77,4	89,8 (56,7)	**59,8**	144,2	699	166	106,3	116,4
(sat)	n	22630	29194 (17665)	17424	54326	251187	57379	32691	34305
40-8-753-	t	225	202 (167)	**82,7**	1553	2112	689	414	118
0,1 (sat)	n	52297	47655 (39839)	16344	447419	585708	189071	96018	26568
40-11-414-	t	**1198**	1246 (1220)	1213	$> 1h$	$> 1h$	$> 1h$	1207	1202
0,2 (unsat)	n	408589	420418 (410815)	410165	-	-	-	407006	402188
40-16-250-	t	3027	3177 (3141)	3050	$> 1h$	$> 1h$	$> 1h$	**2214**	3083
0,35 (unsat)	n	908917	923908 (911684)	910892	-	-	-	683167	916726
40-25-180-	t	3056	2777 (2711)	1900	$> 1h$	$> 1h$	$> 1h$	**1748**	3454
0,5 (unsat)	n	477096	488035 (470890)	342571	-	-	-	321811	610021

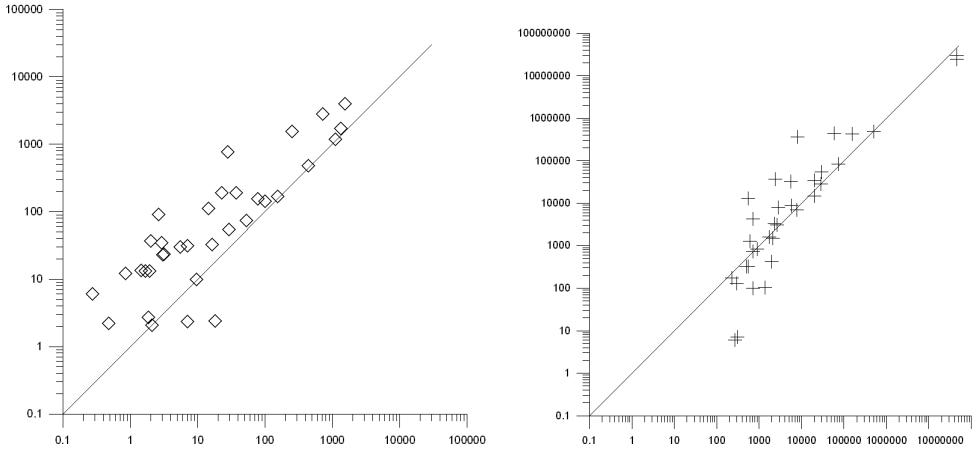

Figure 1: A summary view of run times (left figure) and nodes visited (right figure), for conflict-driven and impact heuristics

Table 4: Cpu times (t), and nodes (n) from boolean problems. Best cpu time is in bold.

Instance		$d/wdeg$	$d/wdeg$ $r.probe$	$d/wdeg$ RSC	$Impact$	$Node$ $Impact$	$Impact$ RSC	$alldel$ $by\ \#del$	$fully$ $assigned$
jnh01	t	10,5	111,8 (8)	14,8	**2,3**	13,8	13,8	4,3	6,4
(sat)	n	966	5466 (667)	563	100	100	100	491	709
jnh17	t	3,2	59,9 (0,7)	18,5	2	10,8	10,7	**1,5**	**1,5**
(sat)	n	477	4503 (107)	1233	132	131	131	216	204
jnh201	t	3,1	77,7 (2,2)	3,5	2,7	13,9	11,8	**1,2**	**1,2**
(sat)	n	336	5492 (203)	168	177	180	180	179	178
jnh301	t	38,4	113,5 (107,2)	38,9	**2,4**	6,7	5,9	7,5	8,3
(sat)	n	2703	5497 (203)	168	105	104	104	626	808
aim-50-1-	t	0,18	0,44 (0,12)	0,25	0,29	0,31	0,25	**0,09**	**0,09**
6-unsat-2	n	1766	6517 (1716)	1243	1760	255	255	794	549
aim-100-1-	t	0,4	1,3 (0,8)	1,6	6	2,7	4,9	**0,18**	0,23
6-unsat-1	n	3965	16067 (859)	8200	36586	1560	3392	1866	1431
aim-200-1-	t	0,8	2,3 (1,4)	2	2,2	2	1,4	**0,28**	0,29
6-sat-1	n	5099	16809 (7953)	1456	8074	216	216	1901	1540
aim-200-1-	t	2,1	3,7 (2,8)	9	12	5,4	9	**0,21**	0,23
6-unsat-1	n	14453	29670 (19565)	27654	32643	1966	3819	1388	1248
pret-60-	t	1517	1501,8 (1501,7)	$> 1h$	1736	$> 1h$	$> 1h$	**1197**	1325
25 (unsat)	n	48,1M	48,3M (48,3)	-	24,7M	-	-	45,2M	46,6M
dubois-20	t	1424	1498,3 (1498,2)	-	4013	$> 1h$	$> 1h$	**1209**	1981
(unsat)	n	48,4M	48,4M (48,4M)	-	29,6M	-	-	43M	47M

Both axes are logarithmic.

As we can clearly see from Figure 1 (left plot), conflict-driven heuristics are almost always faster. Concerning node visits, the right plot does not reflect an identical performance. Although it seems that in most cases conflict-driven heuristics are making a better exploration in the search tree, these exists a considerable set of instances where the Impact heuristic visit less nodes.

6 Conclusions

In this paper we experimentally evaluate the most recent and powerful variable ordering heuristics, and new variants of them, over a wide range of academic, random and real world problems. These heuristics can be divided in two main categories: heuristics that exploit information about failures gathered throughout search and recorded in the form of constraint weights and heuristics that measure the importance/impact of variable assignments for reducing the search space. Results demonstrate that, in general, heuristics based on failures have much better cpu performance. Although impact based heuristics are in general slow, there are some cases where they perform a smarter exploration of the search tree resulting in fewer node visits.

This paper is only a preliminary step towards evaluating and (crucially)

understanding the behavior of modern variable ordering heuristics. As results presented here demonstrated, both major strategies for variable selection, conflicts and impacts, have strengths and weaknesses. We believe that the development of hybrid techniques that exploit the strengths of both strategies should be a important topic for future research.

References

[1] T. Balafoutis and K. Stergiou. On conflict-driven variable ordering heuristucs. In *Proceedings of the ERCIM workshop - CSCLP*, 2008.

[2] C. Bessière, A. Chmeiss, and L. Sais. Neighborhood-based variable ordering heuristics for the contraint satisfaction problem. In *Proceedings of CP'01*, pages 61–75, 2001.

[3] C. Bessière and J.C. Régin. MAC and combined heuristics: two reasons to forsake FC (and CBJ?). In *In Proceedings of CP-1996*, pages 61–75, Cambridge MA, 1996.

[4] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *In Proceedings of 16th European Conference on Artificial Intelligence (ECAI '04)*, Valencia, Spain, 2004.

[5] D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22:251–256, 1979.

[6] H. Cambazard and N. Jussien. Identifying and Exploiting Problem Structures Using Explanation-based Constraint Programming. *Constraints*, 11:295–313, 2006.

[7] M. Correia and P. Barahona. On the integration of singleton consistency and look-ahead heuristics. In *Proceedings of the ERCIM workshop - CSCLP*, 2007.

[8] R. Dechter and I. Meiri. Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In *In Proceedings of IJCAI'89*, pages 271–277, 1989.

[9] E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.

[10] D. Frost and R. Dechter. Look-ahead value ordering for constraint satisfaction problems. In *In Proceedings of IJCAI'95*, pages 572–578, 1995.

[11] D. Grimes and R.J. Wallace. Sampling strategies and variable selection in weighted degree heuristics. In *In Proceedings of CP 2007*, pages 831–838, 2007.

[12] R.M. Haralick and Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–314, 1980.

[13] A. Mackworth. On reading sketch maps. In *In Proceedings of IJCAI'77*, pages 598–606, Cambridge MA, 1977.

[14] R. Mohr and T.C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[15] P. Prosser, K. Stergiou, and T. Walsh. Singleton consistencies. In *Proceedings of CP'00*, pages 353–368, 2000.

[16] P. Refalo. Impact-based search strategies for constraint programming. In *In Proceedings of CP 2004*, pages 556–571, 2004.

[17] D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *In Proceedings of CP '94*, pages 10–20, 1994.

[18] B.M. Smith. The brelaz heuristic and optimal static orderings. In *In Proceedings of CP'99*, pages 405–418, 1999.

[19] B.M. Smith and S.A. Grant. Trying harder to fail first. In *In Proceedings of ECAI'98*, pages 249–253, 1998.

[20] R. Zabih. Some applications of graph bandwith to constraint satisfaction problems. In *In Proceedings of AAAI'90*, pages 46–51, 1990.