Components of Information Flow Underlying Heuristic Decisions in Constraint Satisfaction Search

Richard J. Wallace

Insight Centre for Data Analytics, School of Computer Science and IT, University College Cork, Ireland

Abstract

This paper is a contribution to the analysis of variable ordering heuristics for constraint satisfaction problems. A model of information flow leading to heuristic decisions is presented that includes four distinguishable components: discriminability, feature and message value quality, and decision quality. Experimental results demonstrate that each of the quality components can have a definite effect on performance when other components are held constant. These results should improve the analysis of CSP heuristics going forward. They also bear on an earlier analysis of heuristic effects, or heuristic actions, since they clarify one of these actions that had not been adequately characterized prior to this work. As a result, we now have the makings of a complete framework for explaining how variable ordering heuristics actually work.

Keywords

constraint satisfaction problems, heuristics, search, information

1. Introduction

Although there is now a wide selection of variable ordering heuristics that can be used to solve constraint satisfaction problems (CSPs) more efficiently, and new heuristics continue to be devised [1, 2, 3], knowledge of how such heuristics work and why one is more effective than another is still rudimentary. (Note: in this paper, the terms “heuristic", “ordering heuristic", and “search heuristic" always refer to variable ordering heuristics.)

Recent analysis of the effects of preprocessing with local consistency algorithms on subsequent solution search revealed a new dimension of this problem, and in doing so yielded new insights into the heuristic side of CSP search [4]. This work showed that in addition to reducing the search space, the pattern of alterations (here, domain reductions) provides information to heuristics such as minimum domain size and minimum domain over degree regarding their selections. While this generally results in improved performance, it may also lead to marked decrements. In contrast, for search heuristics based on static properties such as static degree, performance is never degraded.

These results prompted further analysis of information flow in CSP algorithms. Although this factor has always been implicit in the use of dynamic ordering heuristics, until now it was never characterized explicitly. In this work the effect of information flow was demonstrated when other factors such as value selection, propagation effects, and the branching factor were controlled for. In addition, it was demonstrated that the information conveyed by domain reductions serves to signal where likely bottlenecks are located, although the signal is unfortunately ambiguous. (Note that this is different from the signal being contaminated by noise, since the signal itself is noise-free.)

Although this work identified information flow as a critical component of CSP search when guided by dynamic variable ordering heuristics, it did not present a full account of the factors that go into information flow. The present work addresses this deficiency. In this paper a full analysis of the components of information flow in this situation is presented. With this deeper understanding one can immediately see many issues that have not been addressed in previous work on heuristics, in particular,

Proceedings of the Workshops and Doctoral Consortium of the 42nd International Conference on Logic Programming (ICLP 2026), co-located with the Federated Logic Conference (FLoC 2026), July 18-19 and 24-25, 2026, Lisbon, Portugal.

wallacerichardj77@gmail.com (R. J. Wallace)

0000-0002-3537-7719 (R. J. Wallace)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC by 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-03

CEUR-WS.org/Vol-4311/RCRA_regular_03.pdf

papers where new variable ordering heuristics are presented. Some of these issues are discussed later in this paper.

In the experimental part of this work, two kinds of preprocessing are used. As in most of the earlier work, we will employ a series of local consistency algorithms based on *arc consistency*, namely arc consistency itself (AC), neighbourhood singleton arc consistency (NSAC), and singleton arc consistency (SAC). For our purposes, the important feature of this series is that successive levels of consistency remove increasingly more domain values on average, which generally improves the quality of information that is provided to the search procedure. (In fact, with respect to values deleted they form a total order, such that SAC $\geq$ NSAC $\geq$ AC.) In addition, a form of preprocessing known as random probing is used that allows us to compare the same type of information flow based on different kinds of information and with heuristics based on degree rather than domain size.

Together, these experimental strategies allow us to delineate the different components of information flow, thus providing a complete (albeit still qualitative) model of the process. Four different components can be identified, which will be named and defined later in the paper.

This work is a contribution to the experimental analysis of algorithms. In keeping with this, most of the evidence presented is experimental in nature. For the most part, this is a necessity because as yet we do not have an adequate handle on the phenomena such that meaningful formal analysis could be undertaken. In this connection, it may be noted that the importance of experimental analysis has been emphasized in earlier work in connection with search heuristics [5, 6].

The present paper is organized as follows. Section 2 provides some basic concepts and definitions in the field of constraint satisfaction search. Section 3 summarizes earlier work on CSP heuristics. Section 4 presents the basic analysis of information flow, including each of its components. Section 5 presents results of experiments on "information quality", showing that its various components can be empirically distinguished and that each can have a significant effect on search efficiency. Section 6 draws conclusions from this work and outlines a scheme for explaining how variable ordering heuristics work.

2. General Background: CSPs and Their Algorithms

Here we review some of the basic concepts in the field that form the background of the present work.

A constraint satisfaction problem (CSP) involves assigning values to a set of variables subject to restrictions on the way that values can go together. More formally, a CSP can be defined as a tuple, (X, D, C) where:

X is a set of *variables*, $X_1, \ldots, X_n$,

D is a set of *domains*, D_i, where each D_i is the set of *values* that may be assigned to variable X_i

C is a set of *constraints*. Each C_i belonging to C consists of a relation R_i and a particular subset of the variables in X, called the *scope* of the constraint. R_i is based on the Cartesian product of the values of the domains of the variables in the scope.

A *solution* to a CSP is an assignment or mapping from variables to values, $A = \{(X_1, a), (X_2, b), ..., (X_k, x)\}$, that includes all variables ($k = n$) and satisfies all the constraint relations in C.

In addition to their defining properties, classes CSPs are often characterized by certain properties, chief of which are number of variables, domain size(s), constraint graph density, or equivalently, average degree of a variable (i.e. the number of constraints it participates in), and the tightness of the constraints. The latter is defined as the proportion of tuples in the Cartesian product of the domains of variables in the scope of a constraints that *do not* satisfy that constraint.

CSPs have an important monotonicity property in that inconsistency with respect to even one constraint implies inconsistency with respect to the entire problem. This has given rise to methods for filtering out values that cannot participate in a solution, based on local inconsistencies, i.e. inconsistencies with respect to subsets of constraints. By doing this, these algorithms establish well-defined forms of local consistency in a problem, even before search is carried out.

The most widely used methods establish *arc consistency*. In problems with binary constraints, arc consistency (AC) refers to the property that for every value a in the domain of variable X_i and for every constraint C_{ij} involving X_i there is at least one value b in the domain of X_j such that (a,b) satisfies that constraint. For non-binary constraints, this requirement is extended to include all the other variables in the constraint, i.e. for every value a, there must be one tuple of values that together satisfy the constraint and include a as the assignment to X_i.

The present paper also makes use of certain forms of *singleton arc consistency*, or SAC. This is a form of AC in which the just-mentioned value a, for example, is considered the sole representative of the domain of X_i. If AC cannot be established under this condition, then there can be no solution with this value, so a can be discarded. If this condition can be established for all values in problem P, then the problem is singleton arc consistent. (Obviously, SAC implies AC, but not vice versa.)

A closely related form of consistency is called *neighbourhood singleton arc consistency*, or NSAC. NSAC algorithms establish SAC with respect to the neighbourhood of the variable whose domain is a singleton.

Definition 1. The *neighbourhood* of a variable X_i is the set $X_N \subseteq X$ of all variables in all constraints whose scope includes X_i, excluding X_i itself. Variables belonging to X_N are called the neighbours of X_i.

If for each value $a \in D_i$, where i is in $\{1 \dots n\}$, singleton arc consistency can be established in the subgraph based on that variable and its neighbours, then the problem is neighbourhood singleton arc consistent.

With regard to finding a full solution to a CSP, in the present paper we only consider backtracking methods. These methods build partial solutions that are consistent, extending them with further assignments until a full solution is obtained. If at some point in this process the partial solution cannot be extended, then search 'backs up' by retracting assignments and replacing them with new ones before attempting to extend the partial solution.

Backtrack search is typically enhanced by interleaving local consistency with search decisions, to form "hybrid search algorithms". Typically, a new bout of local consistency is done after assignment of a value to a variable in order to simplify the problem further before proceeding to the next decision.

The efficiency of backtrack search is further enhanced by using heuristics (rules of thumb based on maximizing or minimizing some problem feature) to choose the next variable or value to add to the partial solution. A well-known heuristic (mindom) chooses a variable with the smallest domain size; another (domain over forward degree or d/fd) extends this idea by choosing the variable for which domain size divided by the number of adjacent variables among the still uninstantiated set is minimized.

3. Related Work on Analysis of CSP Heuristics

Although there have been a number of papers that propose new heuristics for choosing variables to instantiate during search, there has been relatively little work on why they work and why one heuristic is superior to another. Perhaps the earliest contribution of this sort was the "Fail-First Principle" of Haralick and Elliott [7], loosely described as: select the variable that is most likely to lead to failure, where "failures" are dead-ends that one must backtrack from.

Later, Beck and co-workers proposed a more general "policy framework" in which search is partitioned into two types of state, called "promise" and "fail-first", depending on whether search is presently on a path that would lead to a full solution or is inside an insoluble subtree in the search graph. Each state has an associated policy. Given this framework, heuristic quality can be assessed by determining the degree to which a heuristic conforms to the appropriate policy given the present state of search. Within this framework, the Fail-First Principle can be viewed as an ideal meta-heuristic, which essentially states that one should always make heuristic choices as if the fail-first policy were in force. These authors also derived measures of promise and fail-firstness and applied them to a number of well-known ordering heuristics.

To my knowledge, the earliest work that specifically examines the *resulting effect* of variable selection with effective heuristics is a paper by Hooker and Vinay on SAT problems [6]. They identified a process that they called “simplification"; differences in simplification, they decided, could account for the differences in search effort that resulted.

A decade later, the present author employed factor analysis to assess the effects of different heuristics on search effort [8, 9]. This work suggested that there were only two fundamental effects of heuristic choice. One could be identified with the “simplification" factor of Hooker and Vinay. The other was called “contention". The former seemed to operate by reducing the number of interactions between different parts of the subgraph of unassigned variables, i.e. it was an implicit divide-and-conquer strategy. The latter seemed to operate by inducing rapid failure if search was inside an insoluble subtree, the upshot being that search withdrew from insoluble subtrees more rapidly.

More recently, it was proposed that all these results could be incorporated into the kind of framework used in decision theory, where three levels of analysis are distinguished: normative, prescriptive and descriptive [10]. Using these distinctions, the policy framework is a normative theory, while the analysis of heuristic effects is at the descriptive level. The prescriptive level consists of heuristics themselves. [1]

As described in the Introduction, a new aspect of heuristic analysis was revealed when the importance of information provided by preprocessing to the search process was demonstrated. An adumbration of this work appeared in the mid-1990s, when Sabin and Freuder observed that with forward checking, applying AC to a problem before search sometimes resulted in degraded performance [11]. However, those authors gave no explanation for why a result of this kind should occur.

One factor identified in recent work was termed “discriminability". This is simply the number of differences among variables with respect to the feature used by the heuristic [12]. Discriminability can be measured by means of the classic entropy formula (see below). In this work it was shown that for the preprocessing series described earlier, higher levels of consistency in addition to leading to greater problem reduction also reliably increased discriminability associated with features based on domain size.

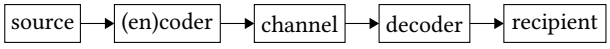
This work also showed that other factors are involved that remained to be properly elucidated. This is done in the present work. In the next section, we describe the basic framework. Then, in the following section, we examine each component in terms of its effects on search effort.

4. The Analysis of Information Flow in CSP Search

To explain the results discussed above, we make use of notions of information flow in the context of CSP search. In the next subsection we review the basic framework, as laid out in earlier work [4, 13, 12]. Then we present a model of the components of information flow. The last subsection considers the problem of information quality and how it can be measured.

4.1. Information channels and CSP heuristics

The reader is probably familiar with the standard view of information transmission [14], which is depicted in the following diagram. In this system, messages from a source are encoded and transmitted over a “channel". They are then decoded and passed to a recipient.



For present purposes, the same process can be viewed somewhat differently:



[1] In the original paper, a serious conceptual error was made in that the Fail-First Principle was located at the prescriptive level; instead, it belongs at the normative level as part of the policy framework.

In this case, the “channel" could be the storage of results from earlier processing and their subsequent retrieval. Information “extraction" can involve selection of certain portions of the stored results as well as any transformations that might be made. This step may also involve combining information from different sources. The “decision" usually involves comparisons of some sort and a selection among different actions on this basis.

Of course, all algorithms make use of information in carrying out their tasks, but since the study of algorithms takes place in a deterministic setting, where each usage is associated with a computation, this aspect is usually not emphasized as such. However, when key choices are not fully determined, as in the case with CSP heuristics, this aspect of the algorithmic procedure becomes an important matter for analysis, as earlier results show.

CSP heuristics operate by extracting information from a problem and using that information to guide decisions during search. As shown by many papers in the past, there are a multitude of problem features that can be used. In the present framework, each of these (and each feature combination) forms a distinct information channel with respect to the extraction process.

As noted in earlier work, we can distinguish two kinds of information channel:

- information is extracted from ‘nature’, i.e. from pre-existing features of the problem, such as the degree of a variable
- information is extracted from results produced by a preprocessing algorithm or propapagation earlier in search

In keeping with the notion of an information channel, we can characterize messages sent across this channel in terms of the feature that a heuristic relies on. Consider, for example, the case where we use the domain reduction obtained during preprocessing as a heuristic. (Note that for problems with constant domain size this is equivalent to mindom, but in other cases it is not.) In this case, domain size reduction can be viewed as a code whose letters are the possible reductions ranging from 0 to $d_{\max} - 1$ (or $d - 1$ for problems with constant domain size). The message sent from preprocessing to the search stage always contains n letters, one for each variable (fixed length code). (We ignore messages containing d as a value, since in these cases search would not be carried out.)

4.2. Components of information flow between preprocessing and search

As demonstrated in the experiments to follow, in the information flow between preprocessing and search, several different aspects or elements can be distinguished. These can be divided into three categories:

- discriminability
- code quality
- decision quality

The first category, discriminability, refers to the number of differences among the letters in a given message. Since this has been described and analyzed in earlier work [13, 12], it is discussed only briefly. As noted earlier, discriminability can be measured using the well-known entropy formula,

$$H(S) \;=\; \sum_{k=1}^{m} p(k) log(1/p(k))$$

but with a slightly different interpretation of its terms. Here, the numbers 1 to m refer to the m different code values that appear in a given message, such as the different amounts of domain reduction. Instead of referring to the probability that value k will occur in any given message, here $p(k)$ is simply the relative frequency of this value in the message at hand. Since the number of differences within a message with respect to a given feature is associated with the amount of information conveyed, the entropy expression gives an appropriate measure of the former value. Note that in contrast to the other

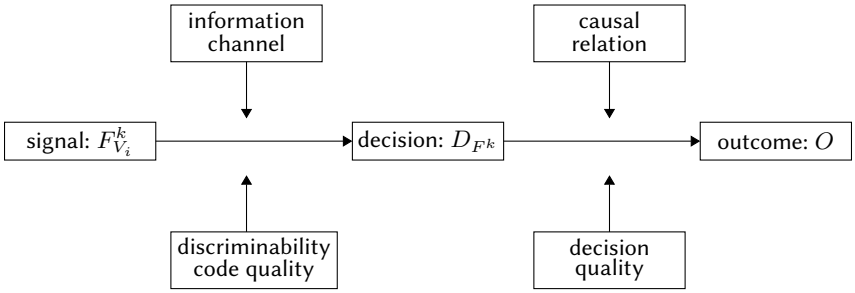


Figure 1: Workflow model of CSP search showing components of information flow underlying differences in search effort.

categories, this is a kind of information theoretic, or syntactical, quantity, in that it refers to a property of the message itself rather than anything that might be inferred from it.

The next category pertains to the quality of the information in a given message In this context, the term “quality" always refers to search effort, which is, of course, our ultimate concern. Hence, in this paper “quality" can always be taken as referring to differences in search performance. Naturally, our main interest is in improvements in quality, but for the analysis to be complete, we must consider reductions in quality as well.

Code quality refers to the general factor of quality of the feature(s) used by a heuristic. This category can, in turn, be subdivided into two separate elements: (1) the quality of the feature used by a heuristic as opposed to some other feature, (2) the quality of a given message as opposed to messages containing different letters (values) associated with the same feature. Of course, differences in message quality can be due to differences in discriminability as well, but they may also be due to the particular letters in the message (e.g. domain reductions of 0 or 1 versus reductions of 0 or 10) as well as the partition of the variables with respect to the associated letters. (Whether the latter actually occurs when discriminability is held constant is an empirical question that is addressed later in this paper.)

The third category pertains to the decision itself. This needs to be considered in addition to the quality of the information given because it is possible to use this information in more than one way, and these differences in the form of different decisions result in differences in search effort. An obvious example is the difference in quality between a heuristic and its anti-heuristic, e.g. choosing the variable from a set based on minimum domain size versus choosing one that has the maximum domain size.

The different elements just described are depicted together in Figure 1, which shows how they fit into the overall sequence of information extraction and decision making that results in a given search outcome. In the figure, $F^k_{V_i}$ refers to the value of feature F^k associated with variable V_i. D_{F^k} denotes the decision based on feature F^k. O is the outcome resulting from this decision, culminating in some degree of search effort.

As the figure shows, there are two basic mappings or relations. The first relation, between signal and decision, is always a mapping, while the second, between decision and outcome is either a mapping or relation depending on whether search depends on static or dynamic heuristics. This figure also indicates the nature of the two relationships (text above the process diagram) as well as the association of properties with these relations (text below).

4.3. Assessing information quality

As noted earlier, code quality is fundamentally different from a property like discriminability because it refers to something that is referred to by a message, or can be inferred from it, rather than the message itself. Because of this, it can be considered a kind of semantic property associated with the signal values. This means that we have to consider what it is that is being referred to, and develop means of assessing it.

As already noted, in the present situation quality refers to search effort, with lower effort representing

higher quality. In this context, the focus is on differences in search effort that depend on different heuristic decisions that can be made at any given stage of search.

Although there may be arguments for assessing search effort using runtime even in this context, in this work search effort measured by number of nodes in the search tree generated in finding a solution. In addition to being arguably a more fundamental measure of search difficulty, the number of search nodes is not affected by the various procedures that were used to ensure proper experimental control (see below). In such cases, differences in runtime due to such factors are irrelevant to the goals of the experiment.

There are numerous issues that might be raised regarding code quality. But here we will simply use search nodes as our basis for comparison. For more general discussions of these issues, see [13, 12].

5. Experimental Analysis of Information Quality

This section presents results showing that each of the different components of information flow that have been identified can be associated with differences in search effort.

5.1. Methods

Here, general methods are described; specific details are given in connection with particular experiments.

The key feature of the present experimental program was that it focuses on information flow between preprocessing and search. Since in this case information flow is between two distinct stages of processing (as opposed to information flows during the course of search), this makes it easier to distinguish generation and reception of information in order to study the properties of the channel itself.

Another important feature of this experimental program was the use of preprocessing algorithms of increasing degrees of effectiveness, i.e. that deleted progressively more domain values. The three arc consistency algorithms described earlier in Section 2 were employed for this purpose. In all cases the subsequent search algorithm was maintained arc consistency (MAC) [11]. Algorithms were implemented in Common Lisp, specifically Macintosh Common Lisp (MCL) version 5.1 running on an iMAC (MAC OS X version 10.2.8) with a Power PC 800MHz CPU.

Problem sets were chosen that met certain experimental requirements, the chief one being that more restrictive forms of preprocessing remove progressively more values that in turn lead to greater reductions in search effort. In addition, care was taken to ensure that more restrictive forms of preprocessing did not make the problems so easy that it became trivial to solve them; in that case, positive results would also be trivial. In addition, all problems had solutions. This was done to avoid marked differences in number of values deleted and cases of zero search nodes, which would render the present analysis meaningless or at least problematic.

Earlier work was done with a variety of problem types and demonstrated the generality of the basic relations between preprocessing and search with dynamic domain heuristics. Problem classes included geometric problems, random relop problems, radio link frequency problems, and driverlog problems (see [4] for further details). Because of time and space limitations (experiments of this type are quite time consuming), results in this paper are for the most part limited to geometric problems, although similar results have been found for other problem classes in some cases. [2]

In this connection, it should be emphasized that the purpose of these experiments is to demonstrate that each kind of quality defined in the previous section can have an impact on search performance; hence, a complete analysis of the effects of a heuristic on performance may have to take into account any or all of these features. For this kind of demonstration, a single class of problems is sufficient.

Geometric problems are generated by selecting points at random within the unit square to represent variables, and adding constraints between those whose Euclidean distance is less than some criterion, called the 'distance' [15]. In the present work, in addition, if there was more than one connected component, separate components were connected via pairs of variables (one in each component) having

[2]Eventually, corresponding results for other problem classes will be made available as supplemental material.

the smallest Euclidean distance between their points in the unit square, in order to make a graph with a single connected component.

For these problems, in addition, degree of support for each value across a given constraint was varied: with an expected value of 0.8, the tightness was 0.3, and with an expected value of 0.2 it was 0.75. Such problems are called “geovarsat problems" to distinguish them from ordinary geometric problems that have uniform constraint tightness.

Three sets of geovarsat problems were tested having either 40, 60 or 80 variables with constant domain sizes of 12, 14, and 15, respectively. To limit variation in graph density, only problems within a small range around a “target" were accepted. For 40, 60 and 80 variable problems, this target was 160 ($\pm$ 3), 280 ($\pm$ 3), and 380 ($\pm$ 5) constraints, respectively. Problems with these specifications were obtained by filtering during problem generation.

(Further details on general methods can be found in [4, 13, 12].)

5.2. Discriminability and code quality

Simple arguments suffice to show that greater discriminability in and of itself does not guarantee less search effort. Consider first, a given problem whose variables are partitioned under a given heuristic, thereby giving a particular value for discriminability. Then the same value for discriminability will hold if the corresponding anti-heuristic is used. Yet in one case search may be improved in comparison to a random ordering while in the other it is likely to be impaired. (See [16] and below for examples.)

Secondly, consider a code based on the usual numerical variable labels. This gives maximum discriminability, although for many problem types, the resulting lexical ordering is essentially a random ordering that gives poor results in comparison to heuristics appropriate for that type of problem.

These cases suggest that greater discriminability will enhance search performance only in connection with enhancement of code and decision quality.

5.3. Analyzing feature and message quality - method 1

In the following experiments, random probing was used to produce a code where each variable has an associated value for the sum of the weights of its adjacent constraints. This was compared with a code based on static degree.

Random probing [17] is a variation of the weighted degree procedure [18]. In the latter procedure, each constraint has an associated “weight", which is the number of times that propagation via that constraint led to a loss of all values in some variable domain (“domain wipeout"). Sums of these weights are then used to give a weighted degree of each variable in order to enhance the maxdeg or d/fd heuristic. Random probing uses this idea in a preprocessing stage, in which a truncated search procedure is carried out by choosing variables and values at random up to a limited cutoff in terms of either nodes or failures (i.e. domain wipeouts). This is done repeatedly prior to full search. During these runs, weights (i.e. failures) are tallied just as in the original weighted degree procedure. This technique allows full search to begin with a set of constraint weights rather than building these from scratch as search progresses.

Using this technique, the quality of messages based on a constraint weight code could be varied by varying the amount of probing done prior to search. During search, constraint weights were no longer updated; this ensured that the information gained by preprocessing was the factor affecting search efficiency. The weighted *static* degree heuristic was used instead of the more commonly used domain over weighted forward degree heuristic to avoid extraneous effects related to changes in current domain size.

Five probing conditions were used in the experiment together with ordinary static degree. The conditions were: 5 probes, each ending when 5 failures had accumulated; 10 probes, each ending with 10 failures; 20 probes, each with 20 failures; 40 probes, each with 40 failures; 80 probes, each with 80 failures. In addition to search efficiency, measured as number of search nodes, discriminability values were obtained in all cases.

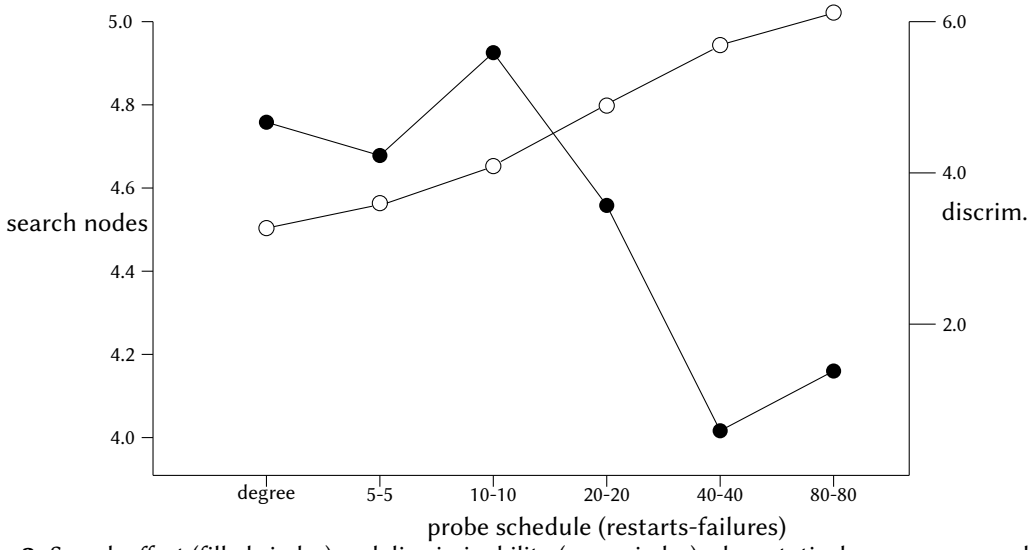


Figure 2: Search effort (filled circles) and discriminability (open circles) when static degree or summed constraint weights after varying amounts of probing are used as variable ordering heuristics. Search effort is graphed logarithmically. (Note. For these problems, the maximum value for discriminability is 6.322.)

Due to the variability inherent in weights due to random probing, each condition was run three times following different numbers of calls to the random generator (0, 1000, and 2000). For each run, each problem was solved 50 times with random value selection and random selection among the best variables according to the equivalence classes produced by the given heuristic at each stage of search.

Results are given in Figure 2. These are the grand means based on the three group means, which were in turn based on the means for each problem of 50 experimental runs with random selections as described above.

Note first, that even after the most limited amount of probing, average discriminability values are higher than they are for static degree. Despite this, limited probing often leads to much worse performance, presumably because of inadequate sampling. However, with greater amounts of probing (commensurate with schedules typically used in the literature), search performance is better than with static degree.

More significant for present purposes is the demonstration that differences in discriminability are not sufficient to account for differences in search performance that are obtained under these conditions. Clearly, code quality must also be considered. It is also clear that in addition to the different features (i.e. degree versus constraint weights), in this case the particular set of code values in a message, i.e. message quality, is also a significant factor affecting search performance. Thus, in the present experiment, message values in a code that was potentially superior to static degree could lead to search performance that was appreciably worse depending on the constraint weights obtained with different amounts of probing.

5.4. Analyzing feature and message quality - method 2

A different approach to demonstrating differences in feature quality was also used that is in some respects more satisfactory. In these experiments, discriminability was kept constant by reducing the number of differences for a heuristic that yielded more discriminations than the heuristic it was being compared with. With this method, it was possible to compare various features; here, we will consider domain reduction, degree, and degree weighted by constraint failures. (Although the latter is normally referred to as weighted degree, in order to keep the present terminology consistent we will continue to refer to it as summed constraint weights or simply constraint weights.)

The present method is based on the following argument. Each heuristic partitions the set of n variables into k equivalence classes, where $k \leq n$. Discriminability, of course, varies with the size of k. Moreover, these classes form a total order based on the feature used by a given heuristic. Therefore,

if we merge classes that are adjacent in this ordering we can reduce discriminability to match that produced by a different heuristic without contraverting the original ordering such that if variable X_i was preferred to variable X_j in the original ordering, X_j will not be preferred to X_i in the new ordering. In this way, if two heuristics h_1 and h_2 are associated with discriminabilities k_1 and k_2, where $k_1 > k_2$, we merge some of the equivalence classes given by h_1 to give a discriminability equal to k_2.

The procedure for equalizing discriminability was as follows. For both heuristics, variables were ordered from highest to lowest preference. For example, for static degree and constraint weight, variables would be ordered from largest value to smallest. Then the pattern of equivalence classes formed by the heuristic with lower discriminability was recorded and this was used to determine the equivalence classes for the heuristic with higher discriminability.

One problem with this technique involves ties between variables under the heuristic with greater discriminability. In such cases, it is possible that, with the equivalence classes imposed by the pattern, tied values will fall into different equivalence classes, thus producing distinctions among certain variables that were indistinguishable under the basic heuristic.

For comparisons with constraint weights this was not a serious problem, since for the probing schedule used there were very few ties and almost none among the top ranked 10-15 variables; this meant that in almost all cases, if this condition occurred at all, it happened after earlier instantiations and constraint propagations had already made the problem tractable or almost so. For the domain reduction-degree comparison, ties occurred more often. Since it was assumed that it was a matter of chance whether this factor would enhance or impair search effort, and it was more likely to be an impairment since for the variables in question one is deprecating the value for degree, no attempt was made to prevent it. In this case, where the degree heuristic would possibly operate with a mild handicap, a clear demonstration of its superiority to domain reduction would still show that feature quality differed for the two heuristics.

For the constraint weight versus degree comparison, the geovarsat80 problem set was used, with a probing schedule of 80 restarts and 80 failures. For each problem and for each set of constraint weights, search was carried out 50 times with random value selection and random selection among the best remaining (uninstantiated) variables at each level of search. Other procedures were the same as in earlier experiments with constraint weights, i.e. static degree was used and weights were frozen after preprocessing. Somewhat surprisingly, under the conditions of the experiment, the constraint weight heuristic showed considerable variability in contrast to earlier results with this probing schedule. Consequently, ten full runs were carried out, i.e. for each problem ten sets of constraint weights were generated and each was tested with 50 randomized runs. In addition, ten runs were carried out with static degree. In both cases, prior to each run, the random generator was called n times, where n was 0, 1000, 2000, ..., or 9000 depending on the run.

The results of this experiment are shown in Figure 3. In the figure the mean across the 25 problems is shown for each experimental run. A single run is represented by a vertical line whose location on the x-axis indicates the mean search nodes.

It can be seen that despite the considerable variation in mean scores for the constraint weight heuristic, there is no overlap with the results for static degree; moreover, in many cases the difference is considerable. We conclude that there is a genuine difference in feature quality between heuristics in favor of constraint weights under this probing schedule.

The next set of experiments compared degree and (maximum) domain reduction (equivalent to minimum domain size at the start of search for these problems). Here, a pattern based on domain reduction was used to create a variable selection strategy for degree with the same discriminability. As already noted, ties across the resulting equivalence classes were allowed, since they would tend to degrade rather than enhance search performance. Note that these are fixed order heuristics since the heuristics are based on the original domain reductions produced by preprocessing.

Again, search was carried out on each problem 50 times with random value selection and random selection among the current best variables. In addition, the entire set of problems was tested in this way up to ten times with the different initial settings of the random number generator (i.e. after 0, 1000, 2000, ... calls to the random generator).

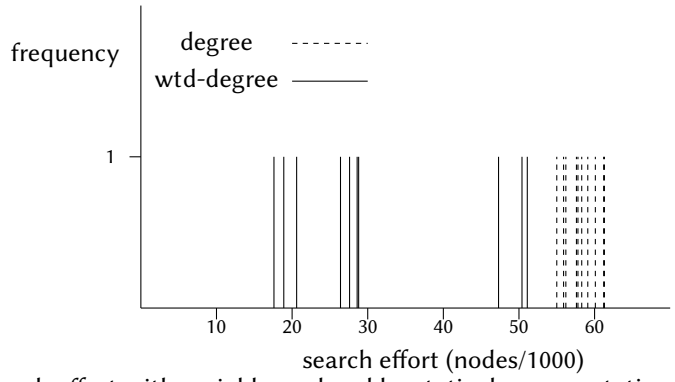


Figure 3: Mean search effort with variables ordered by static degree or static weighted degree with discriminability equalized. Each vertical line represents a group mean using a different randomized selection of values and equivalent variables. Further details in text.

Table 1
Domain Reduction Versus Static Degree After AC and NSAC with Discriminability Controlled

	AC pre		NSAC pre	
probs	degpatt	domreduct	degpatt	domreduct
geo40	14,017	32,020	299	326
geo60	>522,963[1]	>1,386,914[2]	689	7610
geo80	—	—	308,327[1]	>1,684,104[2]

Notes. Grand means for search nodes across repeated experiments, problem sets and repeated tests of individual problems (see text for details). [1] based on 3 repetitions. [2] based on 1 full experiment. > indicates that 10^7 node limit was reached on some runs.

In this experiment, 40-, 60- and 80-variable geovarsat problems were used, with either AC or NSAC as the preprocessing algorithm. Because of the difficulty in obtaining data under some conditions, not all combinations of problem X preprocessing algorithm could be tested. In addition, in some cases a cap of 10^7 nodes was used to limit search.

Results are shown in Table 1. Unless otherwise noted, means are for ten repetitions of the full experiment, each beginning with a different setting of the random number generator. (In cases where fewer repetitions were carried out, this was because the time required to solve many of the problems was too great for more repetitions to be completed; separate testing showed that under some conditions more than 10^8 search nodes were required to complete an individual run.)

Once again, we conclude that there is a real difference in code quality between degree and domain reduction. In addition, the magnitude of the difference can be altered depending on the degree of domain reduction resulting from preprocessing.

5.5. Assessing decision quality

For completeness, we include a simple experiment comparing a heuristic and its anti-heuristic on the 40-variable geovarsat problem sets. For this purpose the static degree and the maximum domain reduction heuristics were used; similar results have been found when other heuristics were tested [16]. In keeping with the restriction of information flows to those between preprocessing and search, the domain reduction heuristic is based on reductions that occurred during preprocessing, making it effectively a fixed ordering.

As with other experiments, each problem was tested 50 times with random selection of the next value to assign and the next variable from the current best set under the ordering. For these tests a search limit of 10^6 nodes was used.

As found in other work (cf. [16]), the difference between heuristics and anti-heuristics was quite pronounced. For the degree heuristic, these problems were usually solved after a few hundred nodes had been opened, while with the corresponding anti-heuristic, an average of 33.4 individual runs per problem (out of 50) exceeded the million-node limit. Similarly, for the domain reduction heuristic an average of 29,663 nodes were required to solve the problem, while for the corresponding anti-heuristic the average was >938,229 search nodes, and the million node limit was exceeded on an average of 45.6 runs.

In these tests discriminability was identical for each comparison, as was the feature used by each of the heuristics compared. In addition, the message values for each feature were also the same. It is clear, then, that decision quality is a factor that can affect heuristic performance, so it must be considered along with the other factors in any complete analysis.

5.6. Comparing different preprocessing methods with respect to quality

The foregoing analysis can be extended to the effects of varying levels of preprocessing on subsequent search. Specifically, we can examine the basis for improvements in search when more stringent levels of local consistency are established. Again, we will use the AC-NSAC-SAC series for this purpose.

Previous work has established that for many problem classes discriminability is increased when NSAC is used for preprocessing as opposed to AC, and for most classes tested SAC results in further increases [12]. In this case the feature coded (domain reduction) is the same for all forms of preprocessing; however, it is an open question whether there are differences in the quality of message values when more stringent forms of local consistency are established. Hence, the following experiment was done to answer this question.

For this purpose, the pattern technique described in previous sections was employed. With this technique, comparisons were made between AC and NSAC, AC and SAC, and NSAC and SAC. In each case, a partition was determined for the weaker form of local consistency, and this partition was the basis for the pattern used for the stronger form. The search itself was preceeded by the weaker form of consistency, so for example, for the AC-NSAC comparison, this meant that AC reductions preceded search in both cases along with identical partitionings. A similar procedure was followed for the NSAC-SAC comparison, i.e. in both conditions being compared, search was preceded by NSAC preprocessing of the problem and an NSAC-based partition was used during search.

In addition, further domain reductions during search were not considered by the heuristic, which based all its decisions on the domain reductions obtained after preprocessing, making this a fixed order heuristic. Thus, the only difference between the AC and NSAC conditions was the set of message-values considered by the heuristic at each stage of search. As in earlier experiments, under each condition, a problem was tested 50 times with random selection of values and best variables in the remaining set according to the partition. In addition, each full experiment was run five times following five different calls to the random number generator.

Note that the conditions of this experiment require that many decisions will be made by the heuristic that it would ordinarily avoid due to enhanced discriminability, with a consequent fall-off in search efficiency. For this reason, the median is likely to be more informative than the mean, which is why this statistic was used in this analysis.

The results of this experiment are summarized in Table 2, which shows grand means for median and the proportion of cases favoring the more stringent form of preprocessing, averaged over the five experimental runs. For these problems, using NSAC or SAC values rather than AC values gave a clear improvement. However, there was no evidence of improvement when SAC values were used rather than NSAC.

From this and experiments with other problem classes carried out in previous work, we conclude that in addition to greater problem reduction, more stringent forms of local consistency can yield greater discriminability as well as superior message values, i.e. values for the feature used by a dynamic heuristic that on average result in more efficient search.

Table 2
Effects of message values produced by different forms of preprocessing on subsequent search

AC-NSAC			AC-SAC			NSAC-SAC		
AC mdn	NSAC mdn	prop	AC mdn	SAC mdn	prop	NSAC mdn	SAC mdn	prop
33,076	19,369	0.83	33,076	5,502	0.77	347	389	0.5

Notes. Grand means for median search nodes and proportion of problems improved. Geovarsat40 problem set (25 problems). Further discussion in text.

6. Prospects for a Complete Framework for Understanding Heuristic Action

This work begins to address an important lacuna in our understanding of the operation of CSP heuristics. As far as the author is aware, this aspect of the problem has not been analyzed before, in part because the role of information flows during search was not properly recognized.

The present work also clarifies the effects of preprocessing algorithms on CSP search. In particular, it is now clear why there are vagaries in search effort after problem reduction through removal of locally inconsistent values, as shown many years ago by Sabin and Freuder [11]. As shown in earlier papers in this series, this is due to deficiencies in the information used by dynamic variable ordering heuristics, which serve to compromise search effort.

Within this information-based framework, one can demonstrate that there are distinct components of information transmission between preprocessing and search, which were termed discriminability (among the letters of a message), code-feature and message-value quality, and decision quality. The present experiments also show that variations in preprocessing, either through stronger forms of local consistency or by more extensive collection of failures for constraint weights, can affect both discriminability and code quality.

This work indicates that we need to revise the way we compare ordering heuristics for CSP search. Up until this time, this has been done by straightforward implementation and testing. However, in determining why one heuristic is better than another one should specify which component of the information flow is being improved. In particular, differences in discriminability should be distinguished from differences in code quality. Such analysis is currently being carried out on several variable ordering heuristics that have been proposed during the last two decades.

However, the significance of this work extends beyond this. As noted earlier, some years ago it was shown using factor analysis that differences in performance due to variable ordering heuristics seem to be due to two independent factors. One was identified with the simplification hypothesis of Hooker and Vinay [6] and seems to involve separating the variables yet to be instantiated into roughly independent parts. The other factor was tentatively called a “contention" factor, although the hypothesis that contention built up more rapidly with heuristics that loaded heavily on this factor was not supported, since at a given level of search the insoluble subtrees were not significantly different in size when compared with simplification heuristics.

With the present work, it becomes clear that contention heuristics (those that load more heavily on the "contention factor") are heuristics that are guided by information concerning contentious, or bottleneck, variables so that these are chosen over others. This information flow occurs within search as well as after preprocessing, since in both cases the same local consistency strategies are used.

In the earlier work the conjecture was made that these two factors hold across all problem types. However, while the properties of homogeneous random problems allow them to be demonstrated clearly in a factor analysis, in other cases interactions between problem features and the magnitude of the effect of the heuristic actions in different parts of a problem can obscure their workings, so that the factor model will reflect other sources of variation which do not reflect fundamental processes.

In the earlier work evidence was given to support this conjecture, although much more needs to be done. Now that we have a much clearer idea of what each factor is this should facilitate such efforts. At any rate, a framework is now in place that can potentially lead to a complete account of variable ordering heuristics for CSPs.

7. Acknowledgements

This publication has emanated from research supported in part by a research grant from Science Foundation Ireland under Grant No 12/RC/2289_P2

8. Declaration on Generative AI

No generative AI tools were used in preparing this paper.

References

[1] D. Habet, C. Terrioux, Conflict history based search for constraint satisfaction problem, in: Thirty-fourth ACM/SIGAPP Symposium on Applied Computing, ACM, 2019, pp. 1117–1122.

[2] H. Li, M. Yin, Z. Li, Failure based variable ordering heuristics for solving CSPs, in: L. D. Michel (Ed.), Principles and Practice of Constraint Programming-CP 2021., Leibnitz International, 2021, pp. 9:1–9.

[3] R. Wang, W. Xia, R. H. C. Yap, Correlation heuristics for constraint programming, in: Twenty-Ninth International Conference on Tools with Artificial Intelligence - ICTAI'17, IEEE Press, 2017, pp. 1037–1041.

[4] R. J. Wallace, Explaining the effects of preprocessing on constraint satisfaction search, in: L. Longo, R. OReilly (Eds.), Thirtieth Irish Conference on Artificial Intelligence and Cognitive Science AICS 2022. Communications in Computer and Information Science No. 1662, Springer, 2023.

[5] J. N. Hooker, Needed: An empirical science of algorithms, Operations Research 42 (1994) 201–212.

[6] J. N. Hooker, V. Vinay, Branching rules for satisfiability, Journal of Automated Reasoning 15 (1995) 359–383.

[7] R. M. Haralick, G. L. Elliott, Increasing tree search efficiency for constraint satisfaction problems, Artificial Intelligence 14 (1980) 263–314.

[8] R. J. Wallace, Factor analytic studies of CSP heuristics, in: Principles and Practice of Constraint Programming-CP'05. LNCS No. 3709, Springer, 2005, pp. 712–726.

[9] R. J. Wallace, Determining the principles underlying performance variation in CSP heuristics, International Journal on Artificial Intelligence Tools 17 (2008) 857–880.

[10] R. J. Wallace, A normative-prescriptive-descriptive approach to analyzing CSP heuristics, in: Thirtieth International FLAIRS Conference, AAAI Press, 2017, pp. 158–163.

[11] D. Sabin, E. C. Freuder, Contradicting conventional wisdom in constraint satisfaction, in: A. Borning (Ed.), Principles and Practice of Constraint Programming - PPCP'94. LNCS No. 874, Springer, 1994, pp. 10–20.

[12] R. J. Wallace, Information flows during constraint satisfaction search with dynamic heuristics: information from preprocessing, in: M. Maratea, L. Serafini, M. Vallati (Eds.), 31st RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion, AI4-IPS-RCRA-SPIRIT 2024, CEUR Workshop Proceedings, Vol-3883, 2024.

[13] R. J. Wallace, Domain reductions after preprocessing: Effects on dynamic variable ordering heuristics in constraint satisfaction search, in: M. F. Santos, J. Machado, P. Novais, P. Cortez, P. M. Moreira (Eds.), Progress in Artificial Intelligence. 23rd EPIA Conference on Artifical Intelligence, EPIA 2024, Part II. LNAI No. 14968, 2024.

[14] D. S. Jones, Elementary Information Theory, Oxford University Press, Oxford, 1979.

[15] D. S. Johnson, C. R. Aragon, L. A. McGeoch, C. Shevron, Optimization by simulated annealing: An experimental evaluation. part II. graph coloring and number partitioning, Operations Research 39 (1991) 378–406.
[16] J. C. Beck, P. Prosser, R. J. Wallace, Toward understanding variable ordering heuristics for constraint satisfaction problems, in: Fourteenth Irish Conference on AI & Cognitive Science-AICS'03, 2003, pp. 11–16.
[17] D. Grimes, Identifying Sources of Global Contention in Constraint Satisfaction Search, Ph.D. thesis, National University of Ireland, Cork, 2012.
[18] F. Boussemart, F. Hemery, C. Lecoutre, L. Sais, Boosting systematic search by weighting constraints, in: Sixteenth European Conference on Artificial Intelligence-ECAI'04, IOS, 2004, pp. 146–150.