Information Flows during Constraint Satisfaction Search with Dynamic Heuristics: Information from Preprocessing

Richard J. Wallace

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

Abstract

We still do not have an adequate understanding of heuristic methods used for solving constraint satisfaction problems (CSPs). An example of this involves the effects of preprocessing, an essential means of improving CSP search. The traditional explanation for its beneficial effect is that the resulting "problem reduction" leaves fewer possibilities to explore. Recently, however, it was shown that when dynamic variable ordering heuristics are used, other factors related to domain size reduction are much more important. This paper extends this earlier work and clarifies what is involved. The key idea is that the pattern of domain reductions produced by preprocessing constitutes information that by itself enables heuristics to make better decisions. Treating domain reduction as a code and the set of reductions as a message of length n transmitted to the search algorithm, we can distinguish two factors, message discriminability and code quality. We provide evidence that greater problem reduction enhances both.

Keywords

constraint satisfaction problems, heuristics, preprocessing, search, information theory

1. Introduction

A fundamental strategy for solving CSPs is to subject them to some form of *preprocessing* before searching for a solution. For this purpose, one uses what are called local consistency techniques. These are polynomial-time algorithms that test for limited forms of consistency within a problem [2].

The best-known examples are based on "arc consistency" (AC) and "path consistency". Arc consistency algorithms test whether a given value or label for a given variable in the problem is supported by each and every constraint that the variable is directly affected by. Path consistency algorithms test that all paths between two variables with consistent values contain values that support both values. In both cases, the thinking is that if a value is not locally consistent, then it cannot be part of any complete set of labelings that satisfy all the constraints in the problem, aka a solution. Therefore, if we are searching for a solution, we don't have to consider such values at all.

In most accounts of constraint solving, it is assumed that this is the (sole) reason that preprocessing is useful. That is, it reduces the number of alternatives that one must examine during search. In other words, preprocessing reduces the search space.

The benefits of such "problem reduction" are summarized by Edward Tsang in an eponymous chapter of his textbook ([3], pp. 79-80):

> The following are possible gains from problem reduction when combined with searching: (1) Reducing the search space. Since the size of the search space is measured by the grand product of all the domain sizes in the problem, problem reduction can help to reduce the search space by reducing the domain sizes. (2) Avoiding repeatedly searching futile subtrees. ... If redundant values and redundant compound labels can be removed through problem reduction, then one can avoid repeatedly searching those futile subtrees. (3) Detecting insoluble problems. (etc.)

AI4CC-IPS-RCRA-SPIRIT 2024: International Workshop on Artificial Intelligence for Climate Change, Italian Workshop on Planning and Scheduling, RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion, and SPIRIT Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy November 25-28, 2024, Bolzano, Italy [1]

richard.wallace@insight-centre.org (R. J. Wallace)

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

Because they do not bear on the issues dealt with in this paper, we will disregard points (2) and (3). This general view is propounded in later texts [4] and other references [5].

However, recent work has shown that for many problems other factors related to preprocessing and domain reduction are sometimes more significant for search efficiency than reduction of the search space [6, 7]. This is because domain reduction as such can guide *dynamic* variable ordering heuristics in their decision making. This implicit guidance occurs because these heuristics take current domain size into account, and this can have an enormous effect on search efficiency. In contrast, while search space reduction is ubiquitous and important, often it gives only modest degrees of improvement.

In the initial work [6], it was suggested that some form of information theoretic analysis might be useful. This idea was elaborated in later work [7], where a simple framework was proposed in which the set of domain reductions is treated as a code of length n, where n is the number of variables in the problem. In addition, that work suggested two ways in which preprocessing can improve search: by increasing the number of differences among variables with respect to the heuristic property, which was called "discriminability", and by enhancing the quality of information conveyed to the dynamic search heuristic.

The present paper extends this work in several respects. First, it presents a more systematic account of both discriminability and code quality. Secondly, it demonstrates that stronger forms of consistency preprocessing do enhance the discriminability of variables. It also shows that some older findings on anomalous effects of preprocessing fall into the present framework.

Considered more generally, this work bears on the question of why ordering heuristics in constraint satisfaction search are effective. Unfortunately, despite the essential role of ordering heuristics in constraint satisfaction search, there is still no systematic analysis that explains how they work and why one heuristic performs better than another on a given problem. So when a new heuristic is proposed (e.g. [8] [9] [10]) it is often difficult to determine the basis for its superiority or to predict the quality of its performance in different contexts.

In recent years, there has been some progress in developing a comprehensive framework based on decision analysis [11]. However, as the present work shows, there are still important gaps in our knowledge. More specifically, this work shows that a decision-theoretic approach must be complemented by a deeper look at the information that is utilized in decision making in this context.

The present paper is organized as follows. Section 2 provides basic concepts and definitions. Section 3 describes the experimental environment used in this work, including algorithms, problems, and implementation. Section 4 describes the empirical findings that gave rise to the present thesis. Section 5 presents a resolution of a well-known 'anomalous' result. Section 6 clarifies the nature of the information transmitted. Section 7 lays out a framework for characterizing information transmission in this context and the ways that it affects search effort. Section 8 gives conclusions.

2. Background Concepts

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 the present paper we only consider backtracking methods for finding solutions to CSPs. 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, local consistency is done after each new value assignment 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. Algorithms and Testing

3.1. Search algorithms

The search algorithm used in this paper is called maintained arc consistency (MAC) [12]. In this version of hybrid backtracking, following each new assignment, AC is re-established in the subproblem formed by the variables not yet given an assignment. If this AC fails, then the last value assigned is retracted; if there are no more values to test for the last variable chosen, then search backs up and tries another value for the next-to-last variable chosen. This process continues until either a complete set of consistent assignments is found (aka a solution) or there are no more values to test at the highest level, which means there are no solutions.

Forward checking is a more circumscribed form of hybrid backtrack search [13]. In this case, after each assignment, domains of variables sharing a constraint with the current variable are checked for consistency with the value just assigned before proceeding to the next assignment.

The *dynamic* variable ordering heuristics used in these experiments were mindom and d/fd. By dynamic, we mean that the heuristic takes account of the current state of the problem at each stage of search. Mindom is a purely dynamic heuristic, in that it relies on a single property (domain size) that is affected by preprocessing as well as local consistency performed during search. It is also a fairly weak heuristic, which limited the difficulty of the problems that could be used in the experiments. d/fd was also used to determine if the same results would be found for a compound heuristic where dynamic domain size is one component.

The *static* variable ordering heuristic used in these experiments is the maximum (static) degree heuristic. This selects variables on the basis of their degree in the constraint graph (i.e. the number of constraints a variable is involved in), choosing the unassigned variable with the highest degree. In this case the feature used by the heuristic remains the same regardless of how much certain domains are reduced during preprocessing.

In these experiments, MAC was performed with d-way branching. Unless otherwise noted, domain values were tested in lexical order.

3.2. Preprocessing algorithms

The experiments to be described involved comparisons of search effort after different amounts of problem reduction. In addition to AC itself, this work made use of certain forms of *singleton arc consistency*, or SAC [14]. SAC is a form of AC in which a value a is considered the sole representative of the domain of X_i. If AC cannot be established for the entire problem under this condition, then there can be no solution with this value, so it can be discarded. If this condition can be established for all values in problem P, then the problem is singleton arc consistent.

A related form of consistency is called *neighbourhood singleton arc consistency*, or NSAC [15]. NSAC algorithms establish SAC with respect to the neighbourhood of the variable whose domain is a singleton as opposed to the entire problem.

For our purposes, the important fact about these algorithms is that with respect to values deleted, they form a series, where AC is dominated by NSAC, which in turn is dominated by SAC.

3.3. Problems used

The goal of these experiments was to examine the effects of problem reduction on search effort with different heuristics. To this end, certain requirements had to be met regarding problem features and algorithm performance: (i) problems had to have variables with different numbers of constraints (different degrees in the constraint graph), (ii) more restrictive forms of preprocessing had to have greater effect (more values removed), and for the same reason, (iii) there had to be differences in search effort (number of search nodes) following different degrees of problem reduction. Care was also taken to ensure that more stringent forms of preprocessing did not reduce problems so much that they were trivially solvable; in such cases 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.

The following types of problems were used, which in each case met these requirements:

- “Geometric problems" whose values had varying degrees of support in different constraints (see [6] for further details),
- Random “relop problems" whose constraints were relational operators,
- Radio Link Frequency Allocation Problems (RLFAPs) derived from a benchmark problem by altering the tightness of a randomly chosen set of greater-than constraints and randomly swapping adjacent variables within constraint pairs,
- Driverlog benchmark problems.

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’ [16]. In the present work, in addition, if there is more than one connected component, separate components are connected via pairs of variables (one in each component) having the smallest Euclidean distance between their points in the unit square, in order to make a graph with a single connected component.

Two sets of problems were tested having either 80 or 40 variables, with constant domain sizes 15 and 12, respectively. To limit variation in graph density (proportion of possible edges in the constraint graph), only problems within a small range around a “target" were accepted; for the 80-variable problems the target was 380 ($\pm$ 5) constraints; for the 40-variable problems, 160 ($\pm$ 3).

Relop problems had 50 variables, domain size 15 and a graph density of 0.4. The binary constraints were either $\neq$, $>$, or $\geq$ chosen randomly according to the ratios, 70:05:25.

Radio Link Frequency Allociation Problems (RLFAPs) are also binary problems, with two kinds of distance (difference) constraint: $|v(X_i) - v(X_j)| = k$ and $|v(X_i) - v(X_j)| > k$, where X_i and X_j are variables, v is an operator that assigns a value to a variable, and k is an integer. These problems had 100 variables with domain sizes of about 40, and constraint graphs of low density.

In order to obtain a larger sample of RLFAPs with the same basic characteristics, the following method was used. A single benchmark problem was selected. From this, a set of problems was generated by taking the original problem and choosing 60% of the $|v(X_i) - v(X_j)| > k$ constraints at random and decrementing the value of k in those constraints by about 20. In addition, 25 pairs of variables were randomly chosen from variables numbered 1 to 50 along with adjacent variables whose labels differed by no more than 7, and the latter were swapped to create different constraints.

Driverlog problems were benchmarks from the University of Artois website.

For all problem sets except the driverlog series, a large number of problems were generated that were filtered for solutions (in a generate-and-test fashion). From each set, the hardest problems were chosen

Table 1
Correlations Based on Proportional Domain and Search Reduction After AC and NSAC

probs	dom vs del	d/fd vs del	deg vs del	dom vs dg	d/fd vs deg
geo80	–	0.294	0.590^{+}	–	0.171
geo40	0.598^{+}	0.375^{*}	0.850^{+}	0.635^{+}	0.685^{+}
relop50	-0.186	0.032	0.361^{+}	-0.160	0.200
RLFAP100	-0.149	-0.102	0.242^{*}	0.153	0.000
driver	-0.371	0.686	0.943^{*}	-0.086	0.686

Notes. “dom", “d/fd", “deg" and “del" refer to fractions based on search nodes and deletions. * $p < .05$, $^{+}$ $p < .01$, two-tailed.

for use in the present experiments. This ensured that the effects of different forms of preprocessing on search were more pronounced than they would have been if random samples of problems had been used due to simple ceiling effects. In other words, if a problem is easy to solve under all conditions, then deleting more values during preprocessing has very little effect, so factors that affect the relation between preprocessing and search are harder to detect.

3.4. Implementation details

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. In these experiments, search was discontinued if it exceeded one million nodes. In these cases, the cutoff value was used as the search node number.

4. Previous Results: More than Problem Reduction

This section presents some of the basic evidence that more is involved in the effects of preprocessing on search than simple problem reduction, material presented at greater length in [7]. It is included here to provide context for subsequent results.

First we note that conditions for demonstrating this were met in these experiments. Thus, for all problem sets, NSAC deleted appreciably more values than AC, and SAC usually deleted appreciably more values than NSAC. In addition, search effort was always less after greater problem reduction. (Data demonstrating these effects can be found in [7], Table 1.)

The main finding in these experiments was that search with dynamic heuristics is usually not strongly related to problem reduction, in contrast to search with a static search heuristic. This was demonstrated by comparing performance after AC and NSAC, specifically, by testing the correlation between relative problem reduction and relative search reduction for each set of problems.

Relative reductions were assessed with these fractions: $\frac{\text{NSAC}-\text{AC}}{\text{NSAC}}$ was used for total deletions per problem, while $\frac{\text{AC}-\text{NSAC}}{\text{AC}}$, was used for search nodes. Given the violations of assumptions for parametric statistics, statistical tests were done with the non-parametric Spearman Rank Correlation Coefficient [17].

The results of these experiments are shown in Table 1. For all problem sets, correlations between search reduction and domain reduction were always statistically significant for max degree (third data column). For dynamic heuristics, the correlation between search and domain reduction was rarely significantly different from zero.

These differences are partly due to the fact that following greater problem reduction, search with dynamic heuristics was not always improved. In fact, for some problems it was appreciably worse despite an overall improvement in the mean. Such effects never occurred with the static degree heuristic.

These results show that when search involves dynamic ordering heuristics, the effects of preprocessing involve other factors than problem reduction, although that of course plays a part. Both the order-of-

magnitude improvements that were sometimes observed after greater problem reduction, and, even more decisively, deterioration in performance, are inexplicable on this basis, and neither of these are found with a non-dynamic heuristic like max degree. Moreover, the correlational analysis yielded some results where a negative correlation was found between search with a dynamic heuristic and the amount of domain reduction; yet NSAC still led to improvement in search on average in these cases

Since there is a close relation between the dynamic character of the heuristic and these 'anomalous' effects, this is consistent with the notion that domain reductions serve to guide these heuristics during search toward making generally better decisions. Section 6 gives further evidence that this explanation is the correct one. But first we will consider an old result that heretofore has not received a proper explanation, and show that it fits within the present framework.

5. Resolving an old conundrum

In some work done in the 1990s it was found that preprocessing before search sometimes resulted in an excess of overall work, and in some cases search itself was negatively affected [12]. These results were found using the forward checking algorithm, either carried out by itself or preceded by arc consistency. Perhaps because MAC was so effective with these problems, a similar anomaly was not reported in this case.

The authors of this work did not try to analyze this effect. However, since they used a dynamic variable ordering heuristic, viz, minimum domain size, one would expect that the effect demonstrated in the previous section would be in operation.

This expectation was verified by, (1) using minimum domain size in combination with forward checking with some of the problems used in the present work, (2) replicating the original Sabin and Freuder results and comparing results obtained with min domain size and max static degree.

Sabin and Freuder used (homogeneous) random problems with 50 variables, domain size eight, tightness = 0.5, and densities of 0.06, 0.07, 0.08 and 0.09, where "density" in this case refers to the proportion of edges added to a complete spanning tree. For the present tests, 25 problems with these parameter values were generated at each density. (Incidentally, this series passes through a critical complexity region, so that while all of the 0.06 problems had solutions, none of the 0.08 and 0.09 problems did.) For these densities in ascending order, with min domain size, 11/25, 8/25, 9/25 and 5/25 problems gave poorer search results (for nodes and constraint checks) when search was preceded by AC. In contrast, with max static degree the number of nodes was either the same or slightly less when search was preceded by AC. A very similar pattern of results was found for MAC with or without AC prior to search, although of course the differences were much smaller. (The latter and the small number of problems (5) may account for the failure to report such differences in [12].) Clearly, the Sabin-Freuder result is due to the same factors demonstrated in the present paper.

6. What Kind of Information Does Domain Reduction Supply?

The previous experiments showed that when dynamic heuristics are used to select variables, preprocessing involves more than search space reduction. This implies that domain reductions affect heuristic decisions so as to enhance search, at least on average. One possibility is that information of some sort is gained from domain reductions that guides these heuristic decisions.

6.1. Ruling out extraneous factors

To establish that an information channel is indeed being set up between preprocessing and search with dynamic ordering heuristics, it is necessary to rule out other factors. These include the branching factor associated with variable selection, which when combined with propagation during search can reduce the search space appreciably. In addition, there are 'noise factors' such as value selection and

Table 2
Backtrack Search with Domain Reduction Heuristics or Lexical Ordering

statistic	max domreduct	min domreduct	lexical
mean	7,207,246	6,796,259	25,495,721
median	439,935	845,579	2,775,788

Notes. Entries are for search nodes, based on 50 problems.

propagation effects, which can affect search dramatically without being relevant to the information flow we are concerned with.

In order to demonstrate that information coming from preprocessing can in and of itself affect search results, the following procedure was used. First, a preprocessing algorithm was used that deleted a sufficient number of values to potentially guide decisions during search. For this purpose, NSAC was used, which eliminated 10-20% of the domain values. Following preprocessing, all the deleted values were returned to their domains, which in this case were all of the same size originally. In addition simple backtracking was used, so there was no propagation during search. With these procedures, the branching factor was held constant, and any confounding effects of propagation were avoided. In order to remove the effect of value selection, each problem was solved 50 times with values chosen at random from each domain.

For these runs, a fixed variable ordering was used based on the size of domain reduction after preprocessing, with larger reductions chosen over smaller ("max domreduct" in Table 2). Under these conditions, if this ordering improves on a lexical ordering of the variables, we can conclude that information was passed from preprocessing to the heuristic used to select variables during search. This information should improve search overall, although as we know from the previous experiments and from earlier work, this is not guaranteed in all cases. In addition, an anti-heuristic was used, where smaller domain reductions were chosen over larger ("min domreduct" in Table 2).

For this experiment, geovarsat problems were used, whose characteristics were described earlier. Because basic backtracking is a weak method for solving such problems, small problems were used, with 20 variables and domain size 7. The target number of constraints was 50 ± 5. There were 50 problems.

Comparisons between the variable ordering based on maximum domain reduction and a lexical ordering were clearcut. With the heuristic, search was between three and four times better in terms of search nodes (and of course runtimes). A paired comparison t-test based on log-transformed values gave significant results (t = 4.223, $p < 0.001$), as did a sign test (35/50 in expected direction $p < 0.01$). From this we can conclude that domain reduction *in and of itself* provides a signal that can guide variable selection so as to improve search effort on average.

At the same time, the anti-heuristic (minimum domain reduction) also performed better than the lexical ordering (cf. Table 2). This was unexpected. In addition, the mean was somewhat lower than with the heuristic. This was due to one problem for which the heuristic performed very badly; hence, it is not reflected in the median (also shown in the table). In this case, the paired-comparison t-test based on log-transformed values was significant (t = 2.282, $p < 0.01$), while the sign test was almost significant (32/50 in expected direction, $p < 0.10$). Tentatively, I would ascribe these anomalies to the relatively small sample size. This seems likely in view of the tremendous variance one obtains with backtrack search, given that there is no amelioration of the effects of bad variable selections by propagation.

6.2. Comparing domain reductions with constraint weights

Further insight into what a dynamic heuristic is being informed of can be obtained by comparing domain reductions with "constraint weights" obtained through random probing [18]. Random probing is a variation of the weighted degree procedure [19]. In that procedure, each constraint has an associated

Table 3
Summary Statistics for Correlations between Weights and Domain Reductions for Geovarsat Problems

	40-var			80-var		
measure	AC	NSAC	SAC	AC	NSAC	SAC
lowest	0.043	0.385	0.436	-0.141	0.152	0.217
highest	0.501	0.863	0.936	0.268	0.586	0.545
mean	0.251	0.613	0.686	0.102	0.356	0.417
mdn	0.231	0.630	0.688	0.135	0.419	0.445
no. $p < .05$	12	25	25	10	23	25
no. $p < .001$	1	23	23	0	18	19

Notes. 25 problems in each set. "no. $p < x$" is number of correlations statistically significant at designated levels.

"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, which is used to enhance the maxdeg or d/fd heuristic.

Random probing is a preprocessing technique in which a search procedure is carried out by choosing variables 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 with the original weighted degree procedure. This procedure allows (full) search to begin with a set of constraint weights (and therefore weighted degrees) rather than building these from scratch during the search process.

In the present work we are interested in examining the profile of weight-sums (aka weighted-degrees) itself and comparing it with the profile of domain reductions for the same problem. For this purpose, the two sets of geovarsat problems were used. For each problem, random probing was carried out using a schedule of 50 probes, with each probe continuing until 40 failures had occurred. Then the profile of weight-sums obtained for the entire set of variables in a problem was correlated with the domain reductions for the same variables after AC, NSAC or SAC. This analysis employed the Spearman Rank Correlation Coefficient, corrected for tied scores [17].

Table 3 shows summary statistics based on correlations between summed weights and domain reductions for the set of variables in each individual problem. For AC, correlations were small (for 80-variable problems, some were negative), and only about half were statistically significant. In contrast, for both NSAC and SAC, correlations were much higher, and all were positive. For the 40-variable problem set, correlations based on NSAC or SAC were statistically significant for all 25 problems; for the 80-variable set, nearly all were. Correlations were usually slightly higher after SAC than NSAC (0.1 for 80-variable problems, 0.2 for 40-variable problems).

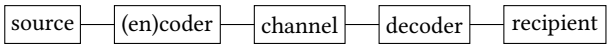
The significance of these results stems from the fact that constraint weights are due to bottlenecks encountered during search, so with random probing higher weights are associated with more basic, possibly global bottlenecks. As is well known, search effort can be greatly reduced by choosing bottleneck variables early in search (at the top of the search tree) [13]. So if these variables can be identified during preprocessing and this information can be transmitted to the search process, then obviously it will make search more efficient. Random probing is an effective procedure for doing this.

If, in turn, domain reduction profiles are positively correlated with weight profiles, this indicates that large domain reductions are often due to the same bottlenecks. Obviously, domain reductions are not as discriminating, nor can they be since the number of possible values is much smaller than the number of possible weight-sums. Moreover, many domains are unchanged (zero reduction). So the fact that correlations are sometimes low to moderate in size is to be expected. However, it is clear that SAC-based forms of consistency produce much better correlations, which in turn means that these stronger consistencies result in significant enhancements of code quality.

7. Characterizing Information Flow with CSP Search Heuristics

7.1. Information channels and CSP heuristics

The diagram below shows the standard view of information transmission [20]. Messages from a source are encoded and transmitted over a “channel", after which they are decoded and passed to a recipient.



But this process can be viewed somewhat differently:

In this case, the “channel" could be the storage of the result and its 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.

It is obvious that most algorithms make use of information in carrying out their tasks. For example, an internal comparison sort requires a list of entities with ordinal properties drawn from some specified set that are presented in an unspecified order. Each entity carries information that guides the decisions made in the series of comparisons and moves that constitutes a realization of the algorithm. So during the operation of the algorithm, information is constantly being extracted from the environment. However, since the channels in this case are noiseless, and there is a 1:1 mapping between transmissions and operations, for deterministic algorithms like this, this feature of the algorithm is not of great theoretical interest and can be ignored.

This is not the case, however, for CSP algorithms, where choices are not deterministic. Here, the information provides only a rough guide and can even be misleading. So in these cases, this aspect of the algorithmic procedure becomes an important matter for analysis, as the results presented earlier show.

All CSP heuristics operate by extracting information from a problem and using that information to guide decisions during search. As is well-known, there are many features that can be used. In the present terms, each of these (and each feature combination) forms a different information channel with respect to the extraction process.

Given the results in this paper, we can distinguish two kinds of information channel:

- information is extracted from ‘nature’, i.e. from pre-existing features of the problem
- information is extracted from results produced by another algorithm

7.2. Measuring the amount of information provided by preprocessing

We can, in fact, characterize the messages sent across each kind of channel in information theoretic terms. 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.) Here, 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 the preprocessing to the search stage always contains n letters, one for each variable (fixed length code). (We ignore messages containing d, since in these cases search would not be carried out.)

For both kinds of channel, one result of information extraction is that it allows search to be more discriminating, i.e. there are more distinctions among variables that allow choices to be made in order to improve search efficiency. Thus, one of the results of preprocessing a problem with a local consistency

Table 4
Mean Discriminability After Different Levels of Preprocessing

problems	AC	NSAC	SAC
geo80	0.705	1.610^{++}	1.821^{++}
geo40	0.948	2.291^{++}	2.350
relop50	1.739	2.137^{++}	2.354^{++}
RLFAP100	1.757	2.609^{++}	4.014^{++}
driver	0.539	0.749^{*}	1.930^{*}

Notes. Statistical comparisons are between NSAC and AC and between SAC and NSAC; in each case results appear beside the former. * $p < .05$, 1-tailed, $^{++}$ $p < .001$.

algorithm is to enhance *discriminability*, i.e. the number of discriminations that can be made (in this case) among variables. This can be measured using the usual formula for the entropy of a system S,

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

where m is the number of different code values, in our case the number of different amounts of domain reduction, and for $p(k)$ we use the relative frequency with which a given domain reduction occurs among these variables.

A few examples will give a sense of what this formula yields. Consider first, cases where preprocessing yields no domain reductions. In these cases the message will contain n zeros and will therefore not allow any discriminations between variables. In this case the amount of information transmitted is

$$p(0) \; log \frac{1}{p(0)} \; = \; 1 \; log \; 1 \; = \; 0$$

Next, consider cases where there are domain reductions. For example, suppose $n \; = \; 5$, and we have two messages, (0, 0, 0, 1, 2) and (0, 1, 2, 3, 4). In the first message there are three variables that cannot be discriminated from each other, while in the second all of them can. (Note that in the latter case $m \; = \; n$.) Using the same measure of information, we calculate the information given to the search algorithm by the first message as:

$$\frac{3}{5} \; log \frac{1}{\frac{3}{5}} \; + \; \frac{1}{5} \; log \frac{1}{\frac{1}{5}} \; + \; \frac{1}{5} \; log \frac{1}{\frac{1}{5}}$$

$$= \frac{3}{5} \; log \frac{5}{3} \; + \; 2(\frac{1}{5} log \; 5) = .7398 \; + \; .9288 = 1.6686$$

and by the second as:

$$5(\frac{1}{5} \; log \; 5) = log \; 5 = 2.3219$$

So the discriminability measure reflects the fact that the second message conveys more information to the heuristic.

Now, we know that more stringent local consistency algorithms generally result in greater domain reductions and consequently a greater range of possible code values. Since the information we are interested in pertains to discriminations between variables, this means that messages produced by more stringent consistency algorithms can transmit more information. That this does occur in practice is shown by an analysis of the problems and preprocessing algorithms used in the experiments that were the subject of Section 4. Results are shown in Table 4. For these problem types at least, stronger forms of preprocessing do result in greater discriminability for the large majority of problems. This means that when compared with AC, SAC-based consistencies yield greater discriminability along with higher code quality.

7.3. Code quality

In this situation, an ideal code will contain up to n distinct letters, for the case where all variables should be distinguished. This means that, unless d is at least as large as $n-1$, there may be some comparisons of potential significance that cannot be encoded through domain reductions. In other words, some (perhaps most) of the actual encodings will *necessarily* be deficient, since not all significant differences can be encoded.

This sort of code will also usually be degenerate; that is, two or more value-pairs can encode the same difference between variables. In fact, if variable X_i should be chosen before variable X_j, there are up to $\frac{d(d-1)}{2}$ ways of signaling this via domain reduction. In addition, in some cases it wll not matter which of two variables is chosen. Fortunately, search is not affected by this aspect of the code.

It might be questioned whether code quality is anything more than discriminability. However, the following argument shows that the latter is only one component of code quality. Suppose we have a message consisting of a set of domain reductions. In addition to informing a dynamic heuristic which variable to select next for assignment, it could also be used to inform an "anti-heuristic", where the opposite decisions are made (e.g. the anit-heuristic of mindom is maxdom). Now, typically, anti-heuristics perform on average worse than selecting variables at random. But in this case, the discriminability is the same. So, in addition to discriminability, we have to consider other aspects of what we are calling "code quality".

A code based on domain size is not perfect, since (1) there are known heuristics that give reliably better performance, (2) as we have seen in the experiments, there are cases where greater domain reductions after more stringent preprocessing leads to reductions in search efficiency. Hence, in terms of efficient search, this code can lead to non-optimal choices. Since the information channel is noiseless, this has to be considered as a problem of interpretation, i.e. a semantic problem, one that can lead to mistakes or decision-faults as opposed to transmission errors.

Although at this time, it is entirely unclear how to rigorously characterize the situation, two approaches seem possible. One is to consider the difference from a 'perfect' ordering. The other is to consider the problem as one of risky choice, where the amount of risk might be characterized in terms of loss functions.

In the first case, we can characterize an ideal code as one that establishes a relational homomorphism between the comparisons between code letters in the message and the ordering relation of variables that yields the smallest search tree. In the case under consideration, where code letters are actually numbers whose quantities can be compared, this means that if quantity (code letter) a is larger than quantity b, then choosing the variable associated with a over the variable associated with b will result in a smaller search tree. And if two or more code letters are the same, then the order in which their associated variables are chosen will not matter.

In this case, the difference from the ideal could be measured by the number of inversions of pairs of variables with respect to a perfect ordering (e.g. if $a > b$ in the perfect ordering, $b > a$ would be an inversion). Unfortunately, in practice we do not know what the best variable ordering is, so we can neither determine if there is a relational homomorphism in a given case nor the degree to which the ordering suggested by domain reductions is discrepant.

An approach through loss functions could potentially incorporate the impact of a given deviation from a 'perfect' ordering in addition to the number of inversions. In addition, it might be possible to estimate the loss function through some sort of sampling. However, it must be left for future work to decide whether methods like these can be made useful in practice.

8. Conclusions and Prospects

The present work takes a few more steps toward the goal of providing a theoretical (and experimental) foundation for the employment of search heuristics. In particular, it is now clear why there are vagaries in search effort after problem reduction through removal of locally inconsistent values.

In the past, this problem was only addressed implicitly, either by introducing more powerful search methods [12] or by ameliorating the situation by combinining domain reduction with other heuristic features such as the degree of a variable [21] (which as we have shown does not solve this particular problem). Since the nature of the problem is now recognized, it may be possible to deal with it more directly.

Using an information-based framework, we were able to demonstrate that there are two factors associated with information transmission between preprocessing and search, which we have termed discriminability (among the letters of a message) and code quality. We have also shown that, at least for AC-based algorithms, stronger forms of local consistency reliably enhance both properties. For discriminability, this was demonstrated directly; for code quality, this was inferred from correlations between domain reductions and failure-based constraint weights, which earlier work has shown can identify bottleneck variables that should be instantiated early in search [22]. Although we were able to propose a well-defined measure for discriminability, defining code quality precisely and devising methods to enhance it reliably remain problems for further study.

This analysis may have implications that go beyond the field of constraint satisfaction. Thus, any problem solving method that includes some kind of problem analysis and/or alteration anterior to heuristic choices may involve the kinds of phenomena described in this paper. And the general phenomenon of information flow applies to a wide range of algorithms that involve non-deterministic choices among alternatives at some steps of their execution.

9. Acknowledgments

Acknowledgments

This publication has emanated from research supported in part by a research grant from Science Foundation Ireland under Grant No 12/RC/2289_P2. Table 1 and portions of Table 3 were first published in Progress in Artificial Intelligence: 23rd EPIA Conference on Artificial Intelligence, Proceedings, Part II, LNAI No. 14968, 2024, by Springer Nature.

References

[1] Diego Aineto, Riccardo De Benedictis, Marco Maratea, Munyque Mittlemann, Gianpero Monaco, Enrico Scala, Luciano Serafini, Ivan Serina, Francesco Spegni, Elisa Tosello, Alessandro Umbrico, Mauro Vallati (Eds.), Proceedings of the International Workshop on Artificial Intelligence for Climate Change, the Italian Workshop on Planning and Scheduling, the RCRA Workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion, and the Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy AI4CC-IPS-RCRA-SPIRIT 2024, co-located with 23rd International Conference of the Italian Association for Artificial Intelligence (AIxIA 2024), CEUR Workshop Proceedings, CEUR-WS.org, 2024.

[2] Chrisian Bessière, Constraint propagation, in: Handbook of Constraint Programming, Elsevier, 2006, pp. 29–83.

[3] Edward Tsang, Foundations of Constraint Satisfaction, Academic Press, 1993.

[4] Rina Dechter, Constraint Processing, Morgan Kaufmann, 2003.

[5] Robert J. Woodward, Higher-level Consistencies: When, Where, and How Much, Ph.D. thesis, University of Nebraska, 2018.

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

[7] Richard J. Wallace, Domain reductions after preprocessing: Effects on dynamic variable ordering heuristics in constraint satisfaction search, in: Manuel F. Santos, José Machado, Paulo Novais,

Paulo Cortez, Pedro M. Moreira (Eds.), Progress in Artificial Intelligence. 23rd EPIA Conference on Artifical Intelligence, EPIA 2024, Part II. LNAI No. 14968, 2024.

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

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

[10] Ruiwei Wang, Wei Xia, Roland 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.

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

[12] Daniel Sabin, Eugene Freuder, Contradicting conventional wisdom in constraint satisfaction, in: Eleventh European Conference on Artificial Intelligence-ECAI'94, Wiley, 1994, pp. 125–129.

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

[14] Christian Bessière, Stéphane Cardon, Romuald Debruyne, Christophe Lecoutre, Efficient algorithms for singleton arc consistency, Constraints 16 (2011) 25–53.

[15] Richard J. Wallace, SAC and neighbourhood SAC, AI Communications 28 (2015) 345–364.

[16] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, Catherine Shevron, Optimization by simulated annealing: An experimental evaluation. part II. graph coloring and number partitioning, Operations Research 39 (1991) 378–406.

[17] Sidney Siegel, Nonparametric Statistics for the Behavioral Sciences, McGraw-Hill, New York, 1956.

[18] Diarmuid Grimes, Identifying Sources of Global Contention in Constraint Satisfaction Search, Ph.D. thesis, National University of Ireland, Cork, 2012.

[19] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, Lakhdar Sais, Boosting systematic search by weighting constraints, in: Sixteenth European Conference on Artificial Intelligence-ECAI'04, IOS, 2004, pp. 146–150.

[20] Douglas S. Jones, Elementary Information Theory, Oxford University Press, Oxford, 1979.

[21] Christian Bessière, Jean-Charles Regin, MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems, in: Principles and Practice of Constraint Programming-CP'96. LNCS No. 1118, Springer, 1996, pp. 61–75.

[22] Richard J. Wallace, Diarmuid Grimes, Experimental studies of variable selection strategies based on constraint weights, Journal of Algorithms: Algorithms in Cognition, Informatics and Logic 63 (2008) 114–129.