Explaining the Effects of Preprocessing on Constraint Satisfaction Search

Richard J. Wallace(✉)

Insight Centre for Data Analytics and Department of Computer Science, University College Cork, Cork, Ireland
richard.wallace@insight-centre.org

Abstract. Preprocessing constraint satisfaction problems is a much studied method for improving the performance of subsequent solution search. The traditional explanation for its beneficial effects is "problem reduction", where possible values that cannot take part in a solution are discarded, leaving fewer possibilities to explore during search. Here, we show that this is not the only or even the main factor when dynamic variable ordering heuristics are used. Multiple lines of evidence indicate that under these conditions domain reductions effected by preprocessing serve to inform the heuristic as to which variables should be chosen for instantiation before others. It is suggested that an information transmission model is needed to account for such effects, and it is argued that an extension of this approach can incorporate simple domain reduction effects as well.

Keywords: Constraint satisfaction · Preprocessing algorithm · Arc consistency · Neighbourhood singleton arc consistency

1 Introduction

The study of constraint satisfaction problems (CSPs) has reached a point where there are deep formal analyses as well as a plethora of effective techniques for solving problems of this type. At the same time, there are many aspects of constraint solving that remain obscure to varying degrees. In part, this is because the combinatorial complexity of these problems gives them features that are hard to comprehend with the standard formal machinery that we have. Instead, we must often fall back on statistical analysis to even discern certain relationships.

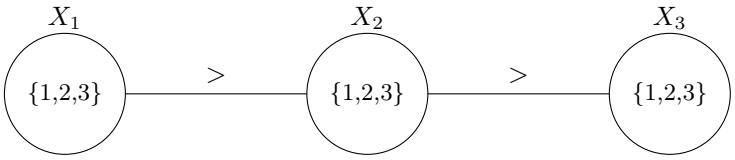
But it can also happen that we simply fail to see the gaps in our explanations. An important example of such an oversight is the topic of the present paper.

In this case, the area of inquiry pertains to what are called local consistency techniques. It not too much to say that these methods form the core of constraint programming and serve to set it off from other approaches to the general problem of combinatorial optimization. Although they can be used in various contexts, a significant one, especially for more stringent forms of consistency, is the processing of a problem in order to simplify it prior to the actual search for a solution.

L. Longo and R. O'Reilly (Eds.): AICS 2022, CCIS 1662, pp. 423–436, 2023.
https://doi.org/10.1007/978-3-031-26438-2_33

Local consistency techniques are polynomial-time algorithms that test for limited forms of consistency within portions of a problem [1]. The best-known example, which is the one to be considered here, is called "arc consistency". This is described more carefully in the following section; here we can simply say that the algorithm tests 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. The thinking here is that if a value is not arc consistent, then it cannot be part of any complete set of labelings that satisfy all the constraints in the problem, aka a solution. Therefore, when we search for a solution, we don't have to consider that value at all.

For example, consider the simple CSP depicted below, where there are three variables, X_1, X_2, and X_3, represented by the circles (graph nodes) in the diagram below. Each variable can take one of three possible values (shown inside the circles). However, there are constraints between variables X_1 and X_2 and between X_2 and X_3, indicated by the lines (graph edges) joining the circles, which specify conditions that must be met. Constraints, in turn, are associated with relations, indicated here by relational operators above the lines. In this case, the relations are that X_1 must be greater than X_2, and the latter must be greater than X_3.



Now, given these constraints, there are values in the domains of some variables that cannot appear in any solution. This can be shown by checking values in the adjacent domain to see if anything can go with (i.e. support) a given value. For example, value 1 in the domain of X_1 is not greater than any value in the domain of X_2; hence, it cannot be a part of any solution, so it can be removed from the problem without any solution being lost. By using this strategy repeatedly, we can reduce the number of values that we have to consider when we search for a solution. In this case we can reduce each domain to a single value, which gives us the unique solution to this problem immediately.

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 much examine during search. Another way of putting this is that preprocessing reduces the search space.

So in a well-known textbook on the subject [9], we find this statement:

> "Problem reduction techniques transform CSPs to equivalent but hopefully easier problems by reducing the size of the domains and constraints in the problems". (p. 33)

And later on:

> "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". (pp. 79–80)

A similar account of the aims of local consistency preprocessing is found in another important reference work [4]:

> "In general, inference, as it is applied to constraints, narrows the search space of possible partial solutions by creating equivalent, yet more explicit networks". (p. 52)

In the present paper we will argue that this is not the full story. In fact, for problems of any difficulty (in contrast to the example above), this is not the chief benefit of preprocessing. For strategies that are sensitive to the current state of the problem, and in particular the state achieved by preprocessing, the latter can impart critical information that allows the search process to operate much more efficiently than it otherwise could.

In the remainder of the paper, we will explicate the last statement and demonstrate it experimentally. Section 2 provides necessary background in the form of basic concepts and definitions. Section 3 describes the algorithms used in this paper. Section 4 describes the experimental environment used in this work, including types of CSPs used. Sections 5 and 6 present the basic argument of the paper together with supporting evidence. Section 7 discusses a related finding that bears on the present thesis. Section 8 considers how the communication process by which preprocessing affects subsequent search might be modelled. Section 9 summarizes work to date and indicates what still must be done to better understand this critical aspect of constraint solving.

2 Basic Concepts

Here, we review some of the basic concepts in the field that form the background to 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 a set of possible *values* for 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. (In the introductory example, one constraint has scope $\{X_1, X_2\}$,

and its relation is based on the Cartesian product of domains D_1 and D_2 i.e. the pairs in $\{(1,1),(1,2)\ldots(3,2),(3,3)\}$ where the first element in greater than the second).

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 does not violate any constraint in C.

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.

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 in the relation associated with the constraint that includes a assigned to X_i.

The present paper also makes use of certain forms of *singleton arc consistency*, or SAC [2,3,10]. 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 for the entire problem 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 [11]. NSAC algorithms establish SAC with respect to the neighbourhood of the variable whose domain is a singleton as opposed to the entire problem.

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 $\{1\ldots n\}$, singleton arc consistency can be established in the subgraph based on that variable and its neighbours, then the problem is neighbourhood singleton arc consistent.

3 Arc Consistency and (N)SAC Algorithms

Pseudocode for a standard arc consistency algorithm is shown in Fig. 1. Basically, the algorithm considers each value in each domain in turn, and for each adjacent constraint, it determines whether that value has a supporting value in the domain

```
   Procedure AC-3
1      Q ← X
2      OK ← true
3      While OK and not empty-Q
4          Select and remove X_i from Q
5          Foreach constraint C_ij that includes X_i
6              Changeflag ← false
7              Foreach value a ∈ domain of X_i
8                  If there is no value b ∈ domain of X_j that supports a
9                      Remove a from domain of X_i
10                     Set Changeflag to true
11             If domain of X_i is empty
12                 OK ← false
13             If OK and Changeflag is true
14                 Update Q to include all variables adjacent to X except X_j
```

Fig. 1. Pseudocode for AC-3, a standard AC algorithm, for binary CSPs.

of the adjacent variable (i.e. the other variable in the constraint). It continues to do this until all values have been checked as many times as they need to be, which may involve re-checking values if some of their neighbouring domains have changed. If a problem is not arc consistent, then eventually some domain of values will become empty. (This is called a domain "wipeout").

The algorithm shown is of the type called "AC-3" [6]. All AC-3 style algorithms use a single queue (although the queue may take different forms) and queue updating procedures as shown in Fig. 1. The version shown here is for problems with binary constraints, but the AC-3 algorithm can be extended to k-ary constraints as well; in this case, the test is: for constraint C and for a given value v in the domain of X_i, is there is an k-tuple in C in which v is assigned to X_i?

SAC and NSAC use AC as a building block. The basic idea is to reduce the domain of a given variable X_i to a single value v, and then determine whether the problem or a subproblem that includes X_i is consistent under these conditions. If it isn't consistent, then v can be removed from the problem without losing any solutions, since if it can't support AC, it can't support any fully consistent solution to the problem.

Figure 2 shows a state-of-the-art NSAC algorithm [12]. This algorithm uses an AC-3 style queue similar to the arc consistency algorithm, except that at each step, the algorithm attempts to establish AC for the neighbourhood subproblem. In addition, if a value 'fails', i.e. at a given step, wipeout occurs, then after removing that value, simple AC is performed for the entire problem (line 10 in the figure). In addition, whenever a value is removed the queue must be updated (lines 12 and 14 in the figure). (The corresponding SAC algorithm is very similar, except that on line 7 AC is established for the entire problem and on line 14 the queue is updated to include all variables in the problem).

```
   Procedure NSACQI
1      Q ← X
2      AC-OK ← AC(P)
3      While AC-OK and not empty-Q
4          Select and remove X_i from Q
5          Foreach v_j ∈ dom(X_i)
6              dom'(X_i) ← {v_j}
7              If AC(X_i + neighbours(X_i)) leads to wipeout
8                  dom(X_i) ← dom(X_i) \ v_j
9                  If AC(P) leads to wipeout
10                     AC-OK ← false
11                 Elseif domains have changed during AC
12                     Update Q to include all neighbours of variables with changed domains
13         If domain of X_i has changed
14             Update Q to include all neighbours of X_i
```

Fig. 2. Pseudocode for NSACQ with interleaved AC. P is always the full problem in its current state.

The search algorithm used in this paper is called maintained arc consistency (MAC) [7]. In this version, after preprocessing, MAC performs a backtrack-style search in which a partial solution is extended by choosing a variable and then a value to assign to that variable that is consistent with all the assignments already chosen. Following each new assignment, MAC also establishes AC 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.

These algorithms will be used to analyze the effects of preprocessing on search. The following empirical studies will attempt to separate problem reduction from other effects that occur when some domain values are eliminated.

4 Experimental Methods

All algorithms were implemented in Common Lisp. Experiments were run using Macintosh Common Lisp (MCL) version 5.1 on an iMAC (MAC OS X version 10.2.8) with a Power PC 800 MHz CPU. Search was carried out using a form of MAC called MAC-3 using different *variable ordering heuristics* for selecting the next variable to assign a value to during search. After a variable was chosen, values in its domain were tested in lexical order (i.e. 1, 2, 3 ...).

One variable ordering heuristic used in these experiments, called minimum domain over forward degree (min d/fd), chooses the variable with the lowest ratio of domain size (number of domain values) to forward degree. The latter is defined as the number of constraints that a variable has with variables that have not yet been assigned a value (hence, the adjective "forward"). The chief

characteristic of this heuristic is that it is *dynamic*, i.e. it takes account of the problem representation at each stage of search. Therefore, the domain size used is the current size of the domain, which will have been reduced during search to those values consistent with prior assignments. Similarly, the value of the forward degree will depend on which variables sharing a constraint with a given variable have not yet been assigned a value.

The other variable ordering heuristic that was used is the maximum static degree heuristic. This chooses 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.

For purposes of demonstration, the following types of problems were utilized. "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' [5]. 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. The present sets of problems had binary constraints, although the scheme can be extended to produce constraints with any number of variables.

Four different problem sizes were tested (here, size = number of variables): 120, 80, 40, and 25. Problems in these sets had identical domain sizes of 20, 15, 12 and 15 values, respectively. To limit variation in graph density (proportion of possible edges), only problems within a small range around a "target" were accepted; for example, for 120-variable problems, the target was 540 constraints ($\pm$ 3), giving a graph density of 0.076.

With geometric problems, constraint relations are generated according to a random scheme for selecting supports. In the problems used in these experiments, domain values had either of two levels of support within each constraint. As a result, values were generally well supported (low tightness), but occasionally they were not (high tightness). For 120-variable problems, a constraint tightness value of 0.3 was selected with a probability of 0.8; otherwise, a constraint tightness of 0.7 was chosen. For other problem sizes, the latter tightness value was 0.75. In the remainder of the paper, these problems will be referred to as "geovarsat" problems.

The second set of problems were a kind of benchmark problem known as Radio Link Frequency Allociation Problems (RLFAPs). These are also binary problems, and have two kinds of distance (difference) constraint, meaning that the difference between two assignments must satisfy a numerical relation. The two forms of distance constraint are $|v(X_i) - v(X_j)| = k$ 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 some integer value. These problems have constraint graphs of low density and domain sizes of about 40. There is also considerable structure to the pattern of constraints and the relation of the k values to the values in the domains. For the present experiments, in order to obtain a larger sample of problems with the

same basic characteristics, the following method was used. A single benchmark problem was used (known as rlfap-graph3), having 200 variables. From this, a set of problems was generated by taking the benchmark problem and randomly choosing ten percent of the $|v(X_i) - v(X_j)| > k$ constraints and altering them by either incrementing or decrementing the value of k by 10. (The decision to increment or decrement was decided randomly, each decision occurring with equal probability).

For all problem sets, a large number of problems were generated that were filtered for solutions (in a generate-and-test fashion). From each set, the 25 hardest problems were chosen for use in the present experiments.

In these experiments, search was discontinued if a cutoff was reached, which was one million search nodes for geovarsat problems and one hundred thousand nodes for RLFAPs; in these cases the search node number was recorded as having the cutoff value.

5 Effects of Preprocessing on Geovarsat Problems

5.1 Results

Results for geovarsat problems of all sizes using the dynamic min d/fd heuristic are shown in Table 1. In this experiment, three different levels of consistency were established prior to search: AC, NSAC, and SAC. Successively higher levels of consistency were associated with successively larger numbers of deleted values, and, on average, successively smaller search trees. For these problems, NSAC deleted about five to ten times as many values as AC, while the difference between NSAC and SAC was sometimes almost double and sometimes just a few percentage points.

Table 2 gives results with min d/fd and max degree for the three smaller problem sizes. Note, first of all that differences in search with max static degree are generally more modest than with min d/fd. Thus for min d/fd, the mean reduction in search nodes after NSAC preprocessing, compared to AC, was 30%, 63%, and 45% for the 80-, 40- and 25-variable problems, respectively, while the corresponding values for max static degree were 23%, 46%, and 11%.

Moreover, when one looks at the data for individual problems, one finds that for d/fd there are occasional cases where following application of a higher level of consistency, search is actually somewhat worse (more search nodes). For the 80-variable problems there were two such cases when NSAC was compared to AC, and eight cases when SAC was compared to NSAC, and there were occasional cases for the two smaller problem sizes as well. In contrast, there were no such cases when the max static degree heuristic was used.

We can go further and consider the relation between the domain reduction after NSAC and AC and the search reduction following these two preprocessing algorithms. This was done by taking the ratio, $\frac{\text{AC - NSAC}}{\text{AC}}$, where AC and NSAC stand for search nodes following AC and NSAC, respectively. A similar

Table 1. Preprocessing and Search With Different Forms of Local Consistency (Geometric Problems, Varying Satisfiability)

problem size	AC		NSAC		SAC	
	Delete	Nodes	Delete	Nodes	Delete	Nodes
120	40	>376,457	229	2915	391	818
80	14	49,075	66	34,500	81	11,921
40	12	335	126	125	211	87
25	4	187	21	102	25	96

Notes. Means of 25 problems. min domain/forward degree variable ordering heuristic. "size" is number of variables. "delete" is number of values deleted. "nodes" is search nodes. ">" indicates cutoff reached on some searches.

Table 2. Search with Different Forms of Local Consistency And Different Variable Ordering Heuristics (Geovarsat Problems)

Heuristic	AC nodes	NSAC nodes	SAC nodes
80-variable probs			
Min d/fd	49,075	34,500	11,921
Max degree	86,379	66,501	58,440
40-variable probs			
Min d/fd	335	125	87
Max degree	334	181	84
25-variable probs			
Min d/fd	187	102	96
max degree	196	174	169

ratio, $\frac{\text{NSAC - AC}}{\text{NSAC}}$ was used for total deletions after NSAC and AC. Pearson Product Moment Correlations were derived after transforming these ratios using the arcsin function [8]. The results are shown in Table 3.

Table 3. Correlations between Proportional Values of Search and Domain Reduction Following AC and NSAC (Geovarsat Problems)

Probs	d/fd vs dels	mxdeg vs dels	d/fd vs mxdg
80 vars	0.40^{*}	0.68^{+}	0.20
40 vars	0.45^{*}	0.83^{+}	0.63^{+}
25 vars	0.07	0.67^{+}	0.26

Notes. * $p < .05$, $^{+}$ $p < .01$, two-tailed.

For all three problem sets, correlations between search reduction and domain reduction were always higher for the max degree heuristic, sometime appreciably so. For one problem set, the correlation for search with d/fd and domain reduction was not even statistically significant. (In fact, it was close to zero). It is also interesting that in two cases the correlation for search reduction with the two heuristics was non-significant.

If one looks at individual problems, the difference is even clearer. To demonstrate this, five problems were chosen at random from the 80-variable set using digit pairs from a random number table; their search tree sizes are shown in Table 4 for each heuristic after preprocessing with either AC or NSAC.

Table 4. Search with different heuristics following AC and NSAC (individual geovarsat problems)

prob#	d/fd		mx dg	
	AC	NSAC	AC	NSAC
11	4577	331	2881	1938
23	1358	1868	1955	1897
54	4446	187	519	374
93	17,213	15,549	835,133	649,133
96	4955	711	8924	8903

Notes. Individual problems chosen at random from 80-variable problem set.

Here, we see that for each problem, search with max degree shows a modest reduction following NSAC in comparison with AC. In contrast, search with d/fd can show modest improvement, drastic improvement, or even worse performance when NSAC is used rather than AC.

Because of space restrictions, results with RLFAPs will not be discussed in detail. However, the same pattern of differences was found, both for the mean results and the correlational analysis.

5.2 Discussion

These results show that when search involves dynamic ordering heuristics, the effects of preprocessing involve other factors than domain reduction, although that of course plays a part. Both order-of-magnitude improvements and, even more decisively, deterioration in performance are inexplicable on this basis. (Note that neither of these are found with a non-dynamic heuristic like max degree). Moreover, the correlational analysis yields one case where there is almost a zero correlation between search with d/fd and the amount of domain reduction; yet NSAC still leads to improvement in search on average in this case (Tables 1 and 2).

Since there is a close relation between the dynamic character of the heuristic and these 'anomalous' effects, an obvious hypothesis is that domain reductions are serving to guide subsequent search in making good heuritic choices of the next variable to assign a value. The following section gives further evidence that this explanation is the correct one.

6 Further Evidence Using Restricted NSAC Testing

During the course of work on the restricted application of NSAC during preprocessing [13], an effect was found that bears on the present issue. In this study, NSAC was performed according to a scheme in which if a randomly generated number between 0 and 1 was less than some criterion, then NSAC was performed; otherwise, the value was skipped over. Otherwise, the procedure was the same as that shown in Fig. 2.

Figure 3 shows search results for one RLFAP problem over 50 separate runs with each of several criterial values (shown under the abscissa). These are frequency distributions of the number of nodes required (i.e. the search effort) for each run for this problem using the d/fd heuristic. (With this problem, using the d/fd heuristic, search after AC preprocessing stopped upon reaching the one hundred thousand node cutoff, while following full NSAC a solution was found after 2205 nodes).

Note that, with a very low probability of performing NSAC, most runs are very bad (in the 10^4 or 10^5 ranges, 10^5 being the cutoff). Then, for slightly higher probabilities, most runs fall in the 10^2 range. In fact, for p = 0.025 or 0.05, most runs required fewer than 211 nodes (200 being a perfect, retraction-free run). This is much better than after full NSAC. For still higher probabilities, there is a shift so that there are increasingly more frequent runs requiring 10^3 nodes.

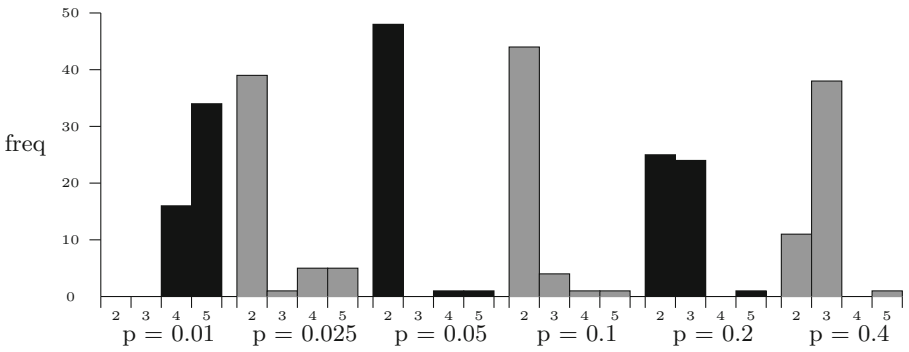


Fig. 3. Frequency histograms for six probabilities of performing NSAC on any given value. RLFAP problem # 1. Log scale on abscissa, expressed as characteristics of the logarithm; e.g. 2 is for total search nodes between 100 and 999, inclusive. Total range is from 100 to over 100 thousand. Successive distributions coloured black or gray to distinguish them.

Again, we see effects on search performance that bear no relation to the degree of problem reduction, which increases monotonically as p increases. Although the full explanation has not yet been worked out, the most reasonable (general) hypothesis is that for this problem, a small proportion of NSAC-based deletions guides the dynamic search heuristic very effectively on the majority of runs.

7 Related Work

Early in the 1990s, there were reports of occasional anomalies in search after preprocessing. In one case this concerned the forward checking algorithm, where backtrack search is interleaved with a more limited form of consistency checking than the full arc consistency that occurs with MAC. When forward checking is used with the minimum domain heuristic (another dynamic variable ordering heuristic), AC prior to search can sometimes result in much worse performance than forward checking alone [7]. Perhaps because this effect was not observed with MAC, this observation has been disregarded for the most part. However, it seems to be another case of domain reduction during preprocessing affecting the choices made by a dynamic variable ordering heuristic.

8 Modelling Preprocessing Effects on Search

The basic finding here is that domain reductions caused by preprocessing affect heuristics for choosing variables made during search when the latter take domain size into consideration. Most often this results in major improvements, but it can also be detrimental. What this means in effect is that preprocessing allows information to be communicated to the search process that, in turn, allows the latter to make better selections from among a set of alternatives.

If there is an information channel, then it would be helpful to characterize the amount of information transmitted, or at least the maximum number of alternatives that can be encoded. Since in this case we are considering the effects of domain reduction, then the maximum number of distinguishable alternatives is $O(d_{max})$, where d_{max} is the maximum domain size in the original problem. (Note that the alternatives we are considering here are different domain sizes).

However, there are two aspects to this. Most simply, the information from preprocessing allows the heuristic to distinguish among choices (variables). In this respect, there is no noise, i.e. no confusion between alternatives. But in the present situation, choices have implications, or further effects, and what we really want are choices that reduce search effort. If we consider an optimal search order as one that results in the smallest possible search tree among all possible orderings, then the information may allow search to better approximate that optimal ordering. In this respect, the information channel is 'noisy', in that a given 'signal' can fail to improve search efficiency or even lead to greater search effort rather than less.

If we consider not only variable but also value selection, we may be able to incorporate domain reduction into the same framework, since preprocessing is informing the search process that certain values should not be tried. In this case, the alternative signals could be considered the possible variable-value pairs, rather than the possible variables.

9 Summary and Conclusions

In this paper we have demonstrated that the classical view of preprocessing in the context of constraint satisfaction search is inadequate, and we have sketched out a more general model of the relation between preprocessing and subsequent search that appears to encompass all of the effects of the former on the latter. This more general model may, in turn, allow us to design and deploy preprocessing algorithms more intelligently, in order to avoid the pitfalls that can arise from insufficient information or even misinformation in the form of misleading signals to decision making processes that occur during search.

Although differences observed in these experiments may have been partly the result of selecting harder problems, this does not affect the basic conclusions regarding the effects of preprocessing on solution search. In fact, it is likely that differences related to preprocessing will be greater when problems are more difficult simply because of a ceiling effect. Hence, using harder problems to demonstrate preprocessing effects is probably a good research strategy. However, since improvement in search due to dynamic variable ordering heuristics is a general phenomenon, the differences observed here should occur regardless of the overall difficulty of the problems.

It seems quite likely that the information transmission conception outlined in this paper has wider application than constraint satisfaction search. Presumably it applies whenever there are heuristic decisions to be made by an algorithmic process and where there is the possibility of altering the problem in order to make the search for a solution more efficient. However, such possible extensions remain for future work.

References

1. Bessière, C.: Constraint propagation. In: Handbook of Constraint Programming, chap. 3, pp. 29–83. Elsevier (2006)
2. Bessière, C., Cardon, S., Debruyne, R., Lecoutre, C.: Efficient algorithms for singleton arc consistency. Constraints **16**, 25–53 (2011)
3. Debruyne, R., Bessière, C.: Some practicable filtering techniques for the constraint satisfaction problem. In: Fifteenth International Joint Conference on Artifcial Intelligence - IJCAI 1997, vol. 1, pp. 412–417. Morgan Kaufmann (1997)
4. Dechter, R.: Constraint Processing. Morgan Kaufmann, Burlington (2003)
5. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Shevron, C.: Optimization by simulated annealing: an experimental evaluation. Part II. Graph coloring and number partitioning. Oper. Res. **39**, 378–406 (1991)

6. Mackworth, A.: Consistency in networks of relations. Artif. Intell. **8**(1), 99–118 (1977)
7. Sabin, D., Freuder, E.: Contradicting conventional wisdom in constraint satisfaction. In: Proceedings of the Eleventh European Conference on Artificial Intelligence-ECAI 1994, pp. 125–129. Wiley (1994)
8. Snedecor, G.W., Cochran, W.G.: Statistical Methods, 7th edn. Iowa State University, Ames (1980)
9. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press (1993)
10. Wallace, R.J.: Light-weight versus heavy-weight algorithms for SAC and neighbourhood SAC. In: Russell, I., Eberle, W. (eds.) Twenty-Eighth International Florida Artificial Intelligence Research Society Conference - FLAIRS-28, pp. 91–96. AAAI Press (2015)
11. Wallace, R.J.: SAC and neighbourhood SAC. AI Commun. **28**, 345–364 (2015)
12. Wallace, Richard J..: Interleaving levels of consistency enforcement for singleton arc consistency in CSPs, with a new best (N)SAC algorithm. In: Baldoni, Matteo, Bandini, Stefania (eds.) AIxIA 2020. LNCS (LNAI), vol. 12414, pp. 301–317. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77091-4_19
13. Wallace, R.J.: Experimental analysis of restricted forms of SAC based reasoning for constraint satisfaction problems: preliminary results. In: Maratea, M., Vallati, M. (eds.) Twenty-ninth RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion - RCRA 2022 (2022)