Symmetry Breaking via Dominance Detection for Lookahead Constraint Solvers

Igor Razgon and Amnon Meisels

Department of Computer Science
Ben-Gurion University of the Negev
Beer-Sheva, 84-105, Israel
{irazgon,am}@cs.bgu.ac.il

Extended Abstract

1 Introduction

A general method for detecting symmetric choice points during search was recently proposed in [3]. The method achieves symmetry breaking by detecting the dominance of search subspaces and eliminating the need to explore them.

Let us sketch the principle of this method applied to a search algorithm returning the first solution found. Assume that a search algorithm currently explores some node P of the search tree. Let S be the remaining search space induced by P [1]. Let P' be some node of the search tree that has already been rejected by the algorithm and S' be the search space induced by P'. If $S \subseteq S'$ than P can be rejected without further search in S. We say that P' dominates P.

The essence of *dominance checking* methods is that before starting search over the search space S induced by the current node P, the algorithm checks whether P is dominated by some previously considered node and if so, P is rejected without further search.

A search algorithm that explores different value choices on variables can potentially eliminate values that are dominated by values

[1] when we say "induced", we mean that S is obtained as the result of application of some procedure maintaining consistency with the current node of the search tree

of the same variable that were already rejected. The present paper proposes a complete search algorithm for constraint satisfaction problems (CSPs) that utilizes lookahead techniques for dominance detection. Lookahead techniques like forward checking (FC) check constraints with all unassigned variables and produce domains of values that change dynamically during search [9]. Our proposed algorithm utilizes the dynamic nature of domains of values of future variables to detect dominance and thereby eliminate values that are dominated by former (rejected) values.

Consider a backtrack based constraint solver, that utilizes some lookahead procedure *Proc*. This means that the solver runs *Proc* after every assignment of a value to a variable to achieve some level of consistency for the remaining constraint network of unassigned variables. In the case of forward checking *Proc* achieves node-consistency, in the case of MAC it achieves arc-consistency [9].

Let P be the current partial solution at some moment of the execution of the solver. Denote by F_P the set of domain values of all unassigned variables for the partial solution P, after application of *Proc*. Let us call F_P the set of *future values* of P.

Let V be the current variable, val_1 and val_2 be values from the current domain of V, and denote by $\langle V, val \rangle$ the assignment of val to V. The dominance detection property can be formulated as follows: if $P \cup \{\langle V, val_1 \rangle\}$ has been rejected and
$F_{P \cup \{\langle V, val_2 \rangle\}} \subseteq F_{P \cup \{\langle V, val_1 \rangle\}}$ then the assignment $\langle V, val_2 \rangle$ can be rejected without further search. In other words, if the assignment $\langle V, val_1 \rangle$ *dominates* the assignment $\langle V, val_2 \rangle$ given the current partial solution P and $\langle V, val_1 \rangle$ is inconsistent, then $\langle V, val_2 \rangle$ is also inconsistent.

Section 2 presents the proposed method of dominance checking. For constraint networks of n variables and maximal domain size of m, the proposed method has an $O(nm^2)$ running time and needs memory bounded by $O(n^2m^2)$.

For dominance detection on a CSP search algorithm the order in which values are checked is important. When values are ordered in decreasing order of future compatible domain sizes, the checking of dominance becomes easier. A new dominance relation that orders values is presented in Section 2. The *minimal dominating set*

construction heuristic requires $O(tm + m \log m)$ time where t is the complexity of the lookahead procedure of the solver.

Section 3 presents preliminary experimental results of FC with minimal dominating set construction for random CSP's. The results demonstrate that the proposed approach is promising. Further developments of the proposed algorithm are discussed.

2 Dominance Detection for Domain Shrinking

2.1 A Dominance Checking for CSP's

Definition 1. *Let P be the current partial solution. Let val be a value from the domain of some unassigned variable. If $val \in F_P$, we say that val is consistent with P.*

Given this definition, we can reformulate the dominance definition as follows.

Definition 2. *Let P, V be the current partial solution and the current variable to be assigned. An assignment $\langle V, val_1 \rangle$ is dominated by assignment $\langle V, val_2 \rangle$ given P if every domain value of an unassigned variable consistent with $P \cup \{\langle V, val_1 \rangle\}$ is also consistent with $P \cup \{\langle V, val_2 \rangle\}$.*

The last definition provides a simple method for dominance checking on constraint networks. Let $\{val_1, \ldots, val_k\}$ be the already rejected subset of the current domain of the current variable. To perform the dominance checking for val_{k+1} we simply check whether val_{k+1} is dominated by either $\langle V, val_1 \rangle, \langle V, val_2 \rangle, ..., \langle V, val_k \rangle$ given P.

The number of values consistent with $P \cup \{\langle V, val_{k+1} \rangle\}$ is $O(nm)$. The number of previously rejected values is $O(m)$. Assuming that the complexity of a consistency check is $O(x)$ we obtain the complexity of the method as $O(nm^2x)$. To decrease the complexity of the proposed method we have to decrease the complexity of the consistency check.

In order to implement an $O(1)$ consistency check method one needs to maintain a data structure for the current partial solution. One possibility is to use a *consistency array* C_P. Let val and val' be a value of the current variable and a value of some other unassigned

variable respectively. Then, $C_P[val][val'] = 1$ if $P \cup \{\langle V, val\rangle\}$ is consistent with val', otherwise $C_P[val][val'] = 0$. To check whether $\langle V, val\rangle$ is dominated by some previous assignment of V, it has to be compared with C_P entries for previous assignments. All entries are computed during the execution of the lookahead procedure at no additional cost. Given this data structure a consistency check is simply reading an entry from the array and takes time $O(1)$.

Note that C_P is deleted only when C_P becomes irrelevant (some of its assignments are deleted or changed). This condition implies that a number of consistency arrays are maintained in the memory simultaneously.

Assume $\{\langle V_1, val_1\rangle, \langle V_2, val_2\rangle, ..., \langle V_k, val_k\rangle\}$ is the partial solution. Then the following consistency arrays corresponding to the growing subsets of the partial solution are maintained simultaneously in memory: $C_{\{\langle V_1,val_1\rangle\}}, C_{\{\langle V_1,val_1\rangle,\langle V_2,val_2\rangle\}}, C_{\{\langle V_1,val_1\rangle,...,\langle V_k,val_k\rangle\}}$. That is, there are $O(n)$ consistency arrays in the general case. Every one of these arrays takes $O(nm^2)$ memory. So all arrays together take $O(n^2m^2)$ memory.

The proposed dominance checking method needs $O(nm^2)$ time and $O(n^2m^2)$ memory.

2.2 The Minimal Dominating Set

The order of assignment of domain values may greatly affect the effectiveness of the dominance detection methods. Let $\{val_1, val_2, \ldots, val_m\}$ be the order of assignments of values to the current variable V enumerated in the order of their consideration.

Assume that $\langle V, val_i\rangle$ is dominated by $\langle V, val_{i+1}\rangle$ for all i, $1 \leq i < m$. This means that in spite of the fact that the domain contains a large number of symmetries they will not be discovered by dominance checking methods. In this subsection we demonstrate how it is possible to overcome this drawback.

Let us define the notion of a *dominating subset* of a domain. Every value in a domain either belongs to the dominating subset or is dominated by one of its members. A *minimal dominating subset* is a dominating subset of a domain of the minimal size.

Lemma 1. *Let P be the current partial solution. Let V be the current variable, and $\{val_1, val_2, \ldots, val_m\}$ be the set of values of the current domain of V enumerated in the order of their instantiation.*

If $|F_{P\cup\{\langle V,val_i\rangle\}}| \geq |F_{P\cup\{\langle V,val_{i+1}\rangle\}}|$ for all $i = 1, \ldots, m-1$, then any dominance checking method (say, the one presented in the previous subsection) will explore a minimal dominating subset of V

Roughly speaking, to consider a dominating subset of minimal size we have to instantiate values in a decreasing order of their sizes of future subspaces. Note that given such an order, no value can dominate a value considered before it unless the sizes of their sets of future values are equal.

Proof.

Assume that the lemma does not hold. Let DS be the dominating subset explored by the proposed method. Let also DS_{min} be a minimal dominating subset of the domain of the current variable and $|DS_{min}| < |DS|$.

Order the values of DS and DS_{min} in the order proposed in the lemma and consider their maximal common prefix. Let val^{DS} and $val^{DS_{min}}$ be the values following this prefix in DS and DS_{min} respectively.

Consider three possible cases:

1. $|F_{P\cup\{\langle V,val^{DS}\rangle\}}| < |F_{P\cup\{\langle V,val^{DS_{min}}\rangle\}}|$
2. $|F_{P\cup\{\langle V,val^{DS}\rangle\}}| > |F_{P\cup\{\langle V,val^{DS_{min}}\rangle\}}|$
3. $|F_{P\cup\{\langle V,val^{DS}\rangle\}}| = |F_{P\cup\{\langle V,val^{DS_{min}}\rangle\}}|$

In the first case, $val^{DS_{min}}$ has been considered already by the proposed method and has not been included in the dominating subset DS. We deduce that $val^{DS_{min}}$ is dominated by a value from the common prefix of DS and DS_{min}. That is, DS_{min} is not minimal which contradicts our initial assumption.

In the second case val^{DS} is not included in DS_{min}. But val^{DS} is not dominated by any value from the common prefix of the considered sets, because otherwise it would not have been included in DS. In addition, val^{DS} is not dominated by any other value in DS_{min}, because all other values have sizes of sets of future values less than val^{DS} (by our ordering condition). So val^{DS} does not belong to DS_{min} and is not dominated by any value from it. Therefore,

DS_{min} is not a dominating subset of the considered domain which contradicts our assumption.

In the third case two subcases are possible:

- $F_{P\cup\{\langle V,val^{DS}\rangle\}} = F_{P\cup\{\langle V,val^{DS_{min}}\rangle\}}$
- $F_{P\cup\{\langle V,val^{DS}\rangle\}} \neq F_{P\cup\{\langle V,val^{DS_{min}}\rangle\}}$

In the first subcase we can replace $val^{DS_{min}}$ by val^{DS} in DS_{min} contradicting our assumption about the maximality of the common prefix. In the second subcase DS_{min} should contain some value val' such that $F_{P\cup\{\langle V,val^{DS}\rangle\}} = F_{P\cup\{\langle V,val'\rangle\}}$. We can change the order of val' and $val^{DS_{min}}$ without violating our ordering condition. In this way we reduce the second subcase to the first one considered above. □

To order values of the considered domain by the presented lemma, we have to compute $|F_{P\cup\{\langle V,val\rangle\}}|$ for every value of the current domain of the current variable V. Therefore, we apply the lookahead procedure $Proc$, $O(m)$ times. If the complexity of $Proc$ is $O(y)$, the complexity of this method is $O(ym + m\log m)$.

Note, that the ordering of domain values of the current variables in the proposed order is a good search heuristic even without consideration of symmetry breaking [2].

3 Realization and Testing

To check the effectiveness of the proposed method, we implemented two versions of FC. The first version has the value ordering proposed in Lemma 1. The second version performs dominance checking in addition to value ordering. Both algorithms use the fail-first variable ordering heuristic [5]. We compared these two algorithms on a set of randomly generated problems characterized by their constraints density p_1 and tightness p_2 [6].

Our preliminary results show that instances of randomly generated problems with a large number of symmetries are mainly concentrated in regions with either low density or low tightness. More formally, these regions are characterized by either $p_1 \leq 0.3$ or ($p_2 \leq 0.3$ and $0.3 < p_1 < 0.8$). For difficult problems in these regions, the FC version with dominance detection outperforms the version without it. For easy problems (without backtracks), the dominance detection

method is not applied at all, that is the computation effort is the same for both these algorithms. The other set of constraint problem is characterized by a small number of symmetries or by their total absence. Dominance detection increases running time of the solver when applied to these problems. But this additional time is not great, because the proposed dominance checking method frequently returns a negative answer in an early stage of its work

Selected results of our experiments are demonstrated in Table 1 and illustrate our observations. All problems have 20 variables and 12 values in all domains. The first 2 columns of Table 1 give the density and tightness of problem instances. The 3rd column presents the number of constrain checks (CCs) for the proposed method and the 4th column presents the CCs performed by FC with just ordering of values. The results in the table are the averages of 10 runs for the same problem parameters.

p_1	p_2	symm-cc	unsymm-cc
0.2	0.1	28595	28595
0.2	0.2	23056	23056
0.2	0.3	17980	17980
0.2	0.4	14081	14081
0.2	0.5	10713	10528
0.2	0.6	29817	33661
0.2	0.7	38402	54512
0.2	0.8	11782	13788
0.2	0.9	5694	4970

Table 1. Comparison of FC's with and without Domiance Checking

These results in Table 1 can be divided into the following three groups

1. Both versions do not execute backtracks. Therefore, the dominance checking mechanism is not applied at all (first four rows of the table).
2. The version with dominance checking outperforms the version without it, by discovering dominances. This group of results occurs for instances with p_2 varying from 0.6 to 0.8.
3. The dominance detection does not succeed to reduce the computational effort ($p_2 = 0.5, 0.9$). Note, that in this case the addi-

tional cost of dominance checking is small relatively to the benefit we gain for the instances of the second group

Based on our experiments, we conclude that domain shrinking methods based on construction of minimal dominating subsets are promising for a subregion of $\langle p_1, p_2 \rangle$ and need further investigation. These methods may be useful for CSP's of large size with low density, which are frequently difficult to solve by complete algorithms [8].

The most important step of the further investigation is to find a successful variable ordering heuristic. Ideally, such a heuristic should be successful for search as well as for construction of small dominating subsets. Heuristics based on constrainedness evaluation [1, 4, 7] seem to be a good choice for this purpose. According to our preliminary experiments, symmetries are concentrated in regions with low constrainedness.

Yet another direction in the development of the proposed method is to find a more effective algorithm for ordering of values in the proposed form of Lemma 1. A naive method applying the lookahead procedure m times can be replaced by a more sophisticated approach.

References

1. C. Bessiere, A. Chmeiss, L. Sais *Neightborhood-Based Variable Ordering Heuristics for the Constraint Satisfaction Problem*, Proceedings of CP2001, pp.565-570, Paphos, 2001.
2. D. Frost, R. Dechter, *Look-ahead value ordering for constraint satisfaction problems*, Proceedings of IJCAI-95, pp. 572-578, Montreal, Canada, August 1995.
3. T.Fahle, S.Schamberger, M. Sellmann *Symmetry Breaking*, Proceedings of CP2001, pp. 93-107, Paphos, 2001.
4. I. P. Gent, E. MacIntyre, P. Prosser, T. Walsh *The Constrainedness of Search*, AAAI/IAAI96, Vol. 1, pp. = 246-252, 1996.
5. R.M. Haralick, G.L.Eliott *Increasing tree search efficiency for constraint satisfaction problems*,Artificial Intelligence, 14:263–313, 1980.
6. P. Prosser *Binary constraint satisfaction problems: some are harder than others* , Proc. ECAI-94, pp.95-99, 1994.
7. I. P. Gent, E. MacIntyre, P. Prosser, B.M. Smith, T. Walsh *An Empirical Study of Dynamic Variable Ordering Heuristics for the Constraint Satisfaction Problem* Proccedings of CP96, pp. 179-193, 1996.
8. B. Smith, S. Grant, *Where the Exceptionally Hard Problems Are*, CP95 Workshop on Really Hard Problems Cassis, September 1995.
9. E.Tsang *Foundations of Constraint Satisfaction*, Academic Press Limited, 1993,