Upward exploitation of constraint propagation in hybrid constraint satisfaction search.

Steven Woods
Working Document Only

Abstract

One method currently proposed for the solution of constraint satisfaction problems (CSPs) is the coupling of traditional breadth-first or depth-first search through the space of all possible variable assignments with some degree of constraint propagation before or during search. Hybrid strategies interleave the work spent on search and the work spent reducing variable domains through some kind of constraint propagation algorithm(s).

Hybrids have been constructed based upon well known search strategies including traditional backtracking, forward checking, backjumping, and backmarking. All of the hybrids utilizing these algorithms have been constructed typically by attempting to achieve some degree of arc consistency during search. For example, forward checking attempts to revise or reduce the domains of remaining variables based upon the most recent instantiated variable value.

It is quite often the case that the work spent in performing constraint propagation is duplicated at various points in the hybrid search space. Essentially identical domain values at disparate points in the search space are removed as a result of identical constraint checks being performed. We present here a brief description of a hybrid-based search strategy that can allow for the sharing of constraint propagation work under certain circumstances. In addition, this strategy can be easily encapsulated in a depth-first structure so as to minimize space requirements of search.

1 Introduction

This work is based on ongoing research into constraint satisfaction search and hybrid application of search and constraint propagation techniques. This work is summarized in [1] and [2].

2 Upward Sharing

The general structure of hybrid search is selection of a variable, assignment of a domain value for that variable (instantiation), some constraint propagation intended to simplify the remaining problem, and repeat. This approach can be nicely formed as a complete depth first strategy which finds some solution as quickly as possible, and yet which is capable of finding all solutions. Forward checking for instance operates precisely in this manner, where the hybrid aspect appears in that all other variables are revised or have their domains "reviewed" with respect to the latest assignment, and any values no longer satisfying a constraint between a particular variable and the newly instantiated one are discarded.

If we were to take the decision point for the assignment to some variable, call it A, and consider the various subtrees resulting from different domain value selections for A, we can notice that the resulting problems under each subtree are quite similar to the parent (before A) problem. In fact, they differ only in that A no longer has a domain, but rather a single value. Each of the A subtrees differs only in the choice of domain value for A.

Figure 1 shows a problem with four variables A, B, C and D in which A has been selected for instantiation, and in fact the value assigned to A is 1 in this subtree. If A has a domain of 1, 2 and 3 in the parent, then there will conceivably be three siblings of this nature. In particular, the subgraph consisting of nodes (and domains) B, C, and D plus their associated constraints C3, C4 and C5 will be identical in each of the three subtrees. Further, this subgraph will exist in earlier ancestors of the parent since in our search method we never add variables or domain values as we progressively instantiate variables.

Now, if we were to perform some constraint propagation in which the domain of B is reduced based upon the newly selected value of A, we can say B's domain has become "A-contaminated". This is illustrated in Figure 2. B's domain has been reduced based upon our new "conjecture" that A's value is 1. Similarly, for any other variable's domain constrained by A, if that domain is reduced based upon

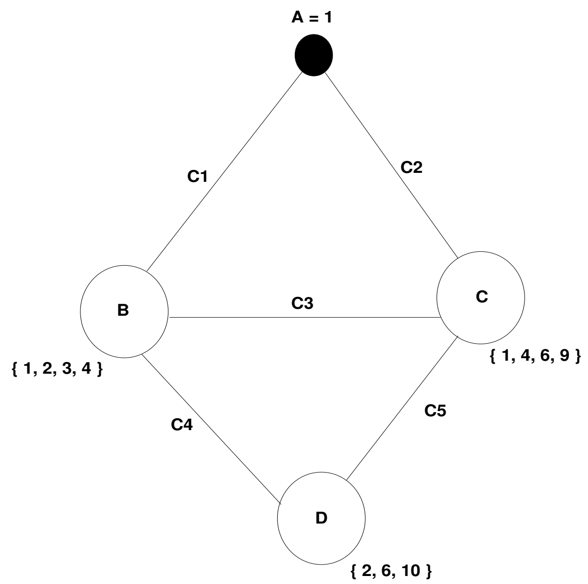

Figure 1: Partially Instantiated Constraint Graph.

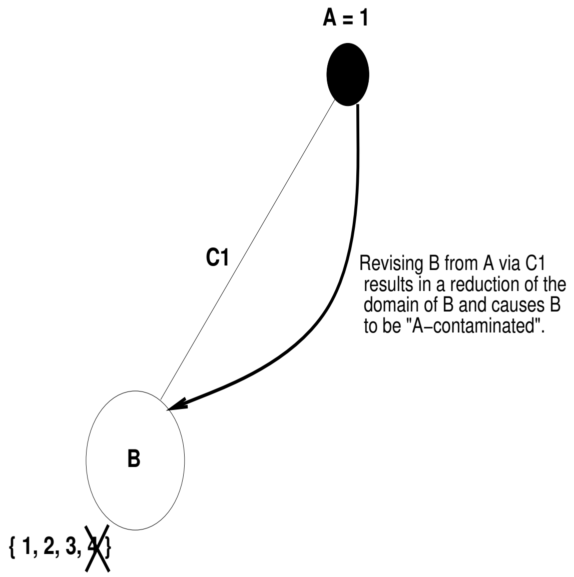

Figure 2: Contamination trickling through graph.

an instantiation of A that domain is considered to be "A-contaminated". Now, if D's domain were to be reduced based upon either B or C (each A-contaminated), then D's domain would be said to be A-contaminated also. The basic intuition of contamination in this context is to allow us to be aware of what variable assignment conjectures have been used in our arrival at a particular domain for each variable.

Assume that the order of instantiation in our example is determined prior to any search. For instance, we will say that the order of variable instantiation is always A, B, C and finally D. Therefore, as we progress down our instantiation order and search space, A is instantiated first, and in all subtrees after that instantiation, the assignment of A is a "fact" that can be used in reasoning about other domain values. If we look at the search space as a whole, the variable commitment point spawns child subtrees, where in each subtree the value of that particular fact differs. The parent of these children has no "given" fact for A, only a set of possible domain values.

The concept of contamination allows us to keep track of which "facts" have been used in reducing some possible domain assignments. Since these assignments occur top-down and one at a time, we can see that the point in time where A is assigned always occurs before B's assignment, and further that at any point in the space after B's assignment, A must be assigned to some particular value also. In fact, when an arbitrary variable has just been assigned a particular domain value, all variables earlier in the order have already been assigned, and all variables earlier in the order have a set of possible values.

If, in a particular instance of constraint propagation during search (say after assigning A=1), we reduced the domain of some variable (say D) based upon the current domain of some other unbound variable (say C) and the constraint set between C and D, then (assuming C is not contaminated) we should be able to "share" this reduction throughout our search space. Figure 3 details this reduction. The reasoning that resulted in a reduction of D's domain was not dependent on any instantiation explicitly or implicitly, and this reduction could have as easily been done prior to the start of any search.

In a backtracking paradigm, sharing this information can be thought of as propagating the domain reduction upwards in the search space. In this particular example, we have just assigned A=1, and the parent is the original (top) problem formulation without assignments. If we propagate the D reduction to each parent up to and including the top prob-

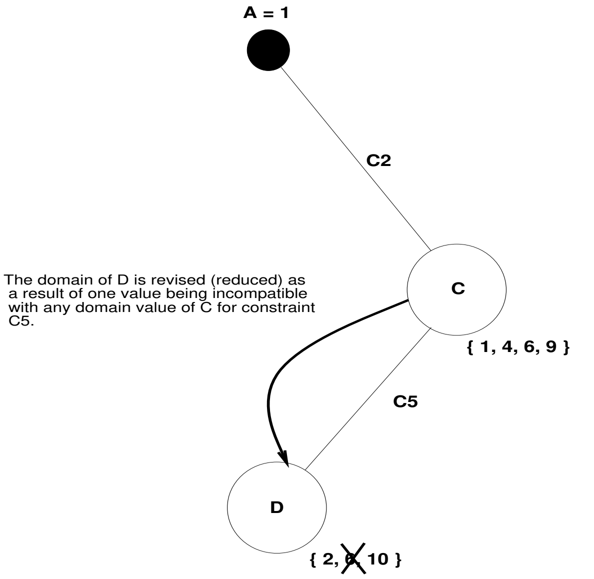

Figure 3: Constraint propagation between unbound variables.

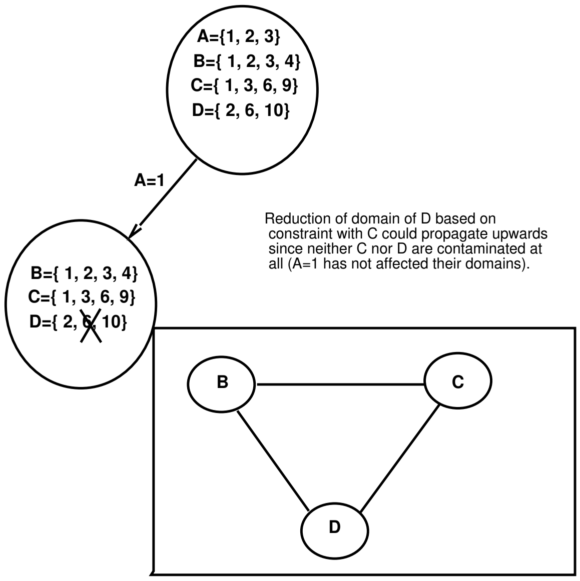

Figure 4: Problem space before upward propagation.

lem, any further successor creations through other instantiations will carry the reduction. The D reduction will therefore have been made locally in the ancestor tree of the node where the constraint propagation occurred, however, it will take global affect as search progresses.

The key to this upward propagation is in the lack of contamination of the domain being propagated. We cannot reduce a domain based upon some variable assignment (directly or indirectly), and propagate this result above the "highest" variable which has taken part in the "contamination". Figure 4 and 5 show the upward propagation to the highest level. Figure 5 shows how this revision of D's domain is later used in a different instantiation of A.

For example consider Figure 6. If we instantiated A=1 and B=2, then reduce the domain of D based only on C where neither D nor C have been contaminated by either A or B, then we may propagate the new D domain to all parents up to and including the whole problem. All subsequent children resulting from backtracking will include this reduction. If we instantiated A=1, and B=3, then reduced the domain of C based on A's value, then C has become A-contaminated. C's new domain may only propagate up to the level where A was instantiated. All subsequent children of the node where A was instantiated will now reflect the reduced C domain. We must be careful since if we attempt to propagate a contaminated reduction past its point of instantiation, we are essentially carrying inferences based

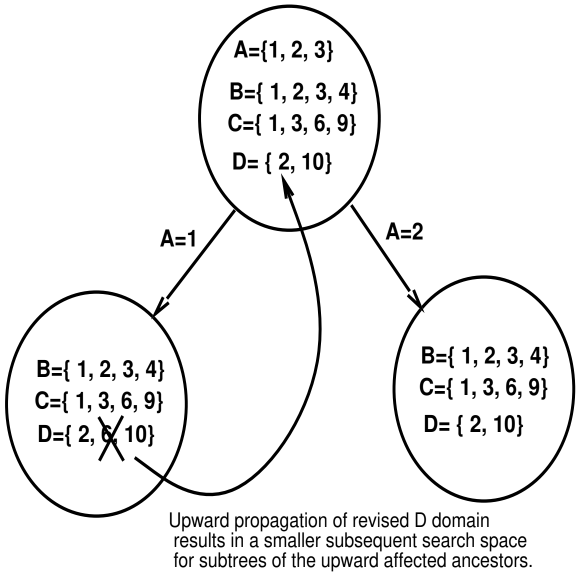

Figure 5: Problem space after upward constraint.

upon assignments past where they occur, and thus would force a reduction in a domain based upon an erroneous assumption, and completeness will have been sacrificed.

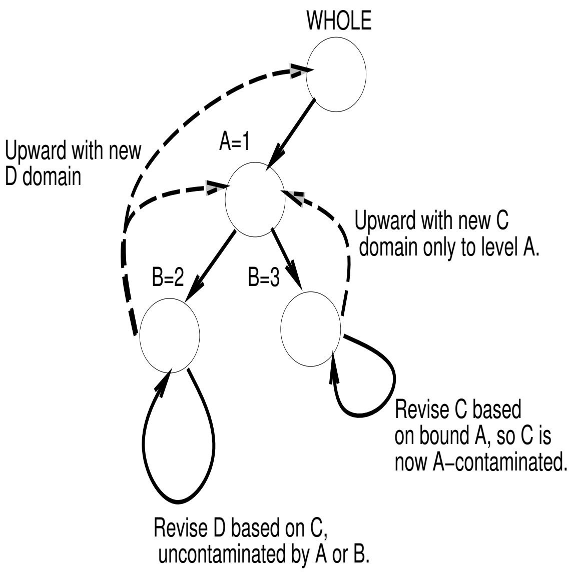

Figure 6: Problem space after upward constraint.

3 Implications

As a result of propagating this constraints upward in the depth first path where possible, we can reduce the size of the search space in subsequent instantiation paths. However, we should note that the work we are sharing could have been performed as part of more elaborate initial consistency checking prior to the start of search. The advantage of this approach is that we may be uncertain initially as to how much effort to spend on preliminary problem reduction. If some factors appear or are noticed during hybrid search, we can take advantage of these conditions locally via application of further consistency propagation, and achieve global improvement for only the cost of replacing ancestor domains where reduction occurs and where the domain contamination determines upward propagation is permissible.

This approach will result in savings only if:

1. A constraint propagation algorithm(s) is utilized which at some point successfully revises some domain based upon constraints between variables with uninstantiated variables. For this point, the tighter the constraints the more likely it is that at least one value will be removed. If no values are removed, there is nothing to propagate.
2. In at least where the above (1) is successful, we are able to propagate the reduction at least one level above the current instantiated variable. For instance, if we have just instantiated B, then in order for a revision to be useful, we must propagate it to at least A so that A's later children (the current node's siblings), reflect this improvement. Of course, the more levels that a removal is propagated, the greater the potential for savings as the scope of the removal is widened.

Please note also :

1. This method evolved from thinking about how to share constraint propagation across the search space in an A or breadth-first search. Further thought will have to be given to this original intent to see to what extent individual node constraint propagtion can be shared in a wider scope than simply depth-first ancestors.
2. Note also that since upward propagations occur alongs the ancestor path of the current node, that the upward propagation of several domain reductions can combine combinatorially within ancestor nodes to produce larger reduction in the size of the search space expanded following upward propagation. These reductions should form a multiplicative reduction in the search required.
3. Search strategies that do not pre-order the variable instantiations can still benefit from this upward ancestor propagation strategy. Since we are careful to keep track of the "contamination" level of each domain during search, we never propagate an erroneous inference above the level in the space where the "trigger instantiation" was first made.
4. This method seems to relate to propagating hypothetical inferences. If identical or highly similar work has been done, we have been unable to locate it.
5. At a later time we will test this constraint satisfaction hybrid algorithm empirically.

Acknowledgements

Thank you to Luc Lamontagne and Denis McGonigal of DREV for comments on this ongoing work.

References

[1] Steven Woods. Interactive recognition of spatially defined model deployment templates. Research memorandum, Defence Research Es-

tablishment Valcartier, Dept. of Defence, Courcelette, Quebec CANADA, August 1993.

[2] Steven Woods. A method of interactive recognition of spatially defined model deployment templates using abstraction. to appear, DND workshop on advanced technologies in Knowledge Based Systems and Robotics, August 1993.