Splitting the atom: A new approach to Neighbourhood Interchangeability in Constraint Satisfaction Problems

James Bowen and **Chavalit Likitvivatanavong**
Cork Constraint Computation Centre, University College Cork, Ireland
{j.bowen, chavalit}@4c.ucc.ie

Abstract

We investigate interchangeability of values in CSPs, based on an approach where a single value in the domain of a variable can be treated as a combination of "sub-values". An algorithm for removing overlapping sub-values is presented. The resulting CSPs take less time to find all solutions and yield a more compactly-representable, but equivalent, solution set. Experimental results show that, especially in loose problems with large numbers of solutions, dramatic savings in search cost are achieved.

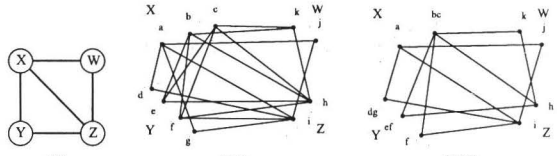
1 Introduction

While most CSP algorithm research is based on the assumption that only one solution to a CSP need be found, the central idea in this paper is motivated by a requirement of certain interactive constraint-based applications: sometimes, before making his next interactive decision, the user needs to know *all* solutions to a sub-CSP. Thus, we need an approach which will reduce the search cost of finding all solutions to a (sub-)CSP and which will enable a more compact representation of the complete solution set. Neighbourhood Interchangeability (NI) [Freuder, 1991] of domain values suggests itself, since merging NI values in a variable domain reduces both thrashing during search and the complexity of enumerating the solution set. Without loss of generality, we restrict our attention here to *binary* CSPs, where the constraints involve two variables.

Our idea is based on a new approach to NI. Normally, each domain value is treated as atomic. Our idea is to "split the atom" – a domain value can be split into several "sub-values" so that interchangeable fragments from other values can then be merged to avoid duplicate search effort during the solving process.

When the values in a domain have been split into fragments, interchangeable fragments of different values can be merged. Figure (i) depicts a constraint network while (ii) shows the microstructure. Figure (iii) shows the result of splitting and merging value fragments, resulting in smaller domains for X and Y.

The approach subsumes NI, as can be seen in X where b and c, which are NI, are merged; similarly, d and g in Y



are merged because they, also, are NI. That the approach surpasses NI can be seen by considering values e and f in Y. Values e and f are not NI because, while both e and f support (b,h) and (c,h), f also supports (b,i) and (c,i). However, we can split f into two fragments, one of which supports (b,h) and (c,h), while the other supports (b,i) and (c,i). The first fragment can be merged with e and the second can remain isolated.

2 Background and Definitions

We will use a cross-product representation (CPR) [Hubbe and Freuder, 1992] to denote sets of solutions to the star-graph CSPs that we must consider when transforming variable domains. For example, consider the star-graph centred at Y in Figure (ii). The set of solutions to this star-graph in which the centre variable, Y, has the value f can be represented by the following CPR: $\{b,c\}\times\{\mathbf{f}\}\times\{h,i\}$.

We define an operator, called "commerge" (commonality extraction and merge), on two CPR sets as follows. The operator extracts, from the two CPR sets, those solutions which have common values for the leaf variables and merges the values for the centre variable. To compute the commerge of two CPR sets, the sets of values for the leaf variables are combined using normal set intersection but the sets of values for the centre variable are combined using normal set union. We denote the commerge operator as $\odot$. Thus, for example, $\{a,b\}\times\{\mathbf{x}\}\times\{c\} \odot \{a\}\times\{\mathbf{y}\}\times\{c,d\}$ is $\{a\}\times\{\mathbf{x,y}\}\times\{c\}$.

We define the "communion" of two CPRs as follows. Two CPRs can be communioned only if the CPRs differ from each other in at most one of the cross-multiplied sets, which cannot be the set for the centre variable. In the communion, all the common cross-multiplied sets are unchanged, but the differing leaf sets are subjected to ordinary set union. We denote the communion operator as $\oplus$. For example, the two CPRs $\{a\}\times\{\mathbf{b}\}\times\{c\}$ and $\{a\}\times\{\mathbf{b}\}\times\{d\}$ can be communioned because they differ only in the right-most subsets which do not

correspond to the centre variable; the communion of these two CPRs is $\{a\}\times\{\mathbf{b}\}\times\{c,d\}$.

We also define a new form of subtraction, the "comminus" of two CPRs, as follows. One CPR can be comminused from another only if they have a common set for the centre variable. Given that provision, the comminus of two CPRs, denoted by $\ominus$, is the same as subtraction. Note that the operator can result in a *set* of CPRs. Consider, for example, the result of subtracting $\{a\}\times\{\mathbf{x}\}\times\{c\}$ (which contains only one solution to a star-graph) from $\{a,b\}\times\{\mathbf{x}\}\times\{c,d\}$ (which contains four solutions). The difference between these two CPRs contains three solutions: *a-x-d*, *b-x-c*, *b-x-d*. In CPR form, this set of three solutions needs two CPRs. That is, $\{a,b\}\times\{\mathbf{x}\}\times\{c,d\} \ominus \{a\}\times\{\mathbf{x}\}\times\{c\}$ is $\{\{a\}\times\{\mathbf{x}\}\times\{d\}, \{b\}\times\{\mathbf{x}\}\times\{c,d\}\}$ or, alternatively, $\{\{a,b\}\times\{\mathbf{x}\}\times\{d\}, \{b\}\times\{\mathbf{x}\}\times\{c\}\}$.

3 Transforming CSPs

The procedure for transforming the domain of a single variable is stated in pseudocode below.

Procedure `transform`(V)

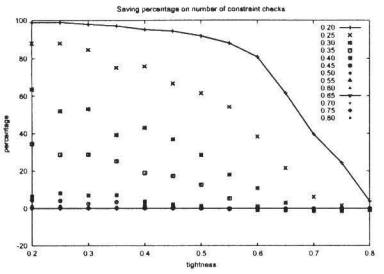
```
queue ← all CPR centered in V;
result ← ∅;
while queue ≠ ∅ do
    Select and delete any S from queue ;
    match ← false ;
    while (not match) and (next T in result exists) do
        if S ⊙ T ≠ ∅ then
            match ← true ;
            C ← S ⊙ T;
            RS ← S ⊖ C;
            RT ← T ⊖ C;
            Replace T in result with C and RT;
            Insert RS to queue ;
    if not match then
        Insert S to result
Where possible, communion the CPRs in result ;
Update the domain of V and its connecting constraints;
```

The size of `result` could be very large if there are many fragments of CPRs, making it unlikely that the resulting CPRs can be compressed so that the resulting domain is smaller than the original. Therefore we impose an upper bound on the size of `result`; if `result` exceeds the upper bound, transformation of the variable's domain is aborted. This cutoff heuristic reduces the transformation time, making the algorithm practical even though the worst-case time complexity is exponential.

The overall transformation process involves transforming each variable domain, one by one, in some order. The order in which variables are processed affects the result. As a heuristic, we have chosen never to accept the result of transforming a variable's domain if the result of the transformation is that the number of members in the domain is increased, and choose the variable whose transformation leads to the maximum domain size reduction.

3.1 Convergence

We only need to transform the domain of each variable once due to the following theorem:



Theorem 1 *If, after the domain of variable V_1 has been transformed, the domain of a neighbouring variable, V_2, is also transformed, any subsequent attempt to transform the domain of V_1 will result in no change.*

4 Experimental Results

We tested the algorithm over randomly generated CSPs with eight variables and six values in each domain, varying density from 0.1 to 0.9 with 0.05 increment step, while tightness ranges from 0.1 to 0.8 with the same increment. For each parameter point, the number of constraint checks required to compute all solutions and the number of representations needed to express these solutions are evaluated, averaged over 100 problem instances. The solver is based on MAC-3 with dom/deg variable ordering. The upper bound of the `result` is ten times the size of the variable's domain.

For the random problem generator, we use model B where density and tightness are fraction instead of probability. Since the transformation process requires arc-consistent CSPs, we also ensure that every problem is arc-consistent by linking each value to at least one value for every constraint in order to guarantee support.

The results are shown in the above graph. While the transformation cannot *guarantee* to reduce the number of constraint checks needed to compute all solutions to a CSP, it can produce huge savings. From the graph, we can see the method works best for problems with low density or low tightness, as either case will lead to a lot of CPR combination. In these cases the work needed for transformed CSPs is less than 1% of the work needed for untransformed CSPs. In those problems where the transformation does not produce large savings, its effect is usually harmless. Indeed, in the worst case we have seen in the experimental data, which occurred only at a few data points, the number of extra constraint checks for a transformed CSP is around 1% more than for the untransformed CSP.

References

[Freuder, 1991] Eugene C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of AAAI'91*, pages 227–233, 1991.

[Hubbe and Freuder, 1992] Paul D. Hubbe and Eugene C. Freuder. An efficient cross-product representation of the constraint satisfaction problem search space. In *Proceedings of AAAI'92*, pages 421–427, 1992.