A Disjunctive Decomposition Control Schema for Constraint Satisfaction*

Eugene C. Freuder
Paul D. Hubbe

Department of Computer Science
University of New Hampshire
Durham, NH 03824, U.S.A.
ecf@cs.unh.edu
pdh@cs.unh.edu

Abstract

The paper presents a control schema for constraint satisfaction. Several algorithms, old and new, are formulated as instances of this schema by specifying different methods of problem decomposition. This formulation facilitates description and comparison of the algorithms and suggests directions for further research. A new decomposition method is presented that is virtually guaranteed to reduce problem size, while always retaining at least one of the solutions to the original problem.

1 Introduction

A *solution* to a *constraint satisfaction problem* (*CSP*) is an assignment of a value to each problem variable that satisfies all the *constraints*, or restrictions, on which combinations of variables are permitted. We will focus here on binary CSPs where the constraints involve two variables. The potential values for a variable constitute its *domain*. We will assume finite domains.

We propose a disjunctive divide and conquer control schema for constraint satisfaction. The schema encompasses a wide variety of specific algorithms, including a new one presented here. It facilitates presentation and comparative analysis of these algorithms and suggests new algorithmic possibilities. In particular, the problem decomposition offers new opportunities for ordered search in a space of alternative problems, and for parallel and distributed processing. A specific new decomposition technique is presented that is virtually guaranteed to reduce the number of possible solutions, i.e. the number of different ways to assign a value to each variable, while always retaining at least one of the actual solutions to the original problem.

Our basic CSP algorithm schema can be stated very simply:

```
Decomposition Algorithm Schema:
Place the initial problem on the Agenda
Until Agenda empty:
    Remove a problem P from Agenda
    If P has only instantiated variables
        then Exit with their values
        else
            Decompose P into a set of subproblems {$P_i$}
            Place each non-empty $P_i$ onto the Agenda
Exit with no solution
```

*This material is based upon work supported by the National Science Foundation under Grant No. IRI-9207633.

Initially all variables are *uninstantiated*; the decomposition methods mark variables as *instantiated*. Intuitively, the instantiated variables are the variables for which we have chosen values. Upon exit the cross product of the instantiated variable domains is the set of reported solutions. (This is not necessarily all the solutions, but some of our algorithms will naturally find sets of solutions even while searching for a first solution.) A problem is *empty* if any of its domains is empty.

We impose the following three conditions on $\{P_i\}$:

1. *Soundness*: Any solution to any P_i is a solution to P.
2. *Termination*: Each of the $\{P_i\}$ is smaller than P. (*Problem size* can be measured as the product of the domain sizes for each variable, i.e. the number of combinations of values that could be generated as potential solutions.)
3. *Semi-completeness*: If there is a solution to P, then there will be a solution to at least one of the $\{P_i\}$.

This *disjunctive decomposition* breaks a problem into subproblems in a manner that guarantees that the decomposition algorithm schema will produce a solution to solvable problems and terminate without a solution when none exists. Notice that if we are only looking for one solution, we do *not* need to require that every solution to P will be a solution to some P_i. (*Conjunctive decompositions* break a problem into subproblems such that all the subproblems must be solved, and the solutions must fit together properly, for the original problem to be solved.)

This schema immediately suggests two avenues of exploration:

1. How is the decomposition performed?
2. How is the agenda organized?

Different answers to these questions produce a family of divergent algorithms.

Notice that if the agenda is maintained as a stack we have a form of depth-first search, which does not need to present a serious space problem. Stack size requirements are O(nD), for n variables and a maximum of D problems in any decomposition $\{P_i\}$. (In fact if we represent all but the first component as a continuation, generating the individual subproblems as needed, stack size can be reduced to O(n).) On the other hand, if we are more flexible in the agenda ordering, it permits opportunities for heuristic ordering that may save processing time. (If ordering is limited to ordering a set $\{P_i\}$ before placing it on the agenda, stack size requirements can still be O(nD).)

All of our examples will assume a stack organization for the agenda. However, that still leaves open questions about which problems to place next on the stack, and in what order.

We will indicate how versions of five specific algorithms can be formulated using this schema:

1. backtracking (BT) [Golumb and Baumert, 1965]: We use this basic algorithm to introduce the schema.

2. forward checking (FC) [Haralick and Elliott, 1980]: This is one of the most successful CSP techniques. The effective minimal domain size variable ordering [Haralick and Elliott, 1980] is naturally incorporated as a decomposition decision.

3. network consistency (NC) [Mackworth, 1977]: This algorithm explicitly operated in a recursive divide and conquer form, alternating local consistency processing with variable domain splitting. The schema formulation helps suggest a variety of NC variations to explore, and clarifies the relationship between NC and FC.

4. backtracking with cross product representation (BT-CPR) [Hubbe and Freuder, 1992]: The schema formulation, similar to that used in the original presentation of the algorithm, facilitates viewing backtracking as a degenerate case of backtracking with CPR. This in turn facilitates the demonstration that adding CPR can not increase the number of *constraint checks* (a standard measure of CSP algorithm performance) when searching for all solutions (and may reduce the number of checks significantly). A similar formulation is possible for forward checking with CPR.

5. inferred disjunctive constraints (IDC) [Freuder and Hubbe, to apear]: The new IDC algorithm was explicitly intended as a decomposition algorithm. It takes advantage of the fact that some solutions may be thrown away in the search for a single solution. The comparison with forward checking that the schema facilitates provides insight into the effective use of this technique. The technique can be shown to virtually always reduce problem size, i.e. the sum of the sizes of the subproblems will virtually always be less than the size of the decomposed problem.

Decomposition methods can be combined. IDC incorporates aspects of FC. An algorithm that uses heuristics to alternatively choose between FC and IDC decomposition during search has proven superior to either IDC or FC alone (testing all three with minimal domain size variable ordering) on some very hard problems [Freuder and Hubbe, to appear].

In the following five sections we specify the decomposition techniques that effectively define each of the five basic algorithms listed above. Plugging each decomposition technique into the decomposition schema produces a version of one of the algorithms. In each case we specify the decomposition abstractly and then illustrate it with a simple example. Section 7 discusses some theoretical aspects of the efficacy and efficiency of these algorithms. The final section proposes some directions for further work.

The abstract decomposition descriptions will refer to a *decomposed problem*, P, with n variables. Each of the subproblems in the decomposition will be specified by describing how to construct them from P. The subproblems are to placed on the decomposition schema's stack so that they reside on the stack in the same order as they are specified. We will often refer to the "first" uninstantiated variable, or the "first" value in a domain, assuming the variables and values are stored in some order. We specify "first" rather than "any" in order to present a more specific algorithm, rather than another schema, parameterized around the method of variable and value choice. However, we could also impose a heuristic variable or value search ordering scheme to choose the variables or values (or provide their initial ordering), and such schemes are of considerable interest.

We will use as an example a simple coloring problem. The variables are countries, the values are colors, the constraints specify that neighboring countries cannot have the same color. We will have four countries (variables): W, X, Y and Z. Each country has three possible colors (values): r(ed), b(lue), and g(reen). The countries are arranged in a "ring": W neighbors X, X neighbors Y, Y neighbors Z and Z neighbors W.

Coloring problems and subproblems will be represented by listing the domains for W, X, Y and Z in order. Thus the original problem can be represented:

```
r b g
r b g
r b g
r b g
```

The domains of instantiated variables will be represented in italics. We will carry out the decomposition depth-first until a solution is found. We will show all the non-empty sibling subproblems for each decomposition, even though in practice we need not generate all siblings at once. Empty subproblems will not be shown.

This example is purposely trivial for pedagogical purposes. It is *not* intended to illustrate the relative merits of the different algorithms. However, it may provide some insight into their potential, as well as their operation.

2 Backtracking

Decomposition:

1. The *instantiated subproblem*.

Mark the first uninstantiated variable, V, instantiated. Make its domain the first value, v, in the domain of V. If v is inconsistent with any of the (single) values in the domains of any of the previously instantiated variables, also remove v from the domain of V, leaving the instantiated problem empty (to be discarded by the algorithm schema).

2. The *remainder subproblem*.

Remove the value v from the domain of V.

Example: (Remember that we are not showing empty subproblems.)

```
r b g
r b g
r b g
r b g
  |                                                         \
r                                                           b g
r b g                                                       r b g
r b g                                                       r b g
r b g                                                       r b g
  |
r
b g
r b g
r b g
  |                                     \
r                                       r
b                                       g
r b g                                   r b g
r b g                                   r b g
  |                 \
r                   r
b                   b
r                   b g
r b g               r b g
  |
r
b
r
bg
  |                 \
r                   r
b                   b
r                   r
b                   g
```

3 Forward Checking

Decomposition:

1. The *precluded subproblem* .

Mark the first uninstantiated variable, V, instantiated. Make its domain the first value, v, in the domain of V. Remove values inconsistent with v from the domains of the remaining uninstantiated variables. As an obvious non-standard refinement, if there is only one uninstantiated variable, it too can be marked instantiated.

2. The *remainder subproblem*.

Remove the value v from the domain of V. (Same as in the backtracking decomposition.)

Example:

```
r b g
r b g
r b g
r b g
  |                                                   \
r                                                     b g
b g                                                   r b g
r b g                                                 r b g
b g                                                   r b g
  |                                 \
r                                   r
b                                   g
r g                                 r b g
b g                                 b g
  |               \
r                 r
b                 b
r                 g
b g               b g
```

4 Network Consistency

Decomposition:

1. The *first divided subproblem* .

Remove half the values from the domain of the first uninstantiated variable, V. Subject the subproblem to arc consistency processing, which may further reduce variable domains. Finally, mark any variables with single value domains instantiated, and if there is only one uninstantiated variable, mark it instantiated also.

2. The *second divided subproblem*.

Remove from the domain of V the values in the domain of V in the first divided subproblem. As with the first divided subproblem: Subject the subproblem to arc consistency processing, which may further reduce variable domains. Finally, mark any variables with single value domains instantiated, and if there if only one uninstantiated variable, mark it instantiated also.

(Note that in employing this decomposition in the decomposition algorithm schema the arc consistency processing does not actually have to be done until we take the subproblem off the agenda.)

Example:

```
r b g
r b g
r b g
r b g
  |                                                   \
r                                                     b g
b g                                                   r b g
r b g                                                 r b g
b g                                                   r b g
  |                                 \
r                                   r
b                                   g
r g                                 r b
b g                                 b g
  |                 \
r                   r
b                   b
r                   g
b g                 b
```

5 Backtracking With Cross Product Representation

Decomposition:

The *CPR subproblems*.

a. Split: For each value v in the first uninstantiated variable, V, create a subproblem where the domain of V is restricted to v, and the domains of each instantiated variable are restricted to those values consistent with v.

b. Merge: If any set of subproblems differ only in the domain of V, merge them into a single subproblem where the domain of V is the union of all their individual V domains (and the other domains are the same as they are in each of these subproblems).

Mark V instantiated in each subproblem.

Example:

```
r b g
r b g
r b g
r b g
  |
r b g
r b g
r b g
r b g
  |             \               \
b g             r g             r b
r               b               g
r b g           r b g           r b g
r b g           r b g           r b g
  |
b g
r
b g
r b g
 |              \               \
b g             g               b
r               r               r
b g             g               b
r               b               g
```

6 Inferred Disjunctive Constraint

Decomposition:

1. The *precluded subproblem*.

Mark the first uninstantiated variable, V_1, instantiated. Make its domain the first value, v, in the domain of V_1. Remove values inconsistent with v from the domains of the remaining uninstantiated variables. If there is only one uninstantiated variable, it too can be marked instantiated. (Same as in the forward checking decomposition.)

2. The *excised subproblems*.

For each remaining uninstantiated variable V_i, i = 2 to n, create a subproblem by removing v from the domain of V_1, removing any values inconsistent with v from the domains of V_2 through V_{i-1} and removing any values consistent with v from the domain of V_i. (Note any variables that do not share a constraint with V_1 will automatically lead to empty subproblems.)

Example:

```
r b g
r b g
r b g
r b g
  |                                     \                                       \
r                                       b g                                     b g
b g                                     r                                       b g
r b g                                   r b g                                   r b g
b g                                     r b g                                   r
  |                 \
r                   r
b                   g
r g                 b
b g                 b g
  |
r
b
r
b g
```

7 Theory

All of these decompositions meet the three conditions laid out in the first section, and thus when the decomposition algorithm schema uses them it will terminate and find a solution to solvable problems. Only the IDC decomposition takes advantage of the fact that some solutions can be thrown away as long as not all are thrown away. The others do not throw away any solutions, and thus if the algorithms employing these decompositions continue to explore the search space after finding solutions ("pretending" to fail) they will find all solutions.

CPR, a relatively new decomposition, is based on the insight that sets of incomplete solutions may be represented and efficiently processed in a cross product representation. It is proven analytically in [Hubbe and Freuder, 1992] that BT-CPR never requires more constraint checks than BT, when searching for all solutions, or proving that none exist. A similar result is obtained for CPR in conjunction with FC.

The new decomposition, IDC, is based on the hypothesis that, if P is solvable, either there will be a solution involving v for V, or there will be a solution involving a value inconsistent with v. It is proven in [Freuder and Hubbe, to appear] that:

1. IDC will find a solution if one exists. It may throw away some solutions, but often we are only looking for one anyway.

2. The size of the IDC decomposition (the sum of the sizes of the subproblems) will always be smaller than the size of the decomposed problem (except in the degenerate case when the instantiated value is the only value in its domain, and all other values are consistent with it). None of the other decomposition schemes can make such a strong reduction claim.

3. The size of the IDC decomposition will always be less than the size of the FC decomposition by an amount equal to the size of the *consistent subproblem*. If v is the value in the domain of the first variable, V, in the precluded subproblem, the consistent subproblem is formed from the decomposed problem by removing v from the domain of V and removing from all other domains any value inconsistent with v.

This theoretical analysis of IDC is facilitated by another decomposition process. We will describe this process and then present an example, again using our coloring problem. (The forward checking decomposition in the example is shown in the reverse order to that shown in Section 3.) Compare the leaves of the tree in the example with the first decomposition in Section 6.

Description:

First carry out a forward checking decomposition. Call the variable instantiated in the precluded subproblem V and the value in its domain v. Now decompose the remainder problem into two subproblems using a variation of the NC decomposition: divide the domain of the variable after V, not in half, but into two pieces, one containing all values inconsistent with v, the other containing all the values consistent with v. Repeat this NC-like decomposition process on the second subproblem, the one containing the consistent values, dividing the domain of the next variable into two pieces. Continue in this manner until all domains have been so divided. The leaves of the resulting decomposition tree will be the subproblems of the IDC decomposition plus the consistent subproblem. Now observe that given any solution to the consistent subproblem we can substitute v for the value of the first variable and still have a solution.

Example:

```
r b g
r b g
r b g
r b g
  |                                                \
b g                                                r
r b g                                              b g
r b g                                              r b g
r b g                                              b g
  |                                \
b g                                b g
b g                                r
r b g                              r b g
r b g                              r b g
  |
b g
b g
r b g
r b g
  |              \
b g              b g
b g              b g
r b g            r b g
b g              r
```

8 Further Work

Viewing these algorithms as instances of the decomposition schema helps us to compare them, and suggests new variations. For example, if we alter NC to subdivide a domain of d values into d pieces instead of 2 pieces, and reduce the amount of arc consistency processing appropriately, we arrive at forward checking. Is there another way to subdivide the domain that outperforms both algorithms for an interesting class of problems?

Ordering heuristics for subproblem consideration provide a new avenue to explore. We can view ourselves as searching in a metalevel "subproblem space". This subproblem space obviously lends itself to distributed and parallel processing, especially given the disjunctive nature of our decompositions. Sophisticated constraint languages may someday mix and match decomposition techniques as most appropriate for the problem at hand. Most intriguing of all is the possibility that useful new forms of decomposition are waiting to be discovered.

References

[Freuder and Hubbe, to appear] E. Freuder and P. Hubbe. Using inferred disjunctive constraints to decompose constraint satisfaction problems. *Proceedings of the Thirteenth IJCAI.*

[Golumb and Baumert, 1965] S. Golumb and L. Baumert. Backtrack programming. *JACM 12*. 516-524.

[Haralick and Elliott, 1980] R. Haralick and G. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence 14*. 263-313.

[Hubbe and Freuder, 1992] P. Hubbe and E. Freuder. An efficient cross product representation of the constraint satisfaction problem search space. *Proceedings of the Tenth National Conference on Artificial Intelligence*. 421-427.

[Mackworth, 1977] A. Mackworth. On reading sketch maps. *Proceedings of the Fifth IJCAI*. 598-606.