Completable Representations of Constraint Satisfaction Problems

Eugene C. Freuder
Computer Science Department
University of New Hampshire
Durham, NH 03824

Abstract

A solution to a constraint satisfaction problem specifies values for a set of variables that satisfy constraints on which combinations of values are permitted. Completability properties specify conditions under which partial solutions, for subsets of variables, will be extensible to full solutions. Different representations for a constraint satisfaction problem can make explicit different sets of constraints. In certain circumstances representations with desirable completability properties can be obtained in polynomial time.

1 INTRODUCTION

1.1 OVERVIEW

Constraints on a set of variables are relations that specify which combinations of values are permitted for those variables. A *constraint satisfaction problem* (CSP) involves finding an instantiation for each variable such that all the constraints are simultaneously satisfied. Many forms of reasoning can be viewed in CSP terms, e.g. temporal reasoning [Dechter, Meiri and Pearl 89], qualitative reasoning [Kuipers and Berleant 88], truth maintenance [de Kleer 89], diagnostic reasoning [Dechter and Pearl 88b] and image interpretation [Reiter and Mackworth 88].

An instantiation of a subset of variables will be called a *solution* if it satisfies all constraints in which any of the variables appear. A solution for all the variables will be called a *complete solution*. If a solution for a subset can be extended to a complete solution I will call it *completable*. For example if a for X, b for Y and c for Z is a complete solution, then a for X and c for Z is a completable solution for the X,Z subset.

Different representations of a CSP can make explicit different sets of constraints, while remaining *equivalent* in the sense that the set of complete solutions remains the same. These representations can have different completability properties based on the extent to which subset solutions are completable. Completability properties can be desirable, especially when these representations are viewed as knowledge bases subject to multiple queries.

I establish here methods for obtaining CSP representations with various completability properties. For appropriate classes of CSPs these methods will operate in polynomial time. As an example of the kind of result this work supports: Suppose we are given a scheduling problem involving a sequence of n event variables, each with 7 value choices, in which each event can only constrain the succeeding 5 events. We can obtain in $O(n^3)$ time a representation in which a solution for *any* subset of more than 5 consecutive events is part of a complete solution.

The notion of completability, and the methods for establishing it, derive from Montanari's concept of *minimality* [Montanari 74]. A CSP is often represented as a *constraint network*, where vertices correspond to variables and edges to constraints. A network of binary constraints is minimal if each pair of values which satisfies the binary constraint on a pair of variables is completable. Montanari viewed the problem of finding a minimal network equivalent to a given network as the *central problem* for networks of constraints. More recently a concept of minimality has played a role in temporal reasoning [van Beek 89]. Completability also serves to generalize global consistency [Dechter 90] and (i,n-i)-consistency [Freuder 85].

Though achieving minimality is in general an NP-complete problem, Montanari established certain conditions under which it can be achieved efficiently. He considered "syntactic" restrictions based on the topology or connectivity structure of the constraints in the networks and "semantic" restrictions on the nature or internal structure of the constraints. Dechter has recently shown how semantic restrictions on the number of potential variable values (or the size of the unary constraints) can be related to a generalization of minimality that she terms global consistency [Dechter 90].

In this paper I consider syntactic restrictions. I begin with CSPs whose constraint networks have the structure of an important class of graphs known as "k-trees" [Freuder 90].

K-trees generalize tree structures (trees are 1-trees). Completability provides appropriate generalizations of minimality in the context of k-trees and I prove completability results for k-tree-structured CSPs. This leads to a complexity bound on obtaining a completable representation for *any* CSP, expressed as a function of the "k-tree embedability" of its constraint network.

K-tree completability may prove especially useful in situations where expressivity can be traded for efficiency. Expressivity/efficiency tradeoffs have been studied in the temporal reasoning domain [Vilain, Kautz and van Beek 89]. Tambe and Rosenbloom have utilized results concerning tree-structured CSPs to trade expressivity for efficiency in a production system pattern matcher [Tambe and Rosenbloom 90].

For simplicity I assume that the original CSP problems we are given are binary CSPs (non-binary CSPs can be expressed as binary ones [Rossi, Dhar and Petri 89]). Higher order, non-binary constraints will in fact be introduced in transforming to completable representations.

The rest of the paper is organized as follows: Section 1.2 provides some CSP background material and presents a simple, informal example of the kind of result we are seeking, and the types of representation transformation that will be used. Section 2 presents the different representations we use and the transformations employed to go from one to another. It includes further motivation for our interest in k-tree structures. Section 3 introduces the concept of i multi-clique completability and proves the central theorem of the paper, which shows how to obtain an i multi-clique completable representation of a k-tree in polynomial time (for fixed k). Section 4 examines special cases of multi-clique completability. It provides a bound for full completability, where a solution for any subset of variables is completable. Section 5 goes beyond k-trees. It discusses "partial k-trees" and uses them to derive results applicable to any CSP. It discusses series-parallel graphs and bandwidth-k graphs, which may be of special interest in CSP applications, e.g. in temporal reasoning. Section 6 contains a final example and a brief summary.

1.2 BACKGROUND AND EXAMPLE

This section presents a simple example to introduce illustrate the idea of completability and the nature of the results we will obtain. A "variable completable" representation of a simple CSP is found. At the same time some basic CSP concepts are reviewed.

Consider the graph coloring problem in Figure 1. The problem is to color the vertices of the graph such that no two vertices that are joined by an edge have the same color. At each vertex we can select from the colors indicated. Blue is indicated by "b", red by "r" and green by "g". The vertices of the graph correspond to CSP variables, the edges to CSP constraints.

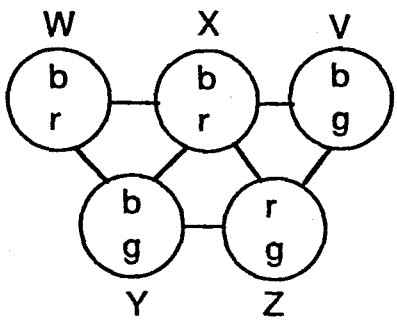

Figure 1: A Graph Coloring Example

It happens that for graph coloring CSPs the constraint network has the same structure as the graph to be colored. This is convenient for devising sample problems with a desired structure.

We will call the constraint network constructed directly from the specification of a CSP the *direct constraint network*. This is to distinguish it from contraint networks we might construct that in some fashion represent the same set of solutions, but with vertices and edges that correspond to a different set of variables and/or constraints. For example, we can modify the direct constraint network by making explicit constraints which are only implicit in the original problem specification, or by removing constraints which are redundant in the original problem specification. (I am trying to avoid in this paper introducing a lot of formal notation and technical detail to specify a CSP and its alternative representations.)

It would be nice to know that any value we pick for any variable that satisfies the constraints directly involving that variable is part of some solution to the problem. We will say that a CSP representation is *variable completable* iff any solution for the singleton subset containing any one variable is completable. This is clearly a nice basic property for a constraint data base to have. However, this is not the case at present. For example, value blue for variable W does not participate in any solution.

To fix this we add further constraints. These are pictured in Figure 2. First, we add ternary (three variable) constraints on the triples of variables WXY, XYZ, XZV. These are the triples of variables that have the property that each variable is constrained by each other variable. The ternary constraints correspond to solutions of three CSP subproblems: each subproblem consists of one of these triples of variables, along with the associated binary constraints between pairs of variables. For example, the ternary constraint on WXY specifies that the triple (red blue green) is a consistent solution to that subproblem.

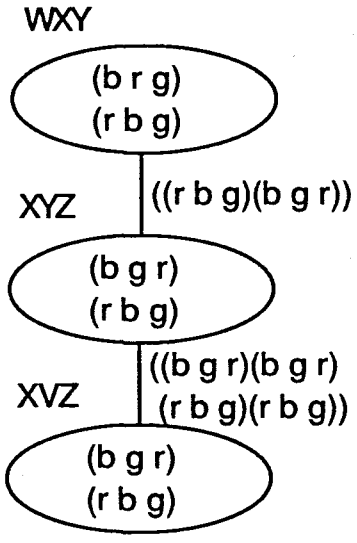

Figure 2: Metaproblem

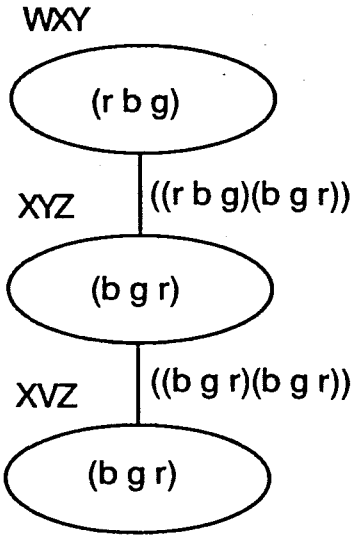

Figure 3: Arc Consistency

Next we add a constraint between the subproblems WXY and XYZ, and another between XYZ and XVZ. These new constraints are effectively 4-ary constraints, between 4 variables of the original CSP, as each pair of subproblems involve four distinct variables. However, for the moment we shall view these new constraints as binary constraints in a "metaproblem". The variables of this metaproblem will correspond to the subproblems identified earlier. The values will be the solutions to the subproblems. The constraints will specify that the solutions to the subproblems must agree on common variables. For example, the triple (red blue green) for WXY is consistent with the triple (blue green red) for XYZ, but (blue red green) for WXY is not consistent with any solution for XYZ: it agrees with one on X and the other on Y, but with neither on both X and Y at once.

In Figure 3 we see the result of achieving *arc consistency* [Mackworth 77], or *2-consistency* [Freuder 78], on the metaproblem: any value for a metavariable not consistent with any value at some neighboring metavariable is eliminated. Notice that initially (red blue green) for XVZ is consistent with (red blue green) for XYZ. However, after the constraint between WXY and XYZ eliminates (red blue green) for XYZ, there is nothing left for the (red blue green) solution of XVZ to go with, and it too is eliminated.

I claim that it is now impossible to pick a value for one of the original CSP variables that satisfies the constraints embodied in Figure 3, but fails to be part of a complete solution. Blue for W, for example, is not in any solution for WXY, i.e does not satisfy the WXY ternary constraint. We could "project" the constraints downward onto the individual variables, e.g. removing blue from W. We can imagine a final representation which contains the original variables, with values removed as called for by the new constraints, along with the new ternary and 4-ary constraints.

At this point you can verify that the representation in Figure 3, is in fact variable completable. However, the process of acquiring this representation remains black magic. Why did I choose those particular subproblems? Why did I link them in that way? How did I know to perform arc consistency processing? These questions will be answered in the rest of the paper. The structure in Figure 3 is an example of what we will define below as a 2-closure of a clique tree representation of the original problem. The original problem has a 2-tree structure. It may help to keep this example in mind as we extend the above process and formulate it in a considerably more general manner to achieve "multi-clique completability".

2 REPRESENTATIONS AND TRANSFORMATIONS

This section defines k-trees, clique trees and i-closures and the representation transformations that take a k-tree into a clique tree and a clique tree into an i-closure of a clique tree. I argue that these transformations do not change the set of solutions. Most of the completability results will concern closures of clique trees of k-trees. I will use a simple example for illustration.

2.1 K-TREES

This section defines k-trees and discusses their importance. I argue that interesting CSPs can have k-tree structure and that to some degree k-tree results apply to *any* CSP.

Call a graph in which any two vertices are joined by an edge a *clique*. A k-tree is formed by starting with a *k-clique*, a clique of k vertices. Vertices are added by linking them to all the vertices of an already present k-clique. Figure 4 illustrates the construction of a 2-tree. The numbering of the vertices reflects the order in which the 2-tree was built.

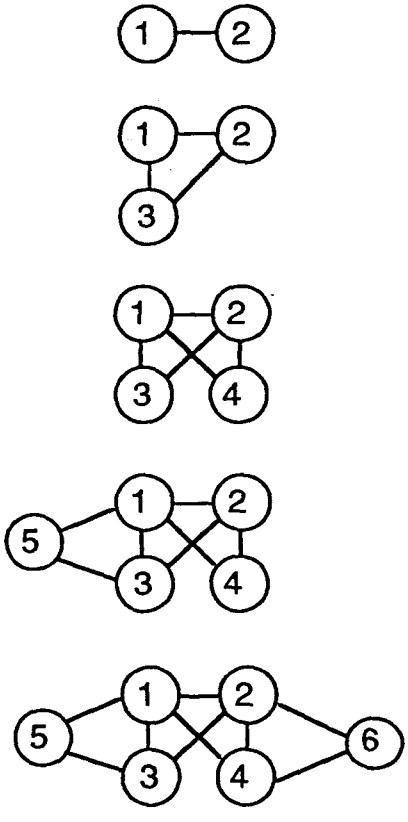

Figure 4: Building a 2-Tree

A graph is a *k-tree* iff:
1. It is a k-clique
or
2. There is a vertex v such that:
   a. v is connected to a k-clique
   b. removing v and its associated edges leaves a k-tree.

A *partial k-tree* is a subgraph of a k-tree. A CSP will be called a *k-tree CSP* if the direct constraint network is a k-tree.

Why should we care about k-tree CSPs?

1. 1-trees are the usual tree structures, which can represent hierarchically structured CSPs.

2. 2-trees are series-parallel networks, which might represent e.g. design problems for series-parallel circuits.

3. K-tree CSPs are solvable in time polynomial in k [Freuder 90].

4. Any constraint network can be embedded in a k-tree for large enough k, by adding appropriate trivial constraints (*trivial constraints* allow all value pairs and are not normally represented by constraint network edges; in this case we make an exception).

5. Any constraint network can be reduced to a partial k-tree for any k by removing enough constraints, settling for a partial solution [Freuder 89].

6. Several techniques have been developed to take advantage of tree substructures in general CSPs; these may be profitably extensible to k-trees [Freuder 90]. In particular k-trees provide target structures for decomposing CSPs into subproblems.

7. K-trees structures offer a hierarchy of choices in trading representational power for computational efficiency; tree structures have already been used for this purpose. In some circumstances we may be in a position to guarantee that problems will be k-tree-structured, in return for limiting the class of problems which we can handle.

8. K-trees, not only trees, have a natural "semantics", that makes it likely such structures will arise in practice. For example, if we have a scheduling, planning or temporal reasoning problem involving a sequence of events, with the plausible restriction that one event cannot affect another more than k events distant, we have a partial k-tree. We can make this a k-tree by simply adding trivial constraints as necessary so that each event is linked to all k events on either side.

2.2 CLIQUE TREES

Given a k-tree we can form a *clique tree* representation. The vertices of the clique tree correspond to the maximal cliques (k+1-cliques) in the k-tree. They are linked in a manner which reflects the construction of the k-tree. Adjacent cliques have a k-clique in common. Figure 5 contains a clique tree for the 2-tree in Figure 4. The 1,2,4 node is a child of the 1,2,3 node reflecting the fact that in building the 2-tree we joined the 4 vertex to vertices 1 and 2. (The 2-tree could have been constructed in a different order and has alternative clique tree representations.) A clique tree can be obtained in O(n) time, for an n vertex k-tree; Wimer [Wimer 87] gives a clique tree construction algorithm. The *leaves* are those vertices with a single adjacent edge.

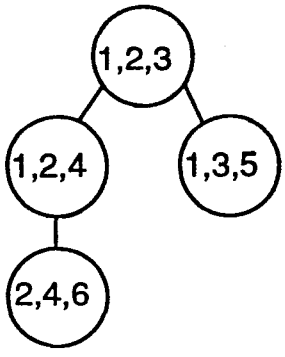

Figure 5: A Clique Tree

This representation is related to acyclic databases and acyclic CSPs [Dechter and Pearl 89], and to tree decompositions [Robertson and Seymour 86]. Its roots go back at least to Beineke and Pippert's early work on k-trees [Beineke and Pippert 71].

The clique tree of a constraint network can be regarded as the constraint network of a *metaproblem* in which the cliques correspond to metavariables whose values are the solutions of the clique subproblems. The constraints of the metaproblem require that consistent values for two adjacent metavariables have identical values for identical variables from the original problem. The clique tree can also be regarded as an expanded version of the original problem, a constraint network in which certain implicit higher level constraints are made explicit; the clique tree vertices correspond to k+1-ary constraints, the clique tree edges to k+2-ary constraints.

2.3 CLOSURES

The *i-closure* of a constraint network results from adding the additional constraints induced by a strong i-consistency algorithm [Freuder 78; Freuder 82; Cooper 89]. If a constraint network is strongly i-consistent then given any consistent values for a subset of fewer than *i* variables, and then any additional variable V, we can find a value for V consistent with the previous values.

The completability results generally involve obtaining the i-closure, for an appropriate i, of the clique tree representation of the direct k-tree constraint network for a CSP. If we wish we can go a step further, adding the original variables back in, reducing their domains of values as required by the constraints (e.g. through consistency propagation as in [Freuder, 78]). We thus finish with a (non-binary) constraint network on the original variables, with the desired completability property.

This final network can be visualized in two ways. All constraints (including the variables viewed as unary constraints) can be represented by vertices. Constraints which share variables are linked by edges at least to the extent needed to ensure that a variable cannot be assigned two different values at the same time to satisfy two constraints. This representation is descended from the constraint network representation I presented in [Freuder 78], and which I view here as a form of metaproblem representation. Alternatively higher order constraints can be viewed as hyperedges in a hypergraph representation [Dechter and Pearl, 89].

We need to know that all this transforming never changes the set of solutions for the original problem variables; however, this is fairly obvious. Similar transformations have been used before in the literature. The clique tree representation incorporates the original constraints, so no new solutions can creep in. The new constraints only state explicitly subproblems of the original problem, and ensure that two subproblem solutions do not assign different values to the same variable; thus no old solutions are ruled out. K-consistency can be used as a preprocessing step in solving CSPs. It only removes inconsistent values. Adding the original variables back only allows established constraints to reflect explicitly restrictions on permissible values that they already express implicitly.

3 MULTI-CLIQUE COMPLETABILITY

This section contains the main result of the paper. A set of variables for a CSP will be called a *clique* if the vertices corresponding to the set of variables, together with the constraints between pairs of these variables, constitute a clique in its direct constraint network representation. A *j-clique* of variables is a clique containing j variables. A CSP representation is *i multi-clique completable* iff any solution for a subset of variables belonging to any i (or fewer) cliques is completable.

The definition allows us to pull, for example, two values from one clique, five from another, all the values from a third. Any cliques will do, though of course they will all be contained in some maximal clique, which in the case of k-tree structure means some k+1-clique. Note that i multi-clique completability implies in particular that a solution for any i *variables* is completable.

The basic idea is to add explicit constraints to the original representation of the CSP sufficient to rule out partial solutions which cannot be completed. This can always be done simply by finding all the complete solutions. The trick is to see if we can get by with less complete processing of the problem. K-tree structures permit that.

Some intuition as to the basic nature of the processing can be gained by thinking of the problem as follows: Given values at a set of cliques in the clique tree we have to a) fill in consistent values in the area "bounded" by those cliques and b) fill in values at the cliques in the subtrees outside this bounded area. The subtrees can be handled recursively. The bounded area can be filled in using the strong i+1-consistency of the i+1-closure of the clique tree.

A primary use of i+1-consistency here is analogous to the use of 3-consistency to fill in values along a path between two constraint network vertices. It has been recognized that 3-consistency corresponds to "path" consistency; if given values for two variables we can find a value for a third variable, we can find values all along a path between them, such that all the constraints represented by the edges of the path are satisfied [Montanari 74]. I suggest here a generalized "physical" implication of i-consistency, which might be termed *tree consistency*: i-consistency guarantees that given values for i-1 variables we can find values for all variables in a tree embedded in the constraint network with these i-1 variables as leaves, such that all the binary constraints represented by the edges of this tree will by satisfied.

Theorem 1: Given a k-tree CSP with n variables, each with at most d values, in $O((nd^{k+1})^{i+1})$ time we can obtain an i multi-clique completable representation.

Proof. We obtain the i+1-closure of a clique tree representation of the k-tree direct constraint network of the given CSP. We want to show that this representation is i multi-clique completable. We have seen that the original problem, the clique tree, and the i+1-closure of the clique tree, all have the same set of solutions. When we are done we can add back vertices corresponding to the original variables if we wish, as suggested in Section 2.3.

We are given a solution for a set of variables chosen from i chosen cliques. I outline an algorithm for completing the solution, finding consistent values for the remaining variables. The algorithm recursively processes the representation. The emphasis is on demonstrating that appropriate values must exist; once we know that a value exists it can always be found, if necessary by exhaustive search. There are two cases in the terminal or basis step of the recursion, and two cases in the non-terminal or recursive step.

Note that in choosing new values we can use the induced constraints in the i+1-closure of the clique tree to insure consistency. However, at the same time we can take advantage of the fact that to form a solution it is sufficient that a set of values satisfy the constraints in the original clique tree.

Preprocessing Step:

If there are any cliques for which some variables, but not all, have been assigned values, assign consistent values to the remaining variables in those cliques. This can be done as the original values chosen must have satisfied the constraint involving all the variables from those cliques induced by i+1-closure of the clique tree.

Terminal Step:

Case 1: All the variables at all the cliques have been assigned consistent values.

Case 2: The variables at only one clique have been assigned consistent values. Make that clique the root of the clique tree. Now 2-consistency guarantees that we can find consistent values at the remaining vertices of the tree, e.g. while traversing the tree in breadth-first order.

Non-terminal Step:

Case 1: At least one of the cliques, C, whose variables have been assigned consistent values is not a leaf of the clique tree. Removing C disconnects the tree into two subtrees, involving two sets of cliques, R and S. The problem we are currently dealing with can be solved by independently and recursively solving two subproblems, involving R plus C and S plus C. We recurse on the subgraphs of the i+1-closure of the clique tree induced by the union of R and C and the union of S and C. The only "communication" in the original clique tree between the cliques of the two subproblems is through C; if we solve the subproblems independently and the solutions agree on C we will have a solution to the full problem.

Case 2: All of the cliques whose variables have been assigned consistent values are leaves of the clique tree. Since not all cliques contain variables with chosen values, there must be a leaf, L, whose one adjacent vertex, P, does not contain variables with chosen values. We use i+1-consistency to move inward in the tree from L. Choose a metavalue for P, i.e. values for all the variables in the clique P, consistent with all the chosen values at the leaves. From this point we can ignore L; it's only contact with the other cliques, in the original clique tree, is through P. Recurse on the representation obtained by removing L.

Recursion in either case results in smaller problems; eventually one of the terminal cases must be reached.

The complexity bound largely follows from the bound on Cooper's k-consistency algorithm [Cooper 89], which is $O(n^k d^k)$ for a CSP with n variables of d values. Our metavariables have at most d^{k+1} values, and there will be n-k k+1-clique metavariables, see [Beineke and Pippert 71]. Q.E.D.

It may be more natural to talk directly about completing solutions for sets of variables rather than in terms of cliques. Since any variable is in some clique we have the following corollary (in this and succeeding theorems n will represent the number of variables, d the maximum number of values for any variable):

Corollary 1: For a k-tree CSP, in $O((nd^{k+1})^{i+1})$ time we can obtain a representation in which any solution for any i variables is completable.

In choosing to achieve i multi-clique completability for a particular i there is clearly a tradeoff between computational effort and level of completability. The level and form of completability appropriate for an individual problem will balance the needs of the user against the computational realities.

4 EXTREMES

This section focusses on i multi-clique completability representations of k-tree CSPs for extreme values of i and k: i=1 and n-k (the number of maximal cliques), and k=1 (the usual tree structures). (For n vertices the maximum k value is n, associated with a complete graph) Improved completability bounds are obtained for the minimal i and k values.

4.1 CLIQUE COMPLETABILITY

A CSP representation is *clique completable* iff any solution for any subset of variables from any one clique is completable.

Theorem 2. A clique completable representation of a k-tree CSP can be obtained in $O(nd^{k+2})$ time.

Proof: This is basically Theorem 1 for i=1. However, for tree-structured CSPs 2-closure can be obtained in time linear in the number of variables and quadratic in the maximum number of values for a variable [Dechter and Pearl 88a]. The latter factor comes from the need to pair all the values in one variable with all those in an adjacent variable. However, we have a special case here. Adjacent metavariables in the clique tree share k out of k+1 variables from the original problem. Assuring consistency between them amounts to establishing a k+2-ary constraint in the original problem, or solving a CSP with k+2 variables. This can be done in $O(d^{k+2})$ time (for fixed k). If we do this for pairs of adjacent metavariables starting at each leaf and moving up to the root, then going back down from the root to each leaf, we achieve 2-consistency in $O(nd^{k+2})$ time. Q.E.D.

It is easy to see, using results from [Freuder 90] and the k-tree literature, that achieving k+1-consistency for k-trees would make all clique solutions completable. As we have seen a k-tree can be built by starting with a k+1-clique and adding vertices which each in turn link to all the vertices in an already present k-clique, forming another k+1-clique. The initial clique is called a *basis*, and any k-clique can serve as a basis for building the k-tree in this way [Proskurowski 84]. Thus to complete a solution for a clique, we could, if we had k+1-consistency, first add values as needed to form the solutions to a k-clique, then choose values for each variable we added in turn as we built the k-tree. At each point we would only need to worry about the consistency of the new value with k already chosen values, and k+1-consistency would take care of that. (If we are given all the values for a k+1-clique to begin with, one of the non-basis values will be predetermined.)

However, for k+1-consistency we have a complexity bound $O(n^{k+1}d^{k+1})$. Comparing with the bound from Theorem 2, essentially we trade a factor of n^k against a factor of d. Clearly as a function of the number of problem variables, n, the bound of Theorem 2 is superior.

Corollary 2: A variable completable representation of a k-tree CSP can be obtained in $O(nd^{k+2})$ time.

Proof. Since every variable is in some clique, this is a special case of clique completability. Q.E.D.

4.2 FULL COMPLETABILITY AND TOTAL-CLIQUE COMPLETABILITY

A CSP representation is *fully completable* iff any solution for any subset of variables is completable. We will see that full completability corresponds to n-k multi-clique completability.

Theorem 3: A fully completable representation of a k-tree CSP can be obtained in $O((nd^{k+1})^{n-k+1})$ time.

Proof: As observed earlier, there are n-k maximal cliques in a k-tree. Any variable must be in at least one of these. Thus n-k multi-clique completability implies full completability; it can be obtained in $O((nd^{k+1})^{n-k+1})$ time by Theorem 1. Q.E.D.

A potentially better bound can be obtained for a limited form of n-k multi-clique completability. Define a representation to be total-clique *completable* iff any solution for a subset of variables that constitutes *all* the variables from any set of maximal cliques is completable.

Theorem 4: Given a clique tree representation of a k-tree CSP, with L leaves, we can obtain a total-clique completable representation in $O((nd^{k+1})^{L+1})$ time.

Proof. Observe that in the proof of Theorem 1, after the preprocessing step, we only need more than 2-consistency to deal with the leaf cliques or some subset of them. If all the variables in the cliques involved have been assigned variables to start with we do not need the preprocessing step. Thus we will never need more than L+1-consistency, where L is the number of leaves in the clique tree, regardless of what i is. Q.E.D.

Note that this result places a premium on finding a clique tree with a minimal number of leaves. The same CSP can have two clique tree representations with very different numbers of leaves. In general a k-tree can have approximately k/(k+1) leaves, so the upper bound outlook is not good: on the other hand, there are "chain" clique trees of arbitrary size that have only two leaves. If the clique tree is a *chain*, a tree with a single branch, we can obtain total-clique completability in $O(nd^{k+1})^3$ time.

4.3 TREES

When k=1 we have ordinary tree structures. For tree CSPs, Corollary 2 says that we can obtain a variable completable representation in $O(nd^3)$ time. The experienced reader may observe here that 2-closure of the

tree structured constraint network itself, *not* the clique tree, is sufficient to guarantee variable completability, and can be obtained in $O(nd^2)$ time. In fact, for k=1, clique completability is Montanari's minimality, and, as Montanari noted, 2-closure of a tree structure is sufficient to achieve minimality [Montanari 74].

Montanari has a further result. He shows that values for any two variables in the 3-closure of a tree CSP, not necessarily from the same clique in the tree (i.e. not necessarily joined by an edge) can be extended to a solution [Montanari 74]. 3-consistency can be obtained in $O(n^3d^3)$ time [Cooper 89]. Furthermore Montanari's method, which first fills in a path between the variables, motivating the approach of Theorem 1, appears to also achieve 2 multi-clique completability (which could extend the solution of as many as four variables).

For k = 1 Theorem 1 gives a bound of $O(n^3d^6)$; avoiding the clique tree transformation gives a better bound. Furthermore, we can use the generalization of path consistency to tree consistency suggested in Section 3 to handle i variables, or cliques. This approach requires i+1-consistency, where we have an $O((nd)^{i+1})$ bound. This compares with the $O((nd^2)^{i+1})$ bound given for i multi-clique consistency for k = 1 by Theorem 1. We have sketched out a proof of the following theorem, generalizing Montanari's results:

Theorem 5. The i+1-closure of a tree CSP is i multi-clique completable. In particular a solution for any i variables is completable.

The question now arises as to whether these results imply a more efficient alternative to the methods of Theorem 1 for k > 1. Theorem 1 already is descended from Montanari's methods. However, it may be possible to extend those methods more directly than Theorem 1 does, perhaps by using the concept of a k-path [Beineke and Pippert 71], or a tree of k-cliques.

5 PARTIAL K-TREES

This section indicates how the previous completability results extend to partial k-trees via their embedding in a k-tree. The embedding transformation again does not change the set of solutions; it involves adding trivial constraints. Partial k-trees allow us to formulate results for general CSPs and for classes of special interest, series-parallel and bandwidth-k CSPs.

Recall that a partial k-tree is a subgraph of a k-tree. As noted above, by adding trivial constraints any constraint network can be made into, or *imbedded* in, a k-tree, for some k. If necessary we can simply add enough trivial constraints to form a complete graph (all possible edges); complete graphs are k-trees. Thus since any constraint network is a partial k-tree for some k, completability results can be provided for *any* CSP. However, if indeed the embedding is the complete graph, the completability results will not be very interesting.

Combining these observations with Theorem 1 gives us:

Theorem 6: Given an arbitrary constraint network G and an embedding of G in a k-tree, in $O((nd^{k+1})^{i+1})$ time we can obtain an i multi-clique completable representation.

It is obviously desirable to obtain an optimal embedding in the sense of one in which k is as small as possible. An optimal embedding may be hard to find. There is an $O(n^{k+2})$ algorithm to find the embedding [Arnborg, Corneil and Proskurowski 87]. For a given CSP, even the optimal k may be close to n, resulting in a k exponent in the complexity bounds that is close to n.

However, if a 2-tree embedding exists it can be found in linear time [Wald and Colbourn 83]. The class of partial 2-trees is the class of series-parallel networks, which are of special interest in engineering. Thus we have:

Theorem 7: Given a series-parallel constraint network: in $O((nd^3)^{i+1})$ time we can obtain an i multi-clique completable representation.

For larger optimal k, heuristic methods that settle for suboptimal embeddings may prove effective. Dechter and Pearl's work on tree-clustering [Dechter and Pearl 89] provides a closely related representation that should be useful in treating arbitrary graphs. In some cases the embedding may be obvious from the semantics. Consider, for example, the scheduling problem in Section 1.1. Each event can constrain up to 5 succeeding events. The embedding into a 5-tree is not hard to find: add trivial constraints as needed so that each event does constrain the 5 succeeding ones.

The direct constraint network of this scheduling problem is a bandwidth-5 graph. A *bandwidth-k graph* is one for which an ordering of the vertices exists with the property that no vertex shares an edge with another more than k vertices before or after it in the ordering. This ordering, call it a *bandwidth-k ordering*, can be found, if it exists, in polynomial time for fixed k [Saxe 80]. Bandwidth has been studied in connection with CSPs [Zabih 90]. Restricting problem structure to fit a bandwidth-k criterion would seem to be a natural way of specifying representational restrictions for classes of CSP problems in, for example, temporal reasoning.

Theorem 8. Given a bandwidth-k ordering of a direct constraint network of a CSP, a representation in which a solution for any set of k+1 or more consecutive variables in the ordering is completable can be obtained in $O((nd^{k+1})^3)$ time.

Proof. Bandwidth-k graphs are easily seen to be partial k-trees; they can be embedded in k-trees by adding trivial constraints as needed to ensure that each vertex is linked to all vertices not more than k vertices away in the ordering.

Moreover, such a k-tree has a clique tree which is a simple chain with two leaves. Any set of k+1 or more consecutive values will constitute all the variables in a set of adjacent cliques in this chain. Apply Theorem 4. Q.E.D.

This theorem establishes the $O(n^3)$ result claimed for the scheduling problem in Section 1.1.

6 CONCLUSION

6.1 ILLUSTRATION

A final example will illustrate various forms of completability. Figure 4 can be viewed as representing another simple graph coloring problem. Suppose that there are three colors available at each vertex: red, blue and green. The graph to be colored, and the 2-tree direct constraint network for this coloring problem, both have the structure shown in the figure.

The clique tree for the 2-tree with the structure shown in Figure 5 is 2-consistent without any further processing. Thus it is clique completable. If we color 1 red, 3 blue and 5 green, for example, we can go on to find consistent colors for 2 (green), 4 (red) and 6 (blue).

However, the representation is not 2 multi-clique completable. Suppose, for example, we choose blue for 3, red for 5 and red for 6. These values do not violate any of the constraints embodied explicitly in the clique tree. However, there is no complete solution to the problem which includes this assignment of values.

The 3-closure of the clique tree *is* 2 multi-clique completable. Obtaining the 3-closure will, in particular, induce a new constraint, C, between the 2,4,6 vertex and the 1,3,5 vertex. The values blue for 3 and red for 5 appear in only one triple at the 1,3,5 vertex: g,b,r. The value red for 6 appears in two triples at the 2,4,6 vertex: g,b,r and b,g,r. However, the new constraint C does not allow either 2,4,6 triple to be paired with the 1,3,5 triple. Thus the incompletable combination of blue for 3, red for 5, and red for 6 is not a solution for the subset of variables 3, 5 and 6 in the 3-closure of the clique-tree.

Since the clique tree has only two leaves, the 3-closure is total-clique completable. Now for this small sample problem it might have been easier simply to find all the complete solutions. However, we could make the coloring problem considerably larger and more complex while still retaining a simple chain clique tree structure.

6.2 SUMMARY

For k-tree CSPs a representation can be obtained in $O((nd^{k+1})^{i+1})$ time in which a solution for variables taken from any i cliques of the direct constraint network can always be extended to a complete solution to the problem. This is called an i multi-clique completable representation. The representation obtained is the i-closure of the clique tree of the direct constraint network of the k-tree CSP. Completable representations for any CSP can be found by first embedding the CSP constraint network in a suitable k-tree.

Tighter bounds than those derivable in the obvious way from i multi-clique completability are available for clique completability, variable completability and for trees (1-trees). The bounds on full and total-clique completability are better than one might expect.

Specific results are available for CSPs with series-parallel and bandwidth-k structure. There is reason to believe that temporal reasoning and scheduling problems, in particular, may admit computationally tractable completability results. Constraint knowledge bases might be preprocessed to provide completability properties.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. IRI-8913040. The Government has certain rights in this material. The author is currently a Visiting Scientist at the MIT Artificial Intelligence Laboratory.

References

[Arnborg, Corneil and Proskurowski 87] Complexity of finding embeddings in a *k*-tree. *Siam J. Alg. Disc. Meth. 8*, 2, 277-284.

[Beineke and Pippert 71] Properties and characterizations of *k*-trees. *Mathematika 18*. 141-151.

[Cooper 89] An optimal *k*-consistency algorithm. *Artificial Intelligence 41*. 89-95.

[Dechter 90] From local to global consistency. *Proc. of the Eighth Biennial Conference of the Canadian Society for Computational Studies of Intelligence*. Morgan Kaufmann.

[Dechter, Meiri and Pearl 89] Temporal constraint networks. In Brachman, Levesque and Reiter (eds.) *Principles of Knowledge Representation and Reasoning: Proceedings of the First International Conference*. San Mateo, CA: Morgan Kaufmann.

[Dechter and Pearl 88a] Network-based heuristics for constraint-satisfaction problems, *Art. Int.* 34(1).

[Dechter and Pearl 88b] Constraint-directed approach to diagnosis. UCLA CS Dept. Tech. Rept. R-72-I.

[Dechter and Pearl 89] Tree clustering for constraint networks. *Artificial Intelligence*. 353-366.

[de Kleer 89] A comparison of ATMS and CSP techniques. *Proc. of the Eleventh International Joint Conference on Artificial Intelligence*, 290-296.

[Freuder 78] Synthesizing constraint expressions. *CACM 21*, 11.

[Freuder 82] A sufficient condition for backtrack-free search. *JACM 29*, 1.

[Freuder 85] A sufficient condition for backtrack-bounded search, *JACM 32*, 4.

[Freuder 89] Partial constraint satisfaction. *Proc. of the Eleventh International Joint Conference on Artificial Intelligence*, 278-283.

[Freuder 90] Complexity of k-tree structured constraint satisfaction problems. *Proc. of the Eighth National Conference on Artificial Intelligence*, AAAI Press, Menlo Park, CA, .

[Kuipers and Berleant 88] Using incomplete quantitative knowledge in qualitative reasoning. *Proc. of the Seventh National Conference on Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA.

[Mackworth 77] Consistency in networks of relations. *Artificial Intelligence 8*, 99-118.

[Montanari 74] Networks of constraints: fundamental properties and applications to picture processing. *Information Sciences 7*. 95-132.

[Proskurowski 84] Separating subgraphs in k-trees: Cables and caterpillars, *Discrete Math. 49*, 275-285.

[Reiter and Mackworth 88] A logical framework for depiction and image interpretation. Univ. of British Columbia Dept. of CS Tech. Rept. 88-17.

[Robertson and Seymour 86] Graph minors II: algorithmic aspects of tree-width. *Journal of Algorithms 7*. 309-322.

[Rossi, Dhar and Petri 89] On the equivalence of constraint satisfaction problems. MCC Technical Report ACT-AI-222-89. MCC, Austin, Texas 78759.

[Saxe 80] Dynamic programming algorithms for recognizing small-bandwidth graphs in polynomial time. *SIAM J. Alg. Disc. Meth. 1*, 363-369.

[Tambe and Rosenbloom 90] A framework for investigating production system formulations with polynomially bounded match. *Proc. of the Eighth National Conference on Artificial Intelligence*, AAAI Press, Menlo Park, CA, 693-700.

[van Beek 89] Approximation algorithms for temporal reasoning. *IJCAI-89*.

[Vilain, Kautz and van Beek 89] Constraint propagation algorithms for temporal reasoning: a revised report. in *Readings in Qualitative Reasoning About Physical Systems*. de Kleer and Weld, editors. Morgan Kaufmann.

[Wald and Colbourn 83] Steiner trees, partial 2-trees, and minimum IFI networks. *Networks 13*, 159-167.

[Wimer 87] Linear algorithms on k-terminal graphs. Ph.D. dissertation, Clemson.

[Zabih 90] Some applications of graph bandwidth to constraint satisfaction problems. *Proc. of the Eighth National Conference on Artificial Intelligence*, AAAI Press, Menlo Park, CA, 46-51.