Complexity of K-Tree Structured Constraint Satisfaction Problems

Eugene C. Freuder[1]

Department of Computer Science
University of New Hampshire
Durham, New Hampshire 03824

Abstract

Trees have played a key role in the study of constraint satisfaction problems because problems with tree structure can be solved efficiently. It is shown here that a family of generalized trees, k-trees, can offer increasing representational complexity for constraint satisfaction problems, while maintaining a bound on computational complexity linear in the number of variables and exponential in k. Additional results are obtained for larger classes of graphs known as partial k-trees. These methods may be helpful even when the original problem does not have k-tree or partial k-tree structure. Specific tradeoffs are suggested between representational power and computational complexity.

1 Introduction

A *constraint satisfaction problem* (*CSP*) involves finding values for variables subject to constraints on what combinations of values are allowed. Constraint satisfaction problems occur widely in artificial intelligence. Recently they have received particular attention in the domains of scheduling and temporal reasoning.

Judea Pearl has observed that a primary advantage of viewing AI problems in CSP terms is our ability to identify certain "islands of tractability" among CSP problems, classes of problems that admit efficient solution. The significance of such classes is increased by techniques for "massaging" problems outside these classes to fit into or utilize these classes.

Up until now there has been one major tractable island, the class of tree structured problems. In this paper I identify a "chain" of such islands, generalizations of tree structures, which considerably expand the "tractable topography" of the CSP world. The tractability of these classes varies inversely with the representational complexity of the problems in the class, but the solution effort is always linear in the number of problem variables.

We will restrict our attention here to binary constraint satisfaction problems, in which the given constraints involve two variables. (Non-binary CSPs are shown to be "reducible" to binary CSPs in [Rossi, Dhar, Petrie, 89].) A binary CSP can be represented by a *constraint network*, in which the nodes represent variables and the links represent constraints.

The general constraint satisfaction problem is NP-complete. However, tree structured problems (those with tree structured constraint networks) have a worst case complexity bound that is linear in the number of problem variables [Mackworth and Freuder, 84]. If there are n variables, and no variable has more than d values to choose from, a tree structured CSP can be solved in $O(nd^2)$ time [Dechter and Pearl, 88].

K-trees [Beineke and Pippert, 71; Rose, 74] generalize trees. Trees are 1-trees. The central result of this paper is the demonstration that CSPs whose constraint networks are k-trees can be solved in $O(nd^{k+1})$ time. Note that we are saying that the complexity of a k-tree structured CSP is bounded by a linear function in the number of variables as long as we maintain the original k-tree structure and d bound when increasing the size of the problem.

Further results are described briefly for larger classes of graphs known as k-chordal graphs and partial k-trees. Partial k-trees are characterized in a manner that generalizes the characterization of forests in [Freuder, 82].

Methods have been developed to utilize tree structures in solving general CSPs [Dechter and Pearl, 88]. K-trees now provide us with a range of "target" structures of increasing complexity that we can consider utilizing in similar fashion.

Jeavons has introduced the issue of the expressive power of constraint networks [Jeavons, 89]. I show that the class of partial p-tree structured networks have less expressive power than the class of partial q-tree structured networks for p less than q. Combined with the complexity bounds indicated above, this suggests a tradeoff between computational complexity and expressive power [Levesque and Brachman, 1985].

Another tradeoff arises in utilizing k-trees as target structures in a *partial* constraint satisfaction process [Freuder, 89]. Any CSP can be solved in time exponential in k by removing enough constraints to give the problem a k-tree structure. In the worst case this would necessitate removing a number of constraints equal to the number of edges in a complete graph of n-k vertices.

The 2-tree case is of some special interest. "Minimum

1. This material is based in part upon work supported by the National Science Foundation under Grant No. IRI-8601209 and Grant No. IRI-8913040.

IFI networks" are 2-trees [Wald and Coburn, 83]. "Regular width-2 constraint graphs" are partial 2-trees; this paper improves on the $O(n^3d^3)$ complexity bound for regular width-2 CSPs reported in [Dechter and Pearl, 88]. "Series-parallel graphs" are partial 2-trees [Arnborg, 85].

Many of the basic concepts used here have been rediscovered in different forms in different fields at different times. This is a testimony to their importance, but makes issues of attribution and terminology difficult. [Dechter and Pearl, 88] provides useful background from an AI CSP perspective; [Arnborg, 85] provides useful background from the perspective of work on combinatorial graph algorithms.

The complexity results exploit the connection between CSP theory and combinatorial graph theory. Some background material in these areas is presented in section 2 along with examples of k-trees. The complexity bounds are derived in section 3. Section 4 discusses representational complexity and suggests how k-trees can be utilized even when the original problem does not have k-tree structure. Section 5 summarizes the results.

2 Background

2.1 Constraint Satisfaction Problems

A *binary constraint satisfaction problem* (*binary CSP*) consists of a set of variables, each with a domain of potential values, and a set of binary constraints, which specify pairs of allowable variable values. In a *constraint graph* variables are represented as vertices and constraints as edges.

The n-queens problem is often used as a CSP example. This problem involves placing n queens on an n by n chessboard such that no two queens attack one another. In the constraint graph for this problem each pair of vertices is connected by an edge.

A *solution* to a CSP is an assignment of one value to each of the problem variables such that any pair of values satisfies (is allowed by) the constraint between the corresponding variables. A constraint graph for a CSP *represents* the set of solutions to the CSP. Two constraint graphs are *equivalent* if they represent the same set of solutions.

An *ordered constraint graph* [Freuder, 82] is a constraint graph together with an ordering of the vertices, $v_1 \ldots v_n$. The *width* [Freuder, 82] *of a vertex* in an ordered constraint graph is the number of edges from that vertex back to vertices preceding it in the ordering. The *width of an ordered constraint graph* is the maximal width of its vertices and the *width of a constraint graph* is the minimal width of all the ordered constraint graphs obtained by utilizing every ordering of the vertices.

The well-known backtrack search process is a standard method for solving CSPs. An ordering of the variables of a CSP is *backtrack-free* [Freuder, 82] if backtrack search using that variable order can always find a value for a variable consistent with previous choices of values for variables.

Dechter and Pearl present an *adaptive-consistency* procedure [Dechter and Pearl, 88], which transforms an ordered constraint graph, O, into an equivalent ordered constraint graph, I, with a backtrack-free variable order. I is called the *induced graph* of O. The width of I is called the *induced width of the ordered constraint graph* O. Define the *induced width of a constraint graph* G to be the minimal induced width of all the ordered constraint graphs of G.

The adaptive-consistency procedure from [Dechter and Pearl, 88] is shown below:

```
Adaptive-consistency($X_1$,...,$X_n$)
1. begin
2.    for r=n to 1 by -1 do
3.    compute PARENTS($X_r$)
4.    perform Consistency($X_r$, PARENTS($X_r$))
5.    connect by arcs
        all elements in PARENTS($X_r$)
        (if they are not yet connected)
6     end
```

The X_i are variables in an ordered constraint graph. PARENTS(X_i) returns the set of variables preceding X_i in the ordering that are connected to X_i by an edge in the graph. (Notice that the procedure may add edges to the graph.) The procedure Consistency(v,S) records a constraint on all the variables of S (if S has j variables, this will be a j-ary constraint). The constraint permits an assignment of values to the variables of S only if the assignment is consistent with at least one (consistent) value for v. To determine consistency Dechter and Pearl specify that all constraints be used (old or new) that involve v and a subset (possibly empty) of variables from S. The algorithm basically works to enforce a backtrack-free ordering by adding constraints that prevent one from making choices that will lead to inconsistency.

Define the *adaptive-consistency level* achieved by the adaptive-consistency procedure to be one greater than the maximum size of the PARENTS sets encountered during the running of the algorithm.

An $O(\exp(W^*(\pi)+1))$ bound is given in [Dechter and Pearl, 88] for adaptive-consistency, where $W^*(\pi)$ is the induced width of the constraint graph with ordering π. The bound on the algorithm can be expressed as $O(nd^{W^*(\pi)+1})$. On each of n-1 passes through the outer loop the consistency procedure needs to check that there is a value for X_r consistent with any consistent assignment of values to the parents of X_r. There are at most $W^*(\pi)$ parents. There are at most $d^{W^*(\pi)+1}$ combinations of values for the parents together with X_r. For each of these there are at most $2^{W^*(\pi)}$ constraints to check that involve X_r and some (possibly empty) set of parents.

2.2 Combinatorial Graph Theory

A *graph*, G=(V,E), consists of a set of vertices, V, and a set of edges, E, where each edge is a two-element subset of V. An edge is *incident* to the two vertices in the edge. Two vertices are *adjacent* if they belong to the same edge. A *complete* graph is one that contains all possible edges. The *subgraph* of a graph G *induced by* a subset of vertices V' from G, written G(V'), consists of the vertices V' together with all edges in G incident only to vertices in V'.

Define the *neighborhood* of a vertex v as the set of vertices adjacent to v, and the *degree* of v as the number of vertices in its neighborhood.

A graph G is a *k-tree* [Beineke and Pippert, 71; Rose, 74] if:

1. G has k vertices and is complete

or

2. a. there is a vertex, v, of degree k, whose neighborhood induces a complete graph

and

b. the graph obtained by removing v and all edges incident to it from G is a k-tree.

A *simplicial* vertex of a k-tree is one that satisfies part 2 of the definition. Call the complete graph of k vertices the *trivial k-tree.*

K-trees are structures that can be built as follows: start with a complete graph of k vertices; each time you add a new vertex add an edge between it and k previous vertices that already form a complete subgraph (each pair of vertices is joined by an edge). For example, 1-trees are just trees, and can be built as follows: start with a single vertex, connect each new vertex to one of the previous vertices.

The constraint graph for the n-queens problem is an n-tree; it is also an (n-1)-tree. For example the constraint graph for the 4-queens problem (place 4 non-attacking queens on a 4 by 4 board) is a 3-tree: it can be viewed as a complete graph of 3 vertices (a triangle) plus a fourth vertex connected to each of those three.

Now consider the following variation on the n-queens problem, which I will call the n/m-queens problem. The n/m-queens problem requires only that a given queen cannot attack (or be attacked by) any queen in the previous m rows. The n/m-queens problem has an m-tree constraint graph. The 4/2-queens problem has a 2-tree constraint graph.

A *partial graph* of G consists of the vertices of G and a subset of the edges of G. A subgraph of G is a partial graph of an induced subgraph of G. A *partial k-tree* [Arnborg, 86], is a subgraph of a k-tree; the k-tree is termed an *embedding* of the partial k-tree. Partial 1-trees are forests. Any graph is a partial k-tree for sufficiently large k.

I define a *weak k-tree* as the class of graphs that result when "k" is replaced by "≤k" and "k-tree" by "weak k-tree" in the k-tree definition. The weak k-trees are the k-chordal graphs [Chandrasekharan and Hedetniemi, 88].

3 Computational Complexity

Tree structured problems have been solved with backtrack-free search by achieving a consistency level greater than their width. (Partial) k-tree structured problems are solved here with backtrack-free search by achieving an adaptive consistency level greater than their induced width.

3.1 K-Trees

Dechter and Pearl point out, citing [Arnborg, Corneil and Proskurowski, 87], that determing the induced width, W*, of a constraint graph is an NP-complete problem. However, we will see that for k-trees W*=k (except for the trivial k-tree with k vertices, for which W*=k-1). If the constraint graph is a k-tree it can be recognized as such, if necessary, and an ordered constraint graph with induced width k can be obtained, in O(n) time (regarding k as a constant). Adaptive-consistency can then obtain an equivalent ordered constraint graph with a backtrack-free variable ordering in $O(nd^{k+1})$ time. The backtrack-free search can be completed in O(nd) time. Taken together this provides the promised $O(nd^{k+1})$ bound for solving k-tree structured CSPs.

Arnborg (citing [Rose, 70]) states that k-trees "are easily recognized by a procedure which successively deletes vertices with completely connected neighborhoods" [Arnborg, 85]. I present here an algorithm that performs this recognition for a graph G, and returns an ordering of G with induced width k (k-1 if G is the trivial k-tree) in time linear in the number of vertices, n, for a fixed k.

```
Algorithm W:
Input: A constraint graph G and an integer k.
Output: If G is a k-tree, the algorithm returns an
ordering of G with induced width k (k-1 for the trivial k-
tree); if G is not a k-tree the algorithm reports failure.
  1. Determine the degree of each vertex, store that
     number with the vertex, and place all the degree k
     vertices in a queue K. Sum the vertex degrees as you
     proceed; if and when the sum exceeds $2n-k-k^2$ (twice
     the number of edges in a k-tree), report failure.
  2. For i = 1 to n-k
       If K is empty, report failure.
       If K is not empty:
         Take a vertex v from K.
         If the graph G(N) induced by the neighborhood
         N of v is not complete, report failure.
         If G(N) is complete:
           record v as the (n-i+1)th vertex in the ordering,
           remove v and its incident edges from G,
           subtract one from the degree count associated
           with the vertices in N and place vertices that
           now have degree k in the K queue.
  3. If G is now a complete graph (of k vertices) choose
     the remaining vertices in any order to complete the
     vertex ordering. If G is not complete, report failure.
```

Theorem 1. A k-tree constraint graph with n vertices can be recognized as a k-tree, and an ordered constraint graph of induced width k found (k-1 for the trivial k-tree), in O(n) time.

Proof. Algorithm W does the job. If G is the trivial k-tree, step 3 produces a width k-1 ordering. Adaptive-consistency will not change this width (there are no edges to add). If G is a k-tree with more than k vertices, for any vertex v with degree k the graph induced by the neighborhood of v must be complete, because the definition of k-trees implies that every vertex in a k-tree of more than k nodes is connected to at least k others that induce a complete graph. Any k-tree vertex with a neighborhood that induces the complete graph of k vertices is simplicial [Arnborg, 85; Rose, 70]. Therefore, by part 2 of the k-tree definition, if G is a k-tree it remains one after each pass through step 2. If G is a k-tree after step 2 it must satisfy part 1 of the definition and be the complete graph of k vertices. The ordering produced is such that the parent set of each vertex will already induce a complete graph, therefore adaptive-consistency does not change the width, which remains k.

A k-tree has k(k-1)/2+(n-k)k edges (count the edges as you apply the definition). That and the use of the K queue are the keys to maintaining an $O(nk^2)$ complexity bound for the algorithm. •

Theorem 2. The induced width, W*, of a non-trivial k-tree is k; for the trivial k-tree, W*=k-1. The induced width of a k-tree is equal to its width.

Proof. The trivial k-tree is the complete graph with k vertices, for which the width and the induced width are obviously equal to k-1. Now consider the non-trivial case. Theorem 1 demonstrates that W*≤k. The width of a graph is obviously ≤ its induced width and thus ≤ k. A k-tree with more than k vertices includes a complete graph of k+1 vertices, and regardless of how they are ordered the last one will have k parents. This demonstrates that the width and therefore the induced width are both ≥ k. Thus the width equals the induced width equals k. •

Theorem 3. Given an induced graph I, with induced width k, of an ordered constraint graph of a k-tree structured CSP; the CSP has n variables each with a domain of at most d values. A backtrack-free search using the ordering of I can be conducted with time bound O(nd).

Proof. The adaptive consistency procedure ensures that the ordering of the induced graph is backtrack-free [Dechter and Pearl, 88]. Thus to instantiate each of the n variables, we need at worst try each of the d possible values. For each value we need check at most 2^k constraints involving that variable and subsets of its parents. •

Theorem 4. A k-tree structured CSP can be solved in time $O(nd^{k+1})$.

Proof. Given a constraint graph G for the CSP we can use algorithm W to obtain an ordered constraint graph O with induced width k in O(n) time (Theorem 1). We can then apply adaptive-consistency to obtain an equivalent induced graph in $O(nd^{k+1})$ time, as discussed in section 2. Finally we can carry out a backtrack-free search in O(nd) time (Theorem 3). •

If we do not assume k to be a constant, we can derive an $O(n(2d)^{k+1})$ bound, taking into account the analysis of adaptive consistency presented in section 2.1.

3.2 Partial K-Trees

This section will sketch some extensions of the k-tree results to partial k-trees. It also contains a characterization of partial k-trees in terms of induced width.

Theorem 4 holds for weak k-trees. (As we will see below weak k-trees are partial k-trees; see also [Chandrasekharan and Hedetniemi, 1988]) The key observation (similar to one made by Arnborg, citing Rose [Arnborg, 85; Rose, 70]) is that the basic idea behind Theorem 1 of deleting vertices whose neighborhoods induce a complete graph still works. We can obtain an O(n) bound by modifying Algorithm W to maintain in place of the queue K a queue C of vertices whose adjacent vertices induce a complete graph of ≤k vertices.

If we have an embedding of a partial k-tree in a k-tree, we can go on to use the above methods on the embedding. Every partial k-tree with at least k vertices can be embedded in a k-tree without adding vertices [Arnborg, 86]. Even if we are given an embedding with more vertices than we started with we can simply throw away the extra and still have a weak k-tree (picture an ordering with induced width k, now throw away the extra vertices).

There is unfortunately no similar analog of Algorithm W known for easily recognizing partial k-trees and finding an embedding. There is an $O(n^{k+2})$ algorithm for doing so [Arnborg, 87], and more efficient methods are known for k=2 [Wald and Coburn, 83] and k=3 [Arnborg, 86] (and, of course, for k=1).

Theorem 2 can be extended to the following characterization of partial k-trees, which generalizes the characterization in [Freuder, 82] of forest-structured constraint graphs (partial 1-trees) as those with width≤1. Forests are 1-trees and the width of a 1-tree is equal to its induced width. (Compare with Arnborg's Theorem 3.1(iv) involving the "dimension" concept in [Arnborg, 85].)

The following theorem also serves to identify exactly when we can "induce" width ≤ k, namely when we start with a partial k-tree. K-tree embedding in that case provides a method for obtaining an ordered constraint graph with this induced width. Adaptive consistency, obtained with no more than $O(nd^{k+1})$ effort, then leads to backtrack-free search.

Theorem 5. A constraint graph has induced width≤k iff it is a partial k-tree.

Proof. If the graph has fewer than k vertices the theorem is trivial; assume there are at least k vertices. Suppose the graph, G, is a partial k-tree. It can be embedded in a k-tree with the same vertices. The k-tree will have width k. The partial k-tree's width will be, if anything, less.

Suppose G has induced width w≤k. Let I be an induced graph of an ordered constraint graph of G with induced width w. G is a partial graph of I. I is a weak k-tree. We will see that it can be extended to a k-tree by adding further edges (if necessary), i.e that weak k-trees are partial graphs of k-trees.

First add edges as needed so that the first k vertices in the ordering will induce a complete graph. If the graph has only k vertices we are done. If not add edges to connect the next vertex to the first k, if necessary. This k+1st vertex will now have a parent set that induces the complete graph of k vertices. I will demonstrate inductively that edges can be added as needed so that any remaining vertices will have such parent sets. The resulting graph will clearly satisfy the k-tree definition.

Assume that edges have been added as needed so that the first i vertices in the ordering after the initial k each have a set of parents that induces the complete graph of k vertices. Consider the i+1st vertex, v. Its parent set induces a complete graph (we have a weak k-tree). If its parent set contains k vertices, this step is complete. If its parents all lie within the first k vertices, simply add edges to connect v to all of the first k vertices. Otherwise, consider the parent set of the parent vertex, p, of v that is closest to v in the ordering. The vertex p will have a parent set that induces a complete graph of k vertices, and that parent set will already include the other parents of v. Simply add edges to connect v with as many of the parents of p as needed to bring the total number of parents of v up to k. The resulting set of parents for v will induce a complete graph of k vertices, as they are all in the complete graph of k+1 vertices induced by p and its parents. •

4 Representational Complexity

The representational complexity of k-trees can be analyzed in ways that suggest tradeoffs between representational power and computational complexity. Problems that do not have (partial) k-tree structure for a given k may still be viewed, in whole or in part, in (partial) k-tree terms.

4.1 Representational Power

It is of course in some sense obvious that 2-trees are more "representationally complex" than trees (1-trees), that representational complexity increases with increasing k. Thus the complexity bounds obtained for (partial) k-trees can be viewed as either a) expanding the range of problems for which we can guarantee a relatively efficient solution or b) expanding the representational complexity we can permit in formulating our problems while still maintaining relatively efficient bounds on the effort required to solve the problems.

In temporal reasoning, consideration has been given to restricting the expressive power of the relational language in order to guarantee computational tractability [van Beek, 89]. The results in this paper suggest another means of trading expressivity for tractability, namely restricting expressibility to k-tree structures. For example, by analogy with the n/m-queens example, one could ensure a partial k-tree structure in the constraint graph of the temporal relations in a narrative by insisting that no event could be temporally compared with any event more than k events previous in the narrative. Similar restrictions could be imposed to ensure tractability in scheduling problems.

A more precise notion of representational power permits a more formal analysis. Jeavons [Jeavons, 89] has raised the issue of the "expressive power" of constraint networks. He develops methods for actually counting the number of different solution sets that certain classes of constraint graphs can represent. The following theorem makes a qualitative rather than quantitative statement about the expressive power of partial k-trees.

Theorem 6. Let S_k be the set of solution sets represented by partial k-tree structured constraint graphs. S_p is a proper subset of S_q for p<q.

Proof. Since a partial p-tree is a partial q-tree (Theorem 5) clearly S_p is a subset of S_q. I will construct a partial q-tree structured constraint graph that represents a solution set that cannot be represented by a partial p-tree structured constraint graph.

Consider the following constraint graph, G, with q+1 vertices. G is complete. The complete graph with q+1 vertices is a q-tree. Each variable represented has a domain of q+1 values, the numbers 1 through q+1. A constraint $c(v_i,v_j)$ between variables v_i and v_j specifies that any pair of values is allowed except the pair (i,j).

I will show that the complete graph with q+1 vertices is the only graph that can be used to represent the solution set, S, of this CSP. Therefore, since the complete graph of q+1 vertices cannot be a partial p-tree no partial p-tree can represent S.

Suppose we remove an edge, e, from G, between v_i and v_j for some i and j. Now the constraint c(i,j) cannot be present to prevent solutions that include i for v_i and j for v_j. How can we make up for that? We cannot. No pairs can be removed from the other constraints, as all those pairs are allowed in solutions in the solution set. Adding pairs, or removing other constraints (edges), can only, if anything, increase the number of solutions; CSPs are monotonic in that respect. There are no new constraints to add. Thus the complete graph with q+1 vertices is the only constraint graph that can represent S. •

4.2 Reduction

Various methods have been proposed for expanding the applicability of tree structures in constraint satisfaction problem solving. These methods suggest analogous techniques for utilizing (partial) k-tree structures, even when the original problem does not have such structure. (Of course, the additional complexity permitted by k-tree structure, as opposed to tree structure, should decrease the need for reduction techniques.)

For example, Dechter and Pearl have observed that if we remove enough vertices from an arbitrary constraint graph (along with edges incident to the vertices) we will be left with a tree or forest structure [Dechter and Pearl, 88]. They have studied how arbitrary CSPs can thus be reduced to CSPs with tree (or forest) structure by removing or instantiating variables that comprise a "cycle-cutset" for the constraint graph.

Analogous methods can be used to reduce arbitrary CSPs to k-trees or partial k-trees. In particular, partial 2-trees can be characterized as graphs that do not contain a subgraph homeomorphic to K_4, the complete graph of 4 vertices (i.e. no subgraph is K_4 or K_4 plus additional degree-two vertices) [Arnborg, 85]. Thus we could utilize "K_4-cutsets" to reduce constraint graphs to partial 2-trees.

The cycle-cutset work can be viewed as providing complexity bounds for CSPs in terms of how close their constraint graphs are to having tree or forest structure. In general the complexity of a CSP can be analyzed in terms of the number of variables whose removal would leave a (partial) k-tree structure. For example, the constraint graph for the 4-queens problem can be reduced to a tree by removing two of the four vertices; it can be reduced to a 2-tree by removing a single vertex.

Closeness to k-tree structure can also be measured in terms of constraints. Freuder [Freuder, 89] discusses partial constraint satisfaction problems, which involve weakening the original problem in order to solve an overconstrained problem, or solve a properly constrained problem faster. The following theorem provides an upper bound on how many constraints we have to ignore in order to force a problem into a k-tree structure, thus ensuring that we can solve it in time linear in the number of variables and only exponential in k.

Theorem 7. Any constraint graph, G, can be made into a k-tree structured constraint graph by removing at most a number of edges (constraints) equal to the number of edges in a complete graph of n-k vertices.

Proof. As indicated earlier the number of edges in a k-tree is k(k-1)/2+(n-k)k. Suppose G is a complete graph with n vertices. It then has n(n-1)/2 edges. We can make G into a k-tree by removing (n(n-1)/2)-(k(k-1)/2+(n-k)k)=(n-k)(n-k-1)/2 edges, i.e. the number of edges in a complete graph with n-k vertices. If G is not complete we can first add edges to make it complete (these will correspond to the trivial constraint that allows all pairs, which is normally not included in the constraint graph). We can form a k-tree by removing edges as before (only in the worst case will they all be edges of the original graph). •

For example, the constraint graph for the 4-queens problem can be reduced to a tree by removing three edges. It can be reduced to a 2-tree by removing a single edge.

5 Conclusion

1. The $O(nd^2)$ bound on the complexity of tree structured CSPs generalizes to an $O(nd^{k+1})$ bound for k-trees.

2. CSPs with partial k-tree structure (and every CSP has partial k-tree structure for some k) can be solved in $O(nd^{k+1})$ time once a k-tree embedding has been found (which may require $O(n^{k+2})$ time).

3. The identification of graphs with width ≤1 as forests generalizes to the identification of graphs with induced width ≤k as partial k-trees.

4. Representational power increases with increasing k for partial k-trees.

5. Reduction methods can be used to broaden the applicability of (partial) k-tree results.

Acknowledgments. This work profited from discussion with Pilar de la Torre and Raymond Greenlaw.

References

[Arnborg, 85] Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey, *BIT* 25: 2-23.

[Arnborg and Proskurowski, 86] Characterization and recognition of partial 3-trees, *Siam J. Alg. Disc. Meth.* 7(2).

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

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

[Chandrasekharan and Hedetniemi, 88] Fast parallel algorithms for tree decomposing and parsing partial k-trees, *Proc. of the 26th Annual Allerton Conference on Comm., Cont. and Comp.*, Urbana-Champaign, Illinois.

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

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

[Freuder, 89] Partial constraint satisfaction, *IJCAI-89*.

[Jeavons, 89] The expressive power of constraint networks, Dept. of C.S., Univ. of London, UK.

[Levesque and Brachman, 85] A fundamental tradeoff in knowledge representation and reasoning (revised version), in *Readings in Knowledge Representation*, Brachman and Levesque, editors, Morgan Kaufmann, Los Altos, CA.

[Mackworth and Freuder, 84] The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, *Art. Int.* 25(1).

[Rose, 70] Triangulated graphs and the elimination process, *Journal of Mathematical Analysis and Applications* 32: 597-609.

[Rose, 74] On simple characterizations of k-trees, *Discrete Math* 7: 317-322.

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

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

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