On the Space-Time Trade-off in Solving Constraint Satisfaction Problems*

Roberto J. Bayardo Jr. and Daniel P. Miranker
Department of Computer Sciences and Applied Research Laboratories
The University of Texas at Austin
Austin, Texas 78712
U. S. A.

Abstract

A common technique for bounding the runtime required to solve a constraint satisfaction problem is to exploit the structure of the problem's constraint graph [Dechter, 92]. We show that a simple structure-based technique with a minimal space requirement, pseudo-tree search [Freuder & Quinn, 85], is capable of bounding runtime almost as effectively as the best exponential space-consuming schemes. Specifically, if we let n denote the number of variables in the problem, w^* denote the exponent in the complexity function of the best structure-based techniques, and h denote the exponent from pseudo-tree search, we show $h < (w^* + 1)(\lg(n) + 1)$. The result should allow reductions in the amount of real-time accessible memory required for predicting runtime when solving CSP equivalent problems.

1 Introduction

The constraint satisfaction problem (CSP) is a combinatorial search problem whose occurrence in AI and other domains is well documented. Briefly, a CSP consists of a set of variables to which values must be assigned without violating constraints that disallow certain value combinations. In the general case, the CSP is NP-hard, and worst-case runtime of a CSP algorithm is typically exponential in the number of variables. Improved bounds are achieved on some problems by algorithms that exploit problem specific features. A common technique is to exploit the structure of the problem's constraint graph [Dechter, 92]. The constraint graph of a CSP graphically depicts constraint relationships by representing each variable with a vertex and each constraint with edges connecting the restricted set of variables. In general, the more sparse the constraint graph, the tighter the bound on runtime. For instance, problems with noncyclic constraint graphs can be solved in time linear in the size of the problem [Dechter & Pearl, 87; Bayardo & Miranker, 94].

This research was partially supported by an AT&T Bell Laboratories Ph.D. fellowship.

This paper investigates the extent of a space-time trade-off in solving the CSP. It is often assumed that extensive use of space is necessary to reduce the potentially rad behavior of backtrack search [Seidel, 81]. We show here that this assumption may be ill-founded. We compare the effectiveness of pseudo-tree search [Freuder & Quinn, 85], a polynomial space-consuming technique for solving the CSP, to that of the best structure-based schemes. Dechter [Dechter, 92] has demonstrated that the best structure-based techniques all have worst-case time and space bounds exponential in a parameter known as induced width (w^*). While the exponent in the runtime complexity function of pseudo-tree search (pseudo-tree height, or h) is always greater than induced width for any particular instance, we show that it is always within a logarithmic factor of induced width despite its low space requirement. Specifically, we demonstrate that $h < (w^* + 1)(\lg(n)+1)$.

We foresee the result having applications in real-time AI. Runtime prediction in production systems is sometimes accomplished by improving the complexity of the rule match phase [Barachini, 94]. Tambe and Rosenbloom [1994] show that complexity of the production match phase can be improved using structure-based constraint satisfaction techniques. Our result could allow reductions in the amount of real-time accessible memory required for predicting runtime in hard real-time systems solving problems equivalent to the CSP. Such reductions could prove critical in systems operating on large knowledge bases where domain size (the base of the exponent) is proportional to the amount of knowledge.

The paper begins with a more formal definition of the CSP and constraint graph, and provides a description of backtrack search for solving the CSP. We then review the concepts of pseudo-tree search and pseudo-tree arrangements of a constraint graph. The next section presents the definitions of induced graph and relates it to partial k-trees. Finally, we establish the above-mentioned relation between pseudo-tree height and induced width and close with concluding remarks. The paper assumes the reader is familiar with elementary graph concepts and terminology [Even, 79].

2 Constraint Satisfaction Problems and Backtracking

A *constraint satisfaction problem* (CSP) is a set of *variables* and a set of *constraints.* Each variable is associated with a finite value domain, and each constraint consists of a subset of the problem variables called its *scheme* and a set of mappings of domain values to variables in the scheme. An *assignment* is a mapping of values to a subset of the problem variables where any value mapped to a particular variable belongs to the domain of the variable. An assignment *A satisfies* a constraint C with scheme *X* if *A* restricted to the variables in *X* is a mapping in *C.* A *partial solution* to a CSP is an assignment that satisfies every constraint whose scheme consists entirely of variables mentioned in the assignment. A *solution* to a CSP is a partial solution mentioning every variable.

The *constraint graph* of a CSP has a vertex for each variable and the property that the variables in the scheme of a constraint are completely connected. This is referred to as the *primal-constraint graph* in [Dechter, 92] when there are constraints with more than two variables in their schemes. Without loss of generality, we assume problems have connected constraint graphs. If it were otherwise, we could simply view the disconnected instance as a set of instances, one for each connected component.

A naive method for solving constraint satisfaction problems is *chronological backtrack.* Chronological backtrack maps values to variables along an ordering of the variables until some constraint is violated by the working assignment. When a constraint is violated, the variable most recently to have been assigned a value is assigned another value from its domain. If ever the values in the domain of a variable are exhausted, a *backtrack* takes place to the previous variable along the ordering. The procedure continues until either the last variable is successfully assigned a value (in which case a solution has been found), or until all values from the domain of the initial variable are exhausted (in which case no solution exists). If we let *k* bound the number of values in any domain and *n* denote the number of variables, then chronological backtrack has a runtime complexity of 0(exp(w)) because a variable can be assigned a value up to *k* times.

3 Pseudo-tree Arrangements and Pseudo-tree Search

Freuder and Quinn [1985] introduce the concept of a pseudo-tree arrangement of a graph. A *pseudo-tree arrangement* of a graph is a rooted tree with the same set of vertices as the original graph and the property that adjacent vertices from the original graph must reside in the same branch of the rooted tree (hereafter called the pseudo tree). A branch is simply a path from the root to some leaf. The concept is illustrated in Figure 1, where a vertex in the rooted tree corresponds to the vertex directly above it in the original graph. The original edges from the graph appear dashed in the pseudo tree to illustrate that adjacent vertices always appear within the same branch. Note that while a depth-first search tree is a pseudo tree, a pseudo-tree is not necessarily a depth-first search tree. For instance, the pseudo tree appearing in the figure is not a depth-first search tree.

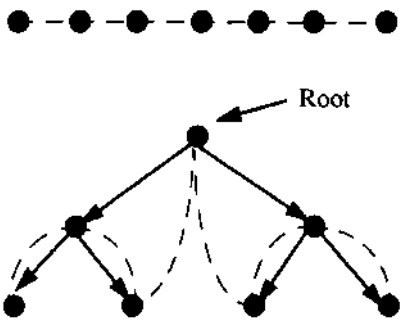

FIGURE 1. A height-3 pseudo-tree arrangement of a chain with 7 vertices.

FIGURE 1. A height-3 pseudo-tree arrangement of a chain with 7 vertices.

Pseudo-tree search [Freuder & Quinn, 85] exploits pseudo-tree arrangements, and has a worst-case complexity function that is exponential in the height of the pseudo tree *(h*). Rather than explain pseudo-tree search in its full detail, we instead describe how to modify chronological backtrack so that it exploits pseudo trees in a similar fashion. Given a pseudo-tree arrangement of the problem's constraint graph, we order the variables of the problem according to a depth-first traversal of the pseudo tree.[1] Now, whenever a backtrack is necessary, instead of backing up to the previous variable in the ordering, we backtrack to the pseudo-tree parent of the current variable, possibly skipping over many variables in the process. This backtrack policy does not sacrifice completeness because, due to the structure of the pseudo tree, the assignments made to the skipped variables are irrelevant with respect to the current failure. We call this algorithm *pseudo-tree backtrack.*

Given the pseudo-tree arrangement, pseudo-tree backtrack has runtime complexity *0(exp(h))* since each variable can be assigned a value a maximum of k^h times. The space requirement of this technique is a mere *0(n)* beyond the size of the problem input to maintain domain value iterators and the variable ordering.

In order for pseudo-tree backtrack to be effective at bounding runtime, it requires a pseudo-tree arrangement with shallow height. An efficient algorithm for finding the

Note that the variable ordering need not be static since there are typically many different depth-first orderings. Dynamic variable ordering involves modifying the unassigned portion of the ordering during backtrack, and can improve average-case performance [Haralick & Elliot, 80; Frost & Dechter, 94],

minimum-height pseudo-tree is not known. Though a proof eludes us, we suspect the problem is NP-hard due to its similarity to various well-known NP-hard problems including minimizing depth-first search tree height.

Freuder and Quinn [1985] provide the following method for heuristically constructing a pseudo-tree arrangement: Find a small set of vertices *S* whose removal from the constraint graph disconnects it (such a set is called a *cutset).* These vertices will form the first \S\ levels of the tree. The remaining levels are formed by recursively applying the procedure to the remaining graph components in order to spawn branches (one for each component) beginning at level |5| + 1 . It is easy to verify that the resulting structure is indeed a pseudo-tree arrangement. The pseudo-tree from Figure 1 can be constructed in this manner by selecting the middle vertex in each chain at each step to disconnect the remaining graph.

Depth-first search can also be regarded as a heuristic technique for pseudo-tree arranging a graph. Although, we have found it unlikely to find an arrangement with small height even when additional heuristics are applied. For example, consider an *n* vertex chain graph. Any depth-first search tree has height at least !~, whereas a pseudo-tree arrangement always exists with neight at most lg(n) + 1 (e.g. as implied by Figure 1).

4 Induced Width

Dechter [1992] demonstrates that the best structure-based algorithms for solving the CSP are exponential-in both time and space—in a constraint graph parameter called induced width *(w*).* The actual exponent in the complexity function of these schemes, e.g. adaptive consistency [Dechter, 92], is w* + 1 . The parameter is obtained from the constraint graph after imposing a variable ordering on which the algorithm operates. A *child* of a vertex in a graph with an ordering of its vertices (an *ordered graph)* is an adjacent vertex that follows it in the ordering. A *parent* is an adjacent vertex that precedes it. The *induced graph* of an ordered graph G is an ordered graph with the same ordered set of vertices as G and the smallest set of edges to contain the edges of *G* and enforce the property that any two vertices sharing a child are adjacent. We can build the induced graph of *G* by iteratively connecting any nonadjacent vertices that share a child. Finally, the *induced width* of an ordered graph G is the maximum number of parents of any vertex in the induced graph of G.

Figure 2 illustrates the process of creating the induced graph. The ordering is assumed to be from top to bottom. Edges added to form the induced graph are dashed. Note that the graph has an induced width of 3.

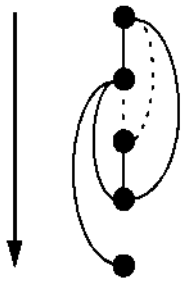

FIGURE 2. Constructing the Induced Graph

Like schemes exploiting pseudo trees, the effectiveness of techniques exponential in induced width depends upon the quality of the constraint graph arrangement. Minimizing induced width is NP-hard [Arnborg, 85], so heuristic techniques are typically applied to produce the ordering.

We can always order a graph so that its induced width is less than the height of any pseudo-tree arrangement. Given a pseudo-tree arrangement of a graph *G* with height *h*, ordering the vertices of *G* according to a depth-first traversal of the pseudo tree produces an ordered graph with induced width $w^* < h - 1$. This is because when creating the induced graph from such an ordering, no edge can be added between vertices in different branches of the pseudo tree. The number of parents of any vertex in the induced graph is thus bounded by the number of its pseudo-tree ancestors.

The above establishes that techniques exponential in induced width can always be made as effective in bounding runtime as techniques exploiting pseudo trees. In fact, for certain classes of problems, induced width is usually better. For instance, any noncyclic graph can be ordered to have an induced width of 1, but pseudo-tree arrangements of chain graphs must increase at least logarithmically with the number of vertices.

In the next section, we bound how much more effective induced width techniques are in the general case. Before proceeding, we review a useful fact relating induced width to k-trees. A graph is a *k-tree* [Beineke & Pippert, 71] if:

- the graph has *k* vertices and is complete (said to be a *trivial* k-tree), or
- there is a vertex of degree *k* whose neighborhood induces a complete graph, and the graph obtained by removing the vertex is a k-tree.

Note that a connected noncyclic graph is a 1-tree. A *partial k-tree* is a partial graph of a k-tree. The following is due to Freuder[1990]:

THEOREM 4.1: An ordered graph with induced width w* is a partial *w**-tree.

5 Relating Pseudo-Tree Height and Induced Width

We now show that given an ordered graph, we can construct a pseudo-tree arrangement of the graph where h is within a logarithmic factor of w*. This implies that no matter how effective we can make induced width techniques solve particular classes of problems, we can make pseudo-tree search solve them almost as effectively without the added burden of an exponential space requirement.

The idea behind our approach is to generalize the case for 1-trees to that of general graphs by making use of k-tree embeddings. It is easy to establish that a 1-tree of n vertices can be split into components that contain at most u vertices by removing a single vertex. We can therefore generate a pseudo-tree arrangement with $h < \lg(n) + 1$ of any 1-tree using the heuristic arrangement method from Section 3. The idea is to always select for the cutset a vertex whose removal splits the remaining graph into components whose sizes (number of vertices) are at most half that of the original. Since such a vertex always exists, splitting recurses at most lg(rt) + 1 levels deep, generating a pseudo-tree arrangement with height at most \g{n) + 1 .

We now generalize to arbitrary graphs. Consider an ordered graph G with induced width w^*. We know from theorem 4.1 that this graph can be embedded into a w^*-tree with the same number of vertices. We establish how to find the appropriate vertices for splitting k-trees, thereby allowing us to generalize the previously described algorithm for pscudo-tree arranging 1-trees.

A clique of size /' is said to be *adjacent* to another clique of size / if they share /' - 1 vertices. Define depth-first search on a non-trivial k-tree to traverse adjacent $k + 1$ cliques instead of adjacent vertices. Figure 3 illustrates a depth-first search tree resulting from traversing a 2-trec in such a manner. We refer to such a depth-first search tree as the *clique tree* of the k-tree. While there may be more than one clique tree depending on where the depth-first search begins and what tie-breaking rule is applied, any clique tree is sufficient for the upcoming claims.

Each node of the clique tree corresponds to some clique in the k-tree, and the edges represent their interconnection. We now state two properties of clique trees, leaving the proofs as exercises. Given a non-trivial k-tree G with n vertices and a clique tree T of G :

- T has exactly n-k vertices.
- Given a vertex v whose removal from T leaves behind connected components of size j, the clique represented by v, when removed from G, leaves behind components of size at most j.

We next use these properties to establish the following lemma:

LEMMA 5.1: Given a non-trivial k-tree G with n vertices, there exists $k + 1$ vertices in G whose removal leaves behind connected components of size at most 2.

Proof: We have already noted that a 1-tree can always be broken into components of a size at most half that of the original graph by removing a single vertex. From a clique tree T of the k-tree, we can therefore find a vertex v whose removal splits T into components of size n-k/2 < n. The $k + 1$ vertices in the clique represented by v, when removed from the k-tree, must therefore leave behind connected components of size at most n. D

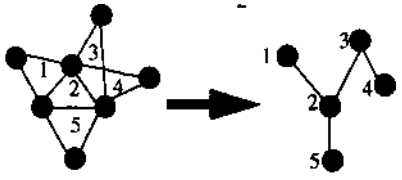

FIGURE 3. A 2-tree and a possible clique tree.

This method for breaking apart k-trees allows us to bound the size of the cutset required for splitting arbitrary graphs and the size of the resulting components:

COROLLARY 5.2: Given an ordered graph G with induced width w^*, $n > w^*$, there exists $w^* + 1$ vertices in G whose removal leaves behind components with size at most n/2.

Proof: Theorem 4.1 tells us we can embed the graph into a w^*-tree with the same number of vertices. Because $n > w^*$, the w^*-tree is non-trivial and the claim thus follows immediately from lemma 5.1. D

We lastly apply the above fact to define a pseudo-tree arrangement procedure, thereby allowing a bound on pseudo-tree height.

THEOREM 5.3: Given an ordered graph with induced width w^*, there exists a pseudo-tree arrangement of the graph with $h < (w^* + 1)(\lg w + 1)$.

Proof: For $n - w^*$, the claim is trivially satisfied by linearly arranging the vertices. Otherwise, by Corollary 5.2, we can find w* + 1 vertices whose removal leaves behind components with less than half the vertices of the original graph. We can use such a set to form the first $w^* + 1$ levels of a pseudo-tree. The resulting components must have induced width of w^* or less (they are simply subgraphs of the original graph). Thus, the remaining variables can be pseudo-tree arranged by applying the procedure recursively on each component. A new pseudo-tree branch is thereby spawned for each remaining component as is done by the heuristic pseudo-tree arrangement procedure from Section 2.

Now, each stage of the recursion leaves behind components whose size are at most half that of the previous graph. Therefore, recursion depth can be bounded by lg(n) + 1 . Since each level of the recursion adds at most

$w^* + 1$ new levels to the tree, the resulting pseudo tree has height at most $(w^* + 1)(\lg(n) + 1)$. □

Theorem 5.3 tells us we can always make the exponent in the complexity function of pseudo-tree backtrack within a logarithmic factor of that of the best exponential space-consuming schemes. Our proofs are constructive and describe a polynomial time-bounded procedure for accomplishing the task. While we did not describe a (polynomial time) procedure for finding the k-tree embedding, such a procedure appears in [Freuder, 90].

6 Conclusions

We have demonstrated that while the best structure-based techniques require exponential space, they are capable of bounding worst-case performance only slightly better than pseudo-tree search, a simple polynomial space-bounded scheme. Attempting to trade space for time is therefore of limited benefit. Interestingly, we have a similar case with respect to average case performance as well. For example, Frost and Dechter [1994] have found that algorithms performing unlimited constraint recording during search (thereby consuming exponential space) barely outperform polynomial space-bounded constraint recording algorithms.

We lastly note the open problem of either (a) finding a polynomial space algorithm that runs in time exponential in induced width, or (b) proving no such algorithm exists. Current approaches for achieving the runtime bound fundamentally require exponential space since they record high-arity constraints or generate all solutions to certain subproblems. On the other hand, proving no such algorithm exists seems like a P = NP related task. We therefore feel that solving the problem will be difficult if not impossible.

References

[Axnborg, 85] Arnborg, S., Efficient algorithms for combinatorial problems on graphs with bounded decomposability-a survey, BIT 25, 2-23, 1985.

[Barachini, 94] Barachini, F., Frontiers in run-time prediction for the production-system paradigm, In AI Magazine, 15(3), 47-61, Fall 1994.

[Bayardo & Miranker, 94] Bayardo, R. J. and Mi ranker, D. P., An optimal backtrack algorithm for tree-structured constraint satisfaction problems. Artificial Intelligence, 71,159-181, 1994.

[Beineke & Pippert, 71] Beineke, L. W. and Pippert, R. E., Properties and characterizations of k-trees, Mathematika 18, 141-151, 1971.

[Dechter, 90] Dechter, R., Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition, Artificial Intelligence 41(3), 273-312, 1990.

[Dechter, 92] Dechter, R., Constraint Networks, Encyclopedia of Artificial Intelligence, Second Edition, 276-285, 1992.

[Dechter & Pearl, 87] Dechter, R. and Pearl, J., Network-based heuristics for constraint-satisfaction problems, Artificial Intelligence 34, 1-38, 1987.

[Dechter & Pearl, 89] Dechter, R. and Pearl, J., Tree clustering for constraint networks, Artificial Intelligence 38, 353-366, 1989.

[Even, 79] Even, S., Graph algorithms (Computer Science Press, Rockville, MD, 1979).

[Freuder, 82] Freuder, E.C., A sufficient condition for backtrack-free search, J. ACM 29(1), 24-32, 1982.

[Freuder, 90] Freuder, E.C., Complexity of k-tree structured constraint satisfaction problems, In Proceedings ofAAAI-90,4-9, 1990.

[Freuder & Quinn, 85] Freuder, E.C. and Quinn, M.J., Taking advantage of stable sets of variables in constraint satisfaction problems, In Proceedings ofIJCAI-85, 1076-1078, 1985.

[Frost & Dechter, 94] Frost, D. and Dechter, R., Dead-end driven learning, In Proceedings ofAAAI-94, 294-300, 1994.

[Haralick & Elliot, 80] Haralick, R. M., and Elliot, G.L., Increasing tree search efficiency for constraint satisfaction problems, Artificial Intelligence 14,263-313, 1980.

[Seidel, 81] Seidel, R., A new method for solving constraint-satisfaction problems, In Proceedings of UCAI-81, Vancouver, B.C., Canada, Morgan-Kaufmann, San Mateo, Calif., 338-341, 1981.

[Tambe & Rosenbloom, 94] Tambe, M. and Rosenbloom, P. S., Investigating production system representations for non-combinatorial match, Artificial Intelligence 68, 155-199, 1994.