Some Applications of Graph Bandwidth to Constraint Satisfaction Problems

Ramin Zabih
Computer Science Department
Stanford University
Stanford, California 94305

Abstract

Bandwidth is a fundamental concept in graph theory which has some surprising applications to a class of AI search problems. Graph bandwidth provides a link between the syntactic structure of a constraint satisfaction problem (CSP) and the complexity of the underlying search task. Bandwidth can be used to define a new class of easy CSP's, namely those that have limited constraint graph bandwidth. These CSP's can be solved in polynomial time, essentially by divide and conquer. This in turn suggests that bandwidth provides a mathematical measure of the decomposability of a search problem. In addition, bandwidth supplies a measure for comparing different search orderings for a given CSP. Statistical analysis suggests that backtracking with small bandwidth orderings leads to a more efficient search than that obtained under orderings with larger bandwidths. Small bandwidth orderings also limit the pruning that can be done by intelligent backtracking. If small bandwidth orderings are indeed advantageous, then a large number of heuristics developed in numerical analysis to find such orderings may find applicability to solving constraint satisfaction problems.

1 Introduction

The bandwidth of an ordering of a graph is the maximum distance between two adjacent vertices, and the bandwidth of a graph is its minimum bandwidth under any ordering.[1] Bandwidth is one of the basic concepts in graph theory, and is related to almost every other mathematical property that graphs possess [**?**]. It is nonetheless surprising to discover some strong connections between graph bandwidth and a class of search problems encountered in AI.

The search problems known as constraint satisfaction problems (CSP's) have an associated constraint graph. The vertices of the graph consist of the variables of the search problem, and there is an edge between two vertices if there is a (non-trivial) constraint between those variables.

The bandwidth of the constraint graph is strongly related to the complexity of the underlying constraint satisfaction problem. In particular, there is evidence for two claims.

- The bandwidth of the constraint graph of a constraint satisfaction problem serves as a measure of its decomposability.

- The bandwidth of a search ordering provides a measure of its quality, as backtracking with a small bandwidth ordering generally results in a smaller search tree.

The first claim is supported by a proof that any problem of limited bandwidth can be solved in polynomial time, essentially by divide and conquer. The second claim is supported by statistical analysis, and by evidence that small bandwidth orderings have additional important properties related to intelligent backtracking. If small bandwidth orderings are indeed useful, then the large body of heuristics that has been developed by numerical analysts for finding such orderings may prove to be useful for solving CSP's.

After a brief review of constraint satisfaction problems, section 3 formally defines bandwidth and relates it to some other graph-theoretic notions in the CSP literature. Section 4 proves that any CSP whose bandwidth can be limited is solvable in polynomial time. Section 5 presents statistical data which suggests that small bandwidth orderings are generally superior, as well as describing some preliminary results about the interaction between such orderings and intelligent backtracking.

[1] These definitions will be made more precise in section 3.

2 Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) has a set of variables and a domain of values,

$$V = \{v_1, v_2, \ldots, v_n\} \text{ the set of variables, } \|V\| = n,$$
$$D = \{d_1, d_2, \ldots, d_m\} \text{ the set of values, } \|D\| = d.$$

Every variable v_i must be assigned a value d_k.

A CSP also consists of some constraints saying which assignments are compatible. Most interesting problems are *binary* CSP's, where the constraints involve pairs of variables. Such a constraint is a (proper) subset of $D \times D$ consisting of the simultaneously permitted assignments. A binary CSP has an associated constraint graph $G = (V, E)$, where

$$E = \{ (v_i, v_j) \mid \text{there is a contraint between } v_i \text{ and } v_j \}.$$

Note that the constraint graph hides a great deal of the information about the search problem, particularly the tightness of the constraints.

The constraint graph G may be assumed to be connected, as otherwise each connected component can be considered separately. Many standard combinatorial problems are binary CSP's, including graph coloring and the n-queens problem.

3 Graph Definitions

Given a graph G with n vertices, an ordering h is a one-to-one map from the vertices of G to the set $\{1, \ldots, n\}$. The bandwidth of a vertex v under an ordering h is the maximum value of $|h(v) - h(w)|$ over all vertices w connected to v. The bandwidth of a graph under an ordering is the maximum bandwidth of any vertex, and the bandwidth of a graph is its minimum bandwidth under any ordering. The bandwidth of G under h will be written as $B(G, h)$.

3.1 Relating bandwidth and other graph properties

There are many notions akin to bandwidth which have been developed in the CSP literature. It is straightforward to determine the relationship between bandwidth and three graph properties: front length (introduced by [**?**]), width [**?**], and induced width [**?**].

Seidel [**?**] defines an invasion of G to be a sequence of subgraphs $G_1, \ldots, G_n$ where G_i is a subgraph of G_{i+1} with i vertices and where $G_n = G$. For a given invasion, a front F_i consists of the vertices in G_i that are adjacent (in G) to vertices not in G_i. The front length of an invasion is the maximum size of F_i. There is an invasion associated with any ordering h, defined by $G_1 = h^{-1}(1)$, $G_{i+1} = G_i \cup h^{-1}(i+1)$. The front length of G under the invasion associated with h will be written as $F(G, h)$.

Freuder [**?**] defines the width of a vertex v under an ordering h to be the number of vertices that are connected to v which occur earlier than v under h. The width of a graph under an ordering is the maximum width of any vertex. The width of G under h will be written as $W(G, h)$.

Dechter and Pearl [**?**] construct the induced graph of G under an ordering h by processing G's vertices in the order $\{h^{-1}(n), \ldots, h^{-1}(1)\}$, and adding an edge between any two neighbors of a vertex v that precede v under h. The induced graph of G under h will be written as G^h.

It is straightforward to analyze the relationships among these quantities. First, for a fixed ordering h, define

$$P_i \quad \stackrel{\text{def}}{=} \quad \{h^{-1}(i - B(G,h)), \ldots, h^{-1}(i-1)\}$$

(assuming $h^{-1}(j) = h^{-1}(1)$ if $j < 1$ for simplicity). The important property of P_i is the following: if there is an edge between v and w, where v occurs before $h(i)$ and w does not, then $v \in P_i$.

Theorem 3.1 *For any graph G and ordering h $B(G, h) \geq F(G, h)$.*

Proof: Consider a front F_i in the invasion defined by h. Clearly $F_i \subseteq P_{i+1}$. But $\|P_{i+1}\| \leq B(G, h)$. □

Theorem 3.2 *For any graph G and ordering h $B(G, h) \geq W(G, h)$.*

Proof: All of the previous vertices that $v = h(i)$ is connected to must be in P_i. □

Theorem 3.3 *For any graph G and ordering h $B(G, h) \geq W(G^h, h)$.*

Proof: The edges added when $v = h(i)$ is processed will be between two elements of P_i. Adding such edges will never increase the bandwidth, so $B(G^h, h) = B(G, h)$. Applying theorem 3.2 with G replaced by G^h gives the desired result. □

Note that the above three inequalities cannot be made strict (i.e., it is invalid to replace '$\geq$' by '$>$').

Theorem 3.4 *There exists a graph G and an ordering h such that $B(G, h) = F(G, h) = W(G, h) = W(G^h, h)$.*

Proof: Consider a graph G that is a triangle. Then, for any ordering h, $B(G, h) = F(G, h) = W(G, h) = W(G^h, h) = 2$. □

4 Limited Bandwidth CSP's

There is no technical definition of what it means for a search problem to be "local" or "nearly decomposable". The intuition behind these terms, however, is that certain search problems can be solved piece by piece, without taking the entire problem into consideration at each point in the search. Coloring a long and narrow graph is an example of such a search problem.

There are several reasons to believe that the bandwidth of a CSP's constraint graph reflects the locality of the search problem. First, note that edges in the constraint graph measure strong interactions between variables. If there is no edge between two variables, then assigning one a value can have no (direct) effect upon the other, so the lack of an edge reflects a sort of independence. If the CSP has limited bandwidth, each vertex in the constraint graph can have no more than a bounded number of neighbors. This suggests that small bandwidth graphs should be solvable by only worrying about a small subset of the variables at any instant.

Additional anecdotal evidence to support this conclusion can be obtained by visual inspection of graphs of various bandwidths. Graphs with small bandwidths tend to look long and thin. The highest bandwidth graph, on the other hand, is the complete graph where there is an edge between every pair of vertices.

A stronger argument can be made on the basis of the claim that nearly decomposable search problems should be easy to solve. In particular, it should be possible to solve them efficiently by solving their subparts more or less independently (gluing the subparts back together into a solution by divide and conquer or by dynamic programming). It turns out that any CSP of limited bandwidth can be solved in polynomial time by dynamic programming.

4.1 Solving bandwidth-limited CSP's in polynomial time

The basic strategy is to find an ordering with minimal bandwidth, and then to use this ordering to solve the CSP. In general, determining the bandwidth of a graph is an NP-complete problem [**?**]. However, if the graph's bandwidth is no larger than b, a minimal bandwidth ordering can be found in time $O(n^b)$ and also space $O(n^b)$. This can be done via an algorithm of Saxe [**?**] as improved by Gurari and Sudborough [**?**].

Given an ordering h with bandwidth no larger than b, the CSP can then be solved via dynamic programming. There are three possible algorithms, with slightly different bounds. Because of the relationships between bandwidth, front length and induced width that are proven in section 3, either Seidel's or Dechter and Pearl's method may be used. In addition, Fernández-Baca has recently produced an algorithm that makes explicit use of bandwidth [**?**]. However, Dechter and Pearl's algorithm gives the bound that seems likely to be lowest in practice.

$B(G,h)$	N	Tree size
8	84	15832 ± 1255
9	628	18271 ± 528
10	1850	19855 ± 379
11	2793	21733 ± 382
12	2908	22703 ± 405

Figure 1: Average tree size for a graph-coloring problem, with 99% confidence intervals.

Dechter and Pearl [**?**] show how to solve a CSP under an ordering h with induced width $w = W(G^h, h)$ in time $O(d^{w+1})$ and space $O(d^w)$. Applying theorem 3.3, $w \leq b$. Combining this with Saxe's algorithm gives the desired result.

Theorem 4.1 *Any CSP whose constraint graph's bandwidth is no larger than b can be solved in time* $O(n^b + d^{b+1})$ *and space* $O(n^b + d^b)$.

One corollary of this theorem is a very simple proof of a result first discovered by Monien and Sudborough [**?**].

Corollary 4.2 *Any graph of limited bandwidth can be colored in polynomial time.*

5 Small Bandwidth Orderings

Since small bandwidth CSP's are easy to solve, the obvious generalization would be to claim that small bandwidth orderings are generally efficient. In fact, the method for solving small bandwidth CSP's described above first finds a small bandwidth ordering (by Saxe's algorithm), and next uses that ordering to solve the search problem. This suggests that there is something special about small bandwidth orderings.

There is both empirical and theoretical evidence that small bandwidth orderings have special properties. Empirically, there is statistical evidence that using small bandwidth orderings for backtrack search is advantageous. There are also theoretical evidence that links small bandwidth orderings with intelligent backtracking.

5.1 Empirical results

Statistical analysis of some preliminary experimental results suggest that small bandwidth orderings are indeed advantageous. The problem that has been examined is the graph-coloring problem described in

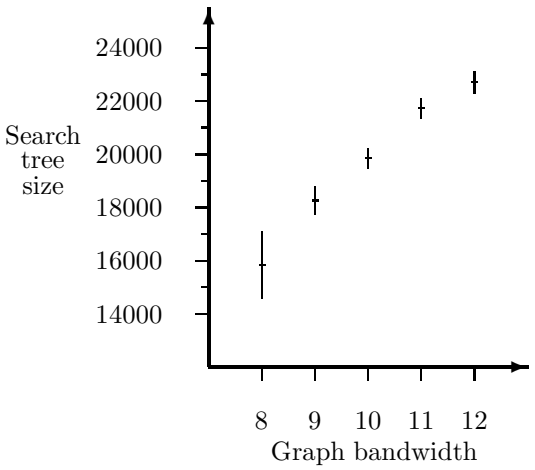

Figure 2: Average tree size as a function of ordering bandwidth, with 99% confidence intervals.

[**?**]. This CSP has 1176 solutions. Depth-first search (sometimes referred to as backtracking, or chronological backtracking) has been run on the problem several thousand times, under orderings with different bandwidths.

Figure 1 shows the average number of nodes in the search tree as a function of the bandwidth of the ordering, together with the 99% confidence intervals. The graph in figure 2 presents the same data in a slightly more readable form. The data supports the hypothesis that the search trees resulting from small bandwidth orderings are significantly smaller than those resulting from orderings with greater bandwidth.

More precisely, the data suggests that as the bandwidth of the ordering increases, the search tree gets larger. Furthermore each increment in bandwidth results in a corresponding increase in search tree size which is statistically significant at the 99% confidence level. For this problem at least, it is not merely the case that the very smallest bandwidth orderings are better han the very largest.

Of course, one would like to have evidence from more than one problem. The large number of trial runs necessary to obtain statistically significant results, however, makes this something of a challenge.

5.2 Ordering bandwidth and intelligent backtracking

Why should it be advantageous to use a small bandwidth ordering? Suppose that a CSP is solved by chronological backtracking under an ordering with bandwidth b. Such a process will construct a search tree in the standard manner: the nodes of the search tree will consist of labelings which assign values to some subset of the variables, the root node will consist of the empty labeling, and the children of a node will assign a value to one variable that was not assigned a value by the parent node. Assuming that the CSP is solved under a fixed ordering, each level in the search tree will correspond to a single variable which is assigned a value at that depth in the tree.[2] Some of the leaf nodes of the search tree will be solutions to the CSP; the remaining leaf nodes are failures, because all the possible values of some unassigned variable are incompatible with the values of the variables that have been assigned values.

At a given node of this search tree, the corresponding labeling will have assigned values to certain variables. However, only the most recent b variables to have been assigned values can have any effect on the remainder of the search. Any variables that were assigned values earlier will have no edges in the constraint graph connecting them to variables that do not yet have values, and hence will have no effect.

More precisely, consider two labelings at the same level in the search tree. If these two labelings assign the same values to the most recent b variables, then the subtrees underneath the two labelings will be isomorphic. This is the property that the dynamic programming schemes mentioned in section 4.1 exploit.

This is also an advantageous property for doing chronological backtracking. In a small bandwidth ordering, at any point in the search the only variables that matter are a small number of recent ones.

When using backtrack search, the decision actually responsible for a failure can occur significantly before the failure itself is detected. This is a well-known malady of backtracking which can lead to extremely poor performance. The greater the number of intervening decisions, the worse backtracking performs.

Various schemes have been proposed to solve this problem. These schemes mostly work by invoking some sort of failure analysis to determine where responsibility for a failure lies. The results of this analysis are then used to discard portions of the search tree which can be shown to contain no solution.

The two most popular such schemes are *dependency-directed backtracking* [**?**] and *intelligent backtracking* [**?**]. The major difference involves how much pruning of the search tree is done. Dependency-directed backtracking achieves a much greater reduction in search tree size, at the cost of a potentially exponential use of

[2] It is rather straightforward to extend all of the results in this paper to rearrangement search strategies such as [**?**]. The bandwidth of a dynamic search ordering is simply the maximum of the bandwidths of the orderings used down any branch of the search tree.

storage space. Intelligent backtracking uses very little space, but does less pruning.

Researchers who have studied dependency-directed backtracking or intelligent backtracking have generally suspected that these schemes have some relationship with search order. For instance, it is very difficult to construct an example of a problem where dependency-directed backtracking is helpful without requiring that the problem be solved in a particular order. There is usually some other ordering which solves the problem without running into this problem. One example is the sample problem described by [**?**, page 136].

Suppose that a search problem is being solved by chronological backtracking under an ordering with bandwidth b. At any point in the search, only the last b variables that have been assigned values matter. A failure occurs because all the values at a particular variable v_i have been eliminated. The responsible variables must be among the last b to be assigned values, as no other variables have an edge connecting them with v_i.

This strongly suggests that the number of intervening decisions between the choice responsible for a failure and the failure itself should be bounded by b. It is possible to prove a theorem that supports this intuition.

The theorem applies to intelligent backtracking schemes, which are in fairly wide use in the PROLOG community. Bruynooghe [**?**] gives one intelligent backtracking algorithm for solving CSP's. There are numerous other very similar schemes, such as that given in [**?**]. The statement of the theorem, which applies to all these schemes, is as follows.

If a CSP is searched in a fixed order using intelligent backtracking, the bandwidth of that ordering provides a bound on the amount of the search tree that intelligent backtracking will prune. This bound holds at almost all nodes in the search tree; the only exception is when intelligent backtracking declares the problem to be insolvable. It is possible to prove restrictions on the nodes in the search tree where these exceptions can occur, in terms of the k-consistency [**?**] of the original CSP.

This theorem and some related results will be discussed in detail in a forthcoming paper. They provide an interesting relationship between search ordering and intelligent backtracking. Intelligent backtracking and search order are clearly not orthogonal. Choosing a good (i.e., small bandwith) ordering reduces the advantages of using intelligent backtracking. A reasonable explanation would be that choosing a small bandwidth ordering ameliorates the problem that intelligent backtracking is designed to solve. This explanation is consistent with the statistical evidence that small bandwidth orderings are useful for (chronological) backtracking.

5.3 Bandwidth and adjacency

One final interesting property of small bandwidth orderings is their relationship to the "adjacency" heuristic. This is one of the simplest and most standard ordering heuristics, which is especially common for graph-coloring problems. Adjacency simply means that the ordering starts at one edge of the graph and looks at new vertices that are adjacent (i.e., connected) to a vertex that has already been examined.

The adjacency heuristic is clearly related to minimizing ordering bandwidth. A small bandwidth ordering, by definition, will examine all of a vertex v's neighbors soon after it examines v. It is not always the case that a small bandwidth ordering will next examine a vertex adjacent to those already examined. However, it is usually true, and in any event the next vertex will be close to the previously examined vertices.

5.4 Consequences

If it is indeed true that small bandwidth orderings are in general efficient, then there is a large set of algorithms developed by numerical analysts that can be applied to CSP's. Bandwidth minimization, as mentioned, is NP-complete. However, because it is an important problem in the field of numerical analysis, many heuristic procedures for finding small bandwidth orderings have been developed. Most of these have relatively low time overhead. [**?**] surveys some of these algorithms; perhaps the most popular one is [**?**].

These algorithms can be applied to CSP's to yield small bandwidth orderings. The CSP may then be solved using chronological backtracking under that ordering; or, as an alternative, it can be solved using the dynamic programming methods mentioned in section 4.1.

6 Conclusions

Graph bandwidth has been defined and shown to have applicability to CSP's. Bandwidth provides a new class of tractably solvable constraint satisfaction problems, and a possible measure of CSP decomposability. In addition, the bandwidth is an important characteristic of a search ordering. Small bandwidth orderings seem to result in a statistically significant increase in backtracking efficiency, and these orderings are also related to intelligent backtracking and to adjacency.

6.1 Acknowledgements

Johan de Kleer, David McAllester and Joe Weening contributed to this research. The author is supported

by a fellowship from the Fannie and John Hertz Foundation.