Branchwidth Heuristics*

Illya V. Hicks †
Industrial Engineering
Texas A & M University

keywords: branchwidth, branch decomposition

Abstract

Branch decompositions were introduced by Robertson and Seymour in their series of papers that proved Wagner's Conjecture. Branch decompositions can be used to solve NP-complete problems modeled on graphs, however finding optimal branch decompositions of graphs is also NP-complete. We present a heuristic to compute near-optimal branch decompositions and compare our heuristic to a heuristic offered by Cook and Seymour.

1 Introduction

Many real-life problems can be modeled as optimization or decision problems on graphs. Also, many of these real-life problems are NP-hard. In the 1980's, Robertson and Seymour conceived of decomposing a graph by using a branch decomposition as a byproduct of their proof of Wagner's conjecture. Branch decompositions open algorithmic possibilities for intractable problems that can be modeled on graphs. Arnborg, Lagergren and Seese [2] showed that several NP-complete problems can be solved in polynomial time using divide and conquer techniques on input graphs with bounded branchwidth, but Seymour and Thomas [13] proved that computing an optimal branch decomposition is also NP-complete. Thus, one has to rely on heuristics to find near-optimal branch decompositions for graphs in general. Previously, Cook and Seymour proposed a heuristic based upon spectral graph theory. We present a comparable heuristic called the *diameter* method.

Definitions and relevant background material are given in Sections 2 and 3. Section 4 describes the tree building techniques needed for both heuristics. Sections 5 and 6 offer the descriptions of the two methods. Finally, Section 7 describe the test instances used and Sections 8 and 9 are reserved for computational results and conclusions respectively.

*research was partially supported by Office of Naval Research, ONR Grant N00014-98-1-0014 and by Texas ATP Grant 99-3604-0034

†The author was supported by AT&T Labs Fellowship Program as a graduate student in Computational and Applied Mathematics Department at Rice University

2 Definitions

Throughout this paper a graph is considered to be undirected and may have loops and multiple edges unless stated otherwise. Let us now define branch decompositions and branchwidth of a graph. Let G be a graph with node set $V(G)$ and edge set $E(G)$. Let T be a tree having $|E(G)|$ leaves in which every non-leaf node has degree 3. Let ν be a bijection from the edges of G to the leaves of T. The pair (T, ν) is called a *branch decomposition* of G. Notice that removing an edge of T, say e, partitions the edges of G into two subsets A_e and B_e. The *middle set* of e, denoted by $mid(e)$, is the set $V(A_e) \cap V(B_e)$. The *width* of a branch decomposition (T, ν) is the maximum cardinality of the middle sets over all edges in T. The *branchwidth* of G, denoted by $\beta(G)$, is the minimum width over all branch decompositions of G. A branch decomposition of G is *optimal* if its width is equal to the branchwidth of G. If there exists a non-leaf node in T with degree greater than three then the pair (T, ν) is called a *partial branch decomposition*. Figure 1 illustrates an optimal branch decomposition of an example graph in addition to providing some of the middle sets of the branch decomposition.

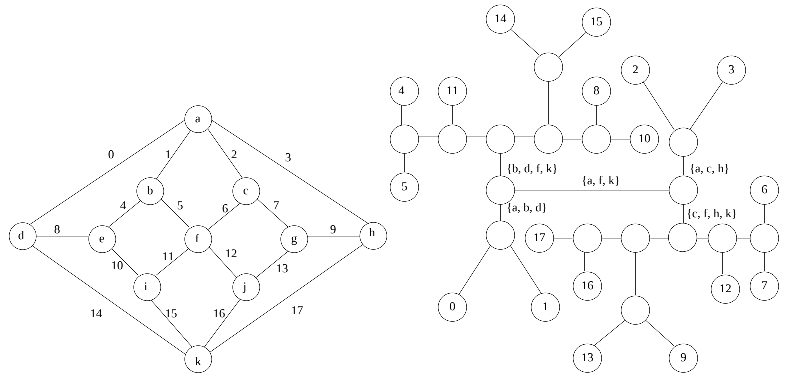

Figure 1: Example graph with optimal branch decomposition

A graph H is a *minor* of a graph G if H can be obtained from a subgraph of G by contracting edges. Graphs of small branchwidth are characterized by the following theorem.

Theorem 1 (Robertson and Seymour [11]) *A graph G has branchwidth:*

1. *0 if and only if every component of G has ≤ 1 edge*
2. *≤ 1 if and only if every component of G has ≤ 1 node of degree ≥ 2*
3. *≤ 2 if and only if G has no K_4 minor.* □

So, 1-factors (matchings) have branchwidth equal to 0; *stars* (trees with only one non-leaf node) are the only connected graphs with branchwidth equal to 1;

all forests that are not in the union of one factors and stars have branchwidth equal to 2. Also, series-parallel and outerplanar graphs have branchwidth at most 2 because these graphs do not have K_4 as a minor.

3 Relevant Background

How does one find near-optimal branch decompositions? In Robertson and Seymour [12] there is an algorithm to estimate the branchwidth of a graph within a factor of 3. For example, the algorithm decides if a graph has branchwidth at least 5 or finds a branch decomposition with width at most 15. This algorithm has not been used in a practical implementation and its improvements by Bodlaender [3], Bodlaender and Kloks [4], and Reed [10] have not been shown to be practical either. Bodlaender and Thilikos [6] did give an algorithm to compute the optimal branch decomposition for any chordal graph with maximum clique size at most 4, on the other hand the algorithm has been only shown practical for a particular type of 3-tree. Bodlaender and Thilikos [5] also offered a linear time algorithm for finding an optimal branch decomposition using a branch decomposition, but that algorithm also appears to be impractical. Seymour and Thomas [13] did give a practical algorithm to compute the branchwidth of planar graphs, yet for general graphs, one has to rely on heuristics. Cook and Seymour [8] gave a heuristic algorithm to produce branch decompositions that shows promise. The *diameter method* and the *hybrid method*, are closely related to the algorithm. All of the heuristics consist of a tree building component and a separation finding component. First, tree building will be discussed. Next, the paper focuses on the Cook and Seymour heuristic, called the eigenvector method, followed by the details of the diameter and hybrid methods and computational results comparing the heuristics.

4 Tree Building

Since computing an optimal branch decomposition is NP-complete one relies on heuristic methods to obtain near-optimal branch decompositions. We may assume that G is a biconnected graph, since one can find a branch decomposition of any graph by first finding branch decompositions of the graph's biconnected components. First, start with a partial branch decomposition (T_1, ν) where T_1 is a star with $|E(G)|$ leaves such that each leaf is associated with a distinct edge of G. Let v be the node of T_1 that is adjacent to all other nodes.

For the initial split, a separation (A, B) is found such that $|E(A)|, |E(B)| \geq 2$ and a new partial branch decomposition (T_2, ν) is created where v is replaced by nodes x and y and edge (x, y) and x has the leaves corresponding to $E(A)$ and y has the leaves corresponding to $E(B)$. An initial split is illustrated in Figure 2. In (T_2, ν), let a be a node of T_2 with degree greater than 3. Let G_a be the subgraph of G associated with the neighbors of a that are leaves. By the construction of T_2, a has only one neighbor that is not a leaf; let e be the edge

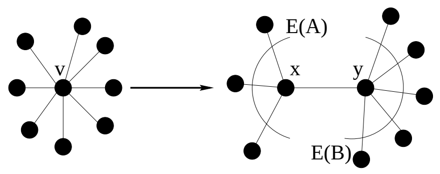

Figure 2: The initial split

between a and this neighbor. Label the nodes of $mid(e)$ in G_a as being *linking* nodes and find a separation (X, Y) of G_a. Let us assume that the cardinality of X is at most the cardinality of Y. If $|X| > 1$, create (T_3, ν) by creating new nodes x and y and edges (a, x) and (a, y) with x having the leaves corresponding to $E(X)$ and y having the leaves corresponding to $E(Y)$. The middle sets of the new edges would be:

$$mid((a,x)) = ((V(X) \cap V(Y)) \cup mid(e)) \cap X \quad (1)$$
$$mid((a,y)) = ((V(X) \cap V(Y)) \cup mid(e)) \cap Y \quad (2)$$

If $|X| = 1$, create (T_3, ν) by creating a new node y and an edge (a, y) with y having the leaves corresponding to $E(Y)$. The middle set of the new edge would be the same as Equation 2. A subsequent split when $|X| > 1$ is given in left illustration of Figure 3 while the right illustration of the figure gives an example of a subsequent split when $|X| = 1$. Notice that the non-leaf nodes of T_3 have at most one neighbor that is not a leaf node. Continue with the process done on T_2 until the resulting partial branch decomposition is a branch decomposition. This algorithm is similar to Cook and Seymour [8] except that every non-leaf of a partial branch decomposition with degree greater than 3 produced by the algorithm has at most one neighbor which is not a leaf node whereas Cook and Seymour [8] allowed multiple neighbors that are non-leaf nodes. This algorithm is also similar to the tree building algorithm given by Robertson and Seymour [12].

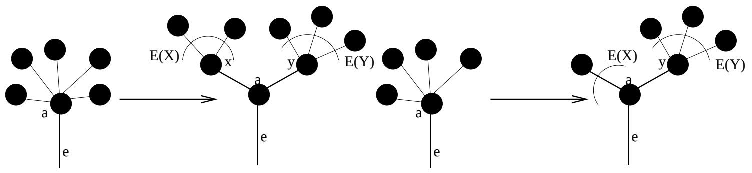

Figure 3: Subsequent splits if $|X| > 1$ or $|X| = 1$

How does one find a good separation? Let $he(S)$ denote a hyperedge where the ends of the hyperedge are S where S is a set of nodes. For a non-leaf node v in a partial branch decomposition, let D_v denote the set of edges incident with v. Define H^v as the hypergraph constructed from the union of hyperedges $he(mid(e))$ where $e \in D_v$. So if v was the non-leaf node of T_1, then H^v would correspond to G. A partial branch decomposition (T, ν) of a graph G is called *extendible* if $\beta(H^v) \leq \beta(G)$ for every non-leaf node $v \in V(T)$. This follows from the fact that if every H^v had branchwidth at most some number k then one could use the optimal branch decompositions of the hypergraphs to build a branch decomposition of G whose width is at most k. One can see that this definition of extendible is synonymous with the definition given by Cook and Seymour [8]. Even though (T_1, ν) is extendible, there is not a way to check if a partial branch decomposition is extendible. A separation is called *greedy* if the next partial branch decomposition created by the use of the separation is extendible if the previous partial branch decomposition is extendible. Some greedy separations will be described along with proofs that they are indeed greedy. Let $G[F]$ denote the subgraph of graph G induced by F, a subset of nodes or edges.

Theorem 2 *Let G be a graph with partial branch decomposition (T_i, ν). Let $a \in V(T_i)$ be a non-leaf node with degree greater than 3. Let e be the edge incident to a and not incident to a leaf node. Let G_a be the induced subgraph of G produced by the leaves adjacent to a. Label the nodes that are in the middle set of e as linking nodes in G_a. Let f denote an edge of G_a that is only incident with linking nodes of G_a. Then $(G_a[f], G_a \setminus f)$ is a greedy separation.*

> **Proof:** *To distinguish between $a \in V(T_i)$ and $a \in V(T_{i+1})$, denote them as a_i and a_{i+1} respectively. Let (T_{i+1}, ν) be the partial branch decomposition created from (T_i, ν) by creating a node y and an edge (a_{i+1}, y) with y having the leaves corresponding to the edges of $G_a \setminus f$. Let (T, μ) be an optimal branch decomposition of H^{a_i}. Create T' from T by deleting the leaf that is mapped to f and for the node of degree 2, contracting one of the node's edges. Let μ' be the restriction of μ to T'. Thus, (T', μ') is a branch decomposition of H^y. So, $\beta(H^y) \leq \beta(H^{a_i})$ and notice that $H^{a_{i+1}}$ has branchwidth at most $|mid(e)|$. Thus, $(G_a[f], G_a \setminus f)$ is greedy.* ∎

Theorem 3 *Let G, (T_i, ν), G_a, a, and e be defined as they were in Theorem 2 with the addition that G is biconnected. Let $v \in V(G_a)$ be a linking node that is only incident to one edge say f and the other end, w, is not a linking node. Then $(G[f], G_a \setminus f)$ is a greedy separation.*

> **Proof:** *Since G is biconnected then the number of linking nodes of G_a is greater than one. To distinguish between $a \in V(T_i)$ and $a \in V(T_{i+1})$, denote them as a_i and a_{i+1} respectively. Let (T_{i+1}, ν) be the partial branch decomposition created from (T_i, ν) by creating a node y and an edge (a_{i+1}, y) with y having the leaves corresponding to the edges of $G_a \setminus f$. Let (T, μ) be an optimal branch decomposition of H^{a_i}. Create T' from T by deleting the leaf that is mapped to f and for the node of degree 2, contracting one of the node's edges. Let μ' be the restriction of μ to T'. If one replaces v with w in every middle set*

of (T', μ') that contained v, then the resulting branch decomposition would be a branch decomposition of H^y. Thus, $\beta(H^y) \leq \beta(H^{a_i})$. Also, notice that $H^{a_{i+1}}$ has branchwidth at most $|mid(e)|$. Thus, $(G[f], G_a \setminus f)$ is greedy. ∎

Theorem 4 *Let G, (T_i, ν), G_a, a, and e be defined as they were in Theorem 2. Suppose $G_a \setminus mid(e)$ has at least two components. Let W be a component of $G_a \setminus mid(e)$. Let $S \subseteq mid(e)$ denote the linking nodes that are adjacent to at least one node of $V(W)$ in G_a. Then $(G_a[V(W) \cup S], G_a \setminus (V(W) \cup E(G_a[V(W) \cup S]))$ is a greedy separation.*

Proof: *To distinguish between $a \in V(T_i)$ and $a \in V(T_{i+1})$, denote them as a_i and a_{i+1} respectively. Let (T_{i+1}, ν) be the partial branch decomposition created from (T_i, ν) by creating nodes x and y and edges (a_{i+1}, x) and (a_{i+1}, y) with x having the leaves corresponding to the edges of $G_a[V(W) \cup S]$ and y having the leaves corresponding to edges of $G_a \setminus (V(W) \cup E(G_a[V(W) \cup S])$. Let (T, μ) be an optimal branch decomposition of H^{a_i}. Create T' by deleting from T the leaves that are not mapped to edges in $H^{a_i}[V(W) \cup S]$ and for every node of degree 2, contracting one of the node's edges. Let μ' be the restriction of μ to T'. So, (T', μ') is a branch decomposition of $H^{a_i}[V(W) \cup S]$. Thus, the width of (T', μ') is at most the branchwidth of H^x. A similar argument can be done for H^y. So, $\beta(H^x), \beta(H^y) \leq \beta(H^{a_i})$. Also, notice that $H^{a_{i+1}}$ has branchwidth at most $|mid(e)|$. $(G_a[V(W) \cup S], G_a \setminus (V(W) \cup E(G_a[V(W) \cup S]))$ is a greedy separation.* ∎

Other than greedy separations, one would want separations (X, Y) such that $max(mid(a, x), mid(a, y))$ is minimized. Cook and Seymour [8] used a heuristic called the *eigenvector method* to find separations for their branch decomposition code. The eigenvector method was implemented along with a method called the *diameter method.*

5 Eigenvector Method

Let G, T_i, a, e and G_a be defined as they were in the previous section. One would like to find a separation (X, Y) of G_a such that the following partial branch decompositions are likely to have greedy separations. One possible way is to find (X, Y) such that $|X|$, $|Y| \geq \delta |E(G_a)|$ for some $0 < \delta < 1$. Also, it would be helpful if $|X \cap Y|$ is small as possible. This is a well known problem.

THE SEPARATOR PROBLEM One is given a graph $G(V, E)$ and wishes to choose a partition (X, Y) of E with $|X|$, $|Y| \geq \delta |E|$ and with $|X \cap Y|$ as small as possible where $0 < \delta < 1$.

The separator problem is also NP-complete. Has anything been gained? The answer is believed to be "yes". The separator problem can be approximated by a heuristic called the eigenvector method, as follows.

Let L be a $|E(G_a)|$ by $|E(G_a)|$ matrix. For a node v, let $adj(v)$ denote the set of edges incident to v and let $deg(v)$ denote the degree of v. For $e, f \in E(G_a)$, let

$$L(e,f) = \begin{cases} 2 & \text{if } e = f, \\ \sum(-(deg(u)-1)^{-1} \;:\; u \in V \; and \; e,f \in adj(u)) & \text{if e and f share an end,} \\ 0 & \text{otherwise.} \end{cases}$$

Compute the eigenvector of L corresponding to the second smallest eigenvalue of L. To achieve this goal, look at the matrix $N = cI - L$ where c is some large constant and I is the identity matrix. So, in order to compute the eigenvector of L corresponding to the second smallest eigenvalue of L one would have to compute the eigenvector of N corresponding to the second largest eigenvalue of N. Since the vector of all 1's, denoted by $\vec{1}$, corresponds to the largest eigenvalue of N, use the power method with a vector y such that $\vec{1}^T y = 0$.

Let x be the eigenvector of N corresponding to the second largest eigenvalue of N. Sort the edges of G_a so that the numbers x_i are in non-decreasing order. Let A be the $\lceil \delta \; |E(G_a)| \rceil$ first terms and B be the $\lceil \delta \; |E(G_a)| \rceil$ last terms for some $0 < \delta < 1$.

Given $A, B \subseteq E(G_a)$, compute a partition (X, Y) of $E(G_a)$ such that $A \subseteq X$, $B \subseteq Y$, and $|X \cap Y|$ is minimized. This procedure is done by a standard network flow problem and is solved as follows. Let H be a minor of G with $V(A)$ identified to node v_A and $V(B)$ identified to node v_B. Next, find the smallest vertex cut in H separating all internally disjoint v_A, v_B-paths, denoted by $sep(v_A, v_B)$. The idea of using eigenvectors to find separators comes from the work of Alon [1] and Chung [7].

6 Diameter Method

The *diameter* of a graph G is the smallest number k such that the shortest path between any two nodes of G has distance at most k where the edges all have length equal to one. The *eccentricity* of a node v is the smallest number k such the shortest path between v and any other node w has distance at most k where the edges of the graph all have length equal to one. Let G be a graph with partial branch decomposition (T, ν) constructed using the construction discussed in Section 4. Let a denote a non-leaf node of T with degree greater than 3. By the tree building construction of (T, ν), a is the only non-leaf node or a has exactly one neighbor that is also a non-leaf node. Let e denote the edge between a and its non-leaf node neighbor (if a has a non-leaf node neighbor). Let G_a denote the subgraph of G given by the edges corresponding to the leaves adjacent to a. If e exists, label the nodes of $mid(e)$ as *linking* nodes in G_a. Again, the separator problem needs to be solved. The diameter method proceeds as follows. First, let the nodes of G_a that have eccentricity equal to the diameter of G_a and those that are labeled linking nodes be labeled as *source* nodes. Let v be a source node and let δ be some number such that $0 < \delta < 1$. Sort all of the nodes such that their distance to v is in non-decreasing order. Let A be the $\lceil |\delta \ V(G_a)| \rceil$ first terms and B be the $\lceil |\delta \ V(G_a)| \rceil$ last terms.

Given $A, B \subseteq V(G_a)$, compute a separation (X, Y) of G_a such that $A \subseteq V(X)$, $B \subseteq V(Y)$, and $|V(X) \cap V(Y)|$ is minimized. This procedure is done by a standard network flow problem and is solved as follows. Let H be a minor of G with A identified to node v_A and B identified to node v_B. Then, find the smallest vertex cut in H intersecting all v_A, v_B-paths, denoted by $sep(v_A, v_B)$. Let us label the nodes of $sep(v_A, v_B)$ in G_a as *separation* nodes and denote the cardinality by new. Let the nodes of G_a that are both labeled linking nodes and separation nodes also be labeled *share* nodes and denote their cardinality by sh. Label the linking nodes that are on one side of the cut but are not labeled separation nodes as *side* nodes and denote their cardinality by s. Let *linking* denote the cardinality of linking nodes in G_a. Let *work* be defined as $max((s + new), (linking - s - sh + new))$ and define *play* as $min((s + new), (linking - s - sh + new))$. So, there is a particular work and play for the pair (v, δ). The method iterates over all source nodes and different values of δ. This method finds the pair (v_o, δ_o) such that the separation produced by them has the smallest work, denoted by *owork*, and the biggest play out of all the pairs (v, δ) with the same work, denoted by *oplay*. So the pair (v_o, δ_o) produces a separation of G_a such that $max(mid(a, x), mid(a, y))$ is minimized and $min(mid(a, x), mid(a, y))$ is maximized and the sets of *owork* and *oplay* will be the middle sets of the two new edges created in the next partial branch decomposition.

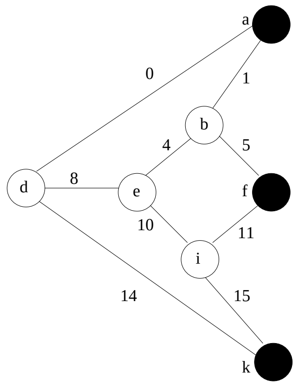

Figure 4: Example G_a

To illustrate the diameter method, let Figure 4 represent G_a with the linking nodes of G_a colored black. So the source nodes of Figure 4 are the linking nodes and nodes i and b. If A consisted of nodes a and b and B consisted of nodes i and k then $sep(v_A, v_B)$ would consist of d, e, and f. The side node would be a and the share node would be f. The work of source node a with δ equal to

2/7 is four and the play is also four. Figure 5 illustrates what G_x and G_y would look like if the pair $(a, 2/7)$ is used as the best pair to give us a separation. The basic premise is that if the shortest path in the graph G between nodes u and v is the diameter and the diameter is large then edges incident with u should not be close in a branch decomposition to edges incident with v.

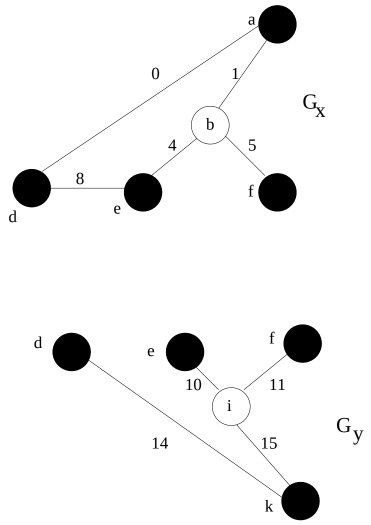

Figure 5: Example G_x and G_y

7 Test Instances

The test instances are partitioned into three classes: planar, compiler, and fill-in. The graph tele133 under the planar class is a 133 node graph from the telecommunications industry. The rest of the planar test instances are Delaunay triangulation of some of the graphs from the TSPLIB95 library of standard test instances for the traveling salesmen problem. For more information about Delaunay triangulation, the reader is referred to Edelsbrunner [9]. Figure 6 illustrates some of the planar test instances with their branchwidth.

The compiler test instances are instances of control-flow graphs from actual C compilations. These test instances were provided by Keith Cooper, Associate Professor of Computer Science at Rice University. As stated earlier the control-flow graph of some program P is a digraph, although the orientation of the edges is unimportant for coloring. So, the graphs are perceived as undirected. Most of the instances are series-parallel graphs, but there are a few that have branchwidth at least 3 and are not planar. Figure 7 illustrates some of the graphs in this class and bounds on their branchwidth.

The fill-in test instances are instances generated by matrices. These instances were provided by Chao Yang, researcher for Oak Ridge National Lab-

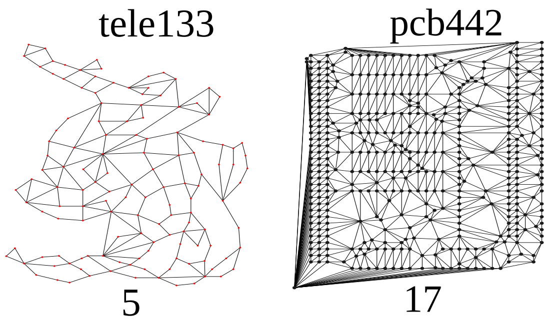

Figure 6: Planar Test Instances with their branchwidth

oratory. Let M be an n by n matrix for some $n > 1$. M can be modeled as a graph in which the nodes represent the different columns and rows of M and an edge between node i and node j represents if $M(i, j)$ is nonzero. Many of the interesting matrices for minimum fill-in are huge (more than 10000 nodes), however, the smaller cases were only considered. Figure 8 illustrates some the graphs in this class and bounds on their branchwidth.

8 Computational Results

All computations were performed on a 143 MHz Sun UltraSparc1 using code written in the C++ programming language. The branchwidth of each graph was calculated using an implementation of the Seymour and Thomas [13] branchwidth algorithm for planar graphs. "NP" under the branchwidth column means that the graph was nonplanar and the branchwidth of that graph is unknown. The diameter method was compared to the eigenvector method due to Cook and Seymour [8] and the resulting widths of the eigenvector method are under the "eigenvector" column while the resulting widths of the diameter method are under the "diameter" column. Under the "eigenvector" column, the eigenvector method had the restriction of only working on connected graphs. So, an

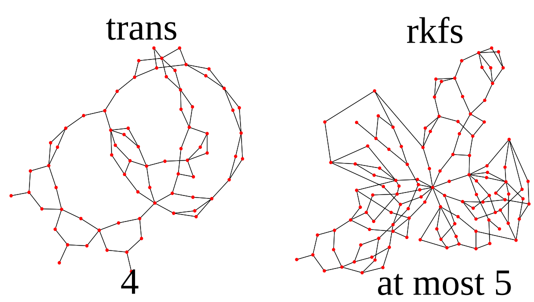

Figure 7: Compiler Test Instances with their branchwidth

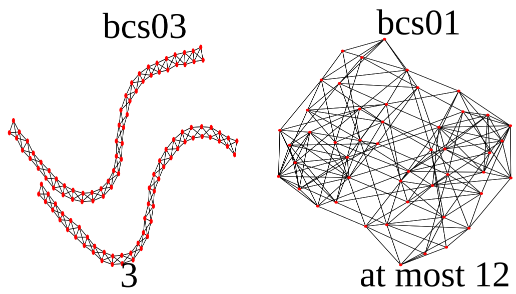

Figure 8: Fill-in Test Instances with their branchwidth

"NC" in this column means the graph was not connected and a branch decomposition was not computed for this graph using the eigenvector method. The "hybrid" column represents a hybrid of both heuristics where the eigenvector method is used on the initial separation and the diameter method is used on the subsequent separation.

For the class labeled "planar graphs" given in Tables 1 and 2, the hybrid method has promising results. The hybrid method did not compute an optimal branch decomposition for only 24 of the 62 planar graphs. Furthermore, 14 of the branch decompositions have widths only one more than the branchwidth of those graphs. Similarly, the diameter method computed an optimal branch decomposition for 37 of the 62 graphs with only 16 branch decompositions width one more than the branchwidths for the graphs.

For the classes labeled "fill-in" and "compiler" given in Tables 3, 4, 5, 6, 7 and 8, the diameter method did not give the optimal branch decomposition for only one graph that was planar. The hybrid method did not give the optimal branch decomposition for two planar graphs. The hybrid method gives more lower widths than the diameter method for the "fill-in" graphs, however the diameter method gave more lower widths than the hybrid method for the "compiler" graphs.

It can be observed that the eigenvector method was faster than the diameter or hybrid method and performed better graphs with small branchwidth. In contrast, the diameter and hybrid method had a better performance in terms of finding smaller widths than the eigenvector method. One reason for both conclusions is the fact that both the diameter and hybrid method search through a number of separations before concluding to use one of those separations for tree building; however, the eigenvector is designed to produce one separation to be used for tree building.

9 Conclusions and Future Work

In conclusion, the diameter and hybrid method for finding near-optimal branch decomposition is comparable and gave a better performance for finding smaller width branch decompositions for graphs with large branchwidth (above 10) than

Table 1: Planar graphs

	diameter		hybrid		eigenvector		
planar graphs	width	time (sec)	width	time (sec)	width	time (sec)	branchwidth
tele133	5	1	5	2	5	0.0	5
a280.tsp.del	13	10	13	52	14	11	13
bier127.tsp.del	14	5	14	7	14	5	14
ch130.tsp.del	10	2	10	11	11	4	10
ch150.tsp.del	12	5	12	15	12	5	12
d1291.tsp.del	30	6,956	30	1,881	38	1,399	29
d1655.tsp.del	32	15,318	31	9,031	45	1,714	29
d198.tsp.del	12	12	12	13	14	11	12
d2103.tsp.del	29	12,025	29	15,336	35	2,332	29
d493.tsp.del	20	385	21	255	24	79	20
d657.tsp.del	22	442	22	356	26	135	22
eil101.tsp.del	10	1	10	4	11	2	10
eil51.tsp.del	8	0.0	8	0.0	8	0.0	8
eil76.tsp.del	10	0.0	10	4	11	1	10
fl1400.tsp.del	13	8,250	14	5,406	14	1,143	13
fl1577.tsp.del	19	11,157	16	87	19	554	16
fl417.tsp.del	9	56	9	41	9	45	9
gil262.tsp.del	15	29	15	41	16	21	15
kroA100.tsp.del	9	1	9	2	9	2	9
kroA150.tsp.del	11	6	11	8	11	5	11
kroA200.tsp.del	11	11	11	12	11	12	11
kroB100.tsp.del	9	1	9	1	9	2	9
kroB150.tsp.del	10	6	10	10	10	6	10
kroB200.tsp.del	12	10	12	11	13	11	12
kroC100.tsp.del	9	1	9	2	9	2	9
kroE100.tsp.del	8	1	8	2	8	2	8
lin105.tsp.del	9	1	8	1	9	2	8
nrw1379.tsp.del	31	8,907	31	8,456	38	1,216	31
p654.tsp.del	12	170	11	365	10	143	10
pcb1173.tsp.del	27	3,511	27	2,744	31	727	25
pcb442.tsp.del	17	106	19	106	21	67	17

the eigenvector method proposed by Cook and Seymour. Future work propels one to investigate the separator problem more thoroughly as a key for better heuristics in this area.

Table 2: Some more planar graphs

	diameter		hybrid		eigenvector		
planar graphs	width	time (sec)	width	time (sec)	width	time (sec)	branchwidth
pr1002.tsp.del	24	1,053	23	1,864	24	450	21
pr107.tsp.del	7	1	7	3	6	2	6
pr124.tsp.del	8	2	8	4	8	1	8
pr136.tsp.del	10	4	10	7	10	3	10
pr144.tsp.del	9	3	9	5	9	3	9
pr152.tsp.del	8	3	8	8	8	4	8
pr226.tsp.del	7	7	7	20	7	5	7
pr2392.tsp.del	34	47,605	33	48,728	34	4,086	29
pr264.tsp.del	14	20	14	30	13	21	13
pr299.tsp.del	11	14	11	24	12	16	11
pr439.tsp.del	17	171	17	167	17	59	16
pr76.tsp.del	9	0.0	9	1	11	1	9
rat195.tsp.del	12	9	12	11	12	9	12
rat575.tsp.del	18	588	18	396	19	105	17
rat783.tsp.del	21	1,065	21	1,416	23	212	20
rat99.tsp.del	9	1	9	1	9	2	9
rd100.tsp.del	11	1	11	4	10	2	10
rd400.tsp.del	18	79	18	92	19	50	17
rl1304.tsp.del	20	15,205	21	2,723	21	906	19
rl1323.tsp.del	23	4,686	23	4,131	26	990	22
rl1889.tsp.del	25	13,130	23	11,707	26	2,487	22
tsp225.tsp.del	12	20	12	15	13	11	12
u1060.tsp.del	26	2,582	20	2,176	37	573	20
u1432.tsp.del	33	11,880	34	5,971	40	1,238	32
u159.tsp.del	10	6	10	10	11	4	10
u1817.tsp.del	29	14,081	29	10,540	34	2,005	28
u2152.tsp.del	36	36,887	37	22,979	39	2,698	31
u2319.tsp.del	46	66,077	46	23,510	72	3,264	44
u574.tsp.del	18	316	17	239	18	100	17
u724.tsp.del	19	2,178	19	547	22	177	18
vm1084.tsp.del	15	2,178	15	1,543	18	176	15
vm1748.tsp.del	23	11,982	24	10,495	26	1,164	22

10 Acknowledgments

The author would like to take the time to acknowledge his advisor, William J. Cook, for his guidance and patience. The author would also like to thank Dr. Nate Dean and Dr. Paul Seymour for our conversations about the subject.

References

[1] N. Alon. Eigenvalues and expanders. *Combinatorica*, 2:83–96, 1986.

[2] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

[3] H. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[4] H. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21:358–402, 1996.

[5] H. Bodlaender and D. Thilikos. Constructive linear time algorithms for branchwidth. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Lecture Notes in Computer Science: Proceedings 24th International Colloquium on Automata, Languages, and Programming*, pages 627–637. Springer Verlag, 1997.

[6] H. Bodlaender and D. Thilikos. Graphs with branchwidth at most three. Technical Report UU-CS-1997-37, Utrecht University, Netherlands, 1997.

[7] F. Chung. *Spectral Graph Theory.* American Mathematical Society, Providence, Rhode Island, 1994.

[8] W. Cook and P. D. Seymour. An algorithm for the ring-router problem. Technical report, Bellcore, 1994.

[9] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, New York, 1987.

[10] B. Reed. Tree width and tangles: A new connectivity measure and some applications. In R. A. Bailey, editor, *Survey in Combinatorics, 1997*, pages 87–162. Cambridge University Press, Cambridge, 1997.

[11] N. Robertson and P. D. Seymour. Graph minors X: Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52:153–190, 1991.

[12] N. Robertson and P. D. Seymour. Graph minors XIII: The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63:65–110, 1995.

[13] P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

Table 3: Minimum Fill-in graphs

	diameter		hybrid		eigenvector		
fill-in graphs	width	time (sec)	width	time (sec)	width	time (sec)	branchwidth
bcs01	13	0.0	12	0.0	13	1.0	NP
bcs03	3	0.0	3	1	NC	0.0	3
bcs04	32	26	32	19	32	1007	NP
bcs05	20	77	21	82	27	144	NP
bcs06	38	272	36	396	40	3255	NP
bcs07	38	272	36	396	40	3255	NP
bcs08	132	4,092	101	11,198	NC	0.0	NP

Table 4: Compiler graphs

	diameter		hybrid		eigenvector		
comp graphs	width	time (sec)	width	time (sec)	width	time (sec)	branchwidth
abrt	2	0.0	2	0.0	2	0.0	2
aclear	2	0.0	2	0.0	2	0.0	2
advbnd	3	3.0	3	8.0	3	0.0	NP
arret	2	0.0	2	0.0	2	0.0	2
bcnd	2	0.0	2	0.0	2	0.0	2
bcndb	2	0.0	2	0.0	2	0.0	2
bcndl	2	0.0	2	0.0	2	0.0	2
bcndr	2	0.0	2	0.0	2	0.0	2
bcndt	2	0.0	2	0.0	2	0.0	2
bilan	3	0.0	3	0.0	3	0.0	3
bilsla	2	0.0	2	0.0	2	0.0	2
blts	2	0.0	2	0.0	2	0.0	2
buts	3	0.0	3	0.0	3	0.0	3
cardeb	3	0.0	3	0.0	3	0.0	3
celbnd	3	1.0	3	4.0	3	0.0	NP
clrdt	2	0.0	2	0.0	2	0.0	2
coeray	2	0.0	2	0.0	2	0.0	2
colbur	3	0.0	3	0.0	3	0.0	NP
cosqb1	2	0.0	2	0.0	2	0.0	2
cosqb	2	0.0	2	0.0	2	0.0	2
cosqf1	2	0.0	2	0.0	2	0.0	2
cosqf	2	0.0	2	0.0	2	0.0	2
cosqi	2	0.0	2	0.0	2	0.0	2
dcoera	2	0.0	2	0.0	2	0.0	2
ddeflu	2	0.0	2	1.0	2	0.0	2
debflu	3	0.0	3	0.0	3	0.0	3
debico	3	0.0	3	0.0	3	0.0	3
decomp	4	1.0	4	1.0	3	0.0	NP
denitl	2	0.0	2	0.0	2	0.0	2
denitr	2	0.0	2	0.0	2	0.0	2
denpt	2	0.0	2	0.0	2	0.0	2
dens	2	0.0	2	0.0	2	0.0	2
densx	2	0.0	2	0.0	2	0.0	2
densy	2	0.0	2	0.0	2	0.0	2
deseco	3	6.0	3	20.0	3	0.0	3

Table 5: More compiler graphs

	diameter		hybrid		eigenvector		
comp graphs	width	time (sec)	width	time (sec)	width	time (sec)	branchwidth
ediagns	3	0.0	3	0.0	3	0.0	NP
drepvi	3	0.0	3	0.0	3	0.0	3
drigl	3	0.0	3	0.0	3	0.0	3
dyeh	3	0.0	3	0.0	3	0.0	3
ecrd	2	0.0	2	0.0	2	0.0	2
ecwr	2	0.0	2	0.0	2	0.0	2
efill	3	0.0	3	1.0	3	0.0	3
endrun	2	0.0	2	0.0	2	0.0	2
energy	2	0.0	2	0.0	2	0.0	2
erf	2	0.0	2	0.0	2	0.0	2
rhs	3	2.0	3	11.0	3	0.0	3
error	2	0.0	2	0.0	2	0.0	2
exact	2	0.0	2	0.0	2	0.0	2
fehl	2	0.0	2	0.0	2	0.0	2
fftb	2	0.0	2	0.0	2	0.0	2
fftf	2	0.0	2	0.0	2	0.0	2
field	4	81.0	4	148.0	4	0.0	NP
fmin	3	0.0	3	0.0	3	0.0	NP
fmtgen	4	0.0	4	0.0	4	0.0	NP
fmtset	3	0.0	3	0.0	3	0.0	3
fpppp	2	0.0	2	0.0	2	0.0	2
gamgen	3	0.0	3	0.0	3	0.0	3
genb	2	0.0	2	0.0	2	0.0	2
genprb	2	0.0	2	0.0	2	0.0	2
getb	3	2.0	3	3.0	3	0.0	3
getdt	2	0.0	2	0.0	2	0.0	2
heat	3	0.0	3	0.0	3	0.0	NP
hmoy	2	0.0	2	0.0	2	0.0	2
ibin	2	0.0	2	0.0	2	0.0	2
ihbtr	3	0.0	3	0.0	3	0.0	3
ilsw	2	0.0	2	0.0	2	0.0	2
inibnd	2	0.0	2	0.0	2	0.0	2
inideb	3	0.0	3	0.0	3	0.0	3
iniset	2	528.0	2	17.0	2	0.0	2
inisla	3	0.0	3	0.0	3	0.0	3

Table 6: More compiler graphs

	diameter		hybrid		eigenvector		
comp graphs	width	time (sec)	width	time (sec)	width	time (sec)	branchwidth
init	3	25.0	3	81.0	3	0.0	NP
inithx	3	1.0	3	1.0	3	0.0	3
injall	4	0.0	4	0.0	3	0.0	3
injbat	2	0.0	4	0.0	2	0.0	2
injchk	2	0.0	2	0.0	2	0.0	2
injcon	3	0.0	3	0.0	3	0.0	NP
integr	3	0.0	3	0.0	3	0.0	NP
inter	2	0.0	2	0.0	2	0.0	2
intowp	2	0.0	2	0.0	2	0.0	2
jacld	2	0.0	2	0.0	2	0.0	2
jacu	2	0.0	2	0.0	2	0.0	2
jobtim	2	0.0	2	0.0	2	0.0	2
l2norm	2	0.0	2	0.0	2	0.0	2
lasden	3	0.0	3	0.0	3	0.0	NP
laser	2	0.0	2	0.0	2	0.0	2
laspow	3	0.0	3	0.0	3	0.0	3
lclear	2	0.0	2	0.0	2	0.0	2
linj	3	0.0	3	0.0	3	0.0	NP
lissag	2	0.0	2	0.0	2	0.0	2
main	2	0.0	2	0.0	2	0.0	2
maxnorm	2	0.0	2	0.0	2	0.0	2
nprio	3	0.0	3	0.0	3	0.0	3
numb	2	0.0	2	0.0	2	0.0	2
orgpar	2	0.0	2	0.0	2	0.0	2
parmov	3	73.0	3	116.0	3	0.0	3
parmve	2	40.0	2	135.0	2	0.0	2
parmvr	3	152.0	3	188.0	3	0.0	3
paroi	3	0.0	3	1.0	3	0.0	3
pastem	3	2.0	3	5.0	3	0.0	3
pdiag	3	2.0	3	4.0	3	0.0	3
pintgr	3	1.0	3	0.0	3	0.0	3
prophy	2	0.0	2	0.0	2	0.0	2
putb	3	0.0	3	2.0	3	0.0	NP
putdt	2	0.0	2	0.0	2	0.0	2
radb2	3	1.0	3	4.0	3	0.0	NP

Table 7: More compiler graphs

	diameter		hybrid		eigenvector		
comp graphs	width	time (sec)	width	time (sec)	width	time (sec)	branchwidth
radb3	3	0.0	3	1.0	3	0.0	NP
radb4	3	1.0	3	4.0	3	0.0	NP
radb5	3	0.0	3	1.0	3	0.0	NP
radbg	3	216.0	3	104.0	3	0.0	NP
radf2	3	1.0	3	4.0	3	0.0	NP
radf3	3	0.0	3	1.0	3	0.0	NP
radf4	3	1.0	3	4.0	3	0.0	NP
radf5	3	0.0	3	1.0	3	0.0	NP
radfg	3	90.0	3	103.0	3	0.0	NP
ranf	2	0.0	2	0.0	2	0.0	2
recre	3	0.0	3	0.0	3	0.0	NP
repvid	2	0.0	2	0.0	2	0.0	2
rewdt	2	0.0	2	0.0	2	0.0	2
rfftb1	2	0.0	2	0.0	2	0.0	2
rfftb	2	0.0	2	0.0	2	0.0	2
rfftf1	2	0.0	2	0.0	2	0.0	2
rfftf	2	0.0	2	0.0	2	0.0	2
rffti1	2	0.0	2	0.0	2	0.0	2
rffti	2	0.0	2	0.0	2	0.0	2
rhs	3	2.0	3	9.0	3	0.0	3
rinj	3	0.0	3	0.0	3	0.0	NP
rkf45	2	0.0	2	0.0	2	0.0	2
rkfs	5	1.0	5	3.0	5	0.0	NP
saturr	2	0.0	2	0.0	2	0.0	2
saxpy	2	0.0	2	0.0	2	0.0	2
setb	2	0.0	2	0.0	2	0.0	2
setbv	2	0.0	2	0.0	2	0.0	2
setinj	2	0.0	2	0.0	2	0.0	2
setiv	2	0.0	2	0.0	2	0.0	2
seval	3	0.0	3	0.0	3	0.0	NP
sgemm	3	0.0	3	0.0	3	0.0	3
sgemv	2	0.0	2	0.0	2	0.0	2
si	3	0.0	3	0.0	3	0.0	3
sigma	2	0.0	2	0.0	2	0.0	2
sinqb	2	0.0	2	0.0	2	0.0	2

Table 8: More compiler graphs

	diameter		hybrid		eigenvector		
comp graphs	width	time (sec)	width	time (sec)	width	time (sec)	branchwidth
sinqf	2	0.0	2	0.0	2	0.0	2
sinqi	2	0.0	2	0.0	2	0.0	2
slv2xy	2	0.0	2	0.0	2	0.0	2
smooth	3	10.0	3	9.0	3	0.0	NP
solv2y	2	0.0	2	0.0	2	0.0	2
solve	2	0.0	2	0.0	2	0.0	2
sortie	3	0.0	3	0.0	3	0.0	NP
spline	3	0.0	3	0.0	3	0.0	3
ssor	3	0.0	3	1.0	3	0.0	3
subb	2	0.0	2	0.0	3	0.0	2
sudtbl	3	0.0	3	0.0	3	0.0	3
supp	2	0.0	2	0.0	2	0.0	2
svd	4	26.0	4	13.0	4	0.0	NP
tcomp	3	0.0	3	0.0	3	0.0	NP
tpart	2	0.0	2	0.0	2	0.0	2
trans	4	0.0	4	0.0	4	0.0	4
twldrv	5	24.0	5	23.0	4	0.0	NP
urand	2	0.0	2	0.0	2	0.0	2
vavg	2	0.0	2	0.0	2	0.0	2
verify	2	0.0	2	0.0	2	0.0	2
vgjyeh	2	0.0	2	0.0	2	0.0	2
vnewl	2	0.0	2	0.0	2	0.0	2
vslv1p	3	15.0	3	77.0	3	0.0	NP
vslv1x	3	9.0	3	31.0	3	0.0	NP
x21y21	2	0.0	2	0.0	2	0.0	2
yeh	3	0.0	3	0.0	3	0.0	3
zeroin	3	0.0	3	0.0	3	0.0	NP