An Effective Upperbound on Treewidth Using Partial Fill-in of Separators

Boi Faltings [*] Martin Charles Golumbic [‡]

June 28, 2009

Abstract

Partitioning a graph using graph separators, and particularly clique separators, are well-known techniques to decompose a graph into smaller units which can be treated independently. It was previously known that the treewidth was bounded above by the sum of the size of the separator plus the treewidth of disjoint components, and this was obtained by the heuristic of filling in all edges of the separator making it into a clique.

In this paper, we present a new, tighter upper bound on the treewidth of a graph obtained by only partially filling in the edges of a separator. In particular, the method completes just those pairs of separator vertices that are adjacent to a common component, and indicates a more effective heuristic than filling in the entire separator.

We discuss the relevance of this result for combinatorial algorithms and give an example of how the tighter bound can be exploited in the domain of constraint satisfaction problems.

Keywords : treewidth, partial k-trees, graph separators

[*]Artificial Intelligence Laboratory (LIA), Ecole Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland. E-mail: boi.faltings@epfl.ch

[‡]Caesarea Rothschild Institute and Department of Computer Science, University of Haifa, Mt. Carmel, Haifa 31905, Israel. E-mail: golumbic@cs.haifa.ac.il

1 Introduction

Let $G = (V, E)$ be an undirected graph. We denote by $G_X = (X, E_X)$ the subgraph of G induced by $X \subseteq V$ where $E_X = \{(u, v) \in E \mid u, v \in X\}$.

A *tree decomposition* for a graph G is defined as a tree T whose nodes are labelled by subsets of V called "clusters" (or "bags") such that

(1) every vertex $v \in V$ appears in at least one cluster,

(2) if $(u, v) \in E$, then u and v co-occur in some cluster, and

(3) for every $v \in V$, the set of nodes of T which include v in their cluster induces a connected subgraph (i.e., a subtree) of T, denoted $T(v)$.

The *width* of a tree decomposition T is the size of the largest cluster minus 1, and is denoted by $width(T)$.

A given graph G may have many possible tree decompositions, including the trival representation as a single node with cluster equal to V. The *treewidth* $tw(G)$ of a graph G is defined to be the minimum width over all tree decompositions for G. Such a tree decomposition is called a *minimum tree decomposition* for G.

Remark 1 *The treewidth of a tree equals 1, of a chordless cycle equals 2, of a clique on k vertices equals $k-1$, and of a stable (independent) set equals zero. It is also well known, that a chordal graph has a minimum tree decomposition where each cluster is a maximal clique of the graph, thus, the treewidth of a chordal graph is the size of its largest clique minus 1.*

The theory of treewidth, introduced by Robinson and Seymour [9], is a very rich topic in discrete mathematics, and has important algorithmic significance, since many NP-complete problems may be solved efficiently on graphs with bounded treewidth. The reader is referred to [1, 2, 7] for further treatment of the subject.

Partitioning a graph using graph separators, and particularly clique separators, is well-known as a method to decompose a graph into smaller components which can be treated independently. It was previously known ([5]) that the treewidth was bounded above by the sum of the size of the separator plus the treewidth of disjoint components, and this was obtained by the heuristic of filling in all edges of the separator making it into a clique.

In Section 2, we present a new, tighter upper bound on the treewidth of a graph whose novelty is by filling in fewer edges of the separator. Our method completes just those pairs of separator vertices that are adjacent to a common component, giving a lower treewidth of the augmented supergraph. This is followed by an example in Section 3 to illustrate our method. In Section 4, we conclude by discussing its application to solving constraint satisfaction problems combining search with dynamic programming, which was our motivation for having studied the question of improving the bounds on treewidth.

2 Our result

We first recall the Helly property which is satisfied by subtrees of a tree [6]. By definition, if $(u, v) \in E$ then $T(u) \cap T(v) \neq \emptyset$. The Helly property for trees states that *if a collection of subtrees of a tree pairwise intersect, then the intersection of the entire collection is nonempty.* This immediately implies the following well-known (folklore) remark [4], which will be used below.

Remark 2 *Let T be a tree decomposition for G. If C is a clique of G, then there is a cluster X (labelling a node of T) such that $C \subseteq X$.*

Let $G = (V, E)$ be an undirected graph and let $S \subseteq V$ be a subset of the vertices. We consider the connected components $H_1, \ldots, H_t$ of $G_{V \setminus S}$, i.e., the connected subgraphs obtained from G by deleting all vertices of S and their incident edges. We denote by V_i the vertices of H_i, that is, $H_i = (V_i, E_{V_i})$. Finally, let $S_i \subseteq S$ denote the subset consisting of all vertices of S which have neighbors in H_i.

Define (x, y) to be a *fill-in* edge if $(x, y) \notin E$ and $x, y \in S_i$ for some i, and let F be the set of all fill-in edges. Define the graph $H = (V, E')$ to be the supergraph of G , where $E' = E \cup F$. In other words, an edge is filled in between $u, v \in S$ in E' if there is a path in G from u to v using only intermediate vertices of some component H_i. Thus, each S_i becomes a clique in H_S.

The following is our new result:

Theorem 1 $tw(G) \leq \max_i\{tw(H_S), |S_i| + \{tw(H_i)\}\}$

Proof. Let T_S be a minimum tree decomposition for the subgraph H_S, and let T_i be a minimum tree decomposition for H_i. We will now construct a tree decomposition T for G.

Since the set S_i forms a clique in H_S, by Remark 2, there is a cluster X_i in T_S containing S_i. To form T, we now (i) add the members of S_i to each cluster of T_i, and (ii) add a new edge from the node x_i with label X_i to an arbitrary node v_i of T_i.

We now show that T is a tree decomposition for H and thus also for G. Condition (1) of the definition of tree decomposition is trivial, and condition (3) is proven as follows: Each $T(v)$ for $v \in V \setminus S$ remains unchanged and is therefore a subtree of T. Also, each $T(x)$ for $x \in S$ is a subtree of T since it consists of the union of its former subtree $T_S(x)$ and, for each i in which x has neighbors in H_i, the entire tree T_i along with the new edge (v_i, x_i) connecting H_i with the node with label X_i.

We prove condition (2) in three cases.

Case 1: $u, v \in V \setminus S$: If $(u, v) \in E$, then u and v are in the same connected component, say H_j, and they appear together in some cluster at a node of T_j.

Case 2: $u \in V \setminus S$ and $v \in S$: If $(u, v) \in E$ where u is in the component H_j, then $v \in S_j$ and they now appear together in some (in fact, in every) cluster of T_j where u appears.

Case 3: $u, v \in S$: If $(u, v) \in E$, then $(u, v) \in E'_S$, so u and v co-occur in some cluster at a node of T_S, hence in T.

Thus, T is a tree decomposition for H and thus also for G.

It now remains to show that $w = width(T)$ is at most $\max\{tw(H_S), |S_i| + tw(H_i) | i = 1, \ldots, t\}$.

We first observe that $tw(G_{V \setminus S}) = \max\{tw(H_i) | i = 1, \ldots, t\}$, since the H_i are disjoint graphs.

Let Y be the largest cluster in T, that is, $w = |Y| - 1$. If Y is the label of a node in T_S, then $w = tw(H_S)$. Otherwise, Y is the new label of a node in T_j for some component H_j, that is, $Y = S_j \cup B$ where B is the largest (original) cluster in T_j, and $tw(H_j) = |B| - 1$. Therefore,

$$w = |Y| - 1 = |S_j| + |B| - 1 = |S_j| + tw(H_j)$$

which proves the claim. Q.E.D.

Corollary 1 $tw(G) \leq tw(H_S) + tw(G_{V \setminus S}) + 1$

Proof. This follows since $|S_i| \leq |X_i| \leq tw(H_S) + 1$ for all i and $tw(G_{V \setminus S}) = \max_i\{tw(H_i)\}$.

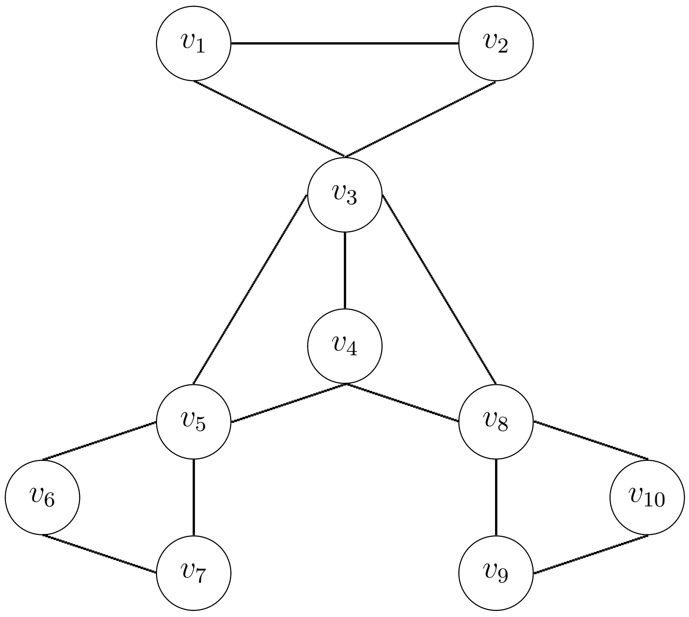

Figure 1: Example graph

Remark 3 *Our result can be seen as a strengthening of the notion of safe separators [3] and of w-cliques [5] where these authors fill-in all pairs of vertices in S making it a clique, and giving the weaker upperbound* $tw(G) \leq |S| + \max_i\{tw(H_i)\} = |S| + tw(G_{V\setminus S})$.

3 Example

Consider the example graph shown in Figure 1. It has a tree decomposition into the following cliques:

$$\begin{aligned} C_1 &= \{v_1, v_2, v_3\} \\ C_2 &= \{v_3, v_4, v_5\} \\ C_3 &= \{v_3, v_4, v_8\} \\ C_4 &= \{v_5, v_6, v_7\} \\ C_5 &= \{v_8, v_9, v_{10}\} \end{aligned}$$

that are all of size 3, and the subgraph with vertices $\{v_1, v_2, v_3\}$ has no tree decomposition into smaller cliques. Thus, its treewidth is 2. (In fact, it a chordal graph, and C_1–C_5 a clique decomposition.)

To illustrate our method, and provide an example for the application described in Section 4, choose $S = \{v_3, v_4, v_5, v_8\}$, thus leaving three connected components $H_1 = \{v_1, v_2\}$, $H_2 = \{v_6, v_7\}$ and $H_3 = \{v_9, v_{10}\}$.

Earlier lemmas give a bound on the treewidth of G as

$$tw(G) \leq |S| + max_i\{tw(H_i)\} = 4 + 1 = 5$$

Using Theorem 2, we obtain a tight bound, as follows. Note that we have $S_1 = \{v_3\}$, $S_2 = \{v_5\}$ and $S_8 = \{v_8\}$, that $H_S = G_S$ since none of the H_i is connected via multiple vertices, and that $tw(H_S) = 2$. Now we have:

$$tw(G) \leq max_i\{tw(H_S), |S_i| + tw(H_i)\} = max_i\{2, 1 + 1\} = 2$$

which is exactly the treewidth of G.

To be fair, we should note that for the earlier result, the best possible choice for S would have been $S' = \{v_3, v_5, v_8\}$, thus leaving an additional disjoint component $H_4 = \{v_4\}$, giving a bound of 4 instead of 5. Using our new Theorem, this separator would not give a bound that is as good because the S_4 associated with the new component H_4 includes all vertices in S', and thus the bound will be 3. While this is still better than the earlier result, it is an indication that S' does not give the best decomposition. This fact is important in the application example below.

4 Application to Constraint Satisfaction Problems

Although this paper may be regarded as purely mathematical, it has its motivation in an important heuristic method for solving various problems including constraint satisfaction ([8]), satisfiability and Bayesian inference ([5]).

For example, when a separator (cutset) S of the constraint graph of a constraint satisfaction problem (CSP) can be found which has certain good treewidth properties, it will allow an efficient solution to the CSP using a hybrid algorithm combining search with dynamic programming.

In search algorithms, there is a tradeoff between (1) the time complexity of searching for a solution, (2) the size of the memory (or cache) to store intermediate computations, and (3) for distributed implementations, the communication complexity for sending and sharing information between parts of the graph. Balancing these three parameters within the resources available is the basis of our motivation.

Using dynamic programming, a CSP can be solved in time and memory exponential in the treewidth of the constraint graph, while with search, it can be solved in time exponential in the number of nodes but space linear in the number of nodes. Let the example graph of Figure 1 represent a CSP with 10 variables with d possible values each, where the arcs correspond to arbitrary unstructured constraints. Dynamic programming would require cubic time and quadratic space in d, whereas tree search would require time on the order of $O(d^{10})$, but memory only linear in d.

When the treewidth of a constraint graph makes the memory required for dynamic programming exceed what is available, it becomes desirable to decompose the problem into pieces with lower treewidth that are solved using dynamic programming, and use search over the variables in the separator. However, the overall complexity is now exponential in the size of the separator plus the largest treewidth of a component. In our example, choosing a decomposition with separator $S' = \{v_3, v_5, v_8\}$, searching through all combinations of values for v_3, v_5 and v_8 and collapsing the remaining nodes using dynamic programming would reduce the space complexity from $O(d^2)$ to $O(d)$, but the time complexity would grow to $O(d^5)$.

The tighter bound given by our theorem allows one decompose the graph recursively to give a more efficient solution. Intuitively, since the complexity of the best known algorithms for solving CSP depends exponentially on the treewidth, a decomposition for which a smaller bound on the treewidth of the original graph can be proven has the potential to better preserve the minimal complexity of the original graph.

In our example, we would pick the larger $S = \{v_3, v_4, v_5, v_8\}$ since it allows to show a bound of $tw(G) \leq 2$ rather than 3. When using S for solving the problem, rather than searching over all combinations of values for variables in S, S would be decomposed again into $S_1 = \{v_3\}$ and $S_2 = \{v_4, v_5, v_8\}$, where $tw(S_2) = 1$.

This shows how to solve the entire CSP in cubic time and linear space in the following steps:

1. first decomposition: remove S and collapse the remaining graph into vertices of S.

    (a) search through all values of v_3 to collapse vertice v_1 into v_2 and then v_3;
    (b) search through all values of v_5 to collapse vertices v_6, v_7 into v_5;
    (c) search through all values of v_8 to collapse verticesv_9, v_{10} into v_8;

2. second decomposition: remove S_1 and use search through all values of v_3 to:
    (a) collapse v_5 into v_4
    (b) collapse v_8 into v_4

3. pick the best solution for v_3 and extend to the collapsed variables in reverse order.

The reader may verify that this algorithm requires only linear space and cubic time in the domain size d, and is thus much better than the decomposition pointed to by earlier results.

We thus believe that Theorem 2 can provide a useful heuristic for decomposing combinatorial problems and solving them efficiently.

Acknowledgements. The authors thank Hans Bodlaender for recommending that we highlight the stronger statement of the Theorem as well as the Corollary. This work was carried out when the second author was a visiting professor at the Artificial Intelligence Laboratory (LIA), Ecole Polytechnique Fédérale de Lausanne (EPFL).

References

[1] Hans L. Bodlaender, A Tourist Guide through Treewidth, *Acta Cybernetica* 11 (1993), 1–21.

[2] Hans L. Bodlaender, Treewidth: Characterizations, Applications, and Computations, *Lecture Notes in Computer Science* LNCS 4271 (2006) 1–14.

[3] Hans L. Bodlaender and Arie M.C.A. Koster, Safe separators for treewidth, *Discrete Mathematics* 306 (2006) 337–350.

[4] Hans L. Bodlaender and Rolf H. Möhring. The pathwidth and treewidth of cographs. *SIAM J. Disc. Meth.* 6 (1993) 181–188.

[5] Bozhena Bidyuk and Rina Dechter. On finding minimal w-cutset problem. In Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI), pp. 4350, Morgan Kaufmann, 2004.

[6] Martin Charles Golumbic, *Algorithmic graph theory and perfect graphs*, Academic Press, New York, 1980. Second edition, *Annals of Discrete Mathematics* **57**, Elsevier, Amsterdam, 2004.

[7] Ton Kloks, Treewidth: Computations and approximations, *Lecture Notes in Computer Science* LNCS 842 (1994) 1–209.

[8] A. Petcu and B. Faltings. MB-DPOP: A New Memory-Bounded Algorithm for Distributed Optimization. Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI-07, Hyderabad, India, Jan, 2007, pp. 1452-1457.

[9] Neil Robertson and Paul D. Seymour, Graph minors. II. Algorithmic aspects of tree-width, *J. Algorithms* 7 (1986), 309–322