CS762: Graph-Theoretic Algorithms
Lecture 19: Separators
February 18, 2002

Scribe: Fred Comeau

Abstract

Algorithms for identifying partial 2-trees are presented, which exploit relationships between partial 2-trees and SP-graphs. $\frac{2}{3}$-separators of graphs are introduced, and rigorously defined in two different ways. These two definitions are then shown to be equivalent. It is also shown that vertex weighted trees always have a $\frac{2}{3}$-separator of size one.

1 Introduction

In this paper we continue with the consideration of k-trees. We consider the relationships discovered previously between partial 2-trees and SP-graphs, and use these to derive algorithms which assist in identifying partial 2-trees. These algorithms are presented without proof.

From here we introduce the concept of graph separators, which is a subset of the vertices of the graph such that the induced graph created by the removal of the separator contains a number of connected components that meet certain conditions. We present two alternative definitions of separators, and then proceed to show they are equivalent. With these definitions established, we then move to the consideration of separators of trees, and prove some results regarding the size of the separator of a tree under certain conditions.

2 Recognizing SP-graphs

We first study how to test whether a given graph G is an SP-graph. We have previously established the following equivalences:

$$\begin{aligned} \text{partial 2-trees} &\iff SP\text{-graphs} \\ &\iff \text{every biconnected component is a 2-terminal } SP\text{-graph} \end{aligned}$$

The second equivalence gives rise to an algorithm to determine if G is an SP-graph (and hence a partial 2-tree, by the first equivalence), *SPTEST*:

SPTEST
- compute all biconnected components
- determine whether each biconnected component is a 2-terminal SP-graph

The first step of *SPTEST* can be done in $O(m+n)$ time [Tar72]. For the second step, we will give two approaches. Assume that C is a biconnected component. The first approach uses triconnected components and the $SPQR$-tree of triconnected components (see [HT73] and [BT96] for details) and works as follows:

2TSPTEST$_1$

compute the *SPQR-tree*, T of C (also gives the triconnected components of C)
if T has a *Q-node* then C is not a 2-terminal SP-tree
otherwise T is the SP-tree of C

We will give no further details of the exact definitions, or why this works correctly. Since the $SPQR$-tree can be computed in $O(m+n)$ time [BT96], this algorithm runs in $O(m+n)$ time, and thus the total running time of *SPTEST* is also $O(m+n)$.

An alternate algorithm that involves replacing the occurrence of certain edge configurations follows:

2TSPTEST$_2$

while G has ≥ 2 edges
  if G has a vertex of degree 2, perform the replacement of Figure 1
  if G has a multiple edge, perform the replacement of Figure 2
  if neither of theses cases apply, G is not a 2-terminal SP-graph
G is a 2-terminal SP-graph

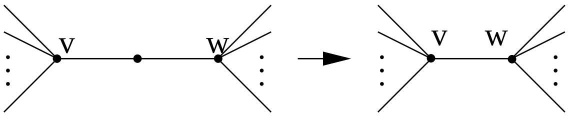

Figure 1: Serial replacement.

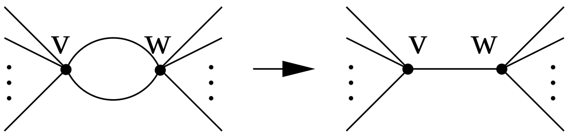

Figure 2: Parallel replacement.

Again we omit the details of why this algorithm works correctly. Clearly it is polynomial; with the right data structure a linear running time can be achieved. There also exist algorithms for recognizing partial 3-trees, but we will not study them here. See, for example, [AP86].

3 Separators

Recall that a tree can be broken into at least two parts by removing just one vertex. In fact, this holds for almost any vertex of the tree. So it is fair to ask whether a tree can be broken into pieces such that additional constraints are satisfied. This motivates the following definition:

Definition 1 *A $\frac{2}{3}$-separator S of a graph G is a set $S \subseteq V$ such that in $G[V-S]$ every connected component contains at most $\frac{2}{3}|V|$ vertices (see Figure 3).*

This definition can be easily extended to a weighted version of the graph. We are given some graph $G=(V,E)$ of n vertices, with nonnegative vertex weights $\{w(1), w(2), \ldots, w(n)\}$. For convenience, given some set $V' \subseteq V$, we define

$$w(V') = \sum_{v \in V'} w(v).$$

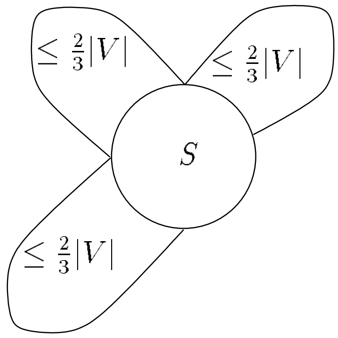

Figure 3: Illustration of a $\frac{2}{3}$-separator.

Then $S \subseteq V$ is a $\frac{2}{3}$-separator of G if for any connected component C formed by the removal of S, $w(C) \leq \frac{2}{3}w(V)$. Notice the weighted version easily reduces to the non-weighted version by setting all vertex weights to 1. This means that if we prove results for the weighted case, the unweighted case trivially follows. More commonly used in the literature is the following alternate definition:

Definition 2 *Given a graph $G = (V, E)$ of n vertices, with non-negative vertex weights $\{w(1), w(2), \ldots, w(n)\}$. A (weighted) $\frac{2}{3}$-separator is a partition $A \cup B \cup S = V$ such that:*

- $w(A) \leq \frac{2}{3}w(V)$
- $w(B) \leq \frac{2}{3}w(V)$
- *There exists no edge between a vertex in A and a vertex in B.*

See Figure 4. Notice there is nothing special about the choice of $\frac{2}{3}$, and for any arbitrary α such that $0 < \alpha < 1$, we can talk about α-separators. We will restrict ourselves to $\alpha = \frac{2}{3}$ for the remainder of this paper, however. Also notice that we can easily choose a separator $S = V$, however we will generally consider cases where S is small. One reason for this is that we can use a small separator to design an efficient divide and conquer algorithm. We use the separator to divide the graph into smaller pieces, solve the problem on these pieces, and then combine the solutions into one for the entire graph. Clearly, a large separator will make combining the solutions more difficult, and hence result in an inefficient algorithm.

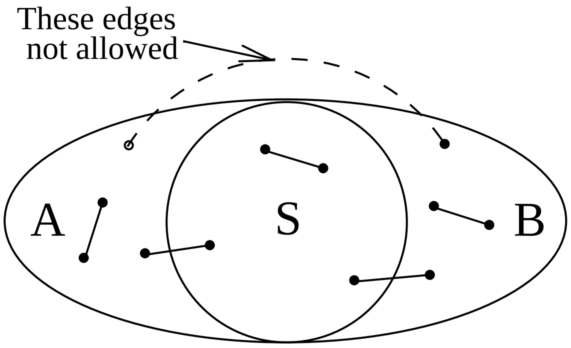

Figure 4: Alternate separator definition.

Claim 1 *Definition 1 and Definition 2 are equivalent.*

Proof: Definition 2 $\Rightarrow$ Definition 1 is trivial, since by definition, the sum of the weights of all connected components in A is at most $\frac{2}{3}w(V)$, thus breaking A into these components we are guaranteed each has weight at most $\frac{2}{3}w(V)$. This also holds for B.

We now examine the other direction. Assume we have a separator S, with corresponding components $C_1, C_2, \ldots, C_k$ of $G[V-S]$ having weight at most $\frac{2}{3}w(V)$ each. We can break the proof into three cases.

Case 1:

$$\sum_{i=1}^{k} w(C_i) \leq \frac{2}{3}w(V).$$

In this case, set $A = \bigcup_{i=1}^{k} C_i$ and $B = \emptyset$. Clearly the conditions hold.

Case 2: There is a component C_j $(1 \leq j \leq k)$ such that $w(C_j) \geq \frac{1}{3}w(V)$. By definition, we have $w(C_j) \leq \frac{2}{3}w(V)$. This means that for all $i \neq j$ we have

$$\sum_{i \neq j} w(C_i) \leq w(V) - w(C_j) \leq \frac{2}{3}w(V).$$

Thus we can then simply set $A = C_j$ and $B = \bigcup_{i \neq j} C_i$, and we have the appropriate sets.

Case 3: For all $1 \leq i \leq k$ we have $w(C_i) < \frac{1}{3}w(V)$. Let j be maximal such that

$$\sum_{i=1}^{j} w(C_i) < \frac{1}{3}w(V).$$

Note that $j < k$ since we are not in Case 1. Since $w(C_{j+1}) \leq \frac{1}{3}w(V)$ (otherwise we would be in Case 2), the sum of the weights of the first $j+1$ components puts us somewhere between the $\frac{1}{3}w(V)$ mark and the $\frac{2}{3}w(V)$ mark, as in Figure 5. This means that the sum of the weights of the remaining components must be less than $\frac{2}{3}w(V)$, as the total of the two groups can be no more than $w(V)$. This gives us our two components:

$$\begin{array}{rl} A = \bigcup_{i=1}^{j+1} C_i & w(A) \leq \frac{2}{3}w(V) \\ B = \bigcup_{i=j+2}^{k} C_i & w(B) \leq \frac{2}{3}w(V). \end{array}$$

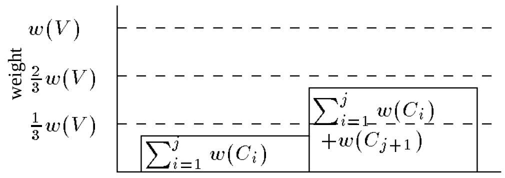

Figure 5: Graph separator.

□

We are now in a position to apply this definition:

Theorem 1 *Every tree T has a (weighted) $\frac{2}{3}$-separator of size 1.*

Proof: We first pick an arbitrary root for the tree. Let T_v represent the subtree rooted at vertex v. After scaling the vertex weights, we assume $w(T) = w(V) = 1$. We then execute the following algorithm:

TREESEP
  set v^* to the root of the tree – we now have $w(T_{v^*}) = w(T) = 1 > \frac{1}{3}$
  while v^* has a child x with $w(T_x) > \frac{1}{3}$
    set $v^* = x$
  return v^*

We will see that v^* is the required separator. We first illustrate the algorithm with the example in Figure 6. We assume that each node of the tree is equally weighted, and the weights are scaled. The algorithm starts at the root, which has a weight of 1. After trying all children of the root, we find that the subtree rooted at the third child has weight $> \frac{1}{3}$. We set v^* to be this child. The weight of the subtree of one of the children of this vertex is still $> \frac{1}{3}$, and thus we set v^* to be this child. None of the trees rooted at the new v^*'s children have weight $> \frac{1}{3}$, and thus we now meet the stopping condition, and return this vertex as the separator.

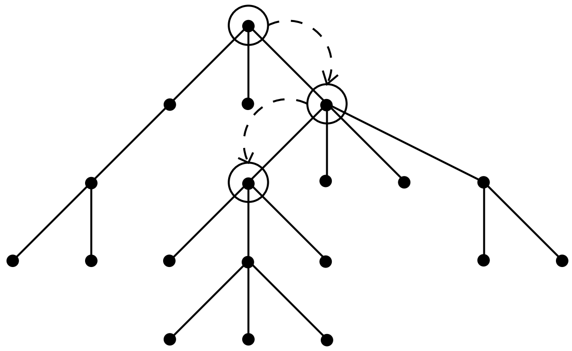

Figure 6: Tree separator example.

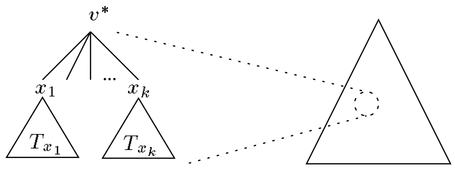

Figure 7: Algorithm snapshot.

To understand why this works, consider the step when the separator is found, as in Figure 7. We have some vertex in the tree, v^*. We know that $w(T_{v^*}) > \frac{1}{3}$, because this is maintained throughout the algorithm. This means that $w(T - T_{v^*}) \leq \frac{2}{3}V$. All children $\{x_1, \ldots, x_k\}$ are such that $w(T_{x_i}) \leq \frac{1}{3}$, otherwise we would not have stopped here, so we have the desired separator. □

Note that we cannot find a $\frac{2}{3}$-separator by removing a single edge. To understand why, consider the tree of Figure 8, where all vertices have equal weight. No matter which edge is removed, we have a component with (scaled) weight of $\frac{1}{9}$, and another with weight of $\frac{8}{9}$.

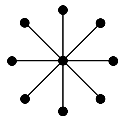

Figure 8: Another tree separator example.

References

[AP86] Stefan Arnborg and Andrzej Proskurowski. Characterization and recognition of partial 3-trees. *SIAM journal on algebraic and discrete methods*, 7(2):305–314, 1986.

[BT96] G. Di Battista and R. Tamassia. On-line maintenance of triconnected components with *SPQR*-trees. *Algorithmica*, 15:302–318, 1996.

[HT73] J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM journal on computing*, 2:135–138, 1973.

[Tar72] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM journal on computing*, 1:146–160, 1972.