CS762: Graph-Theoretic Algorithms
Lecture 17: Partial 1-trees and 2-trees
February 13, 2002

Scribe: Alastair Farrugia

Abstract

We show that 1-trees are trees, and therefore partial 1-trees are forests. The main purpose of this lecture is to introduce series-parallel graphs (SP-graphs); and show that every 2-tree is a simple 2-terminal SP-graph, and every 2-terminal SP-graph is a partial 1-tree or partial 2-tree.

1 Introduction

In this lecture we start exploring the structure of (partial) k-trees for small values of k. For $k = 1$ it is not too difficult to show that 1-trees are exactly[1] trees, and partial 1-trees are exactly forests. For $k = 2$ we introduce the notion of series-parallel graph, and establish the following relationships:

$$2 - \text{trees} \subsetneqq \text{2-terminal SP-graphs} \subsetneqq \text{partial 2-trees or partial 1-trees} \qquad (*)$$

We recall that a k-tree is constructed as follows:

(a) $v_1, \ldots, v_k$ is a k-clique,

(b) for all $i > k$, v_i's predecessors (its neighbours among $\{v_1, \ldots, v_{i-1}\}$) form a k-clique.

We can orient each edge (v_i, v_j) from v_i to v_j, where $i < j$. Then $v_1, \ldots, v_n$ is a perfect elimination order such that $\text{indeg}(v_i) = k$ for all $i > k$.

A partial k-tree is anything that can be obtained from a k-tree by deleting edges (but not vertices); in other words, a partial k-tree is a spanning subgraph of a k-tree.

2 Trees, forests and (partial) 1-trees

Since a 1-clique is just a vertex, the definition above becomes very simple for $k = 1$: a 1-tree is a graph whose edges can be oriented so that $\text{indeg}(v_i) = 1$ for $i > 1$.

Lemma 1 *Every 1-tree is a tree.*

Proof: We proceed by induction on $|V(G)|$. If $|V(G)| = 1$, G is just a single vertex. Otherwise, $G - v_n$ is a 1-tree on $n - 1$ vertices, and thus a tree by induction; that is, $G - v_n$ is connected and has no cycles. Now v_n is adjacent to exactly one neighbour in $G - v_n$, so G is also connected, and

[1] We use the term 'exactly' to mean that every 1-tree is a tree *and* every tree is a 1-tree. Similarly, every partial 1-tree is a forest, and every forest is a partial 1-tree.

cannot contain any cycles passing through v_n (so it cannot have any cycles at all). Thus G is a tree.

Alternatively, note that since $G - v_n$ is a tree, it is connected and has exactly $n-2$ edges; therefore G is also connected and has $n-1$ edges, so it is also a tree. □

Lemma 2 *Every tree is a* 1*-tree.*

Proof: We again proceed by induction on $|V(G)|$. If $|V(G)| = 1$, G is just a single vertex, which is a 1-tree. Otherwise, let w be a leaf of the tree. $G - w$ is also a tree, and thus it is a 1-tree by induction, and we can label its vertices $v_1, \ldots, v_{n-1}$. We now label w as v_n, and note that its neighbours among $v_1, \ldots, v_{n-1}$ consist of just one vertex (that is, a 1-clique). So G is a 1-tree. □

Since forests are exactly the subgraphs of trees, and partial 1-trees are exactly subgraphs of 1-trees, it follows that partial 1-trees are exactly forests.

3 (s,t)-series-parallel graphs

An (s,t)*-series-parallel graph* (also called (s,t) *SP-graph*, or 2*-terminal SP-graph*) is defined recursively as follows:

Base case: A single edge (s,t) is an (s,t)-series-parallel graph
Step: If G_1 is an (s_1,t_1)-series-parallel graph and G_2 is an (s_2,t_2)-series-parallel graph, we can combine them in one of the two following ways:

- Combination in series. The graph formed by identifying t_1 with s_2 given an (s_1,t_2) SP-graph (Fig. 1).
- Combination in parallel. The graph formed by identifying s_1 with s_2, and t_1 with t_2, is an (s_1,t_1) SP-graph (Fig. 2). Note that this may create parallel edges, so SP-graphs need not be simple.

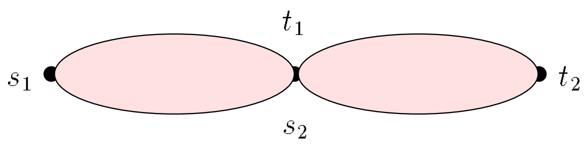

Figure 1: Combination in series.

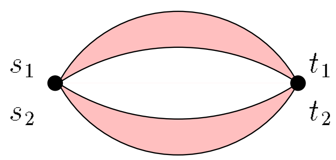

Figure 2: Combination in parallel.

There is a convenient way of representing the sequence of combinations used in constructing such a graph. An SP-tree T for G is defined as follows:

Base Case: If G is just an edge, T is a single node containing an edge whose terminals are labelled s and t.

Step: If G is the combination of G_1 and G_2, in series or in parallel, then T has a node labelled S or P; this node contains G (with its terminals labelled), and its children are the SP-trees for G_1 and G_2.

Note that in S-nodes and P-nodes we can omit the graph G, so long as we keep track of which terminals from the children correspond to which terminals in the S- or P-node. The terminals are important because non-isomorphic (s,t)-series-parallel graphs can be obtained by combining the same graphs with the terminals labelled differently, as shown in Fig. 3).

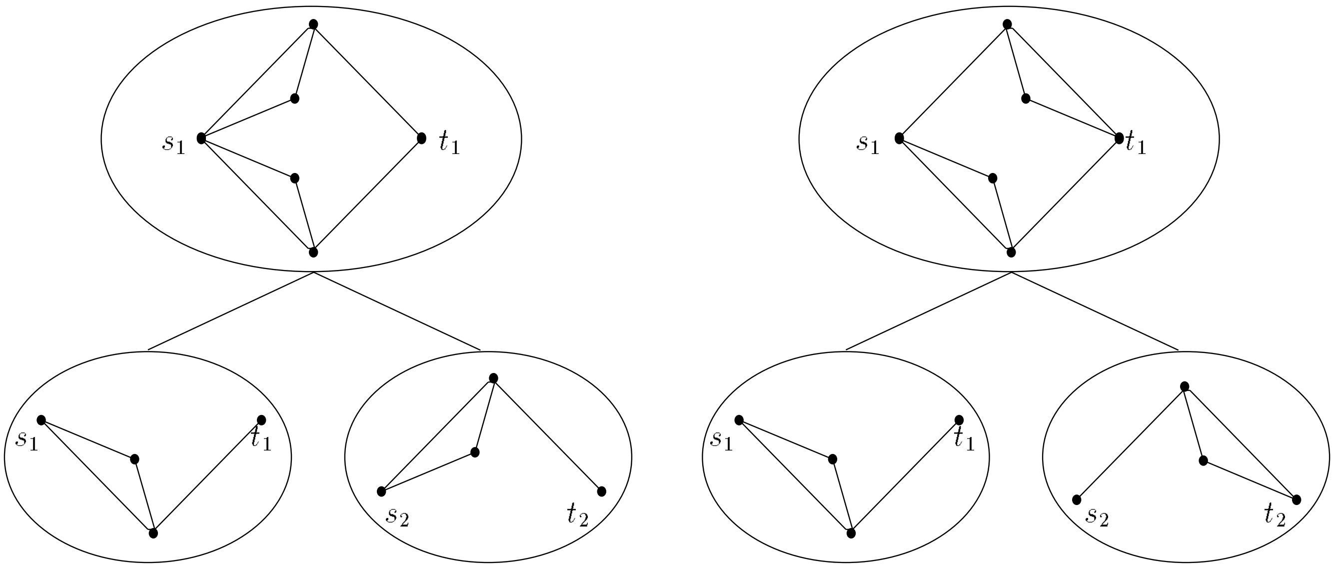

Figure 3: Applying the same combinations with different terminals may lead to different results.

So long as we keep track of the terminals, every SP-tree corresponds to a unique 2-terminal SP-graph. In Fig. 4 we show SP-trees for all 2-terminal SP-graphs with 3-edges, while an SP-tree for C_4 is displayed in Fig. 5.

We can think of 's' and 't' as 'source' and 'target', and direct edges from s to t. We can also draw 2-terminal SP-graphs in the plane, with s on the left and t on the right. The two combination methods preserve these properties, and do not add or delete edges.

4 2-terminal SP-graphs and 2-trees

In this section we explore the relationship between 2-trees, partial 2-trees, and simple 2-terminal SP-graphs. More specifically, we will establish the containments given in the Introduction.

Note that the last SP-tree in Fig. 4 gives us a simple 2-terminal SP-graph that is not a 2-tree (because, for example, every 2-tree with more than 2 vertices contains a triangle). Then again, one would not expect a path to be a 2-tree because all 2-trees can be shown to be biconnected. However, C_4 is a biconnected 2-terminal SP-graph (its SP-tree is shown in Fig. 5) that is not a

2-tree (because it has no triangle). So even with biconnectivity, the first inclusion in $(*)$ is still not an equality. As regards the second inclusion, $K_{1,3}$ is a 1-tree that is not given by any of the SP-trees in Fig. 4.

This shows that the reverse containments in $(*)$ do not hold. We now proceed to establish the forward containments.

Theorem 1 *Every* 2*-tree is a simple* 2*-terminal series-parallel graph.*

Proof: We will show this by induction on n, the number of vertices of the 2-tree. The smallest 2-tree is the edge with two vertices; this is a 2-terminal SP-graph by definition.

For $n \geq 3$, let G be a 2-tree constructed by adding vertices in the order $v_1, \ldots, v_n$. Then $G - v_n$ is also a 2-tree, and by induction it is a 2-terminal SP-graph. When we add v_n, we make it adjacent to the end-vertices of some edge, say (x, y). The SP-tree for $G - v_n$ must contain a leaf with the edge (x, y). We can now obtain an SP-tree for G by replacing this leaf with a node containing a triangle with vertices x, y, v_n (where x, y are the terminals). This new node has two children — a leaf with the edge (x, y), and the SP-tree for the path x, v_n, y. See Fig. 6. $\square$

Theorem 2 *Every simple* 2*-terminal SP-graph* G *is a partial* 1*-tree or partial* 2*-tree.*

Proof: We recall (from a previous lecture) that showing G to be a partial 1- or 2-tree is equivalent to showing that it has a tree-decomposition of treewidth at most 2. In other words, there is a tree T where:

- each node is labelled with at most $2 + 1 = 3$ vertices of G
- for each edge (v, w) of G, there is a node of T containing both v and w
- for each vertex v of G, the set of nodes of T that contain v form a connected subgraph of T.

For an (s, t) SP-graph G we will construct a tree T that also satisfies:

- T has a node containing s and t.

Loosely speaking, the SP-tree defines such a tree decomposition. The precise proof is by induction on h, the height of the SP-tree for G. When $h = 1$, the SP-tree consists of a single node containing an edge (s, t); we define T to have a single node labelled with s and t. When $h > 1$, the top-node is either an S-node or a P-node.

Case: S-node (Fig. 1). G is the combination in series of a (u, v) SP-graph G_1 and a (v, w) SP-graph G_2; by induction they have tree-decompositions given by T_1, T_2. Define T to have a new node i labelled u, v, w, and make i adjacent to a node of T_1 containing v and u, and to a node of T_2 containing v and w.

Case: P-node (Fig. 2). G is the combination in parallel of an (s, t) SP-graph G_1 and an (s, t) SP-graph G_2. By induction they have tree-decompositions given by T_1 and T_2, which each have a node labelled with s, t (and possibly some third vertex v_1 or v_2). Making these two nodes adjacent gives us T as required. $\square$

Note that in the case of a P-node, if the T_1 node is labelled *only* with s and t, we could even merge it with the T_2 node, which would make the tree decomposition a bit more compact.

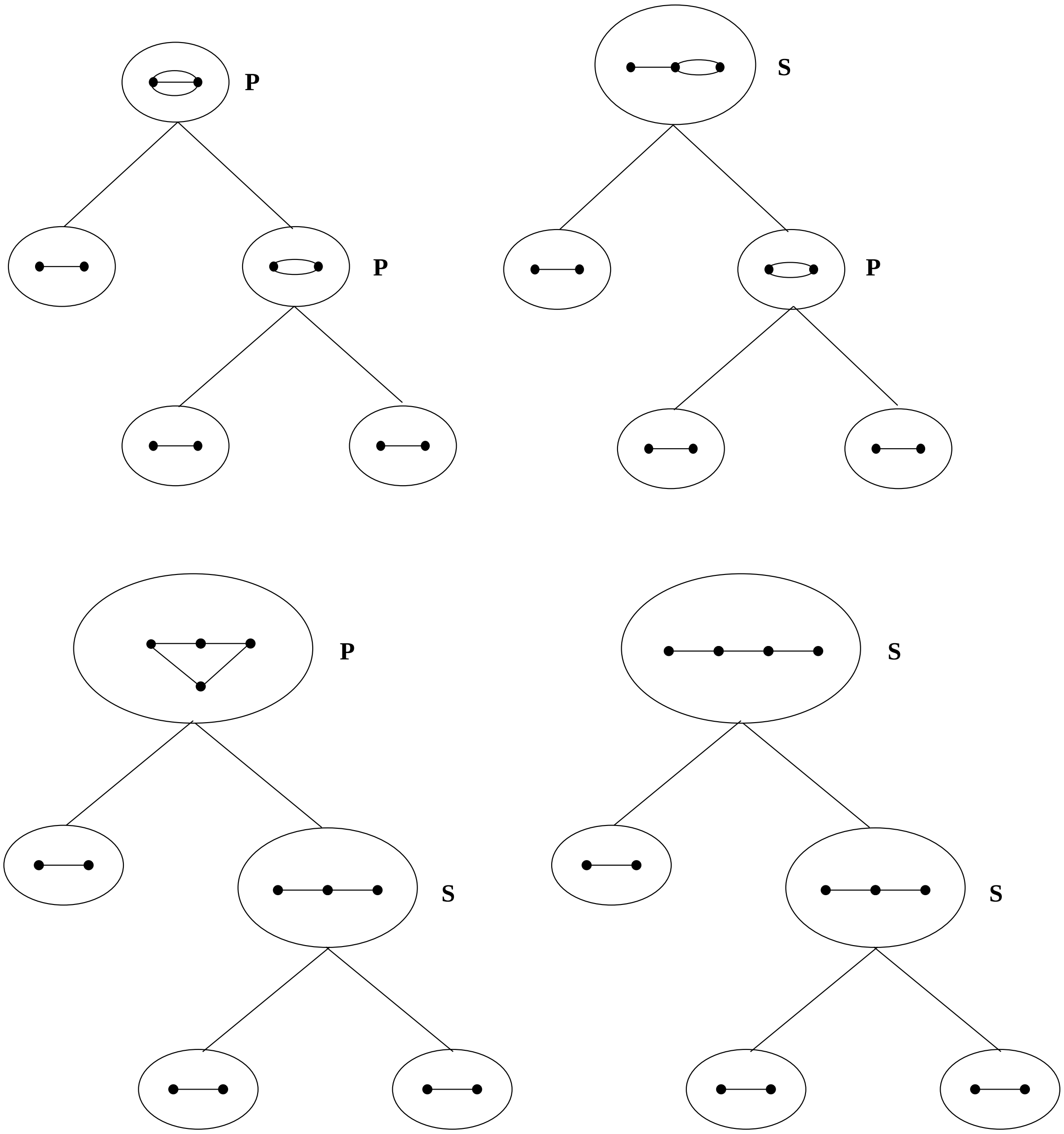

Figure 4: SP-graphs with 3 edges

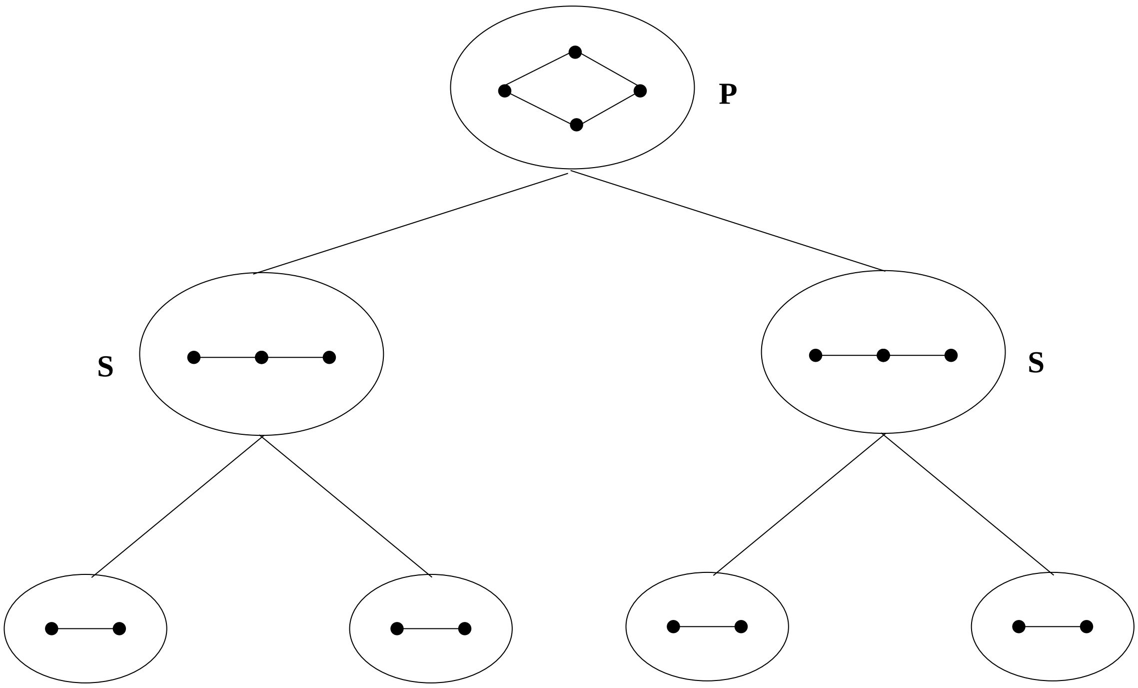

Figure 5: C_4 is a 2-terminal SP-graph

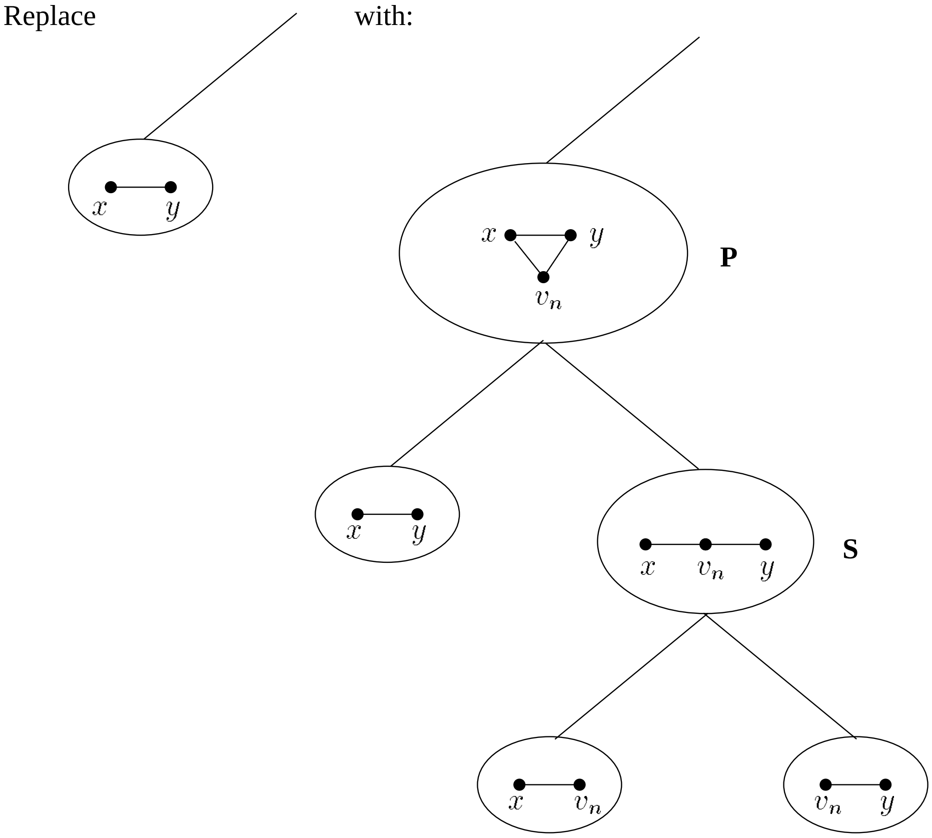

Figure 6: Building an SP-tree for G from that for $G - v_n$.