A Fast Practical Algorithm for the Vertex Separation of Unicyclic Graphs

by

Minko Marinov Markov
B.Sc., Technical University-Sofia, Bulgaria, 1994

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming to the required standard

© Minko Marinov Markov, 2004

University of Victoria

All rights reserved. This thesis may not be reproduced in whole or in part, by photocopy or any other means, without permission of the author.

Supervisor: Dr. John A. Ellis

ABSTRACT

The vertex separation of a graph is the minimum vertex separation of a linear layout of that graph over all its linear layouts. A linear layout of a graph is an arrangement of its vertices in a line and the vertex separation of a linear layout is maximum number of vertices to the left of any intervertex "gap" that are adjacent to vertices to the right of that gap, over all gaps. A unicyclic graph is a connected graph with precisely one cycle that is, a tree plus an extra edge. We present a $\mathcal{O}(n \lg n)$ algorithm to compute the optimal vertex separation of unicyclic graphs. The algorithm is "practical" in the sense that it is easily implementable. Furthermore, the algorithm outputs a layout for the unicyclic graph of minimum vertex separation.

Examiners:

Contents

List of Figures

List of Definitions

Notation

We use without defining them, the following symbols to denote operations: "$\cup$" denotes set union, "$\cap$" denotes set intersection, and "$\setminus$" denotes set difference. Besides that, "$\cap$", "$\cup$", and "$-$" denote operations on graphs, to be defined later.

We use curly brackets "$\{$" and "$\}$" to denote sets. Parentheses "$($" and "$)$" are used for undirected graph edges, as in "(u, v)", and for ordered sets too. Angle brackets "$\langle$" and "$\rangle$" denote either ordered tuples of different-type objects, *e.g.* our notation for graph is "$G = \langle V, E \rangle$".

The center-dot "$\cdot$" symbol denotes the operation list concatenation.

We write the names of computational problems in small capital letters, for instance "PATHWIDTH". The names of graph parameters are typed in small letter, *e.g.* "pathwidth".

By "n" we denote the number of vertices of the graph under consideration unless we specify otherwise.

Chapter 1

Introduction

1.1 The Vertex Separation Problem

This thesis presents computational results on the graph theory problem VERTEX SEPARATION. It is equivalent to the well-known problem PATHWIDTH. The concept of pathwidth of a graph was introduced by Robertson and Seymour in [RS83] and [RS86] in a purely theoretical context, but turned out to be useful in practice as well. Several computational problems that were apparently conceived independently turned out to be equivalent to PATHWIDTH and thus to VERTEX SEPARATION (see [Kin92] for the equivalence between PATHWIDTH and VERTEX SEPARATION). Some of them are GATE MATRIX LAYOUT, introduced by Lopez and Law [LL80], NODE SEARCH NUMBER, introduced by Kirousis and Papadimitriou (see [KP85] and [KP86]), INTERVAL GRAPH EXTENSION, introduced by Fujisawa and Kashiwabara [FK79], and INTERVAL THICKNESS [KP85]. EDGE SEARCH NUMBER, introduced by Parsons [Par78], is also closely related to the above. These ideas have arisen independently in seemingly unrelated areas, with one thing in common: there is an underlying graph structure, and the goal is to find a linear ordering of the vertices such that the graph with this ordering

is, informally, as thin as possible. In VLSI design, the MATRIX PERMUTATION problem [Möh90] has a graph structure determined by the gates of a chip plus the interconnections between them in a matrix-like arrangement, and the goal is to find an ordering (permutation) of the gates which minimizes the width of the matrix. Weinberger arrays, gate matrix layouts and programmable logic arrays are architectures whose efficient implementation is related to MATRIX PERMUTATION [Möh90]. In natural language processing, the graph structure is the dependency graph of a sentence [KT92], and the linear ordering of its vertices is related to the sequential manner in which the brain does the initial processing of a sentence [KT92]. In computational biology, more specifically in generating physical mappings for DNA, an important problem is GENERATING PHYSICAL MAPPINGS IN THE PRESENCE OF FALSE NEGATIVES which is equivalent to VERTEX SEPARATION [GGKS95]. The linear ordering there comes from the linear structure of the DNA molecule. Graph searching problems were initially non-discrete pursuit-evasion problems of an intruder in a graph, independently investigated in the West [Par78] and in the East [Pet82], [Pet83]. In them, the graph structure is part of the definition of the problem, and the linear ordering of the vertices is related to the sequential nature of time—since searching is a temporal process.

Our research is on the vertex separation of a class of graphs and that is why the thesis has that name in its title. The survey chapter presents results on the pathwidth, not vertex separation, of graph classes. The reason is that PATHWIDTH is more widely known and we like to preserve the original name those researchers used.

Besides being an interesting parameter itself, the vertex separation number of a graph is important in one more way. Some problems that are NP-hard in general are solvable in polynomial time when restricted to graphs of bounded

vertex separation (pathwidth, in the original formulation, for instance NETWORK RELIABILITY and 2-EDGE-CONNECTED RELIABILITY [LMC00]).

Computing the vertex separation number of a graph is computationally hard. In fact, VERTEX SEPARATION is NP-complete even for rather resticted classes of graphs, such as planar graphs with maximum degree three [MS88], chordal graphs [Gus93], and bipartite graphs [GGKS95]. Fujisawa and Kashiwabara were the first to prove the NP-completeness for arbitrary graphs [FK79], using a formulation that is close to MINIMUM INTERVAL THICKNESS. The NP-completeness for the formulation of PATHWIDTH was proven first by Arnborg, Corneil and Proskurowski [ACP87, Corollary 4.3.ii].

The NP-completeness of a computational problem is a very strong evidence that it is intractable in general. No one has proved formally that NP-complete problems are intractable, but it is something that practically all specialists in the field believe: under the current conventional notions of what a computing device is, it is impossible to compute on any conceivable physical computer the answer to an NP-complete problem for large inputs that can easily arise in practice, *e.g.* a graph with a thousand vertices.

A positive result on the pathwidth of general graphs was obtained by Ton Kloks. He proves that PATHWIDTH is solvable in linear time for a fixed value of the parameter [Klo94, Chapter 13]. That is, when k is constant, we can answer in linear time whether the pathwidth of any graph is at most k. However, that does not help our research because we investigate graphs such that their pathwidth is not bound by any constant.

A lot of progress in terms of algorithms has been made on the exact computation of the pathwidth of some types of graphs, especially in the perfect graphs hierarchy. Unfortunately, a lot of these algorithms are not feasible from a prac-

tical point of view, although they are polynomial-time. We call "practical" an algorithm that can be implemented in software to produce results on a real computer, in a reasonable amount of time. There are numerous algorithms that are theoretically excellent but impractical, for instance the linear time algorithm of Bodlaender and Kloks on graphs of bounded treewidth [BK96]. At the moment there are practical algorithms for the vertex separation of trees, interval graphs, cographs, and, with the current work, unicyclic graphs.

Substantial progress has been made on the approximation of pathwidth [DKL87], [GLY98], [BGHK95], [KB92], [BF00]. Approximation in the context of hard computational problems (not restricted to graphs only) means the following. We construct an algorithm that produces answer that is not necessarily optimum but is guaranteed to be no worse than $f(n)$ times optimal or $f(n)$ plus optimal, where $f(n)$ is some function of n or a constant. It is known that pathwidth can not be approximated within an additive constant of the optimum [DKL87], unless P equals NP. However, it is not known whether pathwidth can be approximated within a multiplicative constant of the optimum.

1.2 The Vertex Separation of Unicyclic Graphs

A unicyclic graph is a tree plus an extra edge. There is a linear time algorithm for VERTEX SEPARATION on trees [EST94] which is practical and runs in $\mathcal{O}(n)$ time. It is perhaps counterintuitive that adding just one edge to a tree makes it considerably more difficult to find the vertex separation but our experience is that that is indeed the case.

The algorithm that we propose for unicyclic graphs runs in time $\mathcal{O}(n \log n)$ and uses the algorithm for trees as a subroutine. Prior to our algorithm it was known that VERTEX SEPARATION is solvable in polynomial time on unicyclic

graphs. The fastest algorithm was the one of Bodlaender and Kloks [BK96] with complexity, in case of unicyclic graphs, of $\Omega(n^{11})$.

The motivation for the current work was constructing exact (not approximating), fast, practical algorithms for VERTEX SEPARATION on sub-planar graphs, that is, classes of graphs that are proper subsets of planar graphs. As we said, the problem is NP-complete on planar graphs which gives practically no hope for such an algorithm on them. We discovered that constructing such algorithms is extraordinarily difficult when one starts from first principles. After a considerable time and effort, our opinion is that the presented algorithm on unicyclic graphs is an important first step towards discovering such algorithms on cactus graphs and even outerplanar graphs, which in turn may be starting points for algorithms on more complex sub-planar graphs. We discuss that in the section on future work.

1.3 Contributions of This Thesis

As we already said, we present the first fast practical algorithm for VERTEX SEPARATION on unicyclic graphs. Furthermore, the algorithm outputs a linear layout on minimum vertex separation; that is, the algorithm is constructive.

Another contribution is the idea of layout reversal. To each linear layout correspond one or more node search strategies. We prove that the reverse of a search strategy using at most k searchers is a valid strategy as well and it uses at most k searchers, too. A reversal of a layout is any layout corresponding to a reverse search strategy.

We use extensively a concept that we call "extensibility". The extensibility of a layout or a graph is always with respect to one or two vertices. The natural definition of extensibility associates the vertex with a specific direction in case of one vertex, or each of the two vertices with a specific direction in case of two

vertices. Here, directions is either left or right. Using our results on reversability of layouts, we prove that the extensibility property is independent on the directions; that is, a layout is left-extensible with respect to a vertex u and a number k if and only if there exists a layout that is right-extensible with respect to u and k, and similarly for two vertices u and v.

A concept analogous to extensibility was used by Skodinis [Sko03]. However, he uses it with respect to one vertex only. We discovered that extensibility with respect to two vertices in opposite directions is essential for the efficient solving of VERTEX SEPARATION on unicyclic graphs.

Our last contribution is the discovery and verification of a necessary and sufficient condition for the extensibility of a unicyclic graph with respect to two cycle vertices being at most k. The analysis has a complicated case, subcase structure.

Chapter 2

Common Definitions

2.1 Graph Theory Definitions

An *undirected graph* $G = \langle V, E \rangle$ is an ordered pair of two disjoint sets, a non-empty finite set V whose elements are called *vertices*, and a possibly empty set E of unordered pairs of vertices, each pair being called *an edge*. We consider only undirected graphs, therefore when we say "graph", we mean undirected graph. The vertex set of G is denoted by $V(G)$, and the edge set by $E(G)$. If u and v are vertices, and $e = (u, v)$ is an edge, we say that u and v are *the endpoints of* e, or *the endvertices of* e, and that u and v are *adjacent*. If e is an edge and u is an endpoint of it, we say that e *is incident with* u. The parentheses in the edge notation stand for an unordered pair, *i.e.* (u, v) is the same object as (v, u). The edges of the graphs that we consider always have distinct endpoints: edges of the form (u, u) are not allowed. We do not consider "multiple edges" or "self loops" because they do not affect graph parameter we are interested in, the vertex separation.

On some occasions when $G = \langle V, E \rangle$ and $u \in V$, we write $u \in G$. Strictly speaking, this is an abuse of the "$\in$" notation, as G is not a set of vertices, but,

if there is no ambiguity, we use that abbreviation.

The set of all vertices adjacent to u is denoted by $\mathrm{adj}(u)$. The *degree* of u is $\deg(u) = |\mathrm{adj}(u)|$.

When we say that a vertex u is adjacent to a set of vertices W, we mean that u is adjacent to each vertex in W. If $W = \{w_1, w_2, \ldots, w_q\} \subset V$, then $\mathrm{adj}(W)$ is the vertex set $\left(\bigcup_{i=1}^{q} \mathrm{adj}(w_i)\right) \setminus \left(\bigcup_{i=1}^{q} \{w_i\}\right)$.

If $G = \langle V, E\rangle$ is a graph, then any graph $G_1 = \langle V_1, E_1\rangle$, where $V_1 \subseteq V$ and $E_1 \subseteq E$, is called *a subgraph of* G, and we denote that by $G_1 \subseteq G$. If $V_1 \subset V$ or $V_1 = V$ but $E_1 \subset E$, then we call G_1 *a proper subgraph of* G. If V_2 is a vertex set such that $V_2 \subseteq V$, the subgraph $G_2 = \langle V_2, E_2\rangle$, where $E_2 = \{\{u, v\} \in E \mid u, v \in V_2\}$, is the subgraph *induced* by V_2.

Let $G = \langle V, E\rangle$ be a graph and $G_1 = \langle V_1, E_1\rangle$ be a subgraph of G. To *delete* G_1 *from* G means to transform G into $G_2 = \langle V_2, E_2\rangle$, where $V_2 = V \setminus V_1$ and $E_2 = \{(u, v) \mid (u, v) \in E \text{ and } \forall (x, y) \in E_1, \{x, y\} \cap \{u, v\} = \emptyset\}$. The result of the deletion of G_1 is denoted by $G_2 = G - G_1$. To *remove an edge* e *from* G means to transform G into $G_3 = \langle V, E \setminus \{e\}\rangle$. We write $G_3 = G - e$. If $E_u = \{(u, z) \in E \mid z \in \mathrm{adj}(u)\}$, then the graph $G_4 = \langle V \setminus \{u\}, E \setminus E_u\rangle$ is obtained from G by *deleting the vertex* u, and we denote that by $G_4 = G - u$.

A path from u_1 to u_n is a sequence $p = u_1, e_1, u_2, e_2, \ldots, e_{n-1}, u_n$ of alternating vertices and edges such that for $1 \leq i < n$, e_i is incident with u_i and u_{i+1}. If $u_1 = u_n$ the path is called *a cycle*. If no two vertices coincide, the path is called *simple*, u_1 and u_n are called *the endpoints of* p, and the remaining vertices are *the internal vertices of* p. If u_1 and u_n are the only two vertices that coincide, the cycle is called *simple*. In this work, by "path" and "cycle" we mean simple path and simple cycle, respectively. When we describe a path or a cycle, we do not mention the names of the edges. *The length* of a path or a cycle is the number

of vertices in it. If p is a path, the length is denoted by $|p|$.

A graph G is *connected* if either $|V(G)| = 1$ or $|V(G)| \geq 2$ and there is a path between any two of its vertices. If G is not connected, then the maximal connected subgraphs of G are called *the connected components of* G. In a connected graph G, a *cut vertex* is a vertex whose deletion leads to more than one connected components.

A connected graph with no cycles is called *a tree.* There is a unique path between any two vertices of a tree. A connected subgraph of a tree is called *a subtree.* Every vertex of a tree with degree one is called *a leaf.*

A tree in which one vertex, called *the root*, is distinguished, is called *a rooted tree.* In a rooted tree, any vertex of degree one is considered to be a leaf, unless it is the root. The root vertex is denoted by root(T). Let T be a rooted tree with root r. The *descendants* of r are all the other vertices in T, and the descendants of any non-leaf vertex $u \neq r$ are all the vertices w such that u is on the path between w and r. Leaf vertices have no descendants. The *children* of any vertex v are all the descendants of v that are adjacent to v. For any vertex v in T, *the subtree rooted at* v is the subgraph induced by v and the descendants of v; it is denoted by "$T[v]$". In the context of rooted trees, a subtree is always a rooted tree and if a subtree contains r then it is the whole tree T. A subtree that does not contain r is called *proper subtree.*

Definition 1 (unicyclic graph). *A* unicyclic graph *is a connected graph with precisely one cycle in it. We picture the unicyclic graph as a cycle* $u_1, u_2, \ldots, u_q$ *for some* $q \geq 3$ *plus a set of non-empty rooted trees called* constituent trees $T_1, T_2, \ldots, T_q$, *where* $u_i = root(T_i)$ *for* $1 \leq i \leq q$. *A unicyclic graph is shown on Figure 2.1.* □

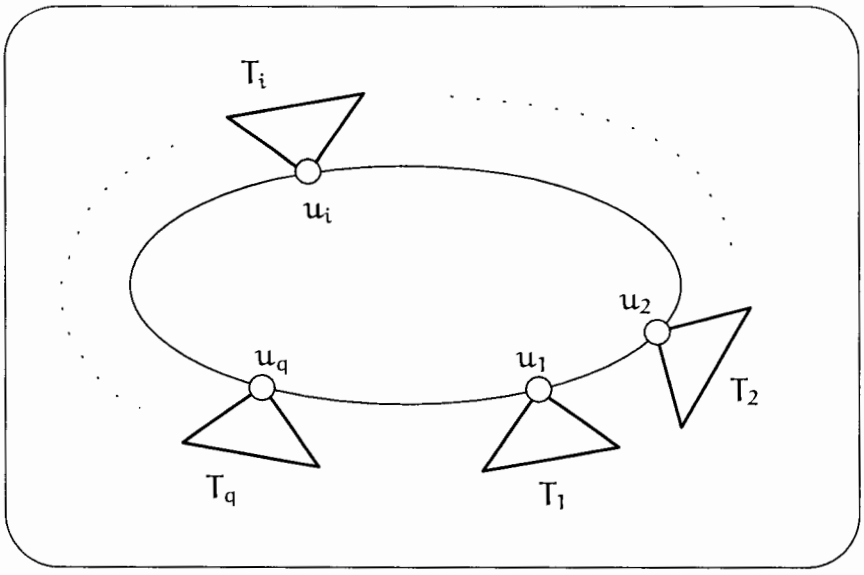


Figure 2.1: A unicyclic graph with constituent trees $T_1, \ldots, T_q$.

2.2 Vertex Separation and Node Search Number

2.2.1 Vertex Separation

A linear layout of a graph $G = \langle V, E \rangle$, or just *a layout*, is a bijective function $L : V \to \{1, 2, \ldots, |V|\}$. For any layout L and vertex u, we define *the separation of* u *under* L to be

$$\pi_L(u) = \{v \in V \mid L(v) \leq L(u), \text{ and for some } w \in V, L(w) > L(u) \text{ and } (v, w) \in E\}$$

The vertex separation of G *under* L is

$$vs_L(G) = \max_{u \in V}(|\pi_L(u)|)$$

and *the vertex separation of* G is

$$vs(G) = \min\{ vs_L(G) \mid L \text{ is a linear layout of } G\}$$

Any layout L such that $vs_L(G) = vs(G)$ is called *optimal.*

2.2.2 Searching and search number

A *node search* on a graph $G = \langle V, E \rangle$ is a *cleaning* of the edges of G, performed according to the following rules. Initially, all the edges are *contaminated.* Then they are *cleaned* by the search in discrete subsequent steps by either *placing* a *searcher* on some vertex, or *removing* a seacher from some vertex that already had a searcher on it. An edge becomes clean when both its endpoints have searchers on them. At any removal of a searcher from a vertex v, if v is on a path p that connects a clean part of G with a contaminated part and there is no other searcher on p, then the clean part becomes contaminated again. In other words, recontamination occurs immediately, at any opportunity, and can only be prevented by the presence of searchers on any path between what has been cleaned and what is contaminated. The goal of the search is to clean all edges of G with as few searchers simultaneously used at any moment as possible. The smallest number of searchers that can perform the cleaning is the *node search number* of G. Since we do not introduce any other searching rules, we say "search" and "search number", rather than "node search" and "node search number". We denote the search number of G by $sn(G)$.

It is obvious that during any search, every vertex must get a searcher at some moment. It is much less obvious that every search that can be done with k searchers, can be done with k searchers so that no vertex gets a searcher more than once. That was proved by LaPaugh in an article called "Recontamination Does Not Help to Search a Graph" [LaP93] for edge searching (edge searching is very similar to node searching). Kirousis and Papadimitriou showed [KP86, Theorem 2.3, pp. 209], using LaPaugh's result, that the same is true for node searching.

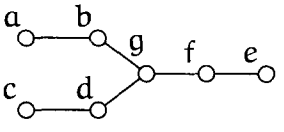
In that context, *recontamination* is defined to be the phenomenon where during a search, an edge that is clean at one moment becomes contaminated later on. Since recontamination does not help, we consider only searches during which every vertex gets a searcher exactly once.

2.2.3 A refinement of the definition of search – based on the fact that recontamination does not help

If $|V| = n$, a search S of G is a sequence of $2n$ moves, two moves for each vertex – a *positive move*, which is associated with the placing of a searcher on that vertex, and a subsequent *negative move*, which is associated with the removal of this searcher. For any vertex u, we denote the positive move with u^+, and the negative move with u^-. We denote the ordinals of u^+ and u^- within the search S by $\mathrm{ord}_S(u^+)$ and $\mathrm{ord}_S(u^-)$, respectively, or simply $\mathrm{ord}(u^+)$ and $\mathrm{ord}(u^-)$, if it is clear which search we have in mind.

If S is a search on G, we define *the reverse of* S as the sequence of $2n$ moves which is the reverse permutation of S with the positive attributes changed to negative and *vice versa*. We denote the reverse of S by $\overline{S}$. In Section 4.2 on page 30 we prove that $\overline{S}$ is a search.

An example of a graph, a search S, and the reverse $\overline{S}$ on it:



$$S = b^+\ a^+\ a^-\ g^+\ b^-\ d^+\ c^+\ c^-\ d^-\ f^+\ g^-\ e^+\ f^-\ e^-$$

$$\overline{S} = e^+\ f^+\ e^-\ g^+\ f^-\ d^+\ c^+\ c^-\ d^-\ b^+\ g^-\ a^+\ a^-\ b^-$$

Clearly, the number of searchers used by a search S at move number i is equal to the difference between the positive and the negative moves up to and incuding it.

2.3 Vertex Separation is Equivalent to Pathwidth

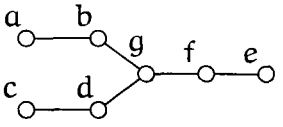
The equality between the pathwidth of a graph and its vertex separation was shown by Kinnersley [Kin92]. In fact, she showed a deeper connection. Not only are the two numbers identical, but there is a correspondence between optimal layouts and path decompositions of minimum width. If $G = \langle V, E \rangle$ is a graph with vertex separation k and $|V| = n$ and L is an optimum layout of G, then we can construct a path decomposition $X_1, X_2, \ldots, X_n$ of width at most k by the following simple rules. $X_1 = \{L^{-1}(1)\}$, and for $2 \leq i \leq n$, $X_i = \pi_L(L^{-1}(i-1)) \cup \{L^{-1}(i)\}$. The details of the proof are in [Kin92, Theorem 3.1]. The correspondence between the two problems allows us to discuss only vertex separation, not pathwidth. Although the problems are equivalent, we find it easier to present and grasp the ideas when they are formulated in terms of linear layouts.

2.4 Vertex Separation is Equivalent to Search Number Minus One

It has been proved that for any graph G, $sn(G) = vs(G) + 1$ [KP86, Theorem 4.1, pp. 216]. To establish this result the authors show that the order, from left to right, of the vertices in any layout of separation k corresponds to the order of the positive moves of some search with $k + 1$ searchers, and *vice versa.*

This allows us to make the following definitions. By *the layout* L *of* G *corresponding to a search* S we mean the layout of the vertices of G in the order of their positive moves in S. And by *a search* S *of* G *corresponding to a layout* L we mean any search of G, such that the order of the positive moves is the order of the vertices under L, and the negative moves are such that the number of used searchers at any moment, *i.e.* the difference between positive and negative

moves, never exceeds $vs_L(G) + 1$. This is achieved by removing searchers as soon as they are not needed to seal off the dirty from the clean edges. Note that in this sense a search determines the layout uniquely, while in general there are many searches corresponding to a layout, because the order of the negative moves is not completely determined by the order of the positive moves. The following picture demontrates that: both S_1 and S_2 are searches that correspond to the layout L:



$$L = b, a, g, d, c, f, e$$

$$S_1 = b^+\ a^+\ a^-\ g^+\ b^-\ d^+\ c^+\ c^-\ d^-\ f^+\ g^-\ e^+\ f^-\ e^-$$

$$S_2 = b^+\ a^+\ a^-\ g^+\ b^-\ d^+\ c^+\ d^-\ c^-\ f^+\ g^-\ e^+\ e^-\ f^-$$

Suppose that L is a layout of some graph G. *A reversal* of L is the layout that corresponds to $\overline{S}$, where S is any search corresponding to L. Continuing with the last example, consider L_1, the layout corresponding to $\overline{S_1}$ and L_2, the layout corresponding to $\overline{S_2}$. That is,

$$L_1 = e, f, g, d, c, b, a$$

$$L_2 = f, e, g, c, d, b, a$$

Both L_1 and L_2 are reversals of L.

Chapter 3

Definitions And Conventions That Are Specific To This Work

3.1 Trees and Unicyclic Graphs

For any rooted tree T with root r, *the body* of T is the possibly empty collection of trees $T - r$.

Let T be an unrooted tree, and $p = u_1, u_2, \cdots, u_n$ be a path in T. If we remove the edges (u_1, u_2), (u_2, u_3), ..., (u_{n-1}, u_n) from T, we get a collection of n trees T_1, T_2, ..., T_n. They are are called *the constituent trees of* T *relative to* p and are considered to be rooted trees, with $u_i = \text{root}(T_i)$ for $1 \leq i \leq n$.

Suppose that G is a unicyclic graph with a cycle $s = u_1, u_2, \ldots, u_q$, and $\{u_{i_1}, u_{i_2}, \ldots, u_{i_p}\} \subset \{u_1, u_2, \ldots, u_q\}$ for some p, such that $2 \leq p < q$. Suppose the constituent trees rooted at u_{i_1}, u_{i_2}, ..., u_{i_p} are deleted from U. What remains is a non-empty set of trees, called *the arches of* U *relative to* $u_{i_1}, u_{i_2}, \ldots, u_{i_p}$. For example, consider the unicyclic graph on Figure 3.1. There are three arches of U relative to u, v, and w, called D_1, D_2, D_3. The arches are outlined by dashed lines.

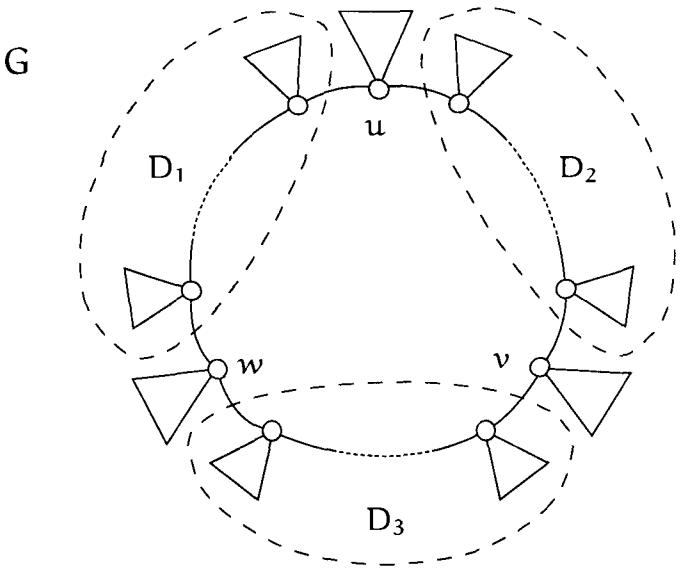


Figure 3.1: We outline the three arches D_1, D_2, and D_3 relative to u, v, and w.

3.2 Vertex Separation

Suppose that G is a graph and L is a layout for G. If for some $u \in G$, $|\pi_L(u)| = vs_L(G)$, we call u *heavy under L*, or simply *heavy* when it is clear which layout we mean.

Let $G = \langle V, E \rangle$ be a graph with a layout L. For any $u \in V$ and $v \in \pi_L(u)$, we say that v *contributes* to the separation of u.

Let L be a linear layout of G and p be a path in G. We say that *a path contributes to the separation of* u if for at least one vertex $v \in p$, $v \in \pi_L(u)$.

For any $u, v \in V$ such that $L(u) < L(v)$, we say that under L, u is *left of* v, and v is *right of* u. Collectively, "left" and "right" are the two *directions.* Let w be the rightmost vertex that is adjacent to u under L. The vertex *right*(u) is defined as follows: if $L(w) > L(u)$, then right(u) is w, and otherwise right(u) is u. Thus, right(u) is the leftmost vertex on the right side of u, to whose separation u does not contribute.

Suppose that $u, v, w, \in V$. If $L(v) < L(u) < L(w)$, we say that u *is between* v

and w. If $L(v) < L(u)$ and $L(\text{right}(u)) \leq L(w)$, we say that u *is surrounded by* v *and* w. Notice that in the latter definition, u may potentially coincide with w when $u = \text{right}(u)$.

Throughout this work, we think of a linear layout as a list of vertices, rather than as a mapping. If $G_1 = \langle V_1, E_1 \rangle$ is a proper subgraph of G, then *the linear sublayout*, or simply *the sublayout*, of G_1 under L, is the ordering of the vertices from V_1 under L. Further, for a vertex $u \in V$, we denote by "$L - u$" the list L with u deleted from it, and the other vertices left in the same relative order. For a set of vertices $U \subset V$, we denote by $L - U$ the list L with all vertices in U deleted from it, and the other vertices remaining in the same relative order.

If $L = u_1, u_2, \ldots, u_n$, then *an interval in* L is a contiguous, possibly empty subsequence $u_i, u_{i+1}, \ldots, u_{i+j}$ of L.

3.3 Extensibility of layouts and graphs

The following two definitions were proposed, in a similar formulation, by Skodinis [Sko03] under the name "extendable". Let $G = \langle V, E \rangle$ be a graph, and L be layout of G. Let k be a positive integer.

Definition 2 (left-extensible layout). *For any* $u \in V$, *we say that* L *is* left-extensible with respect to k and u, *if all vertices left of* u *have separation strictly less than* k, *and the remaining vertices have separation less than or equal to* k. *We denote that by "*L *is* (k)*-left*(u)*-ext". If* $k = vs_L(G)$, *we say that* L *is* left-extensible with respect to u, *denoted by "*L *is left*(u)*-ext".* □

Definition 3 (right-extensible layout). *For any* $u \in V$, *we say that* L *is* right-extensible with respect to k and u, *if all vertices right of and including right*(u) *are of separation strictly less than* k, *and the other vertices are of separation less*

than or equal to k. *That is denoted by "*L *is* (k)-*right*(u)-*ext". If* $k = vs_L(G)$, *we say that* L *is* right-extensible with respect to u, *denoted by "*L *is right*(u)-*ext".* □

Observation 1. *If* $L(u) = 1$, *then* L *is left-extensible with respect to* u. *If* L *is* (k)-*left*(u)-*ext and* L *is modified by placing* u *at the leftmost position and keeping the relative order of the other vertices, the modified layout is still* (k)-*left*(u)-*ext.* □

The second claim is evident, but, if in doubt, consider the vertices to the left of u in the original L, and observe that their separation may indeed increase by at most one after u is moved, but the resulting layout still conforms to Definition 2. Observation 1 allows us to assume without loss of generality that any left-extensible with respect to u and k layout L starts with u.

Theorem 2 on page 32 implies that there exists a (k)-left(u)-ext layout of G if and only if there exists a (k)-right(u)-ext layout of G. In other words, the extensibility property with respect to a vertex is reversible. Therefore, we can make the following definition.

Definition 4. *If* L *is* (k)-*left*(u)-*ext or* (k)-*right*(u)-*ext, we say that* L is k-extensible with respect to u, *denoted by "*L *is* (k)-(u)-*ext". When* $k = vs_L(G)$, *we say that* L is extensible with respect to u, *denoted by "*L *is* (u)-*ext".* □

If G has a layout L that is extensible in a certain way, we say that G is extensible in that way, too. When we say that G is not extensible in a certain way, we mean that there is no layout of G, extensible in this way.

3.4 Stretchability With Respect to a Pair of Vertices

Definition 5 (stretchable layout). *Let* u *and* v *be two vertices from* L*, not necessarily distinct. Let* $\mathcal{I}_1$ *be the possibly empty interval* $\left[L^{-1}(1), \ldots, L^{-1}(L(u)-1)\right]$ *and* $\mathcal{I}_2$ *be the non-empty interval* $\left[right(v), \ldots, L^{-1}(|V|)\right]$*. Let* $\mathcal{J}_1$ *be the possibly empty interval* $\left[L^{-1}(1), \ldots, L^{-1}(L(v)-1)\right]$ *and* $\mathcal{J}_2$ *be the non-empty interval* $\left[right(u), \ldots, L^{-1}(|V|)\right]$*. We say that* L is k-stretchable with respect to u and v *if at least one of the following holds*

- *the separation of any vertex in* L *is at most* k *minus the number of intervals from* $\mathcal{I}_1, \mathcal{I}_2$ *that it is in;*
- *the separation of any vertex in* L *is at most* k *minus the number of intervals from* $\mathcal{J}_1, \mathcal{J}_2$ *that it is in.*

In the former case, we say also that u is associated with the left direction and v with the right direction, *and in the latter case we say the opposite.* □

By Theorem 2 on page 32, there is a k-stretchable with respect to u and v layout for G where u is associated with the left direction and v, with the right direction, if and only if there is a k-stretchable with respect to u and v layout for G where v is associated with the left direction and u, with the right direction. Therefore, when we talk about a k-stretchable graph with respect to two vertices, we do not associate the vertices with directions, since there exist layouts for either case.

We denote the fact that G is k-stretchable with respect to u and v by "G is (k)-(u,v)-stretchable". Similarly to Observation 1, if L is a k-stretchable with respect to u and v layout where u is associated with the left direction, we can

modify L by placing u at the leftmost position and keeping the relative order of the other vertices – the modified layout is still (k)-(u, v)-stretchable.

Chapter 4

Earlier Results On Vertex Separation

Our algorithms are stated in terms of vertex separations and linear layouts. There are several lemmas on separation and extensibility that are used in the justification of all our algorithms, *e.g.* the reversibility of layouts and extensible layouts, that we prove now. Also, we describe the $\mathcal{O}(n)$ algorithm of Ellis, Sudborough, and Turner [EST94], which we call "the EST algorithm", for the vertex separation of trees, and its modification [EM04] that outputs in linear time an optimal layout as well.

4.1 The EST Algorithm

Their algorithm is based on the following theorem:

Theorem 1 ([EST94], Section 3.1, Theorem 3.1). *Let* T *be a tree and* $k \geq 1$. $vs(T) \leq k$ *if and only if for all vertices* x *in* T *at most two of the subtrees induced by* x *have vertex separation* k *and all other subtrees have vertex separation* $\leq k-1$.

□

The authors of the EST algorithm use the phrase "induced by a vertex" with the non-traditional meaning of "remaining after the vertex is deleted". Informally, it is "at most two", because in the linear ordering of the vertices there are two directions relative to each vertex.

The theorem has an important immediate consequence:

Corollary 1 ([EST94], Section 3.1, Corollary 3.1). $vs(T) > k$ *if and only if there exists a vertex which induces* ≥ 3 *subtrees* T' *such that* $vs(T') \geq k$. □

Let T be a rooted tree. By $root(T)$ we denote the root vertex of T. If $u \in T$, by $T[u]$ we denote the subtree of T rooted at u. Let $r = root(T)$, and $u_1, u_2, \ldots, u_k \in T$, $u_i \neq r$ for $1 \leq i \leq k$. Then "$T[r, u_1, u_2, \ldots, u_k]$" denotes the rooted tree with root r that remains after the deletion of $T[u_1]$, $T[u_2]$, $\ldots T[u_k]$ from T. That is,

$$T[r, u_1, u_2, \ldots, u_k] = ((\ldots(T - T[u_1]) - T[u_2])\ldots - T[u_k])$$

If T is an unrooted tree and $u \in T$, by $T[u]$ we denote the rooted tree, obtained from T by choosing u to be the root.

The EST algorithm and its analysis use the concept of *criticality of a tree.* Let T be a rooted tree of separation at least k with root r. A vertex $x \in T$ that may or may not be the root is k-*critical* if $vs(T) = k$ and x has two children u, v, such that $vs(T[u]) = vs(T[v]) = k$. Because of Theorem 1, it follows immediately that

- for every other child w of x, $vs(T[w]) < k$, and
- none of $T[u]$ and $T[v]$ can have a k-critical vertex.

A separation k tree T is *critical* if it has a k-critical vertex. Otherwise, T is *non-critical.* Note that criticality is a property of rooted trees. This is illustrated on Figure 4.1, where choosing a to be the root yields a critical tree, and choosing b

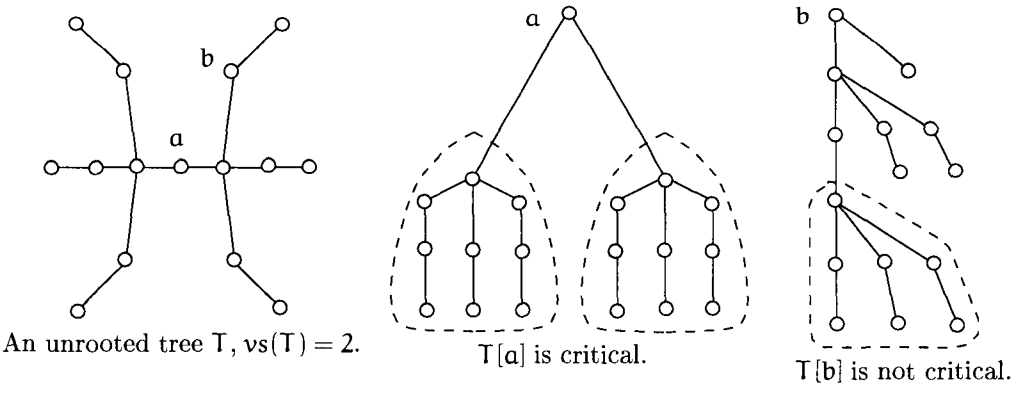


Figure 4.1: Criticality is a property of rooted trees.

to be the root yields a non-critical tree. On the Figure, the minimum subtrees of separation 2 are outlined by dashed lines.

The EST algorithm on trees uses a bottom-up approach, for each vertex u computing the separation of $T[u]$ from the separations of the subtrees rooted at children of u. The final answer is the separation number computed at the root r. In the progress of the algorithm, each vertex gets a label which describes the subtree rooted at it. Here we introduce the labels informally; a formal definition follows. Each label is a non-empty list of elements, where each element is a tuple of a non-negative integer and a flag that can have two values: "critical" and "non-critical". The largest integer in the label of vertex u is equal to the vertex separation, say k, of $T[u]$.

- If $T[u]$ is not critical, the label of u has only one entry, namely k with flag "non-critical".

- Otherwise, $T[u]$ has a k-critical vertex v. Then the first label element is k with flag "critical" and it corresponds to $T[v]$; the remainder of the label, if any, correspond to $T[u, v]$. Thus, if $v = u$, the label of u has only one element, namely k, with flag "critical". If $v \neq u$, the label of u contains more than one element, the first being k with flag "critical" and the remainder

describing the tree $T[u] - T[v]$ in the same way.

4.1.1 The Modified EST – Optimal Linear Layouts of Trees

The EST algorithm does not construct layouts. The article suggests another algorithm for the layouts [EST94, Section 3.5], but it runs in time $\Theta(n \log n)$. Linear time algorithms that compute both the separation number and an optimal layout were proposed in [Sko03] and [EM04]. Here we discuss briefly the latter one, which is an extension of the EST algorithm. An additional data structure, a list of layouts, is maintained. Each label element has a corresponding layout in the layout list, and each vertex of the tree belongs to exactly one layout from that list. A vertex u is called *singular under a layout* L when $\pi_L(u) = \{u\}$. Here is the formal definition of label from [EST94], plus the definition of layout list:

Definition 6. *For any tree* $T[u]$, *the* label λ *is a list of integers* $(a_1, a_2, \ldots, a_p)$ *for some* $p > 0$, *where* $a_1 > a_2 > \cdots > a_p$, *such that there exists a set of vertices* $\{v_1, v_2, \ldots, v_p\}$ *in* T, *such that:*

- $vs(T[u]) = a_1$,
- *for* $1 \leq i < p$, $vs(T[u, v_1, v_2, \ldots, v_i]) = a_{i+1}$,
- *for* $1 \leq i < p$, v_i *is an* a_i*-critical vertex in* $T[u, v_1, v_2, \ldots, v_{i-1}]$,
- $v_p = u$,
- *The last integer* a_p *has a flag that indicates whether* $T[u, v_1, \ldots, v_{p-1}]$ *is* a_p*-critical or not.*

We write the said flag with an apostrophe (′) . For example, if λ is $(9, 6, 5')$ then subtree associated with element 5 is not 5-critical.

Definition 7. *A layout list $\mathcal{L}$ for $T[u]$, where $\lambda = (a_1, a_2, \ldots, a_p)$ is the label, is a list of layouts $(L_1, L_2, \ldots, L_p)$, such that*

- *for $1 \leq i \leq p$, L_i is an optimal layout for $T[v_i]$,*
- *for $1 \leq i \leq p$, v_i is singular in L_i,*
- *If $T[u, v_1, v_2, \ldots, v_{p-1}]$ is not critical, then L_p is left-extensible, or right-extensible, or both, with respect to the root u.*

Furthermore, each label element a_i has a pointer to the corresponding singular vertex v_i in L_i, for $1 \leq i \leq p$. □

4.1.2 Methods for Constructing Tree Layouts

Let T be a tree with $r = \text{root}(T)$. Let r have children $u_1, u_2, \ldots, u_q$. Call T_i the subtree $T[u_i]$, for $1 \leq i \leq q$. The article [EM04] proposes five methods for the construction of a layout with certain properties L for T out of layouts of subtrees:

Method 1. *Given a layout L_i of T_i for $1 \leq i \leq q$, such that $vs_{L_i}(T_i) \leq k-1$, the output is:*

$$L = r, L_1, L_2, \ldots, L_q$$

The resulting layout L is both (k)-left(r)-ext and (k)-right(r)-ext, which follows immediately from the definitions. The root r is singular in L.

The ordering of the layouts $L_1, L_2, \ldots, L_q$, is not important. The construction is illustrated on Figure 4.2.

Method 2. *Given a (k)-left(u_1)-ext layout L_1 of T_1 and for $2 \leq i \leq q$ a layout L_i for T_i such that $vs_{L_i}(T_i) \leq k-1$, the output is a layout L for T:*

$$L = r, L_2, L_3, \ldots, L_q, L_1$$

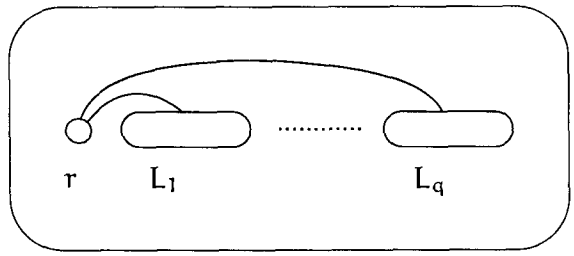


Figure 4.2: Method 1.

The resulting L *is* (k)*-left*(r)*-ext, which follows from the definitions. The root* r *is singular in* L.

The construction is illustrated on Figure 4.3.

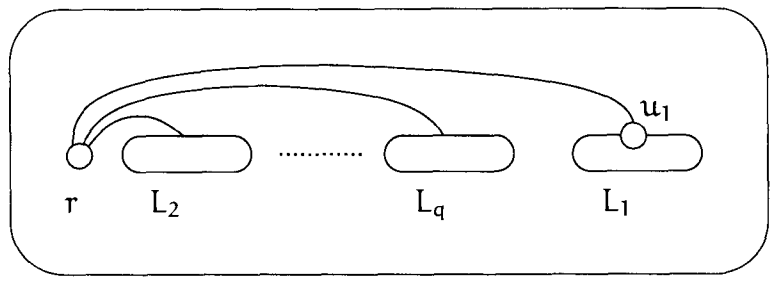


Figure 4.3: Method 2.

Method 3. *Given a* (k)*-right*(u_1)*-ext layout* L_1 *of* T_1 *and for* $2 \leq i \leq q$ *a layout* L_i *for* T_i *such that* $vs_{L_i}(T_i) \leq k-1$ *the output is a layout* L *for* T*:*

$$L = L_1, u, L_2, L_3, \ldots, L_q$$

The resulting L *is* (k)*-right*(r)*-ext, which follows from the definitions. The root* r *is singular in* L.

The construction is illustrated on Figure 4.4.

Method 4. *Given a* (k)*-right*(u_1)*-ext layout* L_1 *for* T_1*, a* (k)*-left*(u_2)*-ext layout* L_2 *for* T_2*, and for* $3 \leq i \leq q$ *a layout* L_i *for* T_i *such that* $vs_{L_i}(T_i) \leq k-1$*, the*

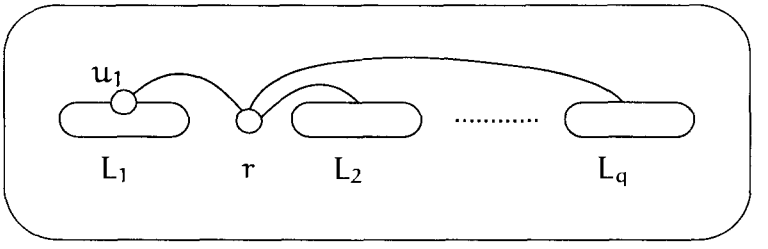


Figure 4.4: Method 3.

output is a layout L *for* T*:*

$$L = L_1, u, L_3, \ldots, L_q, L_2$$

L *has separation at most* k*. If* $vs_{L_1}(T_1) = k$ *and* $vs_{L_2}(T_2) = k$ *then* L *is neither* (k)*-left*(r)*-ext nor* (k)*-right*(r)*-ext, that is, not extensible with respect to* k *and* r*. The properties of* L *follow from the definitions. The root* r *is singular in* L*.*

The construction is illustrated on Figure 4.5.

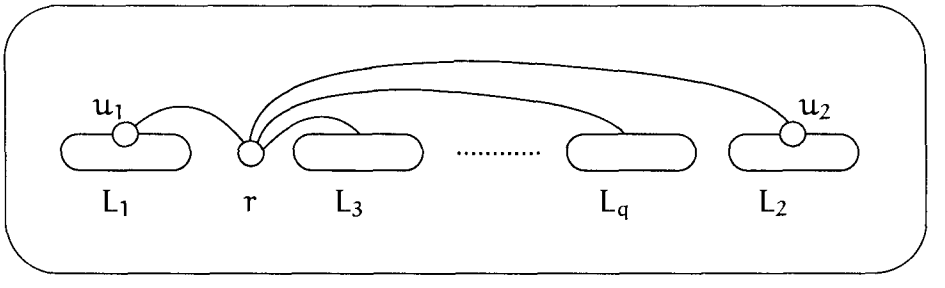


Figure 4.5: Method 4.

Method 5. *There is a vertex* $u \neq r$ *in* T*, such that* $vs(T[u]) = k$ *and we are given an optimal layout* L_u *of* $T[u]$*, such that* u *is singular in* L_u*. Further, we are given a layout* L_1 *of* $T[r, u]$ *of separation at most* $k - 1$*. The part of* L_u *that is strictly to the left of* u *is called* L'_u*, and the part that is strictly to the right of* u *is called* L''_u*,* i.e. $L = L'_u, u, L''_u$*. The output is a layout* L *for* T*:*

$$L = L'_u, u, L_1, L''_u$$

L *has separation* k. *Furthermore, if* $vs_{L'_u} = k$ *and* $vs_{L''_u} = k$ *then* L *is not extensible in any direction with respect to neither* r *nor* u. *Vertex* u *is singular in* L, *while* r *is not singular.*

The construction is illustrated on Figure 4.6.

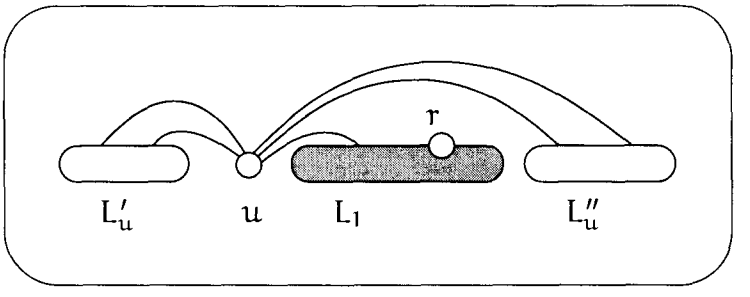


Figure 4.6: Illustrating Method 5. The root r is somewhere in L_1, not necessarily adjacent to u. In general, there are several connecting edges between u and the vertices from L''_u.

4.1.3 Computing Separation and Layout Lists Simultaneously

We discuss briefly, without justifications, how separation numbers and layouts of trees are computed by the modified EST algorithm [EM04]. The computation starts from the leaves, each leaf is marked as having separation number 0 and an optimal layout that is just this leaf.

For any non-leaf node u with children $v_1, v_2, \ldots, v_q$, we are given labels λ_i of $T[v_i]$, and layout lists L_i of $T[v_i]$, for $1 \leq i \leq q$. Recall that each label is a list of numbers, where each label element is marked as critical or non-critical.

Each label element in λ_i has a corresponding layout – an element in the layout list L_i.

Suppose the maximum element over all labels is m.

The computation is done by scanning the labels $\lambda_1, \lambda_2, \ldots, \lambda_q$ from smaller

to larger values, and proceeds according to the number of labels that contain a particular value k. The algorithm is building a label λ of $T[u]$, and a layout list L of $T[u]$. When counting how many labels contain k, λ must be considered too, unless λ is the only label containing k. In the latter case, no action is taken. Let Λ denote the set of labels $\{\lambda_1, \lambda_2, \ldots, \lambda_q, \lambda\}$. Let T^k denote the subtree of $T[u]$ that is obtained by deleting all subtrees that correspond to label elements in λ_1, λ_2, ..., λ_q, that are strictly greater than k. Thus, at the end of the k^{th} iteration, λ is the label of T^k. Suppose that L^k denotes the layout list of T^k, and L is a variable of type layout. Both the EST and its modification consider the following six cases.

Case i. There are at least three labels in Λ that contain value k. Then $vs(T^k)$ is $k+1$, and $\lambda := ((k+1)')$. L is constructed by Method 1 and L^k is set to L.

Case ii. There are precisely two labels in Λ containing value k, and at least one of these elements is marked as critical. Then $vs(T^k)$ is $k+1$, and $\lambda := ((k+1)')$. L is constructed by Method 1 and L^k is set to L.

Case iii. There are precisely two labels in Λ containing value k, and none of these elements is marked as critical. Then $vs(T^k) = k$, and $\lambda := (k)$. L is constructed by Method 4, and L^k is set to L.

Case iv. There is exactly one label in λ_1, λ_2, ..., λ_q containing k, this element is marked as critical, and λ contains k, *i.e.*, $vs(T^{k-1}) = k$. Then $vs(T^k) = k+1$, and $\lambda := ((k+1)')$. L is constructed by Method 1 and L^k is set to L.

Case v. There is exactly one label in λ_1, λ_2, ..., λ_q containing k, this element is marked as critical, and λ does not contain k, *i.e.*, $vs(T^{k-1}) < k$. Then $vs(T^k) = k$, and $\lambda := (k) \cdot \lambda$. In this case, none of Methods 1 through 5 is

invoked. L^k is set to $L^* \cdot L^{k-1}$, where L^* is the layout associated with the mentioned one label.

Case vi. There is exactly one label in $\lambda_1, \lambda_2, \ldots, \lambda_q$ containing k, this element is marked as non-critical. Then $vs(T^k)$ is k, $\lambda := (k')$. A (k)-left(u)-ext layout $\widetilde{L_1}$ and a (k)-right(u)-ext layout $\widetilde{L_2}$ are constructed by Method 2 and Method 3, respectively, and L^k is set to $\{\widetilde{L_1}, \widetilde{L_2}\}$.

After running the EST algorithm we get a label $(a_1, a_2, \ldots, a_{p-1}, a_p)$, and a layout list $(L_1, L_2, \ldots, L_{p-1}, L_p)$. We can obtain a layout L of T of separation a_1 by applying Method 5, $(p-1)$ times [EM04], scanning the label and layout list from the bottom to top. Method 5 inserts L_p right after v_{p-1} in L_{p-1}, and so on, until the obtained layout of $T[v_p, v_1]$ is inserted in L_1 right after v_1. The repeated application of Method 5 is called *nesting procedure.*

4.2 Reversibility of Layouts

Lemma 1. *Suppose that* $G = \langle V, E \rangle$ *is a connected graph and* S *is a search on it. Then* $\overline{S}$ *is a search on* G *as well. Furthermore,* $\overline{S}$ *uses the same number of searchers as* S.

Proof:

By induction on $|V|$. For $|V| = 1$ the claim holds. Assume that it holds for any graph of n vertices, $n \geq 2$. Consider any connected graph $G = \langle V, E \rangle$ such that $|V| = n + 1$. Let S be any search on G. Then S uses k searchers for some k, $2 \leq k \leq n$. Let u be any vertex from V, such that the deletion of u from G yields a connected graph G_1. Let S_1 be obtained from S by deleting both u^+ and u^-. It is obvious that S_1 is a search on G_1. By the inductive hypothesis, $\overline{S_1}$ is a search on G_1.

Assume that $\overline{S}$ is not a search on G. It is clear that, if S is a search, under both S and $\overline{S}$ every edge of G has two searchers at its endpoints at some moment. So, each edge of G is clean, at some moment, under $\overline{S}$. Then the assumption that $\overline{S}$ is not a search implies that recontamination occurs under $\overline{S}$. Consider $\mathrm{ord}_{\overline{S}}(u^-)$. If every edge incident with u is clean at that moment, then $\overline{S}$ is a search because $\overline{S_1}$ is a valid search. Then, our assumption implies that there exists an edge incident with u that is contaminated just before the move $\mathrm{ord}_{\overline{S}}(u^-)$. Call this edge (u, x), for some $x \in G_1$. Next we examine the three possible placements of $\mathrm{ord}_{\overline{S}}(x^+)$ and $\mathrm{ord}_{\overline{S}}(x^-)$, with respect to $\mathrm{ord}_{\overline{S}}(u^-)$. Remember that $\mathrm{ord}_{\overline{S}}(x^+) < \mathrm{ord}_{\overline{S}}(x^-)$, since $\mathrm{ord}_S(x^+) < \mathrm{ord}_S(x^-)$.

- $\mathrm{ord}_{\overline{S}}(x^-) < \mathrm{ord}_{\overline{S}}(u^-)$. We conclude that $\mathrm{adj}(x) \setminus \{u\} \neq \emptyset$, and that for some vertex $z \in \mathrm{adj}(x)$, $z \neq u$, the edge (x, z) is contaminated under $\overline{S}$ at the move $\mathrm{ord}_{\overline{S}}(x^-)$. Then the edge (x, z) is contaminated under $\overline{S_1}$ at the move $\mathrm{ord}_{\overline{S_1}}(x^-)$. So, there is recontamination under S_1, contrary to the inductive hypothesis.

- $\mathrm{ord}_{\overline{S}}(x^+) < \mathrm{ord}_{\overline{S}}(u^-) < \mathrm{ord}_{\overline{S}}(x^-)$. Under this relative placement, x has a searcher on it when the searcher from u is removed. This makes it is impossible (u, x) to be contaminated just before the move $\mathrm{ord}_{\overline{S}}(u^-)$.

- $\mathrm{ord}_{\overline{S}}(u^-) < \mathrm{ord}_{\overline{S}}(x^+)$. This placement of moves under $\overline{S}$ implies $\mathrm{ord}_S(u^+) > \mathrm{ord}_S(x^-)$. In other words, in S vertex x loses its searcher before u gets a searcher. So, the edge (u, x) is not cleaned by S, which contradicts the assumption that S is a search.

Then $\overline{S}$ is indeed a search of G whenever S is a search. It is easy to see that $\overline{S}$ and S use the same number of searchers. □

Corollary 2. *Suppose that* $G = \langle V, E \rangle$ *is a connected graph and* L *is a layout for it. Suppose that* $\overline{L}$ *is a reversal of* L. *Then* $vs_L(G) \leq k$ *implies that* $vs_{\overline{L}}(G) \leq k$ *and* vice versa. □

Observation 2. *Suppose that* $G = \langle V, E \rangle$ *is a connected graph. Suppose that* u *and* v *are two not necessarily distinct vertices from* V. *Suppose that* a *and* b *are two new vertices that are not from* V. *Suppose that* G_1 *is the graph* $G_1 = \langle V \cup \{a, b\}, E \cup \{(u, a), (v, b)\}\rangle$.

Then, G *has a layout* L *that is* k*-stretchable with respect to* u *and* v *where* u *is associated with the left direction and* v *is associated with the right direction, if and only if the layout* $L_1 = a, L, b$ *for* G_1 *has separation at most* k. □

Theorem 2. *Suppose that* $G = \langle V, E \rangle$ *is a connected graph. Suppose that* u *and* v *are two not necessarily distinct vertices from* V. *Then* G *has a layout that is* k*-stretchable with respect to* u *and* v *where* u *is associated with the left direction and* v *with the right direction, if and only if* G *has a layout that is* k*-stretchable with respect to* u *and* v *where* u *is asociated with the right direction and* v *is associated with the left direction.*

Proof:

Suppose that L is a layout that is k-stretchable with respect to u and v where u is associated with the left direction and v with the right direction. By Observation 1, assume that L starts with u.

Suppose that a and b are two vertices not from V. Suppose that G_1 is the graph $G_1 = \langle V \cup \{a, b\}, E \cup \{(u, a), (v, b)\}\rangle$. Suppose that $L_1 = a, L, b$. Call z the last vertex in L. By Observation 2 we know that $vs_{L_1}(G_1) \leq k$. As L starts with u, we have

$$L_1 = a, u, \ldots\ldots\ldots, z, b$$

Suppose that S_1 is a search corresponding to L_1. By the result of Kirousis and Papadimitriou [KP86, Theorem 4.1, pp. 216], we know that S_1 uses at most $k+1$ searchers. S_1 has the form

$$S_1 = a^+, u^+, \ldots\ldots\ldots, z^+, b^+, \underbrace{\ldots\ldots\ldots\ldots}_{\text{only negative moves}}$$

Assume without loss of generality that in S_1 the last three moves are b^+, z^-, b^-:

$$S_1 = a^+, u^+, a^-, \ldots\ldots\ldots, b^+, z^-, b^-$$

Now consider $\overline{S_1}$. By Lemma 1, $\overline{S_1}$ is a search using at most $k+1$ searchers:

$$\overline{S_1} = b^+, z^+, b^-, \ldots\ldots\ldots, a^+, u^-, a^-$$

Suppose that $\overline{L_1}$ is the layout for G_1 corresponding to $\overline{S_1}$. By the result in [KP86] we know that $vs_{\overline{L_1}}(G_1) \leq k$. It is clear that $\overline{L_1}$ has the form

$$\overline{L_1} = b, z, \ldots\ldots\ldots, a$$

Suppose that $\overline{L}$ is a layout for G obtained from $\overline{L_1}$ by deleting the vertices a and b. Thus,

$$\overline{L_1} = b, \overline{L}, a$$

Recall that by construction b is adjacent to and only to v, and a is adjacent to and only to u. Now apply Observation 2 and conclude that $\overline{L}$ is a k-stretchable with respect to u and v layout for G, such that v is associated with the left direction and u is associated with the right direction.

The proof in the other direction can be done completely analogously. □

Chapter 5

The Vertex Separation of Unicyclic Graphs

5.1 A Classification of Trees

The results in [EST94] suggest the following classification of the rooted trees of a given separation k into four classes. Suppose that T is a rooted tree of separation k.

- First suppose that T is non-critical.
  - If there is no child u of root(T), such that $vs(T[u]) = k$, we say that T is *type* $\mathcal{NC}$, which stands for "**n**on-**c**ritical".
  - If there is a child u of root(T), such that $vs(T[u]) = k$, we say that T is *type* $\mathcal{NCB}$, which stands for "**n**on-**c**ritical, **b**uried". In Lemma 3 we show that every tree of type $\mathcal{NCB}$ must have a unique subtree T_1 of type $\mathcal{NC}$. The subtree T_1 is the minimum subtree of T of separation k. That is the motivation for the "buried" part of the name—there is a minimum subtree of separation k, which is "buried inside" T.

- Now suppose that T is critical, and the k-critical vertex is u. Then u has two children v_1 and v_2, such that $vs(T[v_1]) = vs(T[v_2]) = k$. By Theorem 1, $T[v_1]$ and $T[v_2]$ do not have a k-critical vertex, so they are of type $\mathcal{NC}$ or $\mathcal{NCB}$.

  - If $u = \text{root}(T)$, we say that T is *type* $\mathcal{C}$, which stands for "**c**ritical".
  - If root(T) is not the k-critical vertex, we say that T is *type* $\mathcal{CB}$, which stands for "**c**ritical, **b**uried". □

5.2 Lemmas on the Vertex Separation of Trees and General Graphs

These are original results that are used in the justification of the algorithm on unicyclic graphs.

Lemma 2. *Suppose that* T *is a separation* k *rooted tree with root* r *that is of type* $\mathcal{NC}$. *Then* T *has a layout that is both* (k)-*left*(r)-*ext and* (k)-*right*(r)-*ext.*

Proof:

The desired layout is constructed by Method 1. □

Lemma 3. *Let* T *be a rooted tree with root* r, *let* $vs(T) = k \geq 1$, *and* T *be of type* $\mathcal{NCB}$. *Then there exists exactly one vertex* $w \in T$ *such that* $vs(T[w]) = k$ *and* $T[w]$ *is of type* $\mathcal{NC}$.

Proof:

Label each vertex in T with the separation of the subtree rooted at it. Let D be the subgraph of T induced by the set of vertices which are labeled by k. D is non-empty as r and exactly one child of r are in it. Note that every leaf of T is

labeled by 0. Also note that along the path from r to any leaf of T the major elements of the labels can not increase. Therefore, D must be connected. So D is an unrooted tree, and r is a leaf. Note that any vertex in T that has the subtree rooted at it of separation k and of type $\mathcal{NC}$ is a leaf of D different from r.

We claim that D is a path. If D is not a path, T would be of separation $> k$. So D is a path and r is one endvertex of it. And w is the other endvertex of D. □

Lemma 4. *Let T be a rooted tree with root r and $vs(T) = k$. If T is of type $\mathcal{NCB}$, then T has an optimal layout L' that is (k)-left(r)-ext such that r is singular in it, and an optimal layout L'' that is (k)-right(r)-ext such that r is singular in it, but no layout that is both (k)-left(r)-ext and (k)-right(r)-ext.*

Proof:

Let the children of r be $v_1, v_2, \ldots, v_q$. Without loss of generality, let v_1 be the child of r such that $vs(T[v_1]) = k$. As T is of type $\mathcal{NCB}$, by Lemma 3 there exists a unique vertex w such that $vs(T[w]) = k$ and $T[w]$ is of type $\mathcal{NC}$. Note that w is in $T[v_1]$ and may possibly coincide with v_1. Throughout this proof we call "p" the path between r and w.

- We prove that L' exists by induction on $|p|$.

  Basis: $|p| = 1$. Then w coincides with v_1 and $T[v_1]$ is of type $\mathcal{NC}$. By Lemma 2, the desired layout L' exists.

  Inductive Hypothesis: Assume that for some number t, for all cases where $|p| \leq t$, L' exists.

  Inductive Step: Let $|p| = t + 1$. Then in $T[v_1]$ the path from v_1 to w is of length t, so the inductive hypothesis holds and there is an optimal (k)-left(r)-ext layout of $T[v_1]$. The desired L' is constructed by Method 2.

- We prove that L'' exists by induction on $|p|$.

  Basis: $|p| = 1$. Then w coincides with v_1 and $T[v_1]$ is of type $\mathcal{NC}$. By Lemma 2, the desired layout L'' exists.

  Inductive hypothesis: Assume that for some number t, for all cases where $|p| \leq t$, L'' exists.

  Inductive Step: Let $|p| = t + 1$. Then in $T[v_1]$ the path from v_1 to w is of length t, so the inductive hypothesis holds and there is an optimal (k)-right(r)-ext layout of $T[v_1]$. L'' is constructed by Method 3.

- Assume there exists an optimal layout $\overline{L}$ of T that is both (k)-left(r)-ext and (k)-right(r)-ext. Let $\overline{L}_1$ be the sublayout of $T[v_1]$ under $\overline{L}$. There exists a vertex a in $T[v_1]$ that has separation k under $\overline{L}_1$. Note that r does not contribute to $\pi_{\overline{L}_1}(a)$. If a is left of r under $\overline{L}$, then $\overline{L}$ is not left-extensible. Then a must be right of r. Vertex a can not be between r and right(r), because then r would contribute to the separation of a, making it $\geq k$. And a can not be right of or coinciding with right(r), because then $\overline{L}$ is not right-extensible. The assumption has to be false. □

Lemma 5. *Let* $G = \langle V, E \rangle$ *be a connected graph,* $vs(G) = k$, *and* $k > 1$. *Let* V_1, V_2, V_3 *be a partition of* V, *and* G_1, G_2, G_3 *be the subgraphs of* G *induced by* V_1, V_2, V_3, *respectively, such the* G_1 *and* G_2 *are connected, and* $vs(G_1) = vs(G_2) = k$. *Let any vertex from* V_3 *be connected to any vertex from* V_1 *by a path that is vertex-disjoint with* V_2, *and to any vertex from* V_2 *by a path that is vertex-disjoint with* V_1. *Let* L *be any optimal layout of* G. *Then* L *has exactly one of the following forms,* L' *or* L'':

$$L' = L'_1,\ \alpha'_1,\ L'_*,\ \alpha'_2,\ L'_2 \tag{5.1}$$

or

$$L'' = L''_2,\ \alpha''_2,\ L''_*,\ \alpha''_1,\ L''_1 \tag{5.2}$$

where

- L'_1 *and* L''_1 *denote intervals that consist of vertices from* G_1 *only,*
- L'_2 *and* L''_2 *denote intervals that consist of vertices from* G_2 *only,*
- α'_1 *is the rightmost heavy vertex from* G_1 *in case 5.1,* α'_2 *is the leftmost heavy vertex from* G_2 *in case 5.1,* α''_2 *is the rightmost heavy vertex from* G_2 *in case 5.2, and* α''_1 *is the rightmost heavy vertex from* G_1 *in case 5.2,*
- *(applicable to case 5.1 only)* L'_* *denotes the interval of* L *that is between* α'_1 *and* α'_2,
- *(applicable to case 5.2 only)* L''_* *denotes the interval of* L *that is between* α''_2 *and* α''_1.

Proof:

Let L_1 be the sublayout of G_1 under L and L_2 be the sublayout of G_2 under L. Let $v_1 \in V_1$ be the leftmost vertex in L_1, and $v_2 \in V_2$ be the leftmost vertex in L_2. Without loss of generality, let v_1 be left of v_2 under L. By the premises of this lemma, there is a separation k vertex from V_1 under L_1 and there is a separation k vertex from V_2 under L_2. We call the rightmost such vertex from V_1, α'_1, the leftmost such vertex from V_2, α'_2, and prove that L has the form L' from case 5.1.

First we prove that α'_1 is left of α'_2. Assume the opposite. There is a path p from v_1 to α'_1, and this path is vertex-disjoint with G_2. Under the current assumptions v_1 is left of α'_2. Then the separation of α'_2 must be $> k$, because $|\pi_{L_2}(\alpha'_2)| = k$ and, under L, some vertex from p contributes to the separation of α'_2 additionally.

Now we show that left of α_1' there are only vertices from V_1. Assume the opposite. Then there is a vertex $w \in (V_2 \cup V_3)$ left of α_1'. By the premises of this lemma, w is connected to α_2' by a path p that is vertex-disjoint with V_1. Then the separation of α_1' must be $> k$, because $|\pi_{L_1}(\alpha_1')| = k$ and, under L, some vertex from p contributes to the separation of α_1' additionally.

Right of α_2' there are vertices from V_2 only. Assume the opposite. Then there is a vertex $z \in (V_1 \cup V_3)$ right of α_2'. By the premises of this lemma, z is connected to α_1' by a path p that is vertex-disjoint with V_2. Then the separation of α_2' must be $> k$, because $|\pi_{L_2}(\alpha_2')| = k$ and, under L, some vertex from p contributes to the separation of α_2' additionally.

Then all the vertices from V_3 are between α_l and α_r, and L is of the form L' from case 5.1. □

Lemma 6. *If* T *is a rooted tree of separation* k *and root* r*, and* T *is critical, then in any optimal layout of* T *there is a heavy vertex left of* r *and a heavy vertex right of, or coinciding with,* $right(r)$.

Proof:

Let L be an optimal layout for T. Call u the critical vertex in T and call T_1 and T_2 the two separation k subtrees rooted at children of u. Lemma 5 implies that L starts with vertices from T_1 and finishes with vertices from T_2, or *vice versa.* Assume L starts with vertices from T_1. Lemma 5 further implies that there is a heavy vertex α_1' from T_1 such that left of it there are only vertices from T_1, and there is a heavy vertex α_2' from T_2 such that left of it there are only vertices from T_2. The Lemma also implies that r is between α_1' and α_2'.

Now observe that since α_2' has separation k under the sublayout of T_2, r cannot contribute to its separation under L. It follows that α_2' cannot be between r and $right(r)$. The desired result follows. □

Corollary 3. *Suppose that* T *is a rooted tree with root* r *and* $vs(T) = k$. *If* T *is critical then* T *is not extensible with respect to* r. □

Having in mind the above corollary on one hand and Lemmas 2 and 4 on the other hand, we see that a rooted tree is extensible if and only if it is not critical. Thus we get a complete structural characterisation of extensibility on trees.

Lemma 7. *Let* G *be a connected graph of vertex separation* $k > 1$. *Let* G_1, G_2, G_3 *be connected, pairwise vertex-disjoint subgraphs of* G, *each one of them of vertex separation at least* k, *such that between any two* G_i, G_j *there is a path that is vertex-disjoint with the third one* G_k. *Then the vertex separation of* G *is at least* $k+1$.

Proof:

Let L be an optimal layout of G, and L_i be the sublayout of G_i under L, for $i = 1, 2, 3$. In each L_i there is a vertex u_i such that $|\pi_{L_i}(u_i)| \leq k$. Without loss of generality, let u_2 be between u_1 and u_3 under L. By the premises, there is a path between u_1 and u_3 that is vertex-disjoint with the vertices of G_2. Therefore, $|\pi_L(u_2)| > k$. □

Lemma 8. *Let* T *be an unrooted tree,* $vs(T) = k$. *Let* u_1 *and* u_n *be two distinct vertices in* T, *and the path between them be* $p = u_1, u_2, \ldots, u_n$. *Let* T_1, T_2, $\ldots$, T_n *be the constituent trees relative to this path. Let* $T[u_1]$ *be* k*-critical and* v_1 *be the* k*-critical vertex in it, and* $T[u_n]$ *be* k*-critical and* v_n *be the* k*-critical vertex in it. Then either precisely one* $T_i[u_i]$ *is* k*-critical, in which case* $v_1 = v_n$, *this vertex is in* T_i, *and the other consituent trees have separation less than* k, *or precisely two distinct* T_i, T_j *have vertex separation* k *and are of type* $\mathcal{NCB}$, *in which case* v_1 *is the root of one of* T_i, T_j *and* v_n *is the root of the other one.*

Proof:

Case i, $v_1 = v_n$

For some i, $1 \leq i \leq n$, $v_1 = v_n$ is in T_i. Clearly, T_i is k-critical within both $T[u_1]$ and $T[u_n]$, and so $T_i[u_i]$ is k-critical. The fact that the other constituent trees must be of separation less than k follows from Corollary 1 and the separation of T being k.

Case ii, $v_1 \neq v_n$

We prove that v_1 and v_n are vertices from p. First we show that if v_1 and v_n are distinct they are in distinct T_i, T_j. Assume that $v_1 \neq v_n$ and both are in the same T_i. Notice that the parent-child order within T_i is the same within both $T[u_1]$ and $T[u_n]$.

If v_1 and v_n have a common ancestor in T_i that is distinct from any of them, then there are four distinct subtrees in T_i that have separation k. If one of v_1, v_n is an ancestor of the other one, there are 3 distinct subtrees in T_i of separation k. In both assumptions, by Corollary 1, $vs(T) > k$, which contradicts the premises.

So, v_1 and v_n are in distinct trees T_i and T_j, respectively. Again, note that the parent-child relations within T_i are the same for both $T[u_1]$ and $T[u_n]$. The same holds for T_j. In $T[u_1]$ there are two distinct subtrees of separation k, call them T_1' and T_2'', rooted at children of v_1. At least one of them, say T_1', is a subtree of T_i. In $T[u_n]$, there are two distinct subtrees of separation k, call them T_n' and T_n'', rooted at children of v_n. At least one of them, say T_n', is a subtree of T_j.

We show that $v_1 = root(T_i)$. Assume that $v_1 \neq root(T_i)$. Then both T_1' and T_1'' are in T_i. Then, with respect to the parent of $root(T_1')$ and $root(T_1'')$, there are three induced subtrees of separation k, and by Corollary 1 this implies $vs(T) > k$. In an analogous way we prove that $v_n = root(T_j)$. Furthermore, both T_i and T_j are of type $\mathcal{NCB}$, as they contain T_1' and T_n', respectively. □

The following definition is applicable only under the premises of Lemma 8.

Definition 8 (heavy subtrees relative to a path). *In* **Case i** *of the proof of Lemma 8, the critical vertex in* $T_i[u_i]$*, namely* $v_1 = v_n$*, has two children, such that the subtrees rooted at them are of separation* k*. These two subtrees are* the heavy subtrees relative to p*. In* **Case ii** *of the proof, both* u_i *and* u_j *have a unique child within* $T_i[u_i]$ *and* $T_j[u_j]$*, respectively, such that the subtree rooted at that child is of separation* k*. These two subtrees are the heavy subtrees relative to* p *in* **Case ii**.
□

Note that the heavy subtrees relative to a path are vertex-disjoint with it.

Lemma 9. *Let* T *be an unrooted tree, and* u_1, u_n *be distinct vertices in it. Let the path between them be* $p = u_1, u_2, \ldots, u_n$*. Let the constituent trees of* T *relative to this path be* $T_1, T_2, \ldots, T_n$*,* $u_i = root(T_i)$*. Let, for each* T_i*, either* $vs(T_i) \leq k - 1$ *and* T_i *be of any type, or* $vs(T_i) = k$ *and* T_i *be of type* $\mathcal{NC}$*. Then there exists a layout* L *of* T *of separation at most* k*, such that* $L(u_1) < L(u_n)$*, and* L *is* (k)-(u_1, u_n)-*stretchable.*

Proof:

We construct the desired layout. For each constituent tree T_i we construct an optimal layout L_i. Furthemore, if $vs(T_i) = k$ then we construct L_i using Method 1 so that it is both (k)-left(u_i)-ext and (k)-right(u_i)-ext. Consider the following layout L for G:

$$L = L_1, L_2, \ldots, L_n$$

Consider any vertex z in any L_i. If $|\pi_{L_i}(z)| < k$ then $|\pi_L(z)| \leq k$. If $|\pi_{L_i}(z)| = k$ then note that L_i is both (k)-left(u_i)-ext and (k)-right(u_i)-ext, therefore in this case $u_i \in \pi_{L_i}(z)$ and so $|\pi_{L_i}(z)| = |\pi_L(z)|$. It follows that $vs_L(T) \leq k$. To see that L is (k)-(u_1, u_n)-stretchable, consider the properties of L_1 and L_n. □

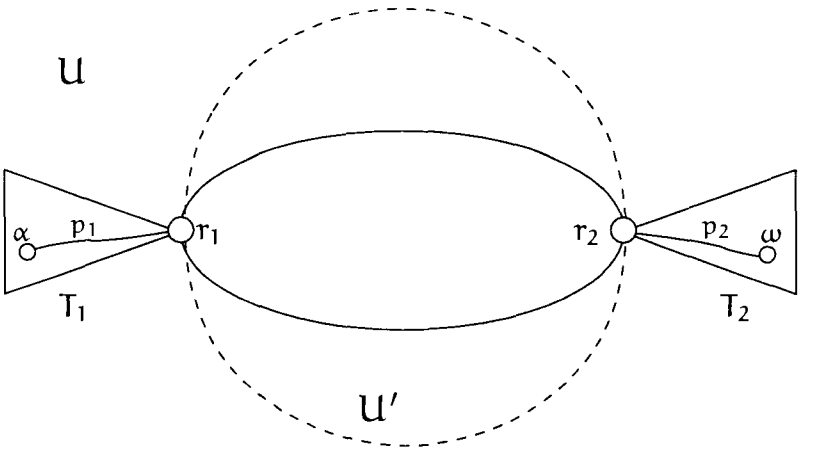


Figure 5.1: A diagram U in Lemma 10.

5.3 Lemmas on the Vertex Separation of Unicyclic Graphs

These are original results that are specific for unicyclic graphs.

Lemma 10. *Suppose that* U *is a unicyclic graph and* L *is a layout for it of separation* k. *Suppose that the leftmost vertex under* L *is* α *and the rightmost one is* ω. *Suppose that* α *and* ω *belong to different constituent trees, called* T_1 *and* T_2, *respectively. Suppose that* $r_1 = root(T_1)$ *and* $r_2 = root(T_2)$. *Suppose that* U' *denotes* U *minus the bodies of* T_1 *and* T_2. *Suppose that* L' *is the sublayout of* U' *under* L. *Then,* L' *is* (k)-(r_1, r_2)-*stretchable.*

Proof:

First observe that $vs_{L'}(U') \leq k$. Then call the path from α to r_1, p_1, and call the path from r_2 to ω, p_2. Because of p_1, the separation of all vertices from U' that are left of r_1 under L' is smaller by at least one relative to their separations under L. Likewise, because of p_2, the separation of all vertices right of or coinciding with right(r_2) under L' is smaller by at least one relative their separations under L. The situation is illustrated on Figure 5.1.

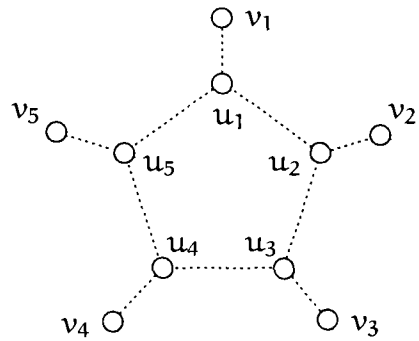


Figure 5.2: A schema of the relative placement of $u_1, \ldots, u_5$ in U in Lemma 11.

The desired result follows by immediate application of the definition of stretchable layout. □

Lemma 11. *Suppose that* U *is a unicyclic graph whose cycle has at least five vertices. Suppose that for five cycle vertices* $u_1, u_2, \ldots, u_5$, *the respective constituent trees* $T[u_1], T[u_2], \ldots, T[u_5]$ *are of separation at least* k *and of type* $\mathcal{NCB}$, $\mathcal{C}$, *or* $\mathcal{CB}$. *Then* $vs(U) \geq k+2$.

Proof:

Let L be any layout for U. For $1 \leq i \leq 5$, $T[u_i]$ has a proper subtree D_i of separation at least k. This follows from the definitions of tree types $\mathcal{NCB}$, $\mathcal{C}$, and $\mathcal{CB}$. Let L_i be the sublayout of D_i under L, for $1 \leq i \leq 5$. Let v_i be a vertex from D_i such that $|\pi_{L_i}(v_i)| \geq k$. Without loss of generality, assume that the relative placement of $u_1, \ldots, u_5$ in the cycle is as shown in Figure 5.2.

Consider all the possible relative placements of $v_1, \ldots, v_5$, in L. There are 120 possibilities, but it suffices to consider the following cases:

$$v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \tag{5.3}$$

$$v_1 \quad v_4 \quad v_3 \quad v_2 \quad v_5 \tag{5.4}$$

$$v_1 \quad v_5 \quad v_3 \quad v_2 \quad v_4 \tag{5.5}$$

These cases are sufficient, because one vertex has to be in the middle and the remaining set of 4 vertices can be partitioned into 2 sets of 2 vertices each in 3 different ways. Without loss of generality, let v_3 be in the middle. Our proof relies on two vertex-disjoint paths, each one having one endpoint left of v_3 and the other endpoint right of v_3. Therefore, the relative order of the two vertices on the left of v_3, for instance v_1 and v_2 in 5.3, is not important for the purposes of our proof. Likewise for the relative order of the two vertices right of v_3. It also does not matter which group of two vertices is to the left of v_3 and which to the right. For instance, in 5.3, we could have chosen v_4 and v_5 to be left of v_3 and v_1 and v_2 to be right of v_3.

Consider case 5.3. There is a path p_1 from v_1 to v_5 and a path p_2 from v_2 to v_4, such that p_1 and p_2 are vertex-disjoint, and both are disjoint with D_3, therefore no vertex from any of them contributes ot the separation of v_3 under L_3. Having in mind that $|\pi_{L_3}(v_3)| \geq k$, we conclude that $|\pi_L(v_3)| \geq k+2$.

In cases 5.4 and 5.5, the desired result follows likewise. In both cases, there is a path p_1 that is between v_1 and v_2 and a path p_2 between v_4 and v_5, such that p_1 and p_2 are vertex-disjoint, and both are disjoint with D_3. We conclude that in both cases, $|\pi_L(v_3)| \geq k+2$. □

Lemma 12. *Let* U *be the unicyclic graph containing four constituent trees of separation* $(k-1)$*, each of type* $\mathcal{NCB}$*,* $\mathcal{C}$*, or* $\mathcal{CB}$*. Let* L *be a separation* k *layout for* U*. Then the leftmost and rightmost vertices in* L *are in two distinct separation* $(k-1)$ *trees.*

Proof

We call the four mentioned trees the *heavy* trees. We establish that if the leftmost and the rightmost vertices, say α and ω, of L are not both in distinct heavy trees, then $vs_L(U) \geq k+1$.

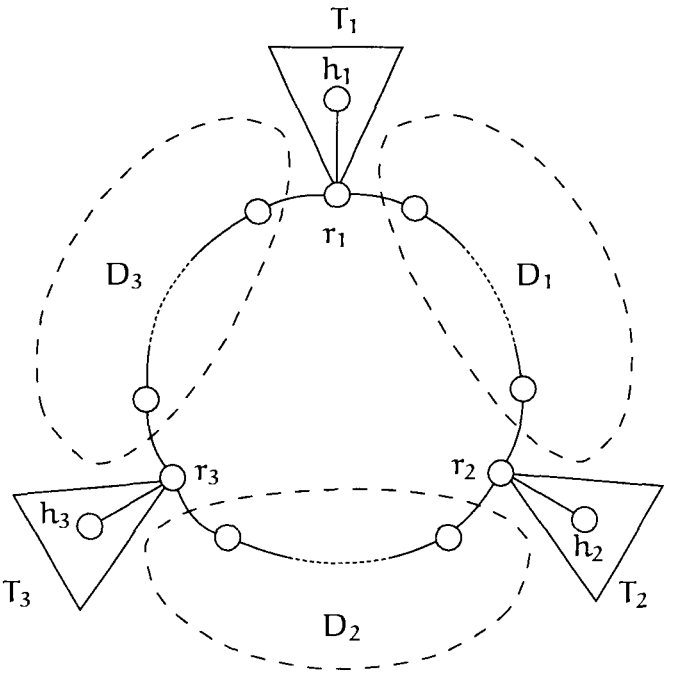


Figure 5.3: A diagram of U in Lemma 12. The assumption is that none of T_1, T_2, T_3 contains any of α or ω.

Suppose only one or neither of α and ω is in a heavy tree. Then there are at least three heavy trees, without loss of generality say T_1, T_2, and T_3, containing neither α nor ω. Let L_i be the sub-layout for T_i under L, for $1 \leq i \leq 3$. Let the arches of the cycle relative to the roots of T_1, T_2, and T_3 be D_1, D_2, and D_3. See Figure 5.3 that shows the structure of U.

Let h_i be a vertex in T_i distinct from the root, such that $|\pi_{L_i}(h_i)| \geq k-1$, for $1 \leq i \leq 3$. Without loss of generality, let h_2 lie between h_1 and h_3 in L. By case analysis we show that two paths p_1, p_2, mutually disjoint and each disjoint with T_2, exist which pass over h_2 in L, and that therefore $vs_L(U) \geq k+1$.

Taking symmetry into account, we need to consider only three cases:

- α and ω are in the same D_i. No matter which D_i is that, there is a path between α and ω confined to that D_i, and there is another path between h_1 and h_3, disjoint with the first one.

- α is in D_1 and ω is in D_2. Then the two paths are one between α and ω through D_1, r_2, and D_2, and the other one between h_1 and h_3 through D_3.
- α is in D_1 and ω is in D_3. Then the two paths are one between α and h_3 through D_1, r_2, and D_2, and the other one between h_1 and ω through D_3.

□

Lemma 13. *Let U be the unicyclic graph containing three constituent trees of separation $(k-1)$, each of type $\mathcal{NCB}$, $\mathcal{C}$, or $\mathcal{CB}$. Let L be a separation k layout for U. Let α be the leftmost and ω be the rightmost vertex in L. Then at least one of α and ω is in one of the mentioned separation $(k-1)$ trees.*

Proof

Assume the contrary and apply exactly the same reasoning as in the proof of Lemma 12 to derive a contradiction. □

In Lemma 14, Lemma 15, and Lemma 16, we use the following assumptions. U is a unicyclic graph and two of the constituent trees T_i, T_j are single vertices r_i and r_j from the cycle. T' and T'' are the arches of U relative to r_i, r_j. The cycle vertices a and b are the vertices of attachment of r_i to T' and T'', respectively. The cycle vertices c and d are the vertices of attachment of r_j to T' and T'', respectively. Figure 5.4 shows the general naming convention we will use.

Assuming that all of a, b, c, and d as shown on Figure 5.4 exist and are distinct, consider the four pairs of vertices from $\{a, b, c, d\}$, such that one vertex is from T' and the other one is from T''. The two pairs a, b and c, d are called *the same side pairs*, and the two pairs a, d and b, c are called *the opposite pairs.*

Definition 9 (complementary extensibilities in unicyclic graphs). *Under the notation from Figure 5.4, when $vs(T') = vs(T'')$, we say that T' and T'' have*

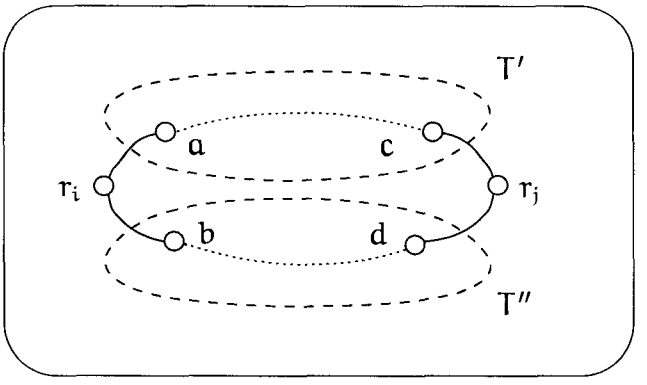


Figure 5.4: A diagram of U in Lemmas 14, 15, and 16.

complementary extensibilities *if both* T' *and* T'' *are extensible when rooted at the vertices from any of the opposite pairs.* □

Lemma 14. *Under the mentioned assumptions, suppose that* $vs(T') = vs(T'') = k-1$. *Suppose that both* T' *and* T'' *are not extensible when rooted each at a vertex from a same side pair. Then there is no* (k)-(r_i, r_j)-*stretchable layout for* U.

Proof:

Recall that not extensible is equivalent to critical for rooted trees. Without loss of generality, assume that $T'[a]$ and $T''[b]$ are critical and there is a layout L for U that is (k)-(r_i, r_j)-stretchable, such that r_i is associated with the left direction and r_j with the right direction. Assume that L starts with r_i and let v be a vertex not in U.

Construct the unicyclic graph U_1 by adding v to U, connecting v to and only to r_j. Let L_1 be a layout for U_1 such that $L_1 = L, v$. Clearly, $vs_{L_1}(U_1) \leq k$. Let L' be the sublayout of T' under L and L_1, and L'' be the sublayout of T'' under L and L_1. Since $T'[a]$ is critical, by Lemma 6 there is vertex $a_1 \in T'$, such that $L'(a_1) < L'(a)$ and $|\pi_{L'}(a_1)| \geq k-1$. Likewise, there is a vertex $b_1 \in T''$, such that $L''(b_1) < L''(b)$ and $|\pi_{L''}(b_1)| \geq k-1$. There are six possible placements of

a_l, a, b_l, b under L:

$$b_l \quad b \quad a_l \quad a \tag{5.6}$$

$$b_l \quad a_l \quad b \quad a \tag{5.7}$$

$$b_l \quad a_l \quad a \quad b \tag{5.8}$$

$$a_l \quad b_l \quad b \quad a \tag{5.9}$$

$$a_l \quad b_l \quad a \quad b \tag{5.10}$$

$$a_l \quad a \quad b_l \quad b \tag{5.11}$$

Recall that under L_1 the leftmost vertex is r_i and the rightmost vertex is v. In case 5.6, because of the edge (r_i, a), r_i is in $\pi_L(a_l)$. However, r_i is not in $\pi_{L'}(a_l)$. Additionally, a vertex from the path $b, \ldots, d, r_j, v$ that is not in $\pi_{L'}(a_l)$, is in $\pi_L(a_l)$. Therefore, $|\pi_L(a_l)| \geq k+1$. Analogously, in case 5.11, the edge (r_i, a) and the path $a, \ldots, c, r_j, v$ cause $|\pi_L(b_l)| \geq k+1$.

In cases 5.7 and 5.8, the path $b_l, \ldots, b$ and the edge (r_i, a) cause $|\pi_L(a_l)| \geq k+1$. In cases 5.9 and 5.10 path $a_l, \ldots, a$ and the edge (r_i, a) cause $|\pi_L(b_l)| \geq k+1$. □

Lemma 15. *Suppose that* U *has a* (k)-(r_i, r_j)*-stretchable layout* L, $vs(T') = vs(T'') = k-1$, *at least one of* $T'[a]$ *and* $T'[c]$ *is extensible, and at least one of* $T''[b]$ *and* $T''[d]$ *is extensible. Then* T' *and* T'' *have complementary extensibilities.*

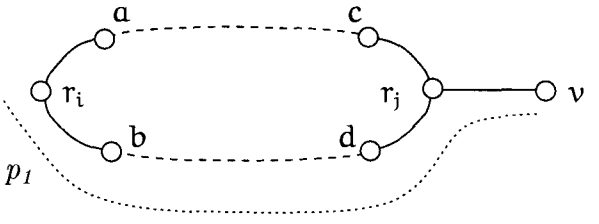
Proof:

The contrapositive claim of Lemma 14 is that if U has a (k)-(r_i, r_j)-stretchable layout then both T' and T'' cannot be not extensible when rooted each at a vertex from a same side pair. The remaining possibility is that T' and T'' are extensible when rooted each at a vertex from an opposite pair. □

Lemma 16. *Suppose that* $vs(T') = k - 1$ *and both* $T'[a]$ *and* $T'[c]$ *are critical. Suppose that* T_1 *and* T_2 *are the heavy subtrees relative to the path from* a *to* c. *Suppose that* L *is a* (k)-(r_i, r_j)-*stretchable layout for* U. *Then there are vertices* z_1 *and* z_2 *in* T', *each laying in a distinct tree from* T_1 *and* T_2, *such that, under* L, *all the vertices of* $(T' - T_1) - T_2$, *all the vertices of* T'', *and the vertex from* r_i, r_j *that is associated with the right direction, are between* z_1 *and* z_2.

Proof:

Suppose that in L, r_i is associated with the left direction and r_j is associated with the right direction. Suppose that L starts with r_i and let v be a vertex not in U. Construct the unicyclic graph U_1 by adding v to U, connecting v to and only to r_j. Let L_1 be a layout for U_1 such that $L_1 = L, v$. Clearly, $vs_{L_1}(U_1) \leq k$. Consider the path $p_1 = r_i, b, \ldots, d, r_j, v$ in U_1:



Call L' the sublayout of T' under L and L_1. Because of p_1, $vs_{L'}(T') < k$. That is, L' is optimal.

Apply Lemma 5 on page 38 to T', with T_1 as G_1 from the Lemma, T_2 as G_2 from the lemma, and $(T' - T_1) - T_2$ as G_3 from the lemma, and conclude that L' has either the form 5.1 or the form 5.2, as defined in Lemma 5. Define z_1 and z_2 to be the following vertices:

- if L' is of form 5.1, then z_1 is α'_1 and z_2 is α'_2;
- if L' is of form 5.2, then z_1 is α''_2 and z_2 is α''_1.

In any event, $L(z_1) < L(z_2)$, all the vertices of $(T' - T_1) - T_2$, which includes a and c, are between z_1 and z_2, and $|\pi_{L'}(z_1)| = |\pi_{L'}(z_2)| = k - 1$.

We prove that if any vertex from T'' is not between z_1 and z_2, then $vs_{L_1}(U_1) > k$. So far we have established the following placement under L_1:

$$L_1 = r_i, \ldots, z_1, \ldots, a, \ldots, c, \ldots, z_2, \ldots, v$$

where a may be right or left of c. We start with two observations:

- r_i contributes to the separation of z_1 under L_1. But r_i does not contribute to the separation of z_1 under L' as r_i is not in T'. Therefore, $|\pi_{L_1}(z_1)| \geq k$ because of the contributions of vertices from T_1 and r_i.
- Vertex c is connected to v by the path c, r_j, v and no vertex from $\{c, r_j, v\}$ is in $\pi_{L'}(z_2)$. It follows that $|\pi_{L_1}(z_2)| \geq k$, because of the contributions of vertices from T' and $\{c, r_j\}$.

Assume that there is a vertex $x \in T''$ that is left of z_1 or right of z_2, under L. First assume that x is left of z_1:

$$L = r_i, \ldots, x, \ldots, z_1, \ldots, a, \ldots, c, \ldots, z_2, \ldots, v$$

That x is connected to v by the path $x, \ldots, r_j, v$. This path is vertex-disjoint with both T' and r_i, and we already established that $|\pi_{L_1}(z_1)| \geq k$ because of the contributions of vertices from T' and r_i. It follows that $|\pi_{L_1}(z_1)| > k$.

Now assume that x is right of z_2:

$$L = r_i, \ldots, z_1, \ldots, a, \ldots, c, \ldots, z_2, \ldots, x, \ldots, v$$

Note that r_i is connected to x by a path that is vertex-disjoint with both T' and

$\{c, r_j\}$. As we already established, vertices from the latter two cause $|\pi_{L_1}(z_2)| \geq k$. Then $|\pi_{L_1}(z_2)| > k$.

So, all vertices of T'' are between z_1 and z_2. We prove that if r_j is not between z_1 and z_2 then $vs_{L_1}(U_1) > k$. Assume that r_j is between z_2 and v. We demontstrated that $|\pi_{L_1}(z_2)| \geq k$ due to the contribution of vertices from T' and $\{c, r_j\}$. The latter result was proven without considering the relative placement of z_2 and r_j, so it holds when r_j is between z_2 and v. But vertex d, which has to be between z_1 and z_2, is connected to r_j, thus additionally contributing at least one to the separation of z_2. Thus $|\pi_{L_1}(z_2)| > k$.

If r_j is between r_i and z_1, a similar argument yields that $|\pi_{L_1}(z_1)| > k$. □

Corollary 4. *Let* $vs(T') = k - 1$, *and both* $T'[a]$ *and* $T'[c]$ *be critical. Suppose that* U *is* (k)-(r_i, r_j)-*stretchable. Then in* T'', *relative to the path from* b *to* d, *all constituent trees are either of separation* $< k - 1$, *or of separation* $k - 1$ *and type* $\mathcal{NC}$.

Proof:

A diagram of U is shown on Figure 5.5.

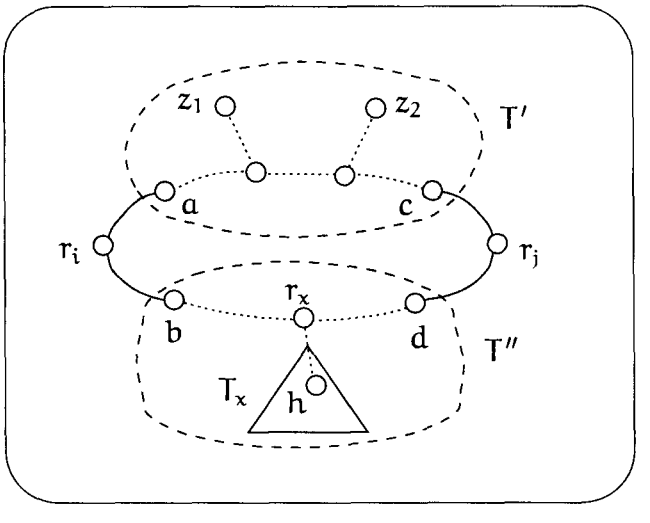


Figure 5.5: A schema of U for the proof of Corollary 4.

Suppose that L, U_1 and L_1 have the meaning from the proof of Lemma 16. Clearly, there can be no constituent tree of separation bigger than $k-1$ in T''. Assume that there is a constituent tree in T'' whose root, call it r_x, has a child such that the subtree rooted at it, call it T_x, is of separation $k-1$. Let L_x be the sublayout of T_x under L_1. Under L_x there is a vertex $h \in T_x$, such that $|\pi_{L_x}(h)| = k-1$. By Lemma 16, there are $z_1, z_2 \in T'$ such that all the vertices from T'' are between z_1 and z_2. Then there are two vertex-disjoint paths in U, namely $p_1 = r_i, b, \ldots, d, r_j, v$ and $p_2 = z_1, \ldots, z_2$, that have one endpoint left of h and the other endpoint right of h, and these paths are vertex-disjoint with $\pi_{L_x}(h)$. Then $|\pi_{L_1}(h)| > k$. □

Corollary 5. *Under the premises of Lemma 16, suppose that the two heavy subtrees relative to the path from* a *to* c *are in the same constituent tree* T_m *that is of type* $\mathcal{CB}$. *Suppose* r_m *is the root of* T_m, f *is the critical vertex in* $T_m[r_m]$, *and* U^* *is* $U - T[f]$. *Then the sublayout* L^* *of* U^* *under* L *is* $(k-1)$-(r_i, r_j)-*stretchable.*

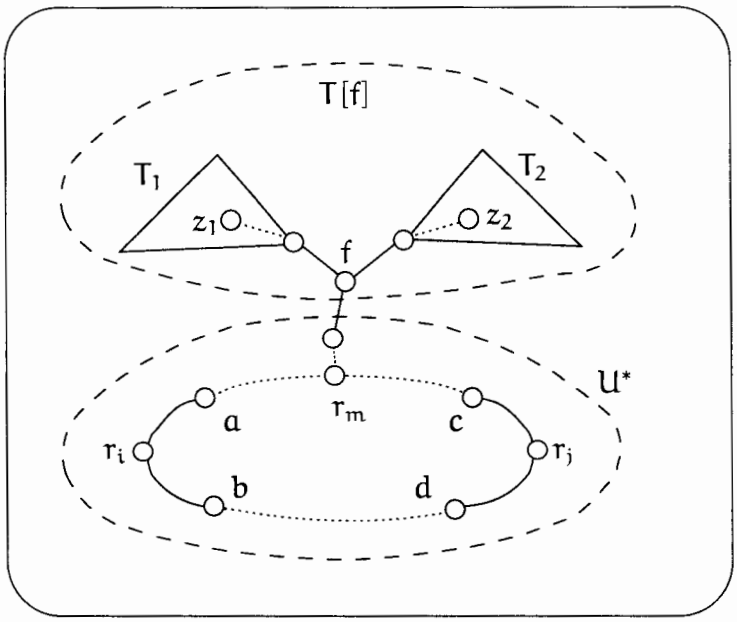


Figure 5.6: A schema of U in Corollary 5.

Proof:

A diagram of U is shown on Figure 5.6. Since $f \in T_m$ and $f \neq r_m$, deleting $T[f]$ from U leaves the cycle intact, *i.e.,* U^* is a unicyclic graph.

Suppose that U_1 and L_1 have the meaning from the proof of Lemma 16. Call U_1^* the unicyclic graph obtained from U^* when v is added. Call L_1^* the sublayout of U_1^* under L_1. Notice that if $vs_{L_1^*}(U_1^*) \leq k-1$, then L^* is $(k-1)$-(r_i, r_j)-stretchable. Our goal is to show that $vs_{L_1^*}(U_1^*) \leq k-1$.

The vertices of U_1^* are v, r_i, r_j, the vertices from T'', and the vertices from $T' - T[f]$. But $|\pi_{L_1^*}(v)| = 0$ and $|\pi_{L_1^*}(r_i)| = 1$, so consider the other mentioned vertices. By Lemma 16, we know that under L, and thus under L_1, all the vertices from $(T' - T_1) - T_2$ and thus all the vertices from $T' - T[f]$, all the vertices from T'', and r_j, are between some $z_1 \in T_1$ and $z_2 \in T_2$. Because of the path from z_1 to z_2 which is disjoint with U_1^*, the separation of any of them under L_1^* is at most $k-1$. □

5.4 The Unicyclic Stretchability Problem

In this section we assume that U is a unicyclic graph such that two distinct cycle vertices r_i and r_j are of degree two. We give necessary and sufficient conditions for U to be (k)-(r_i, r_j)-stretchable. We follow the naming convention of the previous section: we call T' and T'' the arches of the cycle relative to r_i and r_j, and the vertex names a, b, c, and d are assigned as shown in Figure 5.7, which is identical to Figure 5.4.

Of course, one (but not both) of T' and T'' may be empty, but we consider the general case when both are non-empty, $a \neq c$, and $b \neq d$. The reader will see that our results can easily accomodate the special cases of one of T', T'' being empty, *etc.*

We assume without loss of generality that

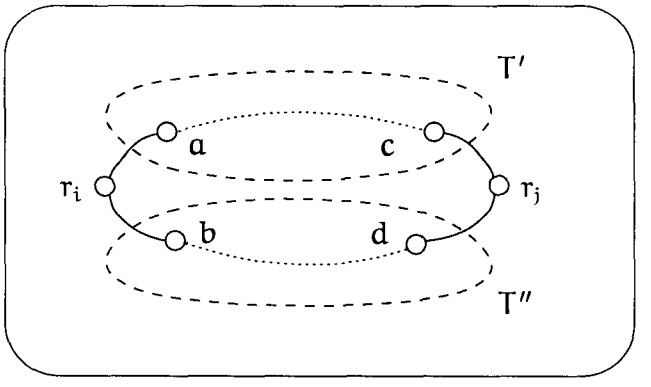


Figure 5.7: A schema of U in Section 5.4.

- $vs(T') \geq vs(T'')$.
- In the case that $vs(T') = vs(T'')$, if one of T', T'' is critical with respect to one or both of a or c, or b or d, whichever is applicable, and the other one is not critical with respect to any of a or c, or b or d, whichever is applicable, it is T' that is the critical tree.
- In the case that $vs(T') = vs(T'')$, if one of T', T'' is critical with respect to both of a or c, or b or d, whichever is applicable, and the other one is not critical with respect to both a or c, or b or d, whichever is applicable, it is T' that is the critical tree with respect to both vertices.

On the topmost level we consider three cases: $vs(T') = k$, $vs(T') < k - 1$, and $vs(T') = k - 1$.

Case 1: $vs(T') = k$.

Lemma 17. *In this case, no* (k)-(r_i, r_j)-*stretchable layout for* U *exists.*

Proof:

Assume there exists a layout L for U that is (k)-(r_i, r_j)-stretchable. Without loss of generality, assume that L starts with r_i and is (k)-right(r_j)-ext. Call L' the sublayout of T' under L. Then there is a vertex $w \in T'$, such that $|\pi_{L'}(w)| = k$.

Now assume that v is a vertex not from U and U_1 is obtained from U by adding v and connecting v to and only to r_j. It is easy to see that $vs_{L_1}(U_1) \leq k$ by the definition of stretchable layout. Call p the path in U_1 from r_i to v that avoids T'. There has to be a vertex from p that is not in $\pi_{L'}(w)$ but is in $\pi_{L_1}(w)$. Thus, $|\pi_{L_1}(w)| > k$, contrary to the earlier conclusion that $vs_{L_1}(U_1) \leq k$. □

Case 2: $vs(T') < k-1$.

Lemma 18. *In this case, a* (k)-(r_i, r_j)-*stretchable layout for* U *exists.*

Proof:

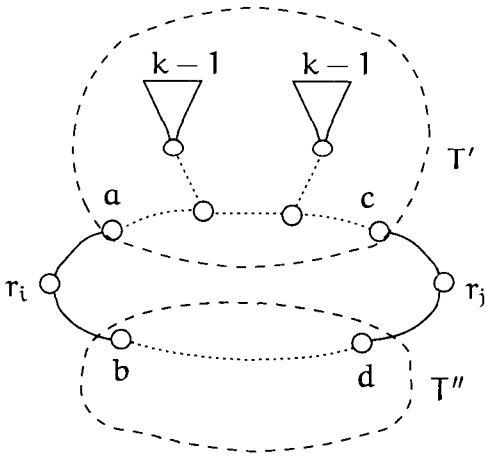
Given that L' is a layout of separation at most $k-2$ for T' and L'' is a layout of separation at most $k-2$ for T'', the construction suggested by Figure 5.11 on page 60 yields a (k)-(r_i, r_j)-stretchable layout for U. □

Case 3: $vs(T') = k-1$. We consider two subcases: either both $T'[a]$ and $T'[c]$ are critical, or at least one of them is not critical.

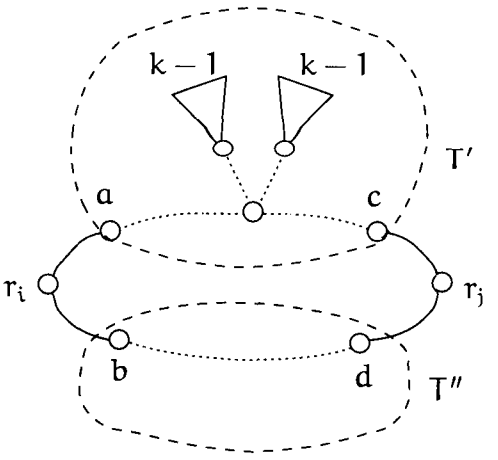
Case 3.1: Both $T'[a]$ and $T'[c]$ are critical. By Lemma 8, applied to T' with the path $a, \ldots, c$, and our knowledge of types of critical trees, we can have three situations:

- The critical vertices in $T'[a]$ and $T'[c]$ are different. Then there are precisely two, separation $k-1$, constituent trees of type $\mathcal{NCB}$ within T'.
- The critical vertex in $T'[a]$ and $T'[c]$ is the same and in T' there is a single, separation $k-1$, constituent tree, which is of type $\mathcal{C}$.
- The critical vertex in $T'[a]$ and $T'[c]$ is the same and in T' there is a single, separation $k-1$, constituent tree, which is of type $\mathcal{CB}$.

In the first two possibilities, the critical vertex in $T'[a]$ and in $T'[c]$ is a root of a constituent tree, *i.e.* a vertex from the path $a, \ldots, c$. In the third possibility, it



5.8.1: The critical vertices in $T'[a]$ and $T'[c]$ are different.



5.8.2: The critical vertices in $T'[a]$ and $T'[c]$ are the same.

Figure 5.8: Illustration of Case 3.1.1.

is not from the path $a, \ldots, c$. Therefore, we treat the first two possibilities in the same way, and so we classify into Case 3.1.1 and Case 3.1.2 as follows.

Case 3.1.1: The critical vertex in $T'[a]$ and in $T'[c]$ is in the path $a, \ldots, c$. Figure 5.8 shows the two possibilities, covered by this case.

Lemma 19. *In this case, there exists a* (k)-(r_i, r_j)-*stretchable layout for* U *if and only if either* $vs(T'') \leq k-2$ *or* $vs(T'') = k-1$ *and* T'' *does not contain separation* $k-1$ *constituent trees of type* $\mathcal{NCB}$, $\mathcal{C}$, *or* $\mathcal{CB}$.

Proof:

First assume that U is (k)-(r_i, r_j)-stretchable. By Corollary 4, all constituent trees in T'' are either of separation $< k-1$, or of separation $k-1$ and type $\mathcal{NC}$.

In the other direction, assume that $vs(T'') = k-1$ and T'' does not contain separation $k-1$ constituent trees of type $\mathcal{NCB}$, $\mathcal{C}$, or $\mathcal{CB}$. We demonstrate a (k)-(r_i, r_j)-stretchable layout for U. We consider only the possibility shown on Figure 5.8.1; the reader will see the solution for the possibility from Figure 5.8.2.

Let us consider the unicyclic graph from Figure 5.8.1 in more detail – see Figure 5.9.

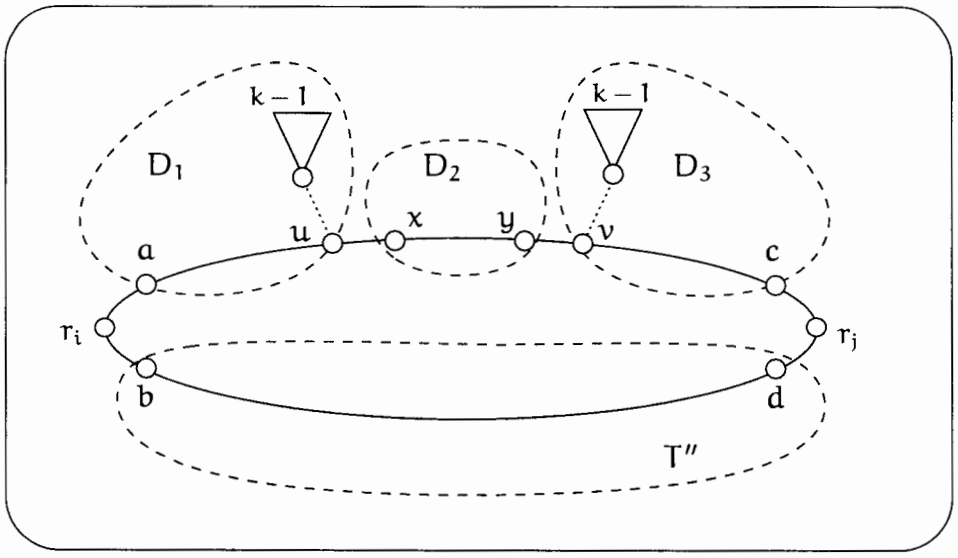


Figure 5.9: A detailed view of the unicyclic graph from Figure 5.8.1.

By Lemma 9, there is a $(k-1)$-(b,d)-stretchable layout L'' for T''. D_2 cannot contain constituent trees of separation $k-1$ and type different from $\mathcal{NC}$, so by Lemma 9 it has a $(k-1)$-(x,y)-stretchable layout L_2. Both $D_1[u]$ and $D_3[v]$ are separation $k-1$, type $\mathcal{NCB}$ trees, so by Lemma 4 there is a $(k-1)$-right(u)-ext layout L_1 for D_1 and a $(k-1)$-left(v)-ext layout L_3 for D_3. Then the layout for U from Figure 5.10 has separation at most k. □

Case 3.1.2: The critical vertex in $T'[a]$ and in $T'[c]$ is not in the path $a,\ldots,c$, *i.e.* there is a separation $k-1$, type $\mathcal{CB}$ constituent tree in T'. Assume the naming convention of Corollary 5 and Figure 5.6 on page 53.

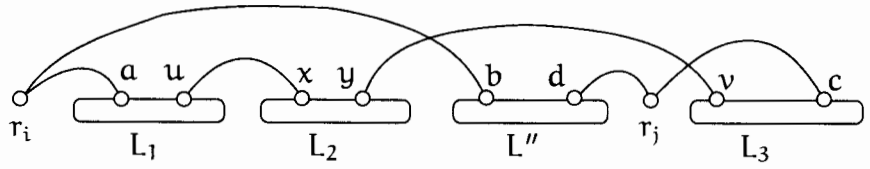


Figure 5.10: The optimal layout from Lemma 19.

Lemma 20. *In this case, there exists a* (k)-(r_i, r_j)-*stretchable layout for* U *if and only if* U^* *is* $(k-1)$-(r_i, r_j)-*stretchable.*

Proof:

If U is (k)-(r_i, r_j)-stretchable, then by Corollary 5, U^* is $(k-1)$-(r_i, r_j)-stretchable.

In the other direction, suppose that L^* is a $(k-1)$-(r_i, r_j)-stretchable layout for U^* where r_i is associated with the left direction and r_j is associated with the right direction. Without loss of generality, suppose that L^* starts with r_i; say that $L^* = r_i, L^1$. Suppose that L^f is an optimal layout for $T[f]$, such that vertex f is singular in it. Say that $L^f = L^f_1, f, L^f_2$. The following layout L for U is (k)-(r_i, r_j)-stretchable:

$$L = r_i, L^f_1, f, L^1, L^f_2$$

□

Case 3.2: At most one of $T'[a]$ and $T'[c]$ is critical.

- If none of them is critical and $vs(T'') = k-1$, by the premises, none of $T''[b]$, $T''[d]$ can be critical, so T' and T'' have layouts that are complementary and the construction shown on Figure 5.11 yields the desired layout for U.
- If none of them is critical and $vs(T'') < k-1$, the said construction yields the desired layout for U again.

So, assume that one of $T'[a]$ and $T'[c]$ is critical.

Lemma 21. *In this case, there exists a* (k)-(r_i, r_j)-*stretchable layout for* U *if and only if either* $vs(T'') \leq k-2$ *or* $vs(T'') = k-1$ *and* T' *and* T'' *have complementary extensibilities.*

Proof:

First assume that $vs(T'') \leq k-2$ or $vs(T'') = k-1$ and T' and T'' have complementary extensibilities. Further, let us assume that it is $T'[c]$ that is not critical

and, in case that $vs(T'') = k-1$, that $T''[b]$ is not critical. Then there is a $(k-1)$--right(c)-ext layout L' for T' and a $(k-1)$-left(b)-ext layout L'' for T''. Then the layout shown on Figure 5.11 is a (k)-(r_i, r_j)-stretchable layout for U.

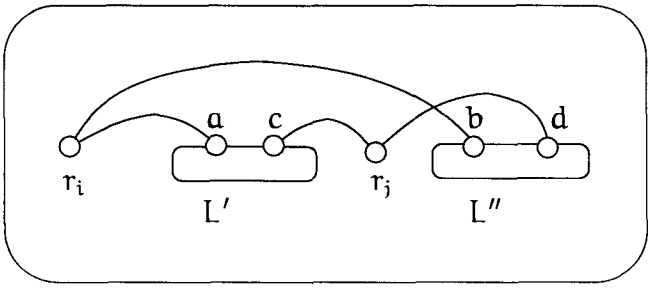


Figure 5.11: A (k)-(r_i, r_j)-stretchable layout for U in Case 2 and Case 3.2 of Lemma 18.

In the other direction, suppose that L is a (k)-(r_i, r_j)-stretchable layout for U. If $vs(T'') = k-1$, by Lemma 15, T' and T'' have complementary extensibilities. □

5.5 Computing Vertex Separation and Optimal Layout of Unicyclic Graphs

The unicyclic graph we consider is U, with a cycle $c = r_1, r_2, \ldots, r_p$ and constituent tree T_1, T_2, ..., T_p, where $r_s = root(T_s)$, for $1 \leq s \leq p$. The computation comprises two stages. As a preprocessing, we compute a label and a layout list for each constituent tree, and the type of each tree. It is straightforward to modify the algorithm in [EM04], so that it determines the type of the tree as well. Another modification of the tree algorithm is necessary: we need the layout lists of the constituent trees, not necessarily their optimal layouts, so the nesting procedure is not invoked at the end. Then we remove an edge e of the cycle and obtain a tree T_U. We choose e as follows: if there are two or more constituent trees of

maximum separation, choose e arbitrarily among the cycle edges. Otherwise, *i.e.* when there is a unique constituent tree T_i of maximum separation, e is chosen arbitrarily among the two cycle edges that are incident to root(T_i).

We apply the modified EST algorithm to T_U and obtain its separation number k and a layout list. Since adding one edge to any graph can not increase the vertex separation by more than one, $vs(U)$ is either k or $k+1$. We propose an algorithm that takes as input the unicyclic graph with the labeled cycle vertices, and k, and outputs $vs(U)$, plus an optimal layout of U. Note that deleting different edges e_1, e_2 of the cycle may lead to trees of different separations k_1 and k_2, where $|k_1 - k_2| = 1$, so the input k is not uniquely determined by U solely, but by the choice of e as well.

U is classified according to k and the labels of the constituent trees:

Case i There is exactly one constituent tree of separation k and it is critical.

Case ii There are three or more constituent trees of separation k and none of them critical.

Case iii There are exactly two constituent trees of separation k and none of them is critical.

Case iv There is exactly one constituent tree of separation k and it is not critical.

Case v There are no constituent trees of separation k.

There no other possibilities. Note that, by Lemma 7, if one of the constituent trees were k-critical, and another constituent tree, critical or not, were of separation k, $vs(T_U)$ would be at least $k+1$. In the remainder of the section we analyse each case.

Also, note that we can construct a separation k layout L_U for T_U, and this L_U is a layout for U as well, because U and T_U have the same vertex sets. So, whenever

we discover that $vs(U) = k + 1$, *i.e.* $vs(U) = vs(T_U) + 1$, it is the case that $vs_{L_U}(U) = vs_{L_U}(T_U) + 1$, and so L_U is an optimal layout for U as well. Therefore, we explain how to construct optimal layout for U only when $vs(U) = k$.

5.5.1 Case i

Let the constituent critical tree of separation k be T_i with root r_i, and the k-critical vertex in it be u. Recall that in this case, the deleted cycle edge in the preprocessing is incident with r_i.

Suppose that $r_i = u$, *i.e.*, T_i is type $\mathcal{C}$. Then $vs(U) = k$, which we prove by demonstrating a layout of separation k. Let T' be the tree $U - T_i$. It has to be the case that $vs(T') < k$: if $vs(T') = k$, then in T_U, r_i is a vertex that induces three subtrees of separation k, which implies $vs(T_U) \geq k+1$. Figure 5.12 illustrates the structure of U. T^1 and T^2 are the subtrees of separation k. Let L_i be the optimal

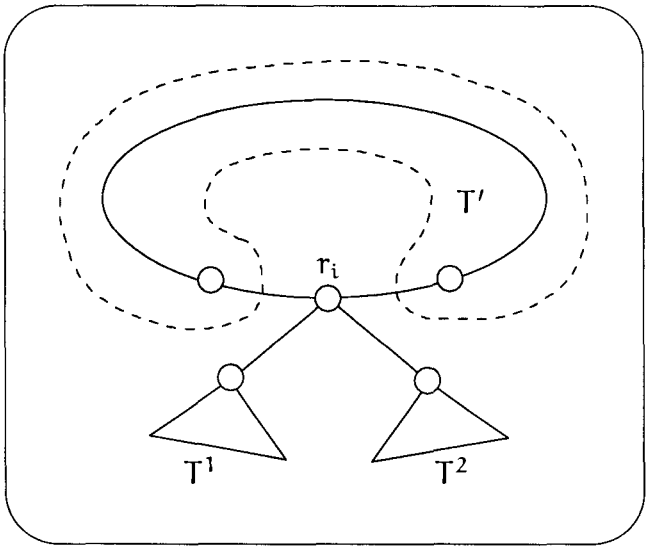


Figure 5.12: Case i, when $r_i = u$.

layout of T_i, as computed by the preprocessing. As T_i is type $\mathcal{C}$, its layout list has only one list entry, namely L_i, and r_i is singular in L_i. Let L_i^{left} be the part of L_i that is strictly left of r_i, and L_i^{right} be the part of L_i that is strictly right of

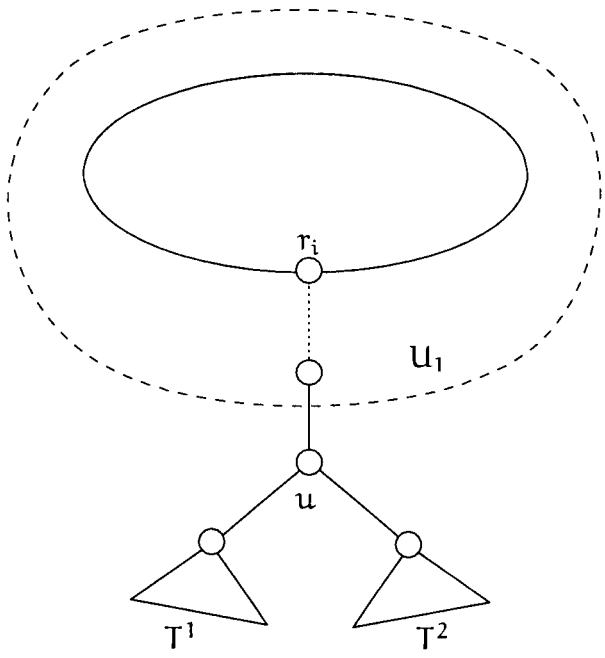


Figure 5.13: Case i, when $r_i \neq u$.

r_i. Compute an optimal layout L' of T', using the labels and layout lists of the constituent trees in T'. Then the following layout of U is of separation k:

$$L = L_i^{\text{left}}, r_i, L', L_i^{\text{right}}$$

Suppose that $r_i \neq u$, *i.e.*, T_i is type $\mathcal{CB}$. Let $U_1 = U - T_i[u]$. Note that U_1 is a unicyclic graph. Figure 5.13 illustrates the structure of U, T^1 and T^2 are the subtrees of separation k.

- If $vs(U_1) \leq k-1$, then $vs(U) = k$. We show that by constructing a layout of U of separation k. Let L_1 be an optimal layout of U_1. Let L_u be the entry in the layout list of T_i that corresponds to $T_i[u]$. Let L_u^{left} be the part of L_u that is strictly left of u, and L_u^{right} be the part of L_u that is strictly right of u. Then the following layout of U is of separation k:

$$L = L_u^{\text{left}}, u, L_1, L_u^{\text{right}}$$

- If $vs(U_1) \geq k$, then $vs(U) = k + 1$ by Lemma 7. □

5.5.2 Case ii

By Lemma 7, $vs(U) = k + 1$. □

For the remaining three cases, we use the following approach. Either we deduce immediately that $vs(U) = k + 1$ or $vs(U) = k$, or identify a set $\mathcal{T}$ of constituent trees. Then we do the following iteration over a number of pairs of trees from $\mathcal{T}$: substitute both trees by single vertices r_i and r_j, run a procedure that returns "yes" if the modified U has a (k)-(r_i, r_j)-stretchable layout, and "no" otherwise; in case of a positive answer, U has separation k, we output an optimal layout and stop, otherwise we restore U and continue with the iteration. If the iteration finishes with a negative answer from the procedure, we conclude that $vs(U) = k + 1$.

5.5.3 Case iii

Let the two constituent trees of separation k be T_i and T_j, with roots r_i, r_j, respectively. We substitute the tree T_i by a single vertex, the root r_i, and we substitute T_j by a single vertex, the root r_j. The resulting graph is unicyclic, which we call U^*.

Lemma 22. *In this case,* U *has a layout* L *of separation* k *if and only if* U^* *has a layout* L^* *that is* (k)-(r_i, r_j)-*stretchable.*

Proof:

Suppose that L^* exists. Since none of T_i, T_j is critical, there is a (k)-right(r_i)--ext layout L_i of T_i and a (k)-left(r_j)-ext layout L_j of T_j. Let $\overline{L^*} = L^* - r_i$, and

$\overline{L_j} = L_j - r_j$. The following layout L of U is of separation k:

$$L = L_i, \overline{L^*}, \overline{L_j}$$

In the other direction, suppose that U has a layout L of separation k. By Lemma 5, applied to U with T_i as G_1, T_j as G_2, and $(U - T_i) - T_j$ as G_3, it follows that the leftmost vertex α in L is from one of T_i, T_j, and the rightmost vertex ω in L is from the other one of T_i, T_j. Without loss of generality, assume that $\alpha \in T_i$ and $\omega \in T_j$. Consider L^*, the sublayout of U^* under L. By Lemma 10, L^* is (k)-(r_i, r_j)-stretchable. □

So, in Case iii we test whether U^* is (k)-(r_i, r_j)-stretchable. If yes, we output that $vs(U) = k$, and the proof of Lemma 22 suggests how to construct an optimal layout for U. If no, $vs(U) = k + 1$.

5.5.4 Case iv

Let the constituent tree of separation k be T_i with root r_i. Let the constituent trees of U of separation $k-1$ and type $\mathcal{NCB}$, $\mathcal{C}$, or $\mathcal{CB}$ be $\{T^1, T^2, \ldots, T^q\}$, with $r^s = \text{root}(T^s)$, for $1 \leq s \leq q$. Let L_i be a (k)-right(r_i)-ext layout for T_i, let L^s be a (k)-left(r^s)-ext layout for T^s, and let U^s denote U minus the bodies of T_i and T^s, for $1 \leq s \leq q$.

- If $q = 0$, $vs(U) = k$. To see why, apply Lemma 9 to the tree $U - T_i$ to conclude that its separation is $\leq k-1$. Suppose that $\widetilde{L}$ is a layout for $U - T_i$ of separation $\leq k-1$. The layout $L = L_i, \widetilde{L}$ for U has separation at most k.

- If $q = 1$, $vs(U) = k$. To see why, call T' and T'' the two arches of the cycle relative to r_i and r^1, and suppose the naming schema of Figure 5.14 holds.

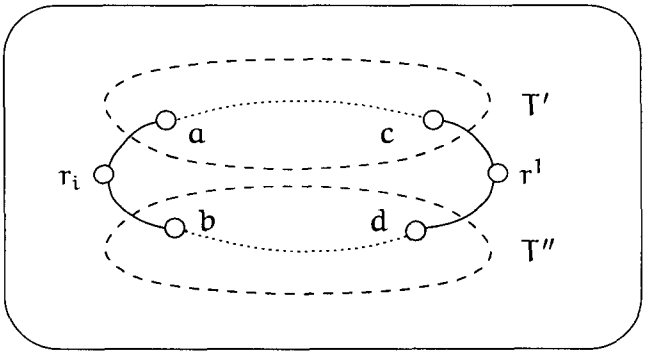


Figure 5.14: The unicyclic U^1.

Assume that T' and T'' are non-empty; if that is not the case, it is easy to see how to modify the following argument. By Lemma 9, T' has a layout L' that is $(k-1)$-(a,c)-stretchable and T'' has a layout L'' that is $(k-1)$--(b,d)-stretchable. Then the layout shown on Figure 5.15 is a (k)-(r_i, r^1)--stretchable layout $\widetilde{L^1}$ for U^1.

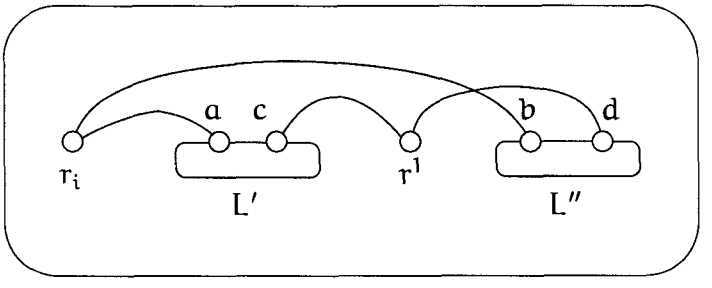


Figure 5.15: The layout $\widetilde{L^1}$.

Having L_i, $\widetilde{L^1}$, and L^1, we construct a separation k layout for U in a way completely analogous to Case iii.

- Suppose that $q = 2$.

Lemma 23. *In this case,* $vs(U) = k$ *if and only if there exists a* (k)-(r_i, r^1)--*stretchable layout for* U^1 *or a* (k)-(r_i, r^2)-*stretchable layout for* U^2.

Proof:

Suppose that the arches relative to r_i, r^1 and r^2 are called D_1, D_2, and D_3, and that the names are assigned as suggested by Figure 5.16.

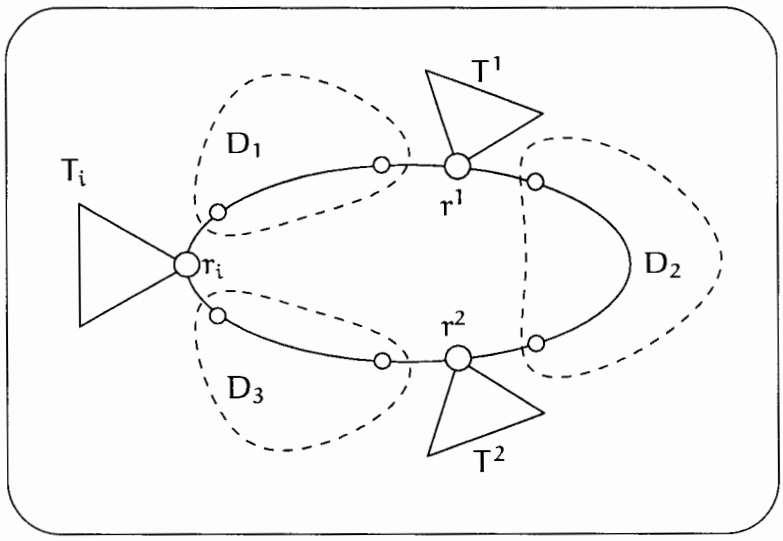


Figure 5.16: The unicyclic graph from Lemma 23.

If there exists a (k)-(r_i, r^1)-stretchable layout for U^1 or a (k)-(r_i, r^2)-stretchable layout for U^2, then a construction completely analogous to Case iii will do to obtain a separation k layout L for U. It remains to show the implication in the other direction.

Suppose that L is a separation k layout for U. We first show that either the leftmost vertex of L, say α, or the rightmost vertex of L, say ω, is in T_i. Assume the contrary – namely, that $\alpha \notin T_i$ and $\omega \notin T_i$. Call L_i the sublayout of G_i under L. There is a heavy vertex h in L_i by the premises. Then, under L, this h must have separation more than k because of the path from α to ω, which contradicts our previous assumptions.

Assume that $\alpha \in T_i$. The argument till the end of the proof is easily modified to accomodate the other possibility, namely $\omega \in T_i$.

If $\omega \in T^1$ or $\omega \in T^2$, the desired result follows from Lemma 10. So, assume

that ω is in some D_i. Call r_j the cycle vertex such that the constituent tree rooted at r_j contains ω. Call T' and T'' the two arches of the cycle relative to r_i and r_j. Call $\widetilde{U}$ the unicyclic graph obtained from U be deleting the bodies of T_i and T_j. By Lemma 10, we know that the sublayout $\widetilde{L}$ of $\widetilde{U}$ under L is (k)-(r_i, r_j)-stretchable.

Next we show that it can not be the case that $vs(D_i) = k - 1$ for all i. Assume that all D_i have separation $k - 1$.

- Suppose that r_j is in D_2. Observe that both T' and T'' have separation $k - 1$, and their extensibilities are not complementary according to Definition 9. To see that they are not complementary, assume the naming convention of Figure 5.4 on page 48 with the current r_i, r_j, T', and T'' corresponding to the ones there. It should be clear that both $T'[c]$ and $T''[d]$ are $(k-1)$-critical and therefore can not have complementary extensibilities. Then Lemma 14 implies that $\widetilde{U}$ cannot be (k)-(r_i, r_j)-stretchable.
- Suppose that r_j is in D_1 or D_3. Then, one of T' or T'' has a vertex z that lies in T^1 or in T^2, such that z the induces three separation $k-1$ trees. It follows that $vs(T') = k$ or $vs(T'') = k$, by Theorem 1. Then, by Lemma 17, $\widetilde{U}$ cannot be (k)-(r_i, r_j)-stretchable.

So, indeed, not all D_i can be of separation $k-1$. Now we show that neither T^1 nor T^2 can be critical. Assume that at least one of them is critical and, without loss of generality, assume that is T^1.

- Suppose that r_j lies in D_1 or D_3. Then, one of T' or T'' has a vertex z that lies in T^1, such that z induces three separation $k - 1$ trees. It follows that $vs(T') = k$ or $vs(T'') = k$, by Theorem 1. Then, by

Lemma 17, $\widetilde{U}$ cannot be (k)-(r_i, r_j)-stretchable.

- Suppose that r_j lies in D_2 and T^1 is type $\mathcal{CB}$. Call f the critical vertex in $T^1[r^1]$. Call X the unicyclic graph $\widetilde{U} - T^1[f]$. Given that $\widetilde{U}$ is (k)--(r_i, r_j)-stretchable, by Corollary 5 it follows that X is $(k-1)$-(r_i, r_j)--stretchable. In X, call T'' the arch relative to r_i and r_j that contains T^2. But in this arch there is a separation $k-1$ constituent, so the whole T'' must be of separation at least $k-1$. Then Lemma 17 implies that X cannot be $(k-1)$-(r_i, r_j)-stretchable.

- Suppose that r_j lies in D_2 and T^1 is type $\mathcal{C}$. In X, call T' the arch relative to r_i and r_j that contains T^1 and T'', the other arch. Call a and c the vertices from T' that are adjacent to r_i and r_j, respectively. Apply Corollary 4 and conclude that $\widetilde{U}$ cannot be (k)-(r_i, r_j)-stretchable, because both $T'[a]$ and $T'[c]$ are $(k-1)$-critical, and T'' contains a constituent, namely T'', of separation $k-1$ and type not $\mathcal{NC}$.

So, indeed, none of T^1, T^2 can be critical.

Now assume that $vs(D_1) < k-1$. We claim that U^2 is (k)-(r_i, r^2)-stretchable. Call T' the arch that contains T^1. Then T'' is D_3. As $vs(D_1) < k-1$ and T^1 can be of separation less than $k-1$ or $k-1$ but not critical, it follows that $vs(T') \leq k-1$. Furthermore, $T'[c]$ can not be $(k-1)$-critical as D_2 can only contain constituent trees of separation less than $k-1$ or $k-1$ and type $\mathcal{NC}$. And $T'' = D_3$ can be of separation less than $k-1$ or $k-1$ but $T''[b]$ and $T''[d]$ cannot be $(k-1)$-critical. By the results from Case 3.2 on page 59, it follows that U^2 is (k)-(r_i, r^2)-stretchable.

By a completely analogous reasoning, assuming that $vs(D_3) < k-1$ implies that U^1 is (k)-(r_i, r^1)-stretchable.

Finally, assume that $vs(D_2) < k-1$. We argue that U^2 is (k)-(r_i, r^2)--stretchable. The proof is analogous to the one for the case when $vs(D_1) < k-1$. The difference is that now it is $T'[a]$ that can not be $(k-1)$-critical, while $T'[c]$ can be $(k-1)$-critical. But $T''[b]$ and $T''[d]$ cannot be $(k-1)$-critical can not be critical in this case either, so the results from Case 3.2 on page 59 are applicable again.

This concludes the proof of Lemma 23. □

Because of Lemma 23, in case that $q = 2$ we test whether U^1 is (k)-(r_i, r^1)--stretchable or U^2 is (k)-(r_i, r^2)-stretchable. If we get a positive answer, then $vs(U) = k$ and an optimal layout for U is constructed as in Case iii. If we get negative answers to both tests, we conclude that $vs(U) = k+1$.

- Suppose that $q = 3$.

Lemma 24. *In this case,* $vs(U) = k$ *if and only if there exists a* (k)--(r_i, r^1)*-stretchable layout for* U^1 *or* (k)-(r_i, r^2)*-stretchable layout for* U^2 *or* (k)-(r_i, r^3)*-stretchable layout for* U^3.

Proof:

If there exists a (k)-(r_i, r^1)-stretchable layout for U^1 or a (k)-(r_i, r^2)-stretchable layout for U^2 or a (k)-(r_i, r^3)-stretchable layout for U^3, then a construction completely analogous to Case iii will do obtain a separation k layout L for U. It remains to show the implication in the other direction.

Call T_i, T^1, T^2, and T^3, the heavy trees. Consider any optimal layout L for U. Call α and ω its endvertices. Apply Lemma 12, with the current heavy trees as the heavy trees there, to conclude that α and ω are in different heavy trees. In the premises of Lemma 12, the four heavy trees are of the

same separation, while in the current Lemma one of them, namely T_i, is of separation one bigger then the other ones. However, the proof of Lemma 12 remains valid even when one of the heavy trees is of separation one bigger than the other three, so the current conclusion remains valid. We can show that one of α and ω must be in T_i in a way completely analogous to the way we showed the analogous claim for $q = 3$. Without loss of generality, suppose that $\alpha \in T_i$.

Since ω is in T^1 or in T^2 or in T^3, by Lemma 10 it follows that at least one some L^s is (k)-(r_i, r^s)-stretchable, for $1 \leq s \leq 3$. □

Because of Lemma 24, in the case that $q = 3$ we test whether U^1 is (k)-(r_i, r^1)-stretchable or U^2 is (k)-(r_i, r^2)-stretchable or U^3 is (k)-(r_i, r^3)--stretchable. If we get a positive answer, then $vs(U) = k$ and an optimal layout for U is constructed as in Case iii. If we get negative answers to both tests, we conclude that $vs(U) = k + 1$.

- If $q \geq 4$, then, by Lemma 11, $vs(U) = k + 1$.

5.5.5 Case v

All constituent trees are of separation at most $k - 1$. There must be at least one separation $k - 1$ constituent tree whose type is not $\mathcal{NC}$, otherwise by Lemma 9 the separation of T_U would not be k.

Let the constituent trees of U of separation $k - 1$ and type $\mathcal{NCB}$, $\mathcal{C}$, or $\mathcal{CB}$ be $\{T^1, T^2, \ldots, T^t\}$, with $r^i = \text{root}(T^i)$, for $1 \leq i \leq t$. Let L^i be a (k)-left(r^i)-ext or (k)-right(r^i)-ext, whichever is convenient at the moment of usage, layout for T^i. Let $U^{i,j}$ denote U minus the bodies of T_i and T^j, for $1 \leq i, j \leq t$, $i \neq j$.

- Suppose that $t = 1$ or $t = 2$. Then $vs(U) = k$, by practically the same

reasoning as in Case iv with $q = 0$ or $q = 1$, respectively.

- Suppose that $t = 3$.

  Lemma 25. *These exists a separation* k *layout for* U *if and only if for some* $T^i, T^j \in \{T^1, T^2, T^3\}$, $U^{i,j}$ *is* (k)-(r^i, r^j)-*stretchable.*

  Proof:

  First, assume that for some $T^i, T^j \in \{T^1, T^2, T^3\}$, $U^{i,j}$ has a (k)-(r^i, r^j)--stretchable layout $L^{i,j}$. Using L^i, $L^{i,j}$, and L^j as in Case iii, we can build a separation k layout L for U.

  In the other direction, suppose a separation k layout L for U exists. Suppose that α and ω are the leftmost and the rightmost vertex, respectively, in L. Apply Lemma 13 to conclude that at least one of α or ω must be in one of T^1, T^2, or T^3. Then proceed as in Case iv with $q = 2$. □

- Suppose that $t = 4$.

  Lemma 26. *These exists a separation* k *layout for* U *if and only if for some* $T^i, T^j \in \{T^1, T^2, T^3, T^4\}$, $U^{i,j}$ *is* (k)-(r^i, r^j)-*stretchable.*

  Proof:

  First, assume that for some $T^i, T^j \in \{T^1, T^2, T^3, T^4\}$, $U^{i,j}$ has a (k)-(r^i, r^j)--stretchable layout $L^{i,j}$. Using L^i, $L^{i,j}$, and L^j as in Case iii, we can build a separation k layout L for U.

  In the other direction, suppose a separation k layout L for U exists. Suppose that α and ω are the leftmost and the rightmost vertex, respectively, in L. Apply Lemma 12 to conclude that both α and ω must be in distinct trees from $T^i, T^j \in \{T^1, T^2, T^3, T^4\}$, $U^{i,j}$. Then proceed as in Case iv with $q = 3$.

  □

- If that $q \geq 5$, then, by Lemma 11, $vs(U) = k + 1$.

5.6 The Algorithm

5.6.1 The main function

The main function takes as input a unicyclic graph U and returns its vertex separation and an optimal layout. The function carries out Stage 1, as described at the beginning of Section 5.5, and then invokes VS-UNICYCLIC(U, k). We assume that the vertex separation, the layout list, and the type of each constituent tree, is recorded during Stage 1.

MAIN-UNICYCLIC(U: unicyclic graph)

```
1  For all constituent trees, compute the separation, layout list and type
2  Choose a cycle edge e as follows
3  if there is a unique constituent tree of maximum separation
4      choose e arbitrarily among the edges, incident with its root
5  else
6      choose e arbitrarily among the edges of the cycle
7  T_U ← U − e
8  k ← vs(T_U)
9  L_U ← an optimal layout for T_U
10 if VS-UNICYCLIC(U, k)
11     return (k, the layout constructed by VS-UNICYCLIC)
12 else
13     return (k + 1, L_U)
```

5.6.2 Computing whether the separation is k or k + 1

At the completion of Stage 1, *i.e.* line 8, we know that $vs(U) \in \{k, k+1\}$. The function VS-UNICYCLIC takes as input a unicyclic graph U and an integer k, and returns TRUE or FALSE, indicating whether $vs(U) = k$ or not. The function summarises the results of Section 5.5. We assume that if it returns TRUE then it also returns an optimal layout using the constructions from Section 5.5. In the pseudo-code of the function below, we use the same case-subcase divsion as in Section 5.5, and the same names.

Note that VS-UNICYCLIC is recursive via the invocation of MAIN-UNICYCLIC at line 10, which calls VS-UNICYCLIC again. VS-UNICYCLIC does not call itself directly because it is designed to compute a boolean value, namely whether the separation is exactly equal to a number, not whether it is at most that number. At line 10 we need the information whether the separation is at most a given number.

VS-UNICYCLIC(U: unicyclic graph, k: positive integer)

```
1  Assume that T_1, T_2, ..., T_p are the constituent trees
2  switch
3      case i  Exactly one T_i has separation k and T_i is critical
4          if T_i is of type C
5              return TRUE
6          else
7              (* T_i is of type CB *)
8              u ← the critical vertex in T_i
9              U_1 ← U − T_i[u]
10             return MAIN-UNICYCLIC(U_1) ≤ k − 1
```

```
11      case ii  Three or more separation k constituent trees
12          return FALSE
13      case iii  Exactly two T_i, T_j have separation k
14          (* T_i and T_j are not critical *)
15          r_i ← root(T_i)
16          r_j ← root(T_j)
17          U* ← U minus the bodies of T_i and T_j
18          return IS-UNI-STR(U*, r_i, r_j, k)
19      case iv  Exactly one T_i has separation k and T_i is not critical
20          T^1, T^2, ..., T^q ← the separation k − 1 constituent trees of type
                                  NCB, C, or CB
21          switch
22              case q = 0 or q = 1
23                  return TRUE
24              case q = 2 or q = 3
25                  for j ← 1, ..., q
26                      r^j ← root(T^j)
27                      U^j ← U minus the bodies of T_i and T^j
28                      z^j ← IS-UNI-STR(U^j, r_i, r^j, k)
29                  return ( z^1 or z^2 or ... or z^q )
30              case q ≥ 4
31                  return FALSE
32      case v  There is no separation k constituent tree
33          T^1, T^2, ..., T^q ← the separation k − 1 constituent trees of type
                                  NCB, C, or CB
34          switch
```

```
35          case q = 1 or q = 2
36              return TRUE
37          case q = 3 or q = 4
38              for i ← 1, ..., q and j ← i + 1, ..., q
39                  r^i ← root(T^i)
40                  r^j ← root(T^j)
41                  U^{i,j} ← U minus the bodies of T^i and T^j
42                  z^{i,j} ← IS-UNI-STR(U^{i,j}, r^i, r^j, k)
43              return ( some z^{i,j} is TRUE )
44          case q ≥ 5
45              return False
```

5.6.3 Testing for stretchability

The function IS-UNI-STR takes as input a unicyclic graph U, two cycle vertices r_i and r_j of degree two, and an integer k, and returns boolean TRUE or FALSE, indicating whether U is k-stretchable with respect to r_i and r_j, or not. The function summarises the results of Section 5.4. We assume that if it returns TRUE, it also returns a stretchable layout, using the constructions from Section 5.4. In the pseudo-code of the function below, we use the same case-subcase divsion as in Section 5.4, and the same names.

We use the names T', T'', a, b, c, and d with the same meaning as in Section 5.4. Figure 5.17, which is identical to Figure 5.4 on page 48, shows the assumed structure of U. Also, the assumptions on page 54 about the separations and criticalities of T' and T'' hold.

Note that the function IS-UNI-STR is recursive too, but IS-UNI-STR and VS-UNICYCLIC are not mutually recursive. That is, once we start testing the stretch-

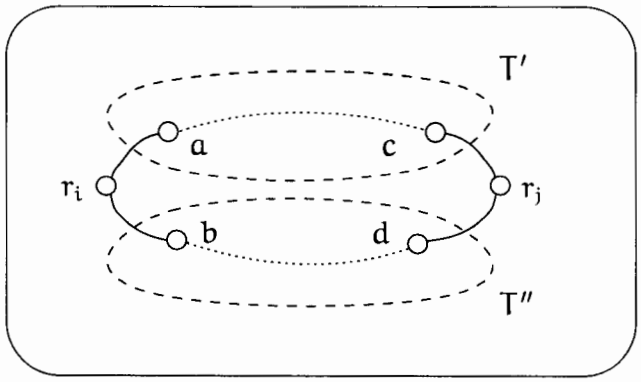


Figure 5.17: This figure is identical to Figure 5.4 on page 48.

ability with IS-UNI-STR, the function VS-UNICYCLIC is never invoked again.

IS-UNI-STR(U: unicyclic graph, r_i, r_j: degree 2 cycle vertices, k: integer)

```
1  switch
2      case vs(T′) = k
3          return FALSE
4      case vs(T′) ≤ k − 2
5          return TRUE
6      case vs(T′) = k − 1
7          if both T′[a] and T′[c] are critical
8              if T′ contains (k − 1)-critical, type CB, constituent tree
9                  f ← the critical vertex in both T′[a] and T′[c]
10                 T[f] ← the subtree of T′[a] (or T′[c]) rooted at f
11                 U* ← U - T[f]
12                 return IS-UNI-STR(U*, r_i, r_j, k − 1)
13             else
14                 if T″ contains no separation k − 1, type NCB, C, or
                       CB, constituent trees
15                     return TRUE
```

```
16                  else
17                      return FALSE
18          else    (* T'[a] or T'[c] is not critical *)
19              if νs(T'') ≤ k − 2
20                  return TRUE
21              else    (* νs(T'') = k − 1 *)
22                  z1 ← T'[a] is not critical and T''[d] is not critical
23                  z2 ← T'[c] is not critical and T''[b] is not critical
24                  return ( z1 or z2 )
```

5.7 Analysis of the Correctness and Time Complexity

The correctness follows immediately from our lemmas in Section 5.4 and Section 5.5. Suppose that U is a unicyclic graph, obtained from a tree T by adding an edge. Suppose that T has n vertices. In Section 3.2 of [EST94], the number of vertices in the smallest tree of separation k is given by the formula:

$$m(k) = \left\lfloor \frac{5 \times 3^k}{6} \right\rfloor, \text{ for all } k \geq 1.$$

Therefore, $\nu s(T) = \mathcal{O}(\log n)$. Since $\nu s(U) \bullet \{\nu s(T), \nu s(T) + 1\}$, it is the case that $\nu s(U) = \mathcal{O}(\log n)$, too.

We already noted that once IS-UNI-STR is called, VS-UNICYCLIC is never called again. In general, MAIN-UNICYCLIC calls VS-UNICYCLIC with number k, which goes into **case i** and calls MAIN-UNICYCLIC again on a smaller unicyclic, in its turn it calls VS-UNICYCLIC with number $k-1$, *etc.*, until at some moment

VS-UNICYCLIC calls the function IS-UNI-STR that tests stretchability, and it either gives an answer or calls itself, but MAIN-UNICYCLIC and VS-UNICYCLIC are not called any more.

Now consider how much computation is done.

- MAIN-UNICYCLIC does $\mathcal{O}(n)$ work, computing the separations and layout lists of the constituents and then $vs(T_U)$ and L_U, and calls VS-UNICYCLIC;
- VS-UNICYCLIC in its turn does an $\mathcal{O}(n)$ amount of work on T'.

This is done s times, say, for some $s \geq 1$, such that $s \leq vs(U)$. When VS-UNICYCLIC goes into case different from **case i**, it either answers immediately (**case ii**), or calls a number of times IS-UNI-STR. Call this number, c, and observe that c is at most $\binom{4}{2} = 6$, *i.e.* a constant.

Therefore, the amount of work done by VS-UNICYCLIC when the case is not **case i**, is $\mathcal{O}(n)$.

Now consider the amount of work in IS-UNI-STR. It does a linear, in terms of the number of vertices of the unicyclic graph it operates on, thus $\mathcal{O}(n)$, amount of work and either return a TRUE or FALSE, or calls itself on a smaller unicyclic graph with a number one less. Suppose the number of times IS-UNI-STR calls itself is some $t \geq 0$. Clearly, $t \leq vs(U) - s$.

So, overall we have $s + c \times t$ recursive invocations, where $s + t \leq vs(U)$ and $0 \leq c \leq 6$. Since $vs(U) = \mathcal{O}(\log n)$, $s+c\times t = \mathcal{O}(\log n)$, and at each invocation we have $\mathcal{O}(n)$ computation. Overall, the time complexity is bounded by $\mathcal{O}(n \log n)$.

Chapter 6

Conclusion and Future Work

We spent quite a lot of time and effort in discovering and verifying the presented algorithm. This may seem strange, having in mind that we solve VERTEX SEPARATION on what is essentially a tree with an extra edge added, and we have a quick and elegant algorithm for trees. However, it seems to us that adding the extra edge, *i.e.* creating the cycle, adds a lot of complexity to the problem.

Currently, we do research on VERTEX SEPARATION on cactus graphs. Informally, a cactus graph is a collection of unicyclic graphs that are "glued together" in a tree-like way. The work we did on unicyclic graphs is fundamental to discovering and verifying an algorithm for the more complicated cactus graphs. A lot of the lemmas that support the unicyclic result have their counterparts in the case of cactus graphs. The current research gives us a better understanding of the unicyclic case. Now we believe it is possible to simplify the unicyclic algorithm to a certain extent. Namely, it may be possible to solve the stretchability problem not by recursive invocation of the function testing the stretchability but rather by bottom-up simultaneous examination of the labels of the constituent trees. However, even if this turns out to be true, testing the stretchability with respect to several pairs of degree two cycle vertices seems inavoidable.

Bibliography

[ACP87] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal of Algebraic Discrete Methods*, 8:277–284, 1987.

[BF00] Hans L. Bodlaender and Fedor V. Fomin. Approximation of pathwidth of outerplanar graphs. Technical Report UU-CS-2000-23, Department of Computer Science, Utrecht University, Utrecht, the Netherlands, 2000.

[BGHK95] Hans L. Bodlaender, John R. Gilbert, Hjálmtýr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, March 1995.

[BK96] Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, September 1996.

[DKL87] N. Deo, M. S. Krishnamoorty, and M. A. Langston. Exact and approximate solutions for the gate matrix layout problem. *IEEE Trans. on Computer Aided Design*, 6:79–84, 1987.

[EM04] J. Ellis and M. Markov. Computing the vertex separation of unicyclic graphs. *Information and Computation*, 192:123–161, 2004.

[EST94] J. A. Ellis, I. H. Sudborough, and J. S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 15 August 1994.

[FK79] T. Fujisawa and T. Kashiwabara. NP-completeness of the problem of finding a minimum-clique-number interval graph containing a given graph as a subgraph. In *IEEE Symposium on Circuits and Systems*, pages 657–660, 1979.

[GGKS95] Paul W. Goldberg, C. Golumbic, Martin, Haim Kaplan, and Ron Shamir. Four strikes against physical mapping of DNA. *Journal of Computational Biology*, 2:139–152, 1995.

[GLY98] Rajeev Govindan, Michael A. Langston, and Xudong Yan. Approximating the pathwidth of outerplanar graphs. *Information Processing Letters*, 68(1):17–23, 15 October 1998.

[Gus93] Jens Gustedt. On the pathwidth of chordal graphs. *Discrete Applied Mathematics*, 45:233–248, 1993.

[KB92] T. Kloks and H. Bodlaender. Approximating treewidth and pathwidth of some classes of perfect graphs. Technical Report RUU-CS-92-29, Univ. Utrecht (Netherlands), Dept. Computer Science, 1992.

[Kin92] Nancy G. Kinnersley. The vertex separation number of a graph equals its path-width. *Information Processing Letters*, 42(6):345–350, 24 July 1992.

[Klo94] T. Kloks. Treewidth: Computations and approximations. *Lecture Notes in Computer Science*, 842:ix + 209, 1994.

[KP85] L. M. Kirousis and C. H. Papadimitriou. Interval graphs and searching. *Discrete Mathematics*, 55:181–184, 1985.

[KP86] L. M. Kirousis and C. H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(2):205–218, 1986.

[KT92] A. Kornai and Z. Tuza. Narrowness, pathwidth, and their application in natural language processing. *Discrete Applied Mathematics*, 36:87–92, 1992.

[LaP93] Andrea S. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2):224–245, April 1993.

[LL80] A. Lopez and H. Law. A dense gate matrix layout method for mos vlsi. In *IEEE Transactions on Electronic Devices*, number 27, pages 1671–1675, 1980.

[LMC00] Lucet, Manouvrier, and Carlier. Evaluating network reliability and 2-edge-connected reliability in linear time for bounded pathwidth graphs. *Algorithmica*, 27:316–336, 2000.

[Möh90] R. H. Möhring. Graph problems related to gate matrix layout and PLA folding. In Gottfried Tinhofer, R. (Rudolf) Albrecht, et al., editors, *Computational graph theory*, volume 7 of *Computing. Supplementum*, pages 17–51, Wien / New York, 1990. Springer.

[MS88] B. Monien and I. H. Sudborough. Min Cut is NP-complete for edge weighted trees. *Theoretical Computer Science*, 58(1-3):209–229, June 1988.

[Par78] T. Parsons. The search number of a connected graph. In *Proc. 9th Southeaster Conference on Combinatorics, Graph Theory, and Computing*, pages 549–554. Utilitas Math. Publ., Winnipeg, 1978.

[Pet82] N. Petrov. Some extremal search problems on graphs, differ. equations, 18 (1982), pp. 591–595, translation from differ. uravn. 18, 821–827 (1982), 1982.

[Pet83] N. Petrov. Pursuit problems with no information concerning the evader, differ. equations, 18 (1983), pp. 944–948. translation from differ. uravn. 18, 1345–1352 (1982), 1983.

[RS83] N. Robertson and P. D. Seymour. Graph minors I, excluding a forest. *Journ. Combinatorial Theory Ser. B*, 35:39–61, 1983.

[RS86] Neil Robertson and Paul D. Seymour. Graph minors II: algorithmic aspects of tree-width. *J. Algorithms*, 7:309–322, 1986.

[Sko03] Konstantin Skodinis. Construction of linear tree-layouts which are optimal with respect to vertex separation in linear time. *Journal of Algorithms*, 47:40–59, 2003.