SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF BASIC AND APPLIED SCIENCES FOR ENGINEERING

PhD Thesis in Mathematical Models for Engineering, Electromagnetism and Nanosciences

Non-Crossing Shortest Paths in Planar Graphs with Applications to Max Flow, and Path Graphs

Supervisor:
Prof. Paolo G. Franciosa

Candidate:
Lorenzo Balzotti

Cycle XXXV

Abstract

This thesis is concerned with non-crossing shortest paths in planar graphs with applications to st-max flow vitality and path graphs.

In the first part we deal with non-crossing shortest paths in a plane graph G, i.e., a planar graph with a fixed planar embedding, whose extremal vertices lie on the same face of G. The first two results are the computation of the lengths of the non-crossing shortest paths knowing their union, and the computation of the union in the unweighted case. Both results require linear time and we use them to describe an efficient algorithm able to give an additive guaranteed approximation of edge and vertex vitalities with respect to the st-max flow in undirected planar graphs, that is the max flow decrease when the edge/vertex is removed from the graph. Indeed, it is well-known that the st-max flow in an undirected planar graph can be reduced to a problem of non-crossing shortest paths in the dual graph. We conclude this part by showing that the union of non-crossing shortest paths in a plane graph can be covered with four forests so that each path is contained in at least one forest.

In the second part of the thesis we deal with path graphs and directed path graphs, where a (directed) path graph is the intersection graph of paths in a (directed) tree. We introduce a new characterization of path graphs that simplifies the existing ones in the literature. This characterization leads to a new list of local forbidden subgraphs of path graphs and to a new algorithm able to recognize path graphs and directed path graphs. This algorithm is more intuitive than the existing ones and does not require sophisticated data structures.

Contents

Chapter 1

Introduction

This thesis gives theoretical and algorithmic results on non-crossing shortest paths in plane graphs, with applications to st-max flow vitality. Moreover, we investigate paths graphs and directed path graphs.

The main algorithmic result, described in Chapter 6, is about vitality of edges and vertices with respect to the st-max flow in undirected planar graphs, that is the max flow decrease when the edge/vertex is removed from the graph. We give efficient algorithms to compute an additive guaranteed approximation of the vitality of edges and vertices in undirected planar graphs. We show that in the general case high vitality values are well approximated in time close to the time currently required to compute the st-max flow.

To reach this, we study the *non-crossing shortest paths* (NCSP) problem on a plane graph, i.e., a planar graph with a fixed planar embedding. Indeed, thanks to Reif [123], the st-max flow in undirected planar graphs can be reduced to a NCSP problem on the dual graphs. The NCSP problem can be formalized as follows: given an undirected plane graph G with positive edge lengths and k terminal pairs that lie on a specified face boundary, find k non-crossing shortest paths in G, each one connecting a terminal pair. It is assumed that terminal pairs appear on the infinite face so that non-crossing paths exist, where two paths in a plane graph G are *non-crossing* if the curves they describe in G's planar embedding do not cross each other.

We give efficient algorithms for st-max flow vitality in planar graphs by a *divide and conquer* approach, where a plane graph derived from the dual graph is split into regions by non-crossing shortest paths. More efficient (and in some case optimal) algorithms for the st-max flow vitality are given on planar graphs with unit capacity.

The NCSP problem was studied by Takahashi *et al.* [131] and solved in $O(n \log n)$ time, then Steiger [130] improved this result to $O(n \log \log k)$ time, where n is the number of the vertices of the input graph. Both algorithms work for graphs with positive edge weights, and maintain the same time complexity also in the unweighted case. In Chapter 5 we solve the NCSP problem on undirected unweighted plane graph in linear time. We use the result by Eisenstat and Klein [48], that gives an implicit representation of a sequence of shortest path trees rooted at the vertices in the infinite face of an undirected unweighted plane graph in linear time.

All algorithms solving the NCSP problem compute the union of the non-crossing shortest

paths and then their distances. In [131] it is claimed that their union is a forest, and the length of each path is computed by using the result by Gabow and Tarjan [54] about lowest common ancestor queries on trees. Actually, in the general case their union may contain cycles, and thus the result in [54] cannot be applied. In Chapter 4 we present an algorithm that, given the union of the non-crossing shortest paths, computes the length of each path in total linear time. This result is obtained by introducing *shortcuts*, that are portions of the boundary of a face that allow us to modify a path without increasing (and possibly decreasing) its length. We show that in this scenario it is possible to establish whether a path is a shortest path by looking at the presence of shortcuts. Thus we reduce, in this particular case, the global property of being a shortest path to a local property, which is the presence of shortcuts.

Since in general the union of shortest paths is not a forest, we asked ourselves how many forests are needed to cover it. Thus, for a set of paths P we define the *Path Covering with Forests Number of* P (PCFN(P)) as the minimum size of a set F of forests such that each path in P is contained in at least one forest in F. In Chapter 7 we prove that if P is a set of non-crossing shortest paths in an undirected plane graph G, whose extremal vertices lie on the same face of G, then PCFN(P) ≤ 4, and this bound is tight. We also describe a linear algorithm able to list these four covering forests.

The second part of the thesis is concerned with path graphs and directed path graphs, where a (directed) path graph is the intersection graph of paths in a (directed) tree. In Chapter 8 we describe two new characterizations of path graphs starting from Monma and Wei's characterization [102]. Our first characterization is the only one, to the best of our knowledge, that directly implies a polynomial recognition algorithm. This algorithm is explained in Chapter 9 and specializes also for directed path graphs. We do not decrease the time complexity for recognition algorithm for path graphs and directed path graphs, but we unify and strictly simplify the study of path graphs and directed path graphs from the algorithmic point of view.

Organization. In Chapter 2 we describe the state of the art about non-crossing shortest paths, max flow in planar graphs, topics related to Path Covering with Forests Number and theoretical and algorithmic results concerning path graphs and directed path graphs. In Chapter 3 we report notation and definition that will be used in the whole thesis and we also summarize the existing characterizations of path graphs and directed path graphs. Central chapters are organized as follows:

- Chapter 4, Chapter 5 and Chapter 7 deal with non-crossing shortest paths in a plane graph: in Chapter 4 we compute the lengths of non-crossing shortest paths in linear time given their union; in Chapter 5 we solve the NCSP problem in linear time in the unweighted case; in Chapter 7 we give a tight bound of PCFN(P) when P is a set of non-crossing shortest paths. These chapters are based on [17, 20, 18].
- in Chapter 6 we give efficient algorithms to compute an additive guaranteed approximation of the *st*-max flow vitality of edges and vertices in undirected planar graphs. We use a *divide and conquer* strategy based on non-crossing shortest paths and results in Chapter 4 and Chapter 5, see [19].

- in Chapter 8 we show two new characterizations of path graphs, and we use them to describe, in Chapter 9, an original recognition algorithm for path graphs and directed path graphs, see [11, 16].

Chapter 2

State of the art

In this chapter we describe the state of the art about treated topics and we compare them with respect to our results. In Section 2.1 we deal with non-crossing shortest paths in plane graphs. In Section 2.2 we report classical results on max flow and we focus the attention on max flow in planar graphs, showing the correlation with the NCSP problem. Section 2.3 is concerned with Path Covering with Forests Number and arboricity. Finally in Section 2.4 we resume theoretical and algorithmic results about path graphs and directed path graphs.

2.1 Non-crossing shortest paths in plane graphs

The problem of computing shortest paths in planar graphs arises in application fields such as intelligent transportation system (ITS) and geographic information system (GIS) [81, 141], route planning [22, 59, 120], logistic [100], traffic simulations [15] and robotics [85]. In particular, non-crossing shortest paths in a plane graph are studied to optimize VLSI layout [26, 95, 96], where two *non-crossing* paths may share edges and vertices, but they do not cross each other in the plane.

The *non-crossing shortest paths (NCSP)* problem on a plane graph can be formalized as follows: given an undirected plane graph G with positive edge lengths and k terminal pairs that lie on a specified face boundary, find k non-crossing shortest paths in G, each one connecting a terminal pair. It is assumed that terminal pairs appear in the infinite face so that non-crossing paths exist; this property can be easily verified in linear time.

Takahashi *et al.* [131] proposed an algorithm for computing k non-crossing shortest paths that requires $O(n \log n)$ time, where n is the size of G. In the same article it is also analyzed the case where the terminal pairs lie on two different face boundaries, and this case is reduced to the previous one within the same computational complexity. The complexity of their solution can be reduced to $O(n \log k)$ by plugging in the linear time algorithm by Henzinger *et al.* [75] for computing a shortest path tree in a planar graph. Their result was improved by Steiger to $O(n \log \log k)$ time [130], exploiting the algorithm by Italiano *et al.* [80]. In the unweighted case we exhibit in Chapter 5 a linear time algorithm; we stress that algorithms in [130, 131] maintain their same time complexity also in this case.

The algorithm proposed in [131] first computes the union of the k non-crossing shortest paths, which is claimed to be a forest. Then they obtain distances between the terminal pairs in $O(n)$ time by using a data structure due to Gabow and Tarjan [54] for efficiently solving least common ancestor (LCA) queries in a forest.

Actually, the union of the k non-crossing shortest paths may in general contain cycles. An instance is shown in Figure 2.1, in which the unique set of shortest paths contains a cycle, hence the distances between terminal pairs cannot always be computed by solving LCA queries in a forest. This limitation was noted first by Polishchuk and Mitchell [119], and it is overcome in Chapter 4, where a linear time algorithm for computing the lengths of non-crossing shortest paths is presented.

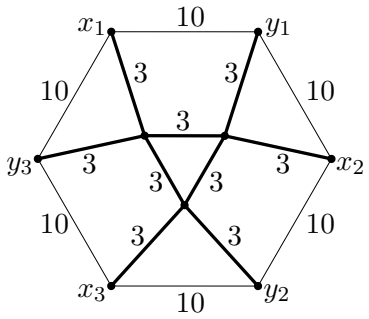


Figure 2.1: in this example the union of shortest paths from x_i to y_i, for $i = 1, 2, 3$, contains a cycle (the union is highlighted with bold edges).

Ericksonn and Nayyeri [49] stated that the union of non-crossing shortest paths can always be covered with two (possibly non edge-disjoint) forests so that each path is contained in at least one forest. They do not describe how to obtain such a covering. This is contradicted by the example in Figure 2.2, where we report the union of 15 non-crossing shortest paths that cannot be covered with two forests so that each path is contained in at least one forest (this is proved by a simple enumeration). This observation led us to introduce the Path Covering with Forests Number treated in Chapter 7.

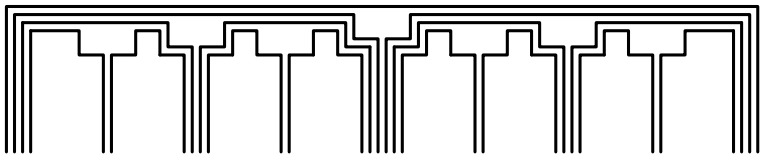


Figure 2.2: union of 15 non-crossing shortest paths that cannot be covered with two forests so that each path is contained in at least one forest (parallel adjacent segments represent overlapping paths).

2.1.1 Related work

The non-crossing shortest paths problem fits into a wider context of computing many distances in planar graphs. In the positive weighted case, the all pairs shortest paths (APSP) problem was solved by Frederickson in $O(n^2)$ time [53], while the single source shortest paths (SSSP) problem

was solved in linear time by Henzinger *et al.* [75]. The best-known algorithm for computing many distances in planar graphs is due to Gawrychowski *et al.* [58] and allows us to compute the distance between any two vertices in $O(\log n)$ time after a preprocessing requiring $O(n^{3/2})$ time. In the plane unweighted case, SSSP trees rooted at vertices in the infinite face can be computed in linear time as in [48]. More results on many distances problems can be found in [32, 38, 46, 51, 105, 109].

If we are interested in distances from any vertex in a fixed face f to any other vertex, then we can use Klein's algorithm [87], that, with a preprocessing of $O(n \log n)$ time, answers to each distance query in $O(\log n)$ time; this result was recently improved by Das *et al.* [42] to $O(n \log |f|)$ preprocessing time and $O(\log |f|)$ query time.

Kowalik and Kurowski [92] dealt with the problem of deciding whether any two query vertices of an unweighted planar graph are closer than a fixed constant r. After a preprocessing of $O(n)$ time, their algorithm answers in $O(1)$ time, and, if so, a shortest path between them is returned.

Wagner and Weihe [136] presented a $O(n)$ time algorithm for finding edge-disjoint (not necessarily shortest) paths in an undirected plane graph such that each path connects two specified vertices on the infinite face of the graph.

In a geometrical setting Papadopoulou [114] found the set of k non-crossing shortest paths between k terminal pairs of points on the boundary of a simple polygon with n vertices in $O(n + k)$ time. Eriksson-Bique *et al.* [50] studied the problem of computing shortest paths in a two-dimensional environment with polygonal obstacles.

More results on disjoint shortest paths for general graphs can be found in [24, 25, 47, 99] and in [43, 89, 128] for the planar case.

2.2 Max flow in planar graphs

Max flow problems have been intensively studied in the last 60 years, we refer to [2, 3] for a comprehensive bibliography. For general graphs, the currently best-known algorithms [86, 111] compute the max flow between two vertices in $O(mn)$ time, where m is the number of edges and n is the number of vertices.

Thanks to the fundamental work by Ford and Fulkerson [52], fixed two vertices s and t—the *source* and the *sink*—the st-max flow value is equal to the capacity of a minimum st-cut. For undirected planar graphs the first algorithm is due to Itai and Shiloach [79] and consists of two phases, each one requires $O(n^2 \log n)$. In the first phase a minimum st-cut is found, in the second a flow with value equal of the capacity of this cut is constructed. Johnson and Venkatesan [82] reached a $O(n^{3/2} \log n)$ time algorithm for both directed and undirected planar graphs by a *divide and conquer* strategy that operates on recursively subdivided regions. Reif [123] improved the method in [79] leading to a $O(n \log^2 n)$ time algorithm for st-min cut. Hassin and Johnson [73] extended the result in [123] giving a $O(n \log^2 n)$ time algorithm also for the st-max flow. Thanks to a new planar graph decomposition, called *r-division*, Frederickson [53] presented improved algorithms for shortest path and related problems as single source problem, all pairs problem, minimum cut and multicommodity flow. In particular he gave a $O(n \log n)$ time algorithm

for st-min cut and st-max flow. Italiano *et al.* [80] solved the st-max flow and st-min cut in $O(n \log\log n)$ time.

For directed st-planar graphs (i.e., graphs allowing a planar embedding with s and t on the same face) finding a max flow was reduced by Hassin [72] to the single source shortest path (SSSP) problem, that can be solved in $O(n)$ time by the algorithm in [75]. For the planar directed case, Borradaile and Klein [31] presented a $O(n \log n)$ time algorithm. In the special case of directed unweighted planar graphs, a linear time algorithm was proposed by Eisenstat and Klein [48].

2.2.1 Vitality

The effect of arcs deletion on the max flow value has been studied since 1963, only a few years after the seminal article by Ford and Fulkerson [52] in 1956. Wollmer [138] presented a method for determining the most vital link (i.e., the arc whose deletion causes the largest decrease of the max flow value) in a railway network. A more general problem was studied by Ratliff *et al.* [121], where an enumerative approach is proposed for finding the k arcs whose simultaneous removal causes the largest decrease in max flow. Wood [139] showed that this problem is NP-hard in the strong sense, while its approximability was studied in [10, 116]. A survey on network interdiction problems can be found in [6].

The *vitality* of an edge e (resp., of a vertex v) measures the max flow decrease observed after the removal of edge e (resp., all edges incident on v) from the graph. A survey on vitality with respect to max flow problems can be found in [12]. In the same article, it is shown that:

- the vitality of all edges in a general undirected graph can be computed by solving $O(n)$ max flow instances, thus giving an overall $O(n^2 m)$ time algorithm by applying the $O(mn)$ max flow algorithms in [86, 111];
- for st-planar graphs (both directed or undirected) the vitality of all edges can be found in optimal $O(n)$ time. The same result holds for determining the vitality of all vertices;
- the problem of determining the max flow vitality of a single edge is at least as hard as the max flow problem, both for general graphs and for the restricted class of st-planar graphs.

Ausiello *et al.* [13] proposed a recursive algorithm that computes the vitality of all edges in an undirected unweighted planar graph in $O(n \log n)$ time.

The above-cited article by Italiano *et al.* [80] also gives a dynamic algorithm that allows the operations described in the following theorem.

Theorem 1 ([80]). *Let G be a planar graph with positive edge capacities. There exists a data structure that after $O(n \log r + \frac{n}{\sqrt{r}} \log n)$ preprocessing time supports: edge insertions and edge deletions in $O((r + \frac{n}{\sqrt{r}}) \log^2 n)$ time; s to t distance queries in $O((r + \frac{n}{\sqrt{r}}) \log^2 n)$ time; max st-flow queries in $O((r + \frac{n}{\sqrt{r}}) \log^3 n)$ time, where $r \in [1, \ldots, n]$.*

The dynamic structure given in Theorem 1 can be used to compute edge and vertex vitality. Thus by properly choosing r, as implicitly shown in Łącki and Sankowski's article [94], the following corollary holds.

Corollary 2 ([80, 94]). *Let G be a planar graph with positive edge capacities. Then it is possible to compute the vitality of h single edges or the vitality of a set of h edges in $O(\min\{\frac{hn}{\log n} + n \log\log n, hn^{2/3}\log^{8/3} n + n\log n\})$.*

In Chapter 6 we give efficient algorithms to compute an additive approximation of the vitality of edges and vertices in undirected planar graphs.

2.3 Graph covering problem

In Chapter 7 we introduce the Path Covering with Forests Number focusing on non-crossing shortest paths in a plane graph. This result gives a structure of a particular set of shortest paths. There is a very restricted literature dealing with this problem for general graph, a first recent result by Bodwin [27] in 2019 develops a structural theory of unique shortest paths in real weighted graphs: the author characterizes exactly which sets of node sequences can be realized as unique shortest paths in a graph with arbitrary real edge weights. The characterizations are based on a new connection between shortest paths and topology; in particular, the new forbidden patterns are in natural correspondence with two-colored topological 2-manifolds, which are visualized as polyhedra.

2.3.1 Arboricity

The Path Covering with Forests Number is strictly linked to the concept of arboricity. The *arboricity* of an undirected graph G is the minimum number of forests $\gamma_f(G)$ into which its edges can be partitioned. It measures how a graph is dense; indeed, graphs with many edges have high arboricity, and graphs with high arboricity must have a dense subgraph. By the well-known Nash-Williams Theorem [107] (proved also independently by Tutte [134])

$$\gamma_f(G) = \max_{X \subseteq V(G)} \left\lceil \frac{|E(G[X])|}{|X| - 1} \right\rceil$$

where $G[X]$ denotes the subgraph of G induced by X. The *fractional arboricity* was introduced by Payan in [115], see also [35, 60]. Arboricity has been studied for general graphs and was specialized for planar graph and subclasses of planar graphs. By the above-cited Nash-Williams Theorem [107], every planar graph has arboricity 3, i.e., every planar graph can be covered with at most 3 forests, and if it has girth greater or equal to 4, then it decomposes into two forests. In [62, 74] planar graphs with girth larger than some constant are decomposed into a forest and a graph with bounded degree. Decomposition of planar graphs into a forest and a matching has been studied in [21, 29, 30, 74, 103, 137].

Arboricity is one of the many faces of *graphs covering* [23, 70, 71, 110] which is a classical problem in graph theory. A recent and complete overview about covering problems can be found in [129]. The classical covering problem asks for covering an input graph H with graphs from a fixed covering class $\mathcal{G}$. Some variants of the problems are in [88]. In the arboricity problem the family $\mathcal{G}$ consists of forests. Other kinds of arboricity have been introduced in literature, as

star arboricity [5, 7, 9], caterpillar arboricity [61, 63], linear arboricity [4, 8, 122, 140], pseudo arboricity [69, 117] in which graphs are covered with star forests, caterpillar forests, linear forests, and pseudoforests (undirected graphs in which every connected component has at most one cycle), respectively.

If the covering class is the class of planar graphs, outerplanar graphs, or interval graphs, then we deal with planar thickness [23], outerplanar thickness [106] or track number [67], respectively.

2.4 Path graphs and directed path graphs

A *path graph* is the intersection graph of paths in a tree. A *directed path graph* is the intersection graph of paths in a directed tree. Note that a directed path graph is an undirected graph.

Path graphs were introduced by Renz [124], who also gave a combinatorial, non-algorithmic characterization. A second characterization is due to Gavril [57] and consists in a specialization of his characterization of *chordal graphs* in [55]. A graph is a chordal graph if it does not contain a *hole* as an induced subgraph, where a hole is a chordless cycle of length at least four. In [57] it is given a first recognition algorithm for path graphs having $O(n^4)$ time complexity. A more general approach is due to Monma and Wei [102], that, inspired by the work by Tarjan [132], characterized several classes of intersection graphs of paths in a tree, varying some variants of trees. In Chapter 8 we simplify the characterization in [102] by reducing it to some 2-coloring subproblems, obtaining the first characterization that directly leads to a polynomial recognition algorithm; we stress that the characterizations in [55, 102, 124] do not give rise to a polynomial recognition algorithm. The characterization in [102] was used by Schäffer to build a faster recognition algorithm [127], that has $O(p(m+n))$ time complexity (where p is the number of *cliques*, i.e., maximal induced complete subgraphs). Later, Chaplick [36] gave a recognition algorithm with the same time complexity that uses PQR-trees. Lévêque *et al.* [98] presented the first characterization by forbidden subgraphs, and in Chapter 8 we show a new smaller characterization by local forbidden subgraphs. A generalization of the *asteroidal triples* (where an asteroidal triple is a stable set of three vertices such that each pair is connected by a path avoiding the neighborhood of the third vertex) was introduced by Mouatadid and Robere [104], where path graphs are characterized by forbidding *sun systems*. Another algorithm is proposed by Dahlhaus and Bailey [41] and claimed to run in $O(m+n)$ time, but it has only appeared as an extended abstract (see comments in [[36], Section 2.1.4]).

Directed path graphs were characterized first by Panda [113] by a list of forbidden induced subgraphs and then by Cameron *et al.* [33, 34] by extending the concept of asteroidal triples. Another characterization is due to Gavril [56], and in the same article he also gave the first recognition algorithms that has $O(n^4)$ time complexity. In the above-cited article, Monma and Wei [102] gave a characterization of directed path graphs that yielded to a recognition algorithm with $O(n^2m)$ time complexity. Chaplick *et al.* [37] presented a linear time algorithm able to establish whether a path graph is a directed path graph (actually, their algorithm requires the *clique path tree* of the input graph, we refer to Section 3.5 for further details). This implies that algorithms in [36, 127] can be used to obtain a recognition algorithm for directed path graphs

with the same time complexity. At the state of art, this technique leads to the fastest algorithms. In Chapter 9, by using the characterization of path graphs in Chapter 8, we present the first recognition algorithm that specializes for path graphs and directed path graphs. It does not require complex data structures and has an easy and intuitive implementation.

Path graphs and directed path graphs are graphs' classes between *interval graphs* and chordal graphs. Gavril [55] proved that a graph is chordal if and only if it is the intersection graph of subtrees of a tree. Chordal graphs can be recognized in $O(m+n)$ time [125, 133]. A graph is an interval graph if it is the intersection graph of a family of intervals on the real line; or, equivalently, the intersection graph of a family of subpaths of a path. Interval graphs were characterized by Lekkerkerker and Boland [97] as chordal graphs with no asteroidal triples. Interval graphs can be recognized in linear time by several algorithms [28, 40, 68, 77, 78, 90, 101].

We introduce a last class of intersection graphs. A *rooted path graph* is the intersection graph of directed paths in a rooted tree. Rooted path graphs can be recognized in linear time by using the algorithm by Dietz [45]. To the best of our knowledge, there does not exist any characterization of rooted path graphs by forbidden subgraphs or by concepts similar to asteroidal triples. The characterizations of these graphs' classes in [102] also describe directly polynomial recognition algorithms for chordal graphs and directed path graphs but not for rooted path graphs and path graphs. All inclusions between introduced graphs' classes are resumed in the following:

interval graphs $\subset$ rooted path graphs $\subset$ directed path graphs $\subset$ path graphs $\subset$ chordal graphs.

Chapter 3

Preliminaries

General definitions and notations used in the whole thesis are given. In Section 3.1 we delineate standard notations about graphs. In Section 3.2 we deal with paths and non-crossing paths. Then in Section 3.3 we define a partial order on a set of terminal pairs lying on the same face of a plane graph, the *genealogy tree*. In Section 3.4 we report some results concerning the max flow problem, focusing on planar graphs. Finally, we describe existing characterizations of path graphs and directed path graphs in Section 3.2.

3.1 General definitions

An *undirected graph* is a pair $G = (V, E)$, where V, $|V| = n$, is a set of *vertices*, and E, $|E| = m$, is a collection of pairs of vertices called *edges*. A *directed graph* is a pair $G = (V, D)$, where V is a set of vertices and D is a collection of ordered pairs of vertices called *darts*.

We recall standard union and intersection operators on graphs.

Definition 3. *Given two undirected (or directed) graphs* $G = (V(G), E(G))$ *and* $H = (V(H), E(H))$*, we define the following operations and relations:*

- $G \cup H = (V(G) \cup V(H), E(G) \cup E(H))$*;*
- $G \cap H = (V(G) \cap V(H), E(G) \cap E(H))$*;*
- $H \subseteq G \iff V(H) \subseteq V(G)$ *and* $E(H) \subseteq E(G)$*;*
- $G \setminus H = (V(G), E(G) \setminus E(H))$.

Given an undirected (resp., directed) graph $G = (V(G), E(G))$, given an edge (resp., dart) e and a vertex v we write, for short, $e \in G$ in place of $e \in E(G)$ and $v \in G$ in place of $v \in V(G)$.

We denote by uv the edge whose endpoints are u and v and we denote by $\overrightarrow{uv}$ the dart from u to v. For each dart $\overrightarrow{uv}$ we define $\text{rev}[\overrightarrow{uv}] = \overrightarrow{vu}$, $\text{head}[\overrightarrow{uv}] = v$ and $\text{tail}[\overrightarrow{uv}] = u$. For each vertex $v \in V(G)$ we define the *degree of* v as $deg(v) = |\{e \in E(G) \mid v \text{ is an endpoint of } e\}|$.

We use round brackets to denote ordered sets. For example, $\{a, b, c\} = \{c, a, b\}$ and $(a, b, c) \neq (c, a, b)$. Moreover, for each $\ell \in \mathbb{N}$ we denote by $[\ell]$ the set $\{1, \ldots, \ell\}$.

We define $\mathrm{dist}_G(u,v)$ as the length of a shortest path in G joining vertices u and v. Moreover, for two sets of vertices $S,T \subseteq V(G)$, we define $\mathrm{dist}_G(S,T) = \min_{u\in S, v\in T} \mathrm{dist}_G(u,v)$.

Let $\omega : E(G) \to \mathbb{R}^+$ be a weight function on edges. The weight function is extended to a subgraph H of G so that $\omega(H) = \sum_{e\in E(H)} \omega(e)$. If G is unweighted, then we denote the weight of a subgraph H as $|H|$, that is the number of edges. When we deal with flow problems we assign to edges a *capacity function* $c : E(G) \to \mathbb{R}^+$.

3.2 Paths and non-crossing paths

A *path* is a graph $p = (V(p), E(p))$ where $V(p) = \{v_1, v_2, \ldots, v_k\}$ and $E(p) = \{v_1v_2, v_2v_3, \ldots, v_{k-1}v_k\}$. A *directed path* is a directed graph $q = (V(q), D(q))$ where $V(q) = \{u_1, u_2, \ldots, u_\ell\}$ and $D(q) = \{\overrightarrow{u_1u_2}, \overrightarrow{u_2u_3}, \ldots, \overrightarrow{u_{\ell-1}u_\ell}\}$. We say that a path (resp. directed path) p is *simple* if there do not exist three edges (resp., darts) in p sharing an extremal vertex. We define a path p as an *ab path* if its extremal vertices are a and b; clearly, if p is a directed path, then p starts in a and ends in b.

Given a directed path p we denote by $\overline{p}$ its undirected version, in which each dart $\overrightarrow{uv}$ is replaced by edge uv; moreover, we denote by $\mathrm{rev}[p]$ its reverse version, in which each dart $\overrightarrow{uv}$ is replaced by dart $\overrightarrow{vu}$.

Given an ab path p and a bc path q, we define $p \circ q$ as the (possibly not simple) ac path obtained by the union of p and q.

Let p be a simple path and let $x, y \in V(p)$. We denote by $p[x,y]$ the subpath of p with extremal vertices x and y.

If G is a plane graph, then we denote by f_G^∞ (or simply f^∞) its unique infinite face. Given a face f of G we denote by ∂f its boundary cycle. Topological and combinatorial definitions of planar graph, embedding and face can be found in [64].

We say that two paths in a plane graph G are *non-crossing* if the curves they describe in the graph embedding do not cross each other; a combinatorial definition of non-crossing paths can be based on the *Heffter-Edmonds-Ringel rotation principle* [118]. We stress that this is a property of the graph embedding, not of the graph itself. Non-crossing paths may share vertices and/or edges.

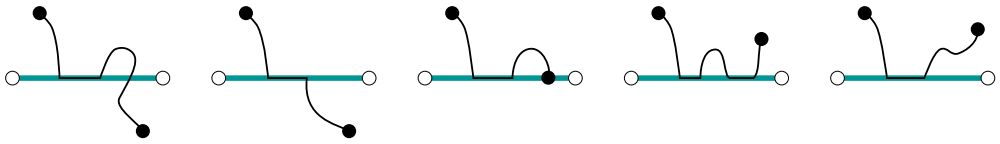
Definition 4. *Two paths p and q are* single-touch *if $\overline{p} \cap \overline{q}$ is a (possibly empty) path.*

Examples of non-crossing paths and single-touch paths are given in Figure 3.1.

Given a (possibly not simple) cycle C in a plane graph G, we define the *region bounded by* C, denoted by R_C, as the maximal subgraph of G whose infinite face has C as boundary. If R is a subgraph of G, then we denote by ∂R the infinite face of R. Finally, we define $\mathring{R} = R \setminus \partial R$.

3.3 Genealogy tree of terminal pairs

Given a set $\{(x_i, y_i)\}_{i\in[k]}$ of distinct terminal pairs on the infinite face f^∞ of a plane graph G, we define γ_i the path in f^∞ that goes clockwise from x_i to y_i, for $i \in [k]$. We deal with non-crossing



(a) (b) (c) (d) (e)

Figure 3.1: paths in (a) and (b) are crossing, while paths in (c), (d), (e) are non-crossing. Moreover, paths in (a), (c) and (d) are not single-touch, while paths in (b) and (e) are single-touch.

path, thus we assume that terminal pairs $\{(x_i, y_i)\}_{i\in[k]}$ are *well-formed*, i.e., for all $j, \ell \in [k]$ either $\gamma_j \subseteq \gamma_\ell$ or $\gamma_j \supseteq \gamma_\ell$ or γ_j and γ_ℓ share no edges. Indeed, if terminal pairs are well-formed, then there exists a set of pairwise non-crossing shortest $x_i y_i$ paths. The reverse is not true if some paths are subpaths of the infinite face of G; this case is not interesting in the applications and has never been studied in literature, where the terminal pairs are always assumed to be well-formed. The well-formed property can be easily verified in linear time, since it corresponds to checking that a string of parentheses is balanced, and it can be done by a sequential scan of the string. We also assume that the terminal pairs are distinct, i.e., there does not exist any pair $i, j \in [k]$ such that $\{x_i, y_i\} = \{x_j, y_j\}$.

We define here a partial ordering as in [131] that represents the inclusion relation between γ_i's. This relation intuitively corresponds to an *adjacency* relation between non-crossing shortest paths joining each pair. Choose an arbitrary i^* such that there are neither x_j nor y_j, with $j \neq i^*$, walking on f^∞ from x_{i^*} to y_{i^*} (either clockwise or counterclockwise), and let e^* be an arbitrary edge on that walk. For each $j \in [k]$, we can assume that $e^* \notin \gamma_j$, indeed, if it is not true, then it suffices to switch x_j and y_j; in this way $\gamma_j \subseteq \gamma_{i^*}$ for every $j \in [k]$. We say that $i \preceq j$ if $\gamma_i \subseteq \gamma_j$. We define the *genealogy tree* T_g of a set of well-formed terminal pairs as the transitive reduction of poset $([k], \preceq)$. By the above choice, i^* is the root of the genealogy tree, and there are as many possible genealogy trees as many leaves of each genealogy tree. W.l.o.g., we assume that $i^* = 1$.

If $i \preceq j$, then we say that i is a *descendant* of j and j is an *ancestor* of i. Moreover, we say that j is the *parent of* i, and we write $p(i) = j$, if $i \preceq j$ and there does not exist r such that $i \preceq r$ and $r \preceq j$. Figure 3.2 shows a set of well-formed terminal pairs and the corresponding genealogy tree. From now on, in all figures we draw f^∞ by a solid light grey line. W.l.o.g., we assume that the infinite face is a simple cycle and that G is a biconnected graph. Indeed, if there is an articulation point z in G, then the solution to a NCSP problem is the union of the solutions of the NCSP problems on the two components; note that if x_i and y_i are in two distinct components, then every $x_i y_i$ path passes through z.

Given $i \in [k]$, we denote by *i-path* an $x_i y_i$ path. We extend $\preceq$ also to i-paths in the following way: given an i-path p and a j-path q, we write $p \preceq q$ if $i \preceq j$. Moreover, if p is an i-path, for any $i \in [k]$, then x_p and y_p denote x_i and y_i, respectively.

For an i-path p, we define Int_p as the internal portion of G with respect to p, i.e., the finite region bounded by the cycle formed by p and γ_i; similarly, we define Ext_p as the external portion

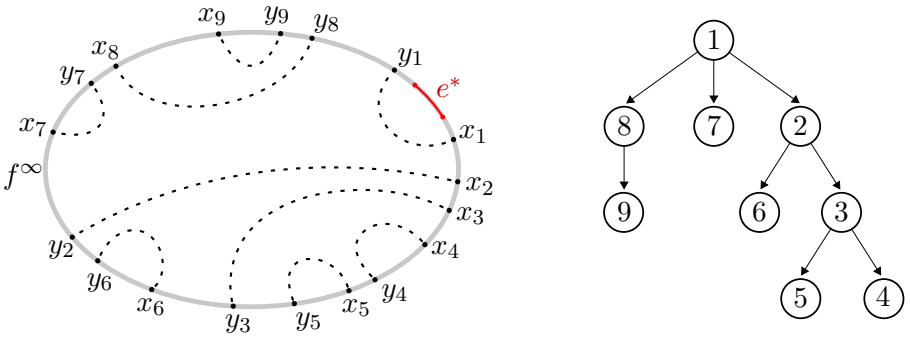


Figure 3.2: on the left a set of well-formed terminal pairs. If we choose $i^* = 1$, then we obtain the genealogy tree on the right.

of G with respect to p, i.e., the finite region bounded by the cycle formed by p and $f^\infty \setminus \gamma_i$.

It is always useful to see each i-path as oriented from x_i to y_i, for $i \in [k]$, even if the path is undirected. For an i-path p and a j-path q, we say that q *is to the right of* p if $q \subseteq \mathrm{Ext}_p$, similarly, we say that q *is to the left of* p if $q \subseteq \mathrm{Int}_p$. Given $R \subseteq G$ and an i-path $p \subseteq R$, for some $i \in [k]$, we say that p is the *leftmost i-path in R* if p is to the left of q for each i-path $q \subseteq R$. Similarly, we say that p is the *rightmost i-path in R* if p is to the right of q for each i-path $q \subseteq R$.

3.4 Max flow in planar graphs

In this section we report some well-known results about max flow, focusing on planar graphs.

Let G be a graph and let s and t be two distinct fixed vertices in G. A *feasible flow* in G assigns to each edge $e = uv \in G$ two real values z_{uv} and z_{vu} such that:

- (*Capacity constraint*) $z_{uv} \in [0, c(e)]$ and $z_{vu} \in [0, c(e)]$;
- (*Conservation of flows*) $\sum_{v:uv\in E(G)} z_{uv} = \sum_{v:uv\in E(G)} z_{vu}$, for each $u \in V(G) \setminus \{s, t\}$;

where we recall that $c : E(G) \to \mathbb{R}^+$ is the capacity function. The *flow from s to t* under a feasible flow assignment x is defined as

$$F(x) = \sum_{v:sv\in E(G)} z_{sv} - \sum_{v:sv\in E(G)} z_{vs}.$$

The *maximum flow* from s to t, denoted by MF, is the maximum value of $F(x)$ over all feasible flow assignments x.

An *st-cut* is a partition of $V(G)$ into two subsets S and T such that $s \in S$ and $t \in T$. The capacity of an st-cut is the sum of the capacities of the edges $uv \in E(G)$ such that $|S \cap \{u, v\}| = 1$ and $|T \cap \{u, v\}| = 1$. The well known Min-Cut Max-Flow Theorem by Ford and Fulkerson [52] states that the maximum flow from s to t is equal to the capacity of a minimum st-cut for any weighted graph G.

We denote by $G - e$ the graph G after the removal of edge e. Similarly, we denote by $G - v$ the graph G after the removal of vertex v and all edges adjacent to v.

Definition 5. *The vitality $vit(e)$ (resp., $vit(v)$) of an edge e (resp., vertex v) with respect to the maximum flow from s to t, according to the general concept of vitality in [91], is defined as the difference between the maximum flow in G and the maximum flow in $G - e$ (resp., $G - v$).*

Now we deal with the max flow in a planar graph. The dual of an undirected plane graph G is an undirected plane multigraph G^* whose vertices correspond to faces of G and such that for each edge e in G there is an edge $e^* = \{u^*, v^*\}$ in G^*, where u^* and v^* are the vertices in G^* that correspond to faces f and g adjacent to e in G. Length $\omega(e^*)$ of e^* equals the capacity of e.

We fix a planar embedding of the planar graph G, and we work on the dual graph G^* defined by this embedding. A vertex v in G generates a face in G^* denoted by f_v^*. We choose in G^* a vertex v_s^* in f_s^* and a vertex v_t^* in f_t^*. A cycle in the dual graph G^* that separates vertex v_s^* from vertex v_t^* is called an *st-separating cycle*. Moreover, we choose a shortest path π in G^* from v_s^* to v_t^*.

Proposition 6 ([79, 123])**.** *A (minimum) st-cut in G corresponds to a (shortest) cycle in G^* that separates vertex v_s^* from vertex v_t^*.*

3.4.1 From max flow to non-crossing shortest paths

According to the approach by Itai and Shiloach in [79] used to find a min-cut by searching for minimum *st*-separating cycles, graph G^* is "cut" along the fixed shortest path π from v_s^* to v_t^*, obtaining graph D_G, in which each vertex v_i^* in π is split into two vertices x_i and y_i; when no confusion arises we omit the subscript G. In Figure 3.3 there is a plane graph G in black continuous lines and in Figure 3.4 on the right graph D. Now we explain the construction of the latter.

Let us assume that $\pi = \{v_1^*, v_2^*, \ldots, v_k^*\}$, with $v_1^* = v_s^*$ and $v_k^* = v_t^*$. For convenience, let π_x be the duplicate of π in D whose vertices are $\{x_1, \ldots, x_k\}$ and let π_y be the duplicate of π in D whose vertices are $\{y_1, \ldots, y_k\}$. For any $i \in [k]$ edges in G^* incident on each v_i^* from below π are moved to y_i and edges incident on v_i^* from above π are moved to x_i. Edges incident on v_s^* and v_t^* are considered above or below π on the basis of two dummy edges: the first joining v_s^* to a dummy vertex α inside face f_s^* and the second joining v_t^* to a dummy vertex β inside face f_t^*. In Figure 3.3 there is a graph G in black continuous line, G^* in red dashed lines and shortest path π from v_1^* to v_k^*. In Figure 3.4, on the left there are the graph G and G^* of Figure 3.3 where path π is doubled.

For each $e^* \in \pi$, we denote by e_x^* the copy of e^* in π_x and e_y^* the copy of e^* in π_y. Note that each $v \in V(G) \setminus \{s, t\}$ generates a face f_v^D in D. There are not faces f_s^D and f_t^D because the dummy vertices α and β are inside faces f_s^* and f_t^*, respectively. Both faces f_s^* and f_t^* "correspond" in D to the leftmost x_1y_1 path and to the rightmost x_ky_k path, respectively. Since we are not interested in removing vertices s and t, then faces f_s^D and f_t^D are not needed in D. In Figure 3.4, on the right there is graph D built on G in Figure 3.3.

If $e^* \notin \pi$, then we denote the corresponding edge in D by e^D. Similarly, if $v_i^* \notin \pi$ (that is, $i > k$), then we denote the corresponding vertex in D by v_i^D.

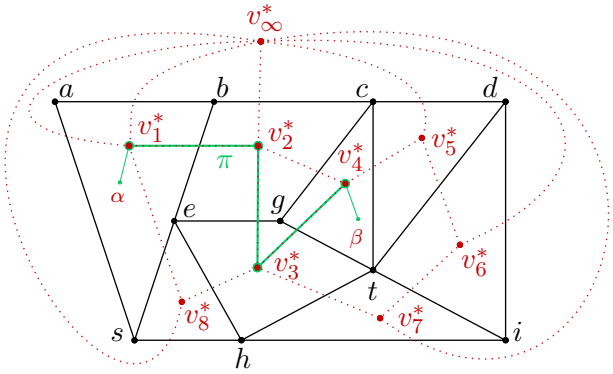


Figure 3.3: graph G in black continuous line, G^* in red dashed lines, shortest path π from v^*_s (v^*_1) to v^*_t (v^*_k) in green, α and β are dummy vertices.

Itai and Shiloach [79] considered only shortest st-separating cycles that cross π exactly once, that correspond in D to paths from x_i to y_i, for some $i \in [k]$. Reif [123] noticed that these paths can be chosen as non-crossing paths, thus he reduced the computation of the max flow in a planar graph G to a problem of non-crossing shortest paths in D.

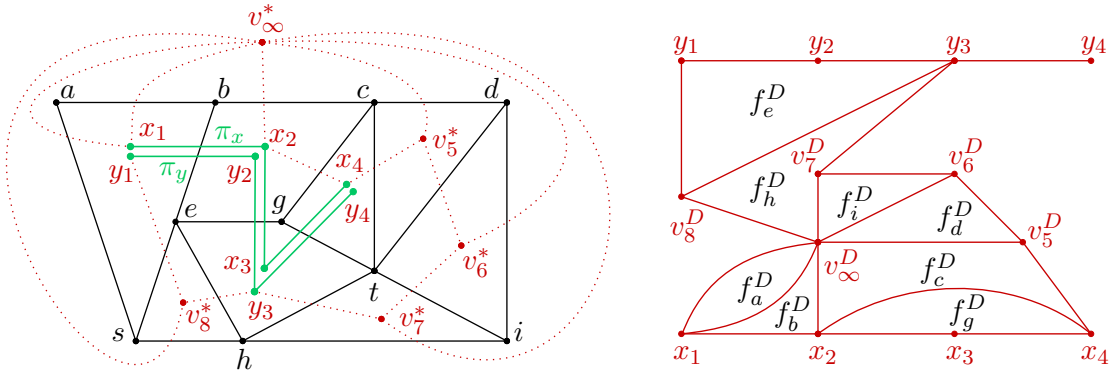


Figure 3.4: on the left green path π is doubled into paths π_x and π_y, and edges incident on x_1, y_1, x_4, y_4 in G^* are moved according to the dummy vertices α and β in Figure 3.3. On the right graph D.

3.5 Existing characterizations of path graphs and directed path graphs

In this section we report the characterizations of path graphs and directed path graphs described by Monma and Wei [102]. We start with a formal definition of these classes of graphs.

Let P be a finite family of nonempty sets. The intersection graph of P is obtained by associating each set in P with a vertex and by connecting two vertices with an edge exactly when their corresponding sets have a nonempty intersection. The intersection graph of a family of paths in a tree is called *path graph*. The intersection graph of a family of directed paths in a directed tree is called *directed path graph*. We consider that two directed or undirected paths intersect each other if and only if they share at least one vertex.

The first characterization of path graphs and directed path graphs is due to Gavril [56, 57]. We let $\mathbf{C}$ denote the set of cliques of a graph G (we recall that a clique is a maximal induced complete subgraph) and for each $v \in V(G)$ let $\mathbf{C}_v = \{C \in \mathbf{C} \mid v \in C\}$. Moreover, for a graph G and for a subset A of $V(G)$, we denote the graph induced by A in G by $G[A]$.

Theorem 7 ([56, 57]). *A graph $G = (V, E)$ is a path graph (resp. directed path graph) if and only if there exists a tree T (resp. directed tree T) with vertex set $\mathbf{C}$, such that for each $v \in V$, $T[\mathbf{C}_v]$ is a path (resp., directed path) in T.*

The tree T of the previous theorem is called the *clique path tree of G* if G is a path graph or the *directed clique path tree of G* if G is a directed path graph. In Figure 3.5, the left part shows a path graph G, and on the right there is a clique path tree of G. Simmetrically, in Figure 3.6, the left part shows a directed path graph G, and on the right there is a directed clique path tree of G.

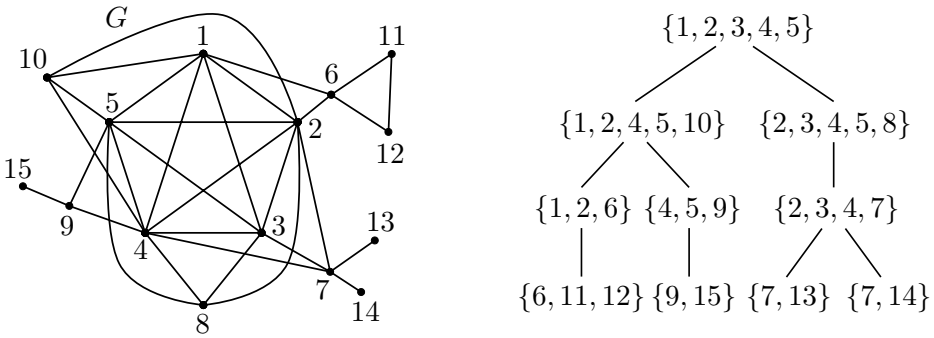


Figure 3.5: on the left a path graph G, and on the right a clique path tree of G.

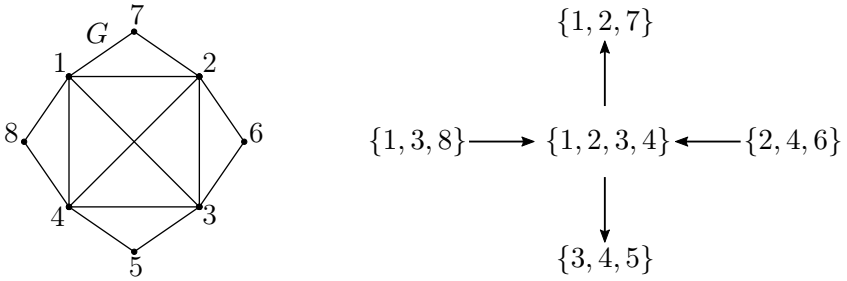


Figure 3.6: on the left a directed path graph G, and on the right a directed clique path tree of G.

Theorem 7 specializes the characterization of chordal graphs, still due to Gavril [55], as those graphs possessing a *clique tree* as stated below.

Theorem 8 ([55]). *A graph G is a chordal graph if and only if it there exists a tree T, called* clique tree*, with vertex set $\mathbf{C}$ such that, for each $v \in V$, $T[\mathbf{C}_v]$ is a tree in T.*

Note that since a clique path tree or a directed clique path tree is a particular clique tree, Theorem 8 also implies that path graphs and directed path graphs are chordal.

As we said, Monma and Wei [102] characterized several class of intersection families, two of these families are path graphs and directed path graph. Now we report some their theorems and lemmas that are concerned with path graphs and directed path graphs.

A clique is a *clique separator* if its removal disconnects the graph in at least two connected components. A graph with no clique separators is called *atom*, (for example, every cycle has no clique separators). In [102] it is proved that an atom is a path graph and/or a directed path graph if and only if it is a chordal graph; moreover, every chordal graph that is an atom has at most two cliques.

Given a clique separator C of a graph G let $G-C$ have s connected components, $s \geq 2$ with vertex-sets $V_1, \ldots, V_s$, respectively. We define $\gamma_i = G[V_i \cup C]$, $i = 1, \ldots, s$ and $\Gamma_C = \{\gamma_1, \ldots, \gamma_s\}$. A clique K of a subgraph γ of Γ_C is called a *relevant clique*, if $K \cap C \neq \emptyset$ and $K \neq C$. A *neighboring subgraph* of a vertex $v \in V(G)$ is a member $\gamma \in \Gamma_C$ such that v belongs to some relevant clique K of γ. For instance, in Figure 3.7 referring to the graph on the left, all the γ_i's but γ_1 are neighboring subgraphs of vertex 4 belonging to clique separator C, while all the γ_i's but γ_4 and γ_5 are neighboring subgraphs of vertex 3. We say that two subgraphs γ and γ' are *neighbouring* if they are neighbouring subgraphs of some vertex $v \in C$; a subset $W \subseteq \Gamma_C$ whose elements are neighbouring subgraphs will be referred to as a *neighbouring set* (e.g, *neighbouring pairs*, *neighbouring triples* etc). In [102] the following binary relations on Γ_C are defined.

- *Attachedness*, denoted by $\bowtie$ and defined by $\gamma \bowtie \gamma'$ if and only if there is a relevant clique K of γ and a relevant clique K' of γ' such that $K \cap K' \cap C \neq \emptyset$. In particular, γ and γ' are neighboring subgraphs of each vertex $v \in K \cap K' \cap C$.

- *Dominance*, denoted by $\leq$ and defined by $\gamma \leq \gamma'$ if and only if $\gamma \bowtie \gamma'$ and for each relevant clique K' of γ' either $K \cap C \subseteq K' \cap C$ for each relevant clique K of γ or $K \cap K' \cap C = \emptyset$ for each relevant clique K of γ. In Figure 3.7, graph on the right, pairs of $\leq$-comparable subgraphs of the graph on the left are joined by a dotted edge.

- *Antipodality*, denoted by $\leftrightarrow$ and defined by $\gamma \leftrightarrow \gamma'$ if and only if there are relevant cliques K of γ and K' of γ' such that $K \cap K' \cap C \neq \emptyset$ and $K \cap C$ and $K' \cap C$ are inclusion-wise incomparable. In Figure 3.7, graph on the right, pairs of antipodal subgraphs of the graph on the left are joined by a continuous edge.

Antipodality and dominance relations are disjoint binary relations on Γ_C whose union is the relation $\bowtie$. Therefore ($\gamma \leq \gamma'$, $\gamma' \leq \gamma$ or $\gamma \leftrightarrow \gamma'$) if and only if ($\gamma \bowtie \gamma'$). Both $\bowtie$ and $\leftrightarrow$ are symmetric and only $\leq$ is irreflexive. Hence, after neglecting reflexive pairs, $(\Gamma_C, \leftrightarrow)$, $(\Gamma_C, \bowtie)$ are simple undirected graphs on Γ_C referred to as, respectively, the *C-antipodality*, and the *C-attachedness graph of G*. The edges of the C-antipodality graph of G are called *antipodal edges* while those edges of the C-attachedness graph of G which are not antipodal edges, are called *dominance edges*. The *C-dominance* of G is the graph on Γ_C having as edges the dominance edges (i.e., the complement of $(\Gamma_C, \leftrightarrow)$ in $(\Gamma_C, \bowtie)$). Hence the edge-sets of the C-antipodality and the C-dominance graphs of G partition the edge-set of the C-attachedness graph of G and the latter is naturally 2-edge colored by the antipodality edges and by the dominance edges. We

adopt the pictorial convention to represent antipodality edges by continuous lines and dominance edges by dotted lines.

To better understand these relations we use Figure 3.7, that shows a clique separator C of G of Figure 3.5 and its connected components. We stress that $C = \{1, 2, 3, 4, 5\}$, $\Gamma_C = \{\gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5\}$, where $\gamma_1 = \{1, 2, 6, 11, 12\}$, $\gamma_2 = \{2, 3, 4, 7, 13, 14\}$, $\gamma_3 = \{2, 3, 4, 5, 8\}$, $\gamma_4 = \{4, 5, 9, 15\}$ and $\gamma_5 = \{1, 2, 4, 5, 10\}$. Clearly, $\gamma_1, \ldots, \gamma_5$ are path graphs. It holds that $\gamma_5 \geq \gamma_1$, $\gamma_5 \geq \gamma_4$, $\gamma_3 \geq \gamma_2$, $\gamma_3 \geq \gamma_4$, $\gamma_1 \leftrightarrow \gamma_2$, $\gamma_1 \leftrightarrow \gamma_3$, $\gamma_3 \leftrightarrow \gamma_5$, $\gamma_2 \leftrightarrow \gamma_5$, $\gamma_2 \leftrightarrow \gamma_4$, and the relation $\bowtie$ follows from above. These relations are depicted in Figure 3.7 on the right.

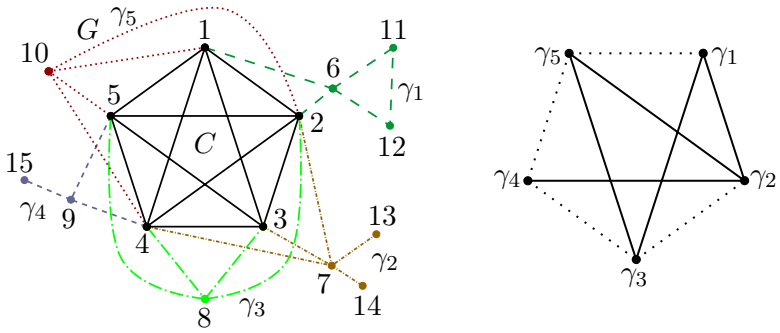


Figure 3.7: on the left a clique separator C of G in Figure 3.5, the connected components $\gamma_1, \ldots, \gamma_5$ are highlighted by different colors and hatchings. On the right relations $\leq$ and $\leftrightarrow$ on connected components $\gamma_1, \ldots, \gamma_5$ are depicted with dotted and continuous lines, respectively.

Now we can report the main result in [102] about path graphs.

Theorem 9 ([102]). *A chordal graph G is a path graph if and only if G is an atom or for a clique separator C each graph $\gamma \in \Gamma_C$ is a path graph and there exists $f : \Gamma_C \to [s]$ such that:*

- *if $\gamma \leftrightarrow \gamma'$, then $f(\gamma) \neq f(\gamma')$;*
- *if γ, γ' and γ'' are neighboring subgraphs of v, for some vertex $v \in C$, then $|f(\{\gamma, \gamma', \gamma''\})| \leq 2$.*

Let us explain the conditions of Theorem 9 in few words. Let T be a clique path tree of G. The removal of clique separator C from G disconnects G in more connected components, but it also disconnects tree T in more subtrees. In a way, the coloring f associates a connected components to the subtrees. The first condition implies that two antipodal connected components γ and γ' need to be in two distinct subtrees, indeed, if not, then for some $v \in (V(\gamma) \cap C) \setminus V(\gamma')$ or $v \in (V(\gamma') \cap C) \setminus V(\gamma)$ the set of cliques of G that contains v does not induce a connected path in T. The second condition says that all connected components that contain v need to be in at most two distinct subtrees, indeed, if not, the set of clique of G that contains v does not induce a path in T.

The following theorem also proved in [102] characterizes directed path graphs.

Theorem 10 ([102]). *A chordal graph G is a directed path graph if and only if G is an atom or for a clique separator C each graph $\gamma \in \Gamma_C$ is a path graph and the γ_i's can be 2-colored such that no antipodal pairs have the same color.*

As before, let us comment briefly Theorem 10. Let C be a clique separator, let $v \in C$, and let f be a 2-coloring of Γ_C. Then all the connected components containing v having the same color through f can be totally ordered by $\leq$. Hence it is possible to create a directed path that starts with all the connected components containing v having the first color through f and ends with all the connected components containing v having the second color through f. The directed clique path tree is build by using such paths.

From an algorithmic point of view, recognition algorithms for path graphs and directed path graphs are joined by the result by Chaplick *et al.* [37]. In this article the authors gave an algorithm that, by starting from the clique path tree of a path graph, build the directed clique path tree, if it exists, in linear time. This result is resumed in the following theorem.

Theorem 11 ([37])**.** *If there exists a polynomial algorithm that tests if a graph G is a path graph and returns a clique path tree of G when the answer is "yes", then there exists an algorithm with the same complexity to test if a graph is a directed path graph.*

Theorem 11 implies that algorithms in [36, 127] can be extended to algorithms able to recognize directed path graph with the same time complexity.

Chapter 4

Computing lengths of non-crossing shortest paths

We deal with the union of non-crossing shortest paths in undirected plane graphs. We introduce the concept of *shortcuts* that allows us to establish whether a path is a shortest path by checking local properties on faces of the graph. By using shortcuts we can compute the length of each shortest path given their union in total linear time, and we can list each shortest path p in $O(\max\{\ell, \ell \log\log \frac{k}{\ell}\})$, where ℓ is the number of edges in p and k the number of shortest paths.

4.1 Introduction

The input of our problem is an undirected plane graph U composed by the union of k non-crossing shortest paths in a plane graph G whose extremal vertices lie on the same face of G. Thus $U = \bigcup_{i\in[k]} p_i$, where, for each $i \in [k]$, p_i is a shortest x_iy_i path in G, x_i and y_i are on the infinite face f^∞ of G, p_i and p_j are non-crossing for every $i, j \in [k]$; hence, we assume that the terminal pairs $\{(x_i, y_i)\}_{i\in[k]}$ are well-formed because of the discussion in Section 3.3. We stress that we know U but we ignore the p_i's, indeed, algorithms solving the NCSP problem in [130, 131] (and also our algorithm for the unweighted case in Chapter 5) compute the union of the non-crossing shortest paths without listing every single path.

We want to compute the lengths of the p_i's shortest paths in linear time, i.e., in time proportional to $|E(U)|$. We also explain an efficient way to list each path. This method will be strictly used in Chapter 7 in the proof of time complexity. We stress that the covering problem solved in Chapter 7 allows us to compute all paths' lengths in total linear time, but it is more complex than the approach shown in this chapter and it requires the results about LCA queries by Gabow and Tarjan in [54].

In this way we prove that if there exists an algorithm able to compute the union of non-crossing shortest paths whose extremal vertices lie on the same face of an undirected plane

graph, then we can compute the lengths of these paths in the same time complexity.

The algorithm we propose can be easily implemented and does not require sophisticated data structures. We follow the same approach by Polishchuck and Mitchell [119], that was inspired by Papadopoulou's work [114]. Their article solves the problem of finding k non-crossing shortest paths in a polygon with n vertices, where distances are defined according to the Euclidean metric.

Our algorithm builds a set of single-touch paths even if the shortest p_i's paths in G composing the input graph $U = \bigcup_{i\in[k]} p_i$ are not pairwise single-touch. This may happen if there are more shortest paths in G joining the same pair of vertices. Uniqueness of shortest paths can be easily ensured by introducing small perturbations in the weight function of G. We wish to point out that the technique we describe in this chapter does not rely on perturbation, but we break ties by choosing rightmost or leftmost paths. This implies that our results can also be used in the unweighted case, as done in Chapter 5. Note that the single-touch property does not depend on the embedding, and if the terminal-pairs are well-formed, then it implies the non-crossing property. This is explained in the following remark, and, for this reason, we can say that the solution found by our algorithm holds for any feasible planar embedding of the graph.

Remark 12. *If $\{\pi_i\}_{i\in[k]}$ is a set of simple single-touch paths, where π_i is an i-path, for $i \in [k]$, then $\{\pi_i\}_{i\in[k]}$ is a set of pairwise non-crossing paths for all the embeddings of U such that the terminal pairs $\{(x_i, y_i)\}_{i\in[k]}$ are well-formed.*

Our approach. We exploit the novel concept of *shortcuts*, that are portions of the boundary of a face that allow us to modify a path without increasing (and possibly decreasing) its length. We show that it is possible to establish whether a path is a shortest path by looking at the presence of shortcuts. Hence while being a shortest path is a global property, we can check it locally by checking a single face at a time for the presence of shortcuts adjacent to the path, ignoring the rest of the graph. Note that this is only possible when the input graph is the union of non-crossing shortest paths, not for general plane graphs. Without this property, finding one distance is as difficult as finding a shortest path in U.

Using shortcuts, we can introduce algorithm `ImplicitPaths` that computes an implicit representation of non-crossing shortest paths. This implicit representation is necessary to compute all distances between terminal pairs in linear time and to solve the problem of listing the edges of a single shortest path. This problem was already discussed in the geometrical case [114, 119] with Euclidean distances, but it is new in non-crossing paths in a weighted plane graph which have a more general structure.

4.2 Shortcuts

Roughly speaking, a shortcut appears if there exists a face f adjacent to a path p so that we can modify p going around f without increasing its length. We show that we can decide whether a path is a shortest path by looking at the existence of shortcuts: in this way, we can check a global property of a path p—i.e., being a shortest path—by checking a local property—i.e., the presence of shortcuts in faces adjacent to p. This result is not true for general plane graphs, but

it only holds when the input graph is the union of shortest paths joining well-formed terminal pairs on the same face. Shortcuts are the main tool of algorithm `ImplicitPaths` described in Section 4.3, and the most important theoretical novelty of this chapter.

We introduce the operator $\rtimes$, explained in Figure 4.1, that allows us to replace a subpath in a path and is used to define the shortcuts.

Definition 13. *Let p be a simple ab path, let $u, v \in V(p)$ be such that a, u, v, b appear in this order in p and let q be a uv path. We denote by $p \rtimes q$ the (possibly not simple) path $p[a, u] \circ q \circ p[v, b]$.*

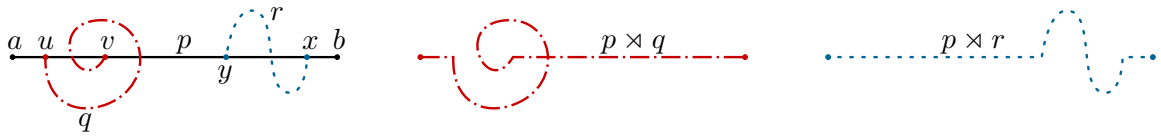


Figure 4.1: illustrating operator $\rtimes$.

Now we can formally define shortcuts, which are clarified in Figure 4.2. The main application of shortcuts is stated in Theorem 15. W.l.o.g., we assume that U is connected, otherwise it suffices to work on each connected component.

Definition 14. *Given a path p and a face f containing two vertices $u, v \in p$, we say that a uv subpath q of ∂f not contained in p is a* shortcut for p *if $\omega(p \rtimes q) \leq \omega(p)$.*

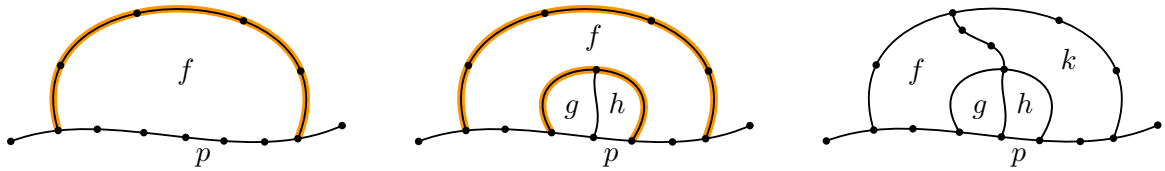


Figure 4.2: all edges have unit weight. On the left, highlighted in orange, there is a shortcut for p contained in ∂f. In the middle there are two shortcuts for p both contained in ∂f. On the right there are no shortcuts for p.

Theorem 15. *Let λ be an i-path, for some $i \in [k]$. If there are no shortcuts for λ, then λ is a shortest i-path.*

Proof. If $\lambda = p_i$ then the thesis holds. Thus let us assume by contradiction that $\omega(p_i) < \omega(\lambda)$ and λ has no shortcuts.

Let $a, b \in V(\lambda) \cap V(p_i)$ be two vertices such that $p_i[a, b]$ and $\lambda[b, a]$ share no edges (such a and b exist because $p_i \neq \lambda$ and they are both i-path). Let C be the simple cycle $p_i[a, b] \circ \lambda[b, a]$, and let R be the region bounded by C. If R is a face of U, then $p_i[a, b]$ is a shortcut for λ, absurdum. Hence we assume that there exist edges in $\mathring{R}$, see Figure 4.3 on the left.

Either $R \subseteq \mathrm{Int}_{p_i}$ or $R \subseteq \mathrm{Ext}_{p_i}$. W.l.o.g., we assume that $R \subseteq \mathrm{Int}_{p_i}$. Being $U = \bigcup_{j \in [k]} p_j$, for every edge $e \in \mathring{R}$ there exists at least one path $q \in P$ such that $e \in q$. Moreover, the extremal vertices of q are in γ_i because paths in P are non-crossing and $R \subseteq \mathrm{Int}_{p_i}$.

Now we show by construction that there exist a path $p \in P$ and a face f such that $f \subseteq R$, ∂f intersects λ on vertices and $\partial f \setminus \lambda \subseteq p$; thus $\partial f \setminus \lambda$ is a shortcut for λ because p is a shortest path.

For all $q \in P$ such that $q \subseteq \mathrm{Int}_{p_i}$ we assume that $\mathrm{Int}_q \subseteq \mathrm{Int}_{p_i}$ (if it is not true, then it suffices to switch the extremal vertices of q).

For each $q \subseteq \mathrm{Int}_{p_i}$, let $F_q = \{f \in F \mid \partial f \subseteq R \cap \mathrm{Int}_q\}$, where F is the set of the faces of U. To complete the proof, we have to find a path p such that $|F_p| = 1$, indeed, the unique face f in F_p satisfies $\partial f \setminus \lambda \subseteq p$, and thus $\partial f \setminus \lambda$ is a shortcut for λ.

Now, let $e_1 \in \mathring{R}$ and let $q_1 \in P$ be such that $e_1 \in q_1$. Being $e_1 \in \mathring{R}$, then $|F_{q_1}| < |F_{p_i}|$ and $|F_{q_1}| > 0$ because $e_1 \in q_1$, see Figure 4.3 on the right. If $|F_{q_1}| = 1$, the proof is completed, otherwise we choose $e_2 \in \mathring{R} \cap \mathring{\mathrm{Int}}_{q_1}$ and $q_2 \in P$ such that $e_2 \in q_2$. It holds that $|F_{q_2}| < |F_{q_1}|$ and $|F_{q_2}| > 0$ because $e_2 \in q_2$. By repeating this reasoning, and being $U = \bigcup_{j \in [k]} p_j$, we find a path p such that $|F_p| = 1$. □

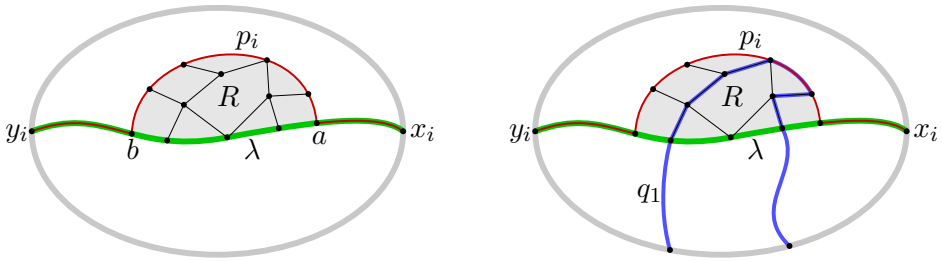


Figure 4.3: paths and regions used in Theorem 15's proof. Path λ is in green, p_i in red, q_1 in blue and region R is highlighted in grey. It holds that $|F_{q_1}| = 4$.

Given a path p, we say that a path q is a *right shortcut for* p if q is a shortcut for p and $q \subseteq \mathrm{Ext}_p$. The following corollary can be proved by the same approach of Theorem 15 and is more useful for our purposes.

Corollary 16. *Let λ be an i-path, for some $i \in [k]$. If there are no right shortcuts for λ, then there does not exist any path $\lambda' \subseteq Ext_\lambda$ satisfying $\omega(\lambda') \leq \omega(\lambda)$.*

4.3 Computing lengths in linear time

In Theorem 19 we show that the distances between terminal pairs can be computed in $O(|E(U)|)$ time by knowing U. This is the main result of this chapter. To achieve it, we introduce algorithm `ImplicitPaths`, that gives us an implicit representation of non-crossing shortest paths used in the proof of Theorem 19. The implicit representation is described in Remark 20.

The main idea behind algorithm `ImplicitPaths` is the following. We build a set of shortest i-paths $\{\lambda_i\}_{i \in [k]}$, by finding λ_i at iteration i, where the terminal pairs are numbered according to a postorder visit of T_g. In particular, at iteration i we find the rightmost shortest i-path in $U_i = \bigcap_{j \in [i-1]} \mathrm{Ext}_{\lambda_j}$, in the following way: first we set λ_i as the leftmost i-path in U_i, then we update λ_i by moving right through right shortcuts (the order in which shortcuts are chosen is

not relevant). When λ_i has no more right shortcuts, then it is the rightmost shortest i-path in U_i by Corollary 16.

Algorithm `ImplicitPaths`:

Input: an undirected plane graph U composed by the union of k non-crossing shortest paths in a plane graph G each one joining a terminal pair on the infinite face of G

Output: an implicit representation of a set of paths $\{\lambda_1, \ldots, \lambda_k\}$, where λ_i is a shortest i-path, for $i \in [k]$

1 Compute T_g and renumber the terminal pairs according to a postorder visit of T_g;
2 **for** $i = 1, \ldots, k$ **do**
3 Let λ_i be the leftmost i-path in $U_i = \bigcap_{j \in [i-1]} \mathrm{Ext}_{\lambda_j}$;
4 **while** *there exists a right shortcut* τ *for* λ_i *in* U_i **do**
5 $\lambda_i := \lambda_i \rtimes \tau$;

Lemma 17. *Let* $\{\lambda_i\}_{i \in [k]}$ *be the set of paths computed by algorithm* `ImplicitPaths`*. Then*

17.(1) λ_i *is the rightmost shortest* i*-path in* U_i*, for* $i \in [k]$*,*

17.(2) $\{\lambda_i\}_{i \in [k]}$ *is a set of single-touch paths.*

Proof. We proceed by induction to prove the first statement. Trivially λ_1 is the rightmost shortest 1-path in U_1 because of Corollary 16. Let us assume that λ_j is the rightmost shortest j-path in U_j, for $j \in [i-1]$, we have to prove that λ_i is the rightmost shortest i-path in U_i.

In Line 3, we initialize λ_i as the leftmost i-path in U_i. By induction and the postorder visit, at this step, there does not exist in U_i any i-path p to the left of λ_i shorter than λ_i. Otherwise λ_i would cross a path λ_j, for some $j < i$, implying that λ_j is not a shortest j-path. We conclude by the while cycle in Line 4 and Corollary 16.

Statement 17.(2) follows from 17.(1); indeed, if λ_i and λ_j are not single-touch, for some $i, j \in [k]$, then 17.(1) is denied either for λ_i or for λ_j. □

Given $i \in [k]$ we define C_i as the set of children of i in the genealogy tree, moreover, we say that λ_j is a *child of* λ_i if $j \in C_i$.

Before stating our main result, we introduce a trivial consequence of non-crossing property and Jordan's Curve Theorem [83], indeed, every i-path π satisfies that $\pi \circ \gamma_i$ is a closed curve.

Remark 18. *Let* $\{\pi_i\}_{i \in [k]}$ *be a set of non-crossing* i*-paths. Let* $i, j, \ell \in [k]$*. If* π_i*,* π_j *and* π_ℓ *share a common edge, then at least two among* $\{i, j, \ell\}$ *are a couple of ancestor/descendant in the genealogy tree.*

Theorem 19. *Given an undirected plane graph* U *composed by the union of* k *non-crossing shortest paths in a plane graph* G *each one joining a terminal pair on the infinite face of* G*, we can compute the length of each shortest path in* $O(|E(U)|)$ *total time.*

Proof. We show that during the execution of algorithm `ImplicitPaths` we can also compute the length of λ_i, for all $i \in [k]$, in linear total time. If we also prove that algorithm `ImplicitPaths` can be executed in linear time, then the thesis follows from Lemma 17.

We define, for all $i \in [k]$, $\lambda_{i,0}$ as λ_i after Line 3, i.e., the leftmost i-path in $U_i = \bigcap_{j\in[i-1]} \mathrm{Ext}_{\lambda_j}$. We have to show that all the $\lambda_{i,0}$'s and all the shortcuts required in Line 4 can be computed in total linear time.

Let's start dealing with the shortcuts. We do a linear preprocessing that visits clockwise every face. Let f be a face of U: after this preprocessing, the lengths of the clockwise path and of the counterclockwise path in ∂f joining any given pair of vertices in f can both be computed in constant time. Now, if the intersection between ∂f and λ_i, for some $i \in [k]$, is contained in λ_j, for some $j \in C_i$, then we know that there are no right shortcuts in f for λ_i, otherwise they would be right shortcuts for λ_j. Thus we ask for a right shortcut in f for λ_i if and only if λ_i visits at least one edge in ∂f that is not contained in its children or $\lambda_i \cap \partial f$ is contained in an least two children of λ_i (consequently at least one more edge of ∂f is visited). In this way, during the execution of algorithm `ImplicitPaths` we ask for a shortcut in f at most $O(|E(f)|)$ times thank also to Remark 18. This implies that finding all the shortcuts requires total linear time.

Now we prove that all the $\lambda_{i,0}$'s can be computed in total linear time. We stress that all the λ_i's and all the $\lambda_{i,0}$'s are represented as list, in this way we can join two paths in constant time. We recall that $\lambda_{i,0}$ is the leftmost i-path in $U_i = \bigcap_{j\in[i-1]} \mathrm{Ext}_{\lambda_j}$, C_i is the set of children of i, and γ_i is the clockwise path on the infinite face of G from x_i to y_i. Let $Y_i = \bigcup_{j\in C_i} \lambda_j$ and let f_i be the infinite face of $Y_i \cup \gamma_i$. We observe that, by its definition, $\lambda_{i,0}$ is the counterclockwise i-path on f_i. Clearly, if all the λ_j's, for $j \in C_i$, are vertex disjoint, then λ_j is contained in f_i, for all $j \in C_i$. If the λ_j's are not vertex disjoint, then some edges of Y_i are not in f_i, and by construction, they are not in f_ℓ, for all $i \preceq \ell$. Thus we can see the sequence of the f_i's as an updating graph for which if an edge is deleted at iteration i, then it does not appear again in f_ℓ for all $i \preceq \ell$. Hence, thanks to Remark 18, every edge appears at most two times in this construction. Consequently, all the $\lambda_{i,0}$'s can be computed in total linear time because also to their list representation.

We have proved that algorithm `ImplicitPaths` requires linear time. We use the same argument to compute paths' lengths. Let $i \in [k]$ and $j \in C_i$. At iteration i we know $\omega(\lambda_j)$, and we compute $\omega(\lambda_{i,0} \cap \lambda_j)$ by subtracting from $\omega(\lambda_j)$ the length of edges of λ_j that are not in $\lambda_{i,0}$. In this way we can compute the lengths of $\lambda_{i,0}$ for all $i \in [k]$ in total linear time because every edge is considered at most two times thanks to Remark 18. Being the shortcuts computable in linear time, the thesis follows. □

By following Theorem 19's proof we obtain the following implicit representation of the λ_i's.

Remark 20. *Paths λ_i's computed by algorithm algorithm `ImplicitPaths` are implicitly represented as follows: $\lambda_i = q_1 \circ \lambda_{j_1}[a_1, b_1] \circ q_2 \circ \lambda_{j_2}[a_2, b_2] \circ \ldots \circ q_r \circ \lambda_{j_r}[a_r, b_r] \circ q_{r+1}$ where $\{j_1, j_2, \ldots, j_r\} \subseteq C_i$ and $E(q_\ell \cap \lambda_z) = \emptyset$ for all $\ell \in [r+1]$ and $z \in C_i$. Note that the q_ℓ's can be empty.*

Now we explain the implicit representation of the λ_i's. If i is a leaf of the genealogy tree, then λ_i is given explicitly. Otherwise we give explicitly edges that do not belong to the children

of λ_i, that is the q_i's paths, and we give the intersection path between λ_i and one of its child by specifying the extremal vertices of this intersection. This representation requires linear space thanks to Remark 18.

4.4 Listing paths

We study the problem of listing the edges in λ_i, for some $i \in [k]$, after the execution of algorithm `ImplicitPaths`. We want to underline the importance of the single-touch property. In Figure 4.4, in (a) four shortest paths are drawn (the graph is unit-weighted), we observe that the single-touch property is clearly not satisfied. A single-touch version of the previous four paths is drawn in (b); it can be obtained by algorithm `ImplicitPaths`. It is clear that the problem of listing the edges in a path in this second case is easier. We stress that in the general case the union of a set of single-touch paths can form cycles, see Figure 2.1 for an example.

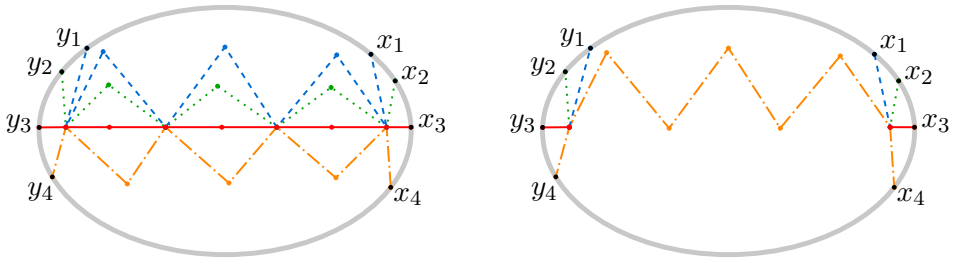


Figure 4.4: (a) the union of shortest i-paths, for $i \in [4]$, in unit-weighted graph, each different path has different style, (b) the union of $\{\lambda_i\}_{i\in[4]}$, the output paths of algorithm `ImplicitPaths`.

Theorem 21. *After $O(n)$ time preprocessing, each shortest path λ_i, for $i \in [k]$, can be listed in $O(\max\{\ell_i, \ell_i \log\log(\frac{k}{\ell_i})\})$ time, where ℓ_i is the number of edges of λ_i.*

Proof. For any $i \in [k]$, we denote by $\overrightarrow{\lambda_i}$ the oriented version of λ_i from x_i to y_i. During the execution of algorithm `ImplicitPaths`, we introduce a function `Mark` that marks a dart d with i if and only if the d is used for the first time in the execution of algorithm `ImplicitPaths` at iteration i. It means that $\texttt{Mark}(d) = i$ if and only if d belongs to $\overrightarrow{\lambda_i}$ and d does not belong to $\overrightarrow{\lambda_j}$, for all $j \preceq i$ and $j \neq i$. This function can be executed within the same time bound of algorithm `ImplicitPaths`. Now we explain how to find darts in $\overrightarrow{\lambda_i}$.

Let us assume that $(d_1, \ldots, d_{\ell_i})$ is the ordered sequence of darts in $\overrightarrow{\lambda_i}$. Let $v = \text{head}[d_{j-1}]$, and let us assume that $deg(v) = r$ in the graph $\bigcup_{j\in[k]} \lambda_j$. We claim that if we know d_{j-1}, then we find d_j in $O(\log\log r)$ time. First we order the outgoing darts in v in clockwise order starting in d_{j-1}, thus let $\text{Out}_v = (g_1, \ldots, g_r)$ be this ordered set (this order is given by the embedding of the input plane graph). We observe that all darts in Out_v that are in Int_{λ_i} are in $\overrightarrow{\lambda_w}$ for some $w \leq i$, thus $\texttt{Mark}(d) \leq i$ for all $d \in \text{Out}_v \cap \text{Int}_{\lambda_i}$. Similarly, all darts in Out_v that are Ext_{λ_i} are in $\overrightarrow{\lambda_z}$ for some $z \geq i$, thus $\texttt{Mark}(d) \geq i$ for all $d \in \text{Out}_v \cap \text{Ext}_{\lambda_i}$. Using this observation, we

have to find the unique $l \in [r]$ such that $\texttt{Mark}(g_l) \leq i$ and $\texttt{Mark}(g_{l+1}) > i$. This can be done in $O(\log \log r)$ by using a van Emde Boas tree [135].

Being the $\overrightarrow{\lambda_i}$'s pairwise single-touch, then $\sum_{v \in V(\lambda_i)} deg(v) \leq 2k$, where the equality holds if and only if every $\overrightarrow{\lambda_j}$, for $j \neq i$, intersects on vertices $\overrightarrow{\lambda_i}$ exactly two times, that is the maximum allowed by the single-touch property.

Finally, if $2k \leq \ell_i$, then we list $\overrightarrow{\lambda_i}$ in $O(\ell_i)$ because the searches of the correct darts do not require more than $O(k)$ time, otherwise we note that

$$\sum_{\substack{j=1,\dots,\ell \\ a_1+\dots+a_\ell \leq 2k}} \log \log a_j \leq \ell \log \log \left(\frac{2k}{\ell} \right),$$

so the time complexity follows. □

Chapter 5

Solving the NCSP problem in the unweighted case in linear time

We solve the non-crossing shortest paths problem on undirected unweighted plane graphs in linear time. We also give a novel concept of *incremental shortest path* subgraph of a plane graph, i.e., a partition of the planar embedding in subregions that preserve distances, that can be of interest itself.

5.1 Introduction

In this chapter, we solve the NCSP problem on an undirected unweighted plane graph in $O(n)$ time. We improve, in the unweighted case, the results in [130, 131]. By applying the technique described in Chapter 4 we can compute distances between all terminal pairs in linear time.

Our algorithm relies on two main results:

- an algorithm due to Eisenstat and Klein [48], that gives in $O(n)$ time an implicit representation of a sequence of shortest path trees in an undirected unweighted plane graph G, where each tree is rooted at a vertex in the infinite face of G. Note that, if we want to compute shortest paths from the implicit representation of shortest path trees given in [48], then we spend $\Theta(kn)$ time; this happens when all k shortest paths share a subpath of $\Theta(n)$ edges;
- the novel concept of *incremental shortest paths (ISP) subgraph* of a plane graph G. We show that an ISP subgraph of G partitions the embedding of G into *distance preserving* regions, i.e., for any two vertices a, b in G lying in the same region R it is always possible to find a shortest path in G joining a and b that is contained in R.

Improved results. We specialize the NCSP problem discussed in [130, 131] to the unweighted case, decreasing the time complexity from $O(n \log\log k)$ to $O(n)$ (for every k). Therefore, in the case of unweighted plane graphs we improve the results in [49, 93].

Erickson and Nayyeri [49] generalized the work in [131] to the case in which the k terminal pairs lie on h face boundaries. They prove that k non-crossing paths, if they exist, can be found in $2^{O(h^2)}n \log k$ time, that becomes $2^{O(h^2)}n \log\log k$ time by plugging in the algorithm in [130]. Applying our results, if the graph is unweighted, then the time complexity decreases to $2^{O(h^2)}n$.

Kusakari *et al.* [93] showed that a set of non-crossing forests in a plane graph can be found in $O(n \log n)$ time, where two forest F_1 and F_2 are *non-crossing* if for any pair of paths $p_1 \subseteq F_1$ and $p_2 \subseteq F_2$, p_1 and p_2 are non-crossing. With our results, if the graph is unweighted, then the time complexity becomes linear.

Our approach. We introduce a new class of graphs, ISP subgraphs, that partition a plane graph into regions that preserve distances. Our algorithm is split in two parts.

In the first part we use Eisenstat and Klein's algorithm that gives a sequence of shortest path trees rooted at the vertices in the infinite face. We choose some specific shortest paths from each tree to obtain a sequence of ISP subgraphs $X_1, \dots X_k$. By using the distance preserving property of regions generated by ISP subgraphs, we prove that X_i contains a shortest $x_i y_i$ path, for all $i \in \{1, \dots, k\}$.

In the second part of our algorithm, we extract from each X_i a shortest $x_i y_i$ path and we obtain a set of non-crossing shortest paths that is our goal. In this part we strongly use the partial order given by the genealogy tree.

5.2 ISP subgraphs

In this section we introduce the concept of *incremental shortest paths (ISP) subgraph* of a plane graph G, that is a subgraph incrementally built by adding a sequence of shortest paths in G starting from f^∞ (see Definition 22). The interest towards ISP subgraphs is due to the fact that for any two vertices a, b in G lying on the same face f of the ISP subgraph there is always a shortest path in G joining a and b contained in f (boundary included). All the results of this section hold for positive weighted graphs, where the length of a path is the sum of edge weights instead of the number of edges.

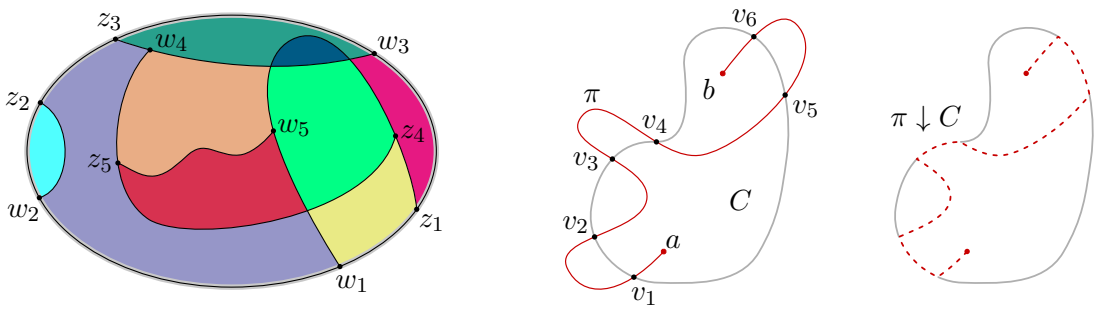
This is the main novel result of this chapter, that allows us to prove that, in order to build the union of shortest paths joining terminal pairs, we can start from the union of some of the shortest paths computed by the algorithm in [48].

Definition 22. *A graph X is an* incremental shortest paths (ISP) subgraph *of an undirected positive weighted plane graph G if $X = X_r$, where X_1, X_2, ..., X_r is a sequence of subgraphs of G built in the following way: $X_1 = f^\infty$ and $X_i = X_{i-1} \cup q_i$, where q_i is a shortest $w_i z_i$ path in G with $w_i, z_i \in X_{i-1}$.*

We define now operator $\downarrow$, that given a path π and a cycle C, in case π crosses C, replaces some subpaths of π with some portions of C, as depicted in Figure 5.1(b). We observe that $\pi \downarrow \partial f$ could be not a simple path even if π is.

Definition 23. *Let C be a cycle in an undirected positive weighted plane graph G. Let a, b be two vertices in R_C and let π be a simple ab path. In case $\pi \subseteq R_C$ we define $\pi \downarrow C = \pi$. Otherwise, let $(v_1, v_2, \ldots, v_{2r})$ be the ordered subset of vertices of π that satisfies the following: $\pi[a, v_1] \subseteq R_C$, $\pi[v_{2r}, b] \subseteq R_C$, $\pi[v_{2i-1}, v_{2i}]$ and R_C have no common edges and $\pi[v_{2i}, v_{2i-1}] \subseteq R_C$, for all $i \in [r]$. For each $i \in [r]$, let μ_i be the $v_{2i-1}v_{2i}$ path on C such that the region bounded by $\mu_i \circ \pi[v_{2i-1}, v_{2i}]$ does not contain R_C. We define $\pi \downarrow C = \pi[a, v_1] \circ \mu_1 \circ \pi[v_2, v_3] \circ \mu_2 \ldots \circ \pi[v_{2r-2}, v_{2r-1}] \circ \mu_r \circ \pi[v_{2r}, b]$.*

Definition 22 and Definition 23 are depicted in Figure 5.1.



(a) (b)

Figure 5.1: (a) an ISP subgraph X of G; extremal vertices w_i, z_i of q_i are drawn, for $i \in [5]$. Different faces of X have different colors. An example of Definition 23 is given in (b).

Note that every face f of an ISP subgraph X of G induces the region R_{∂_f} in G, and this region may contain vertices in G that are not in X. In the following theorem we show that, given any face f of an ISP subgraph X of G, every path π in G whose extremal vertices are in $R_{\partial f}$ is not shorter than $\pi \downarrow \partial f$.

Theorem 24. *Let X be an ISP subgraph of an undirected positive weighted plane graph G. Let f be any face of X, and let a, b be two distinct vertices in $R_{\partial f}$. For any ab path π we have $\omega(\pi \downarrow \partial f) \leq \omega(\pi)$.*

Proof. Let $\{X_i\}_{i \in [r]}$ be the sequence of ISP subgraphs such that $X = X_r$, and let q_i be the path that builds X_i from X_{i-1}. We assume that q_i has no vertices in X_{i-1} other than its endpoints w_i and z_i, otherwise we can split q_i on intersections with X_{i-1} and repeatedly apply the same proof to each portion of q_i. We prove the thesis by induction on r for every choice of a face f of X_r, $a, b \in R_{\partial f}$ and ab path π.

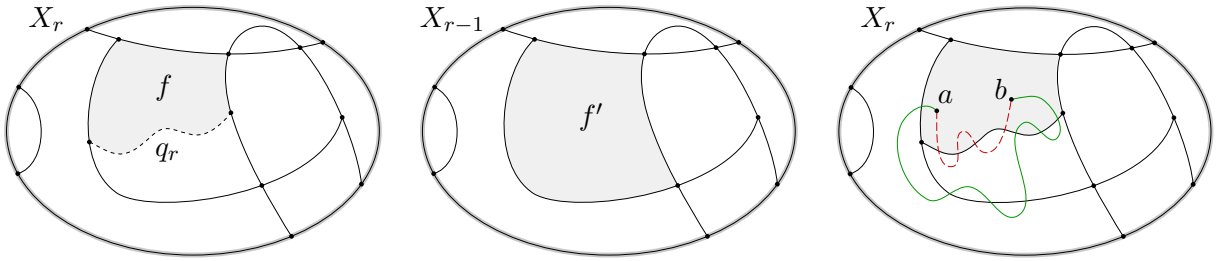
In the base case, where $r = 1$, X_r is equal to f^∞ by Definition 22, thus for every path π we trivially have that $\pi \downarrow \partial f^\infty = \pi$. Hence, $\omega(\pi \downarrow \partial f) = \omega(\pi)$ and the thesis holds. Let us assume that the thesis is true for $r - 1$ and let us prove it for r.

Let f be a face of X_r and let f' be the unique face of X_{r-1} such that $f \subset f'$ (Figure 5.2(a) and Figure 5.2(b) show faces f and f', respectively). Let $a, b \in V(R_{\partial f})$ and let π be an ab path. Three cases may occur:

case $\pi \subseteq R_{\partial f}$: the thesis trivial holds, since $\pi \downarrow \partial f = \pi$;

case $\pi \subseteq R_{\partial f'}$ and $\pi \not\subseteq R_{\partial f}$: since $\pi \subseteq R_{\partial f'}$ and $\pi \not\subseteq R_{\partial f}$, then π crosses q_r an even number of times, thus $\pi \downarrow \partial f$ is not longer than π, since some subpaths of π have been replaced by subpaths of q_r with the same extremal vertices and q_r is a shortest path (see Figure 5.2(c) where π is the red dashed path);

case $\pi \not\subseteq R_{\partial f'}$: since $f \subseteq f'$, it is easy to see that $\pi \downarrow \partial f = (\pi \downarrow \partial f') \downarrow \partial f$. Let us consider $\pi' = \pi \downarrow \partial f'$. By induction, it holds that $\omega(\pi') \leq \omega(\pi)$. We observe now that $\pi' \subseteq R_{\partial f'}$ and $\pi' \not\subseteq R_{\partial f}$, hence the previous case applies, showing that $\omega(\pi' \downarrow \partial f) \leq \omega(\pi')$. Finally, the two previous inequalities imply $\omega(\pi \downarrow \partial f) = \omega((\pi \downarrow \partial f') \downarrow \partial f) = \omega(\pi' \downarrow \partial f) \leq \omega(\pi') \leq \omega(\pi)$ (see Figure 5.2(c) where π is the green continue path). □



(a) a face f in X_r

(b) a face f' in X_{r-1}

(c) two examples of π in the second case (dashed red) and third case (continuous green)

Figure 5.2: in (a) and (b) faces f and f' build on the ISP graph in Figure 5.1(a). In (c) we depict the second and third case of the proof of Theorem 24.

We can state now the main property of ISP subgraphs.

Corollary 25. *Let X be an ISP subgraph of an undirected positive weighted plane graph G and let f be a face of X. For every vertices $a, b \in R_{\partial f}$ there exists a shortest ab path of G contained in $R_{\partial f}$.*

5.3 Our algorithm

We summarize in Subsection 5.3.1 the result of Eisenstat and Klein's article [48], that deals with the multiple-source shortest paths problem. For the sake of clarity, we split our algorithm in two parts:

- in Subsection 5.3.2 we present algorithm `NCSPsupergraph`, that builds a sequence $\{X_i\}_{i\in[k]}$ of subgraphs of G such that X_k contains a shortest path for each terminal pair, and it possibly contains some extra edges. We anticipate that $X_i \cup f^\infty$ is an ISP subgraph of G, for all $i \in [k]$;
- in Subsection 5.3.3 we describe algorithm `NCSPunion` that, by using the sequence of graphs $\{X_i\}_{i\in[k]}$ found by algorithm `NCSPsupergraph`, builds a directed graph that is exactly the union of the shortest directed paths joining each terminal pair contained in the output of algorithm `NCSPsupergraph`.

5.3.1 Eisenstat and Klein's result

The algorithm in [48] takes as input an undirected unweighted plane graph G, where $v_1, v_2, \ldots, v_r$ is the sequence of vertices in the infinite face of G in clockwise order, and returns an implicit representation of a sequence of shortest path trees $\mathcal{T}_{v_i}$, for $i \in [r]$, where each $\mathcal{T}_{v_i}$ is rooted at v_i.

The sequence of trees $\mathcal{T}_{v_i}$, for $i \in [r]$, is represented by explicitly listing the darts in $\mathcal{T}_{v_1}$, and listing the darts that are added to transform $\mathcal{T}_{v_i}$ into $\mathcal{T}_{v_{i+1}}$, for $1 < i \leq r$ (for each added dart from x to y, the unique dart that goes to y in $\mathcal{T}_{v_i}$ is deleted; with the only two exceptions of the added dart leading to v_i, and the deleted dart leading to v_{i+1}). Hence, the output of their algorithm is $\mathcal{T}_{v_1}$ and a sequence of sets of darts. A key result in [48] shows that if a dart d appears in $\mathcal{T}_{v_{i+1}} \setminus \mathcal{T}_{v_i}$, then d cannot appear in any $\mathcal{T}_{v_{j+1}} \setminus \mathcal{T}_{v_j}$, for $j > i$. Thus the implicit representation of the sequence of shortest path trees has size $O(n)$. This representation can be computed in $O(n)$ time.

5.3.2 Algorithm `NCSPsupergraph`

Algorithm `NCSPsupergraph` builds a sequence $\{X_i\}_{i\in[k]}$ of subgraphs of G by using the sequence of shortest path trees given by Eisenstat and Klein's algorithm. We point out that we are not interested in the shortest path trees rooted at every vertex in f^∞, but we only need the shortest path trees rooted at x_i's. So, we define T_i as the shortest path tree rooted at x_i, for $i \in [k]$, i.e., $T_i = \mathcal{T}_{x_i}$. We denote by $T_i[v]$ the path in T_i from x_i to v.

The algorithm starts by computing the first subgraph X_1, that is just the undirected 1-path in T_1, i.e., $\overline{T_1[y_1]}$ (we recall that all T_i's trees given by algorithm in [48] are rooted directed trees, thus $\overline{T_1}$ is the undirected version of T_1). Then the sequence of subgraphs X_i, for $i = 2, \ldots, k$ is computed by adding some undirected paths extracted from the shortest path trees T_i's defined by Eisenstat and Klein's algorithm.

We define the set $H_i \subseteq V(X_i)$ of vertices h such that at least one dart d is added while transforming T_{i-1} into T_i such that $\text{head}[d] = h$. Hence, H_i is the set of vertices of X_i whose parent in T_i differs from the parent in T_{i-1}. At iteration i, we add path $\overline{T_i[h]}$ to X_i, for each h in H_i.

Lemma 26. *Algorithm `NCSPsupergraph` has $O(n)$ time complexity.*

Proof. Eisenstat and Klein's algorithm requires $O(n)$ time, implying that the H_i's and the T_i's can be found in $O(n)$ time. Algorithm `NCSPsupergraph` visits each edge of G at most $O(1)$ times because it builds X_i without visiting edges in X_{i-1}; indeed, in Line 7, $\overline{T_i[h]}$ can be found by starting in h and by walking backwards on T_i until a vertex of X_i is found. The thesis follows. □

Figure 5.3 shows how algorithm `NCSPsupergraph` builds X_4 starting from X_3. Starting from X_3 in Figure 5.3(a), Figure 5.3(b) shows the darts whose head is in H_4. Consider the unique dart d whose head is the vertex u: we observe that $\overline{d}$ is already in X_3, this happens because $\text{rev}[d] \in T_3[y_3]$. Indeed, it is possible that at iteration i some portions of some undirected paths that we add in Line 7 are already in X_{i-1}. Figure 5.3(c) highlights $\bigcup_{h\in H_4} T_4[h]$ and η_4, while in Figure 5.3(d) X_4 is drawn.

Algorithm `NCSPsupergraph`**:**

Input: an undirected unweighted plane graph G and k well-formed terminal pairs $\{(x_i, y_i)\}_{i\in[k]}$ on the infinite face of G

Output: an undirected graph X_k that contains a set of non-crossing paths $P = \{\pi_1, \ldots, \pi_k\}$, where π_i is a shortest $x_i y_i$ path, for $i \in [k]$

1 Compute a shortest path tree T_1 rooted at x_1;
2 $X_1 = \overline{T_1[y_1]}$;
3 **for** $i = 2, \ldots, k$ **do**
4 $X_i = X_{i-1}$;
5 Compute T_i from T_{i-1} by the algorithm by Eisenstat and Klein [48];
6 Compute the set H_i of vertices of X_i whose parent in T_i differs from the parent in T_{i-1};
7 For all $h \in H_i$, $X_i = X_i \cup \overline{T_i[h]}$;
8 Let η_i be the undirected path on T_i that starts in y_i and walks backwards until a vertex in X_i is reached;
9 $X_i = X_i \cup \eta_i$;

Subgraphs $\{X_i\}_{i\in[k]}$ built by algorithm `NCSPsupergraph`, joined with f^∞, satisfy all the hypothesis of Theorem 24. Indeed, paths added in Line 7 and Line 9 are shortest paths in G joining vertices in X_{i-1}, thus fulfilling Definition 22. So, we exploit Theorem 24 to prove that X_i contains an i-path, for $i \in [k]$, and, in particular, X_k contains a set of non-crossing paths $P = \{\pi_1, \ldots, \pi_k\}$, where π_i is a shortest i-path, for $i \in [k]$. The main idea is to show that X_i contains an undirected path that has the same length as the shortest i-path found by the algorithm by Eisenstat and Klein. This is proved in Theorem 27.

Given a subgraph X of G, we say that an i-path p is the *leftmost i-path in X* if for every i-path $q \subseteq X$ it holds $R_{p\circ\gamma_i} \subseteq R_{q\circ\gamma_i}$.

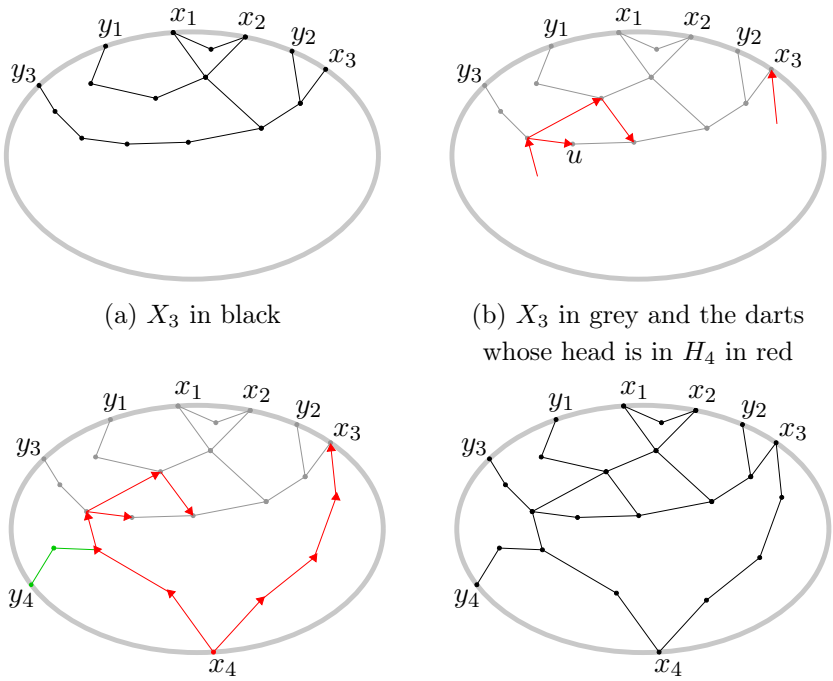
We say that an undirected path p *always turns left* if p chooses the leftmost edge, w.r.t. the fixed embedding, in each vertex going from a to b, where a and b are the extremal vertices of p. Note that the leftmost ab path is not necessarily the path that starts in a and always turns left until b is reached. To better understand the previous definition and the π_i's paths defined in the following theorem, we refer to Figure 5.4, showing paths $\pi_1, \pi_2, \pi_3, \pi_4$ built on graph X_4 in Figure 5.3(d).

Note that the following theorem describes a set of non-crossing shortest paths, but it does not solve our problem. Indeed, if we extract π_i from X_i, for each $i \in [k]$, then we may spend $O(kn)$ time. We show how do it in linear time in the next section.

Theorem 27. *Let π_i be the undirected leftmost i-path in X_i, for $i \in [k]$. The following statements hold:*

27.(1) π_i is the $x_i y_i$ path in X_i that always turns left, for $i \in [k]$,

27.(2) π_i is a shortest i-path, for $i \in [k]$,



(a) X_3 in black

(b) X_3 in grey and the darts whose head is in H_4 in red

(c) $\bigcup_{h \in H_4} T_4[h]$ in red and η_4 in green

(d) X_4 in black

Figure 5.3: how algorithm `NCSPsupergraph` builds graph X_4 starting from X_3.

27.(3) for all $i, j \in [k]$, π_i and π_j are non-crossing.

Proof. We prove all the statements separately.

27.(1) For convenience, for each $i \in [k]$, let λ_i be the undirected path on X_i that starts in x_i and always turns left until it reaches either y_i or a vertex of degree one in X_i; we observe that λ_i is well defined. We have to prove that $\lambda_i = \pi_i$.

Let $i \in [k]$. First, we observe that $x_i \in X_i$ because $x_{i-1} \in H_i$, thus, by Line 7, $\overline{T_i[x_{i-1}]} \subseteq X_i$. This implies $x_i \in X_i$ as we have claimed.

Let u be the extremal vertex of λ_i other than x_i; by construction of λ_i, $u \in f^\infty$. Assume by contradiction that $u \neq y_i$. Two cases are possible: either $u \in V(f^\infty) \setminus V(\gamma_i)$ or $u \in V(\gamma_i) \setminus \{y_i\}$.

The first case cannot occur because Line 7 and Line 9 imply $\overline{T_i[y_i]} \subseteq X_i$, thus λ_i would cross η_i, absurdum. In the second case, let us assume by contradiction that $u \in V(\gamma_i) \setminus \{y_i\}$. Let $d \in \lambda_i$ be the dart such that $\mathrm{head}[d] = u$. By definition of λ_i, vertex u has degree one in X_i. By Line 2, Line 7 and Line 9, all vertices with degree one are equal to either x_ℓ or y_ℓ, for some $\ell \in [k]$, and this implies that there exists $j < i$ such that $u \in \{x_j, y_j\}$. This is absurdum because there is not x_j or y_j in $V(\gamma_i) \setminus \{x_i, y_i\}$ such that $j < i$. Hence λ_i is an i-path, and, by its definition, λ_i is the leftmost i-path in X_i. Therefore $\lambda_i = \pi_i$.

27.(2) We prove that π_i is a shortest i-path by using Theorem 24, indeed, $X_i \cup f^\infty$ is an ISP subgraph of G by construction. Let G' be the graph obtained from G by adding a dummy

path q from x_i to y_i in f^∞ with high length (for example, $|q| = |E(G)|$). Let C be the cycle $\pi_i \circ q$. We observe that $\overline{T_i[y_i]} \downarrow C = \pi_i$ and C is the boundary of a face of G'. Thus, by Theorem 24, $|\pi_i| \leq |\overline{T_i[y_i]}|$. Since $\overline{T_i[y_i]}$ is a shortest path, then π_i is a shortest path in G', hence it also is a shortest path in G.

27.(3) Let us assume by contradiction that there exist $i, j \in [k]$ such that π_i and π_j are crossing, with $i < j$. Thus π_j has not turned always left in X_j, absurdum. □

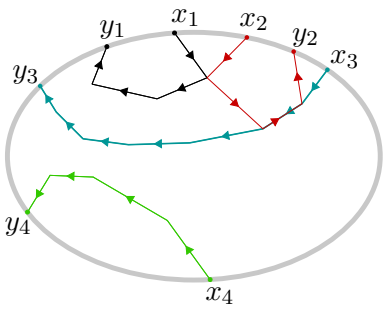


Figure 5.4: paths $\pi_1, \pi_2, \pi_3, \pi_4$ built on graph X_4 in Figure 5.3(d).

5.3.3 Algorithm `NCSPunion`

The graph X_k given by algorithm `NCSPsupergraph` contains a shortest path for each terminal pair, but X_k may also contain edges that do not belong to any shortest path. To overcome this problem we apply algorithm `NCSPunion`, that builds a directed graph $Y_k = \bigcup_{i\in[k]} \rho_i$, where ρ_i is a directed shortest i-path, for $i \in [k]$. Moreover, we prove that Y_k can be built in linear time. This implies that, by using the results in Theorem 19, we can compute the length of all shortest i-paths, for $i \in [k]$, in $O(n)$ time (see Theorem 32).

We use the sequence of subgraphs $\{X_i\}_{i\in[k]}$. By Theorem 27, we know that X_i contains a shortest undirected i-path π_i and we can list its edges in $O(|\pi_i|)$ time. But if an edge e is shared by many π_i's, then e is visited many times. Thus obtaining $\bigcup_{i\in[k]} \pi_i$ by this easy procedure requires $O(kn)$ time. To overcome this problem, we should visit each edge in $\bigcup_{i\in[k]} \pi_i$ only a constant number of times.

Now we introduce two useful lemmas. The first lemma shows that two directed paths λ_i and λ_j that are uncomparable in the genealogy tree T_g (i.e., such that $i \not\preceq j$ and $j \not\preceq i$) cannot share a dart, although it is possible that $\overrightarrow{ab} \in \lambda_i$ and $\overrightarrow{ba} \in \lambda_j$. The second lemma deals with the intersection of non-crossing paths joining comparable pairs.

Lemma 28. *Let λ_i be a shortest directed i-path and let λ_j be a shortest directed j-path, for some $i, j \in [k]$. If j is not an ancestor neither a descendant of i in T_g, then λ_i and λ_j have no common darts.*

Proof. Let us assume by contradiction that λ_i and λ_j have at least one common dart, and let d be the dart in $\lambda_i \cap \lambda_j$ that appears first in λ_i. Let R be the region bounded by $\overline{\lambda_j[x_j, \text{tail}[d]]}$, $\overline{\lambda_i[x_i, \text{tail}[d]]}$ and the clockwise undirected $x_i x_j$ path in f^∞ (Figure 5.5 shows λ_i, λ_j and R).

Being λ_j a simple path, then λ_j crosses λ_i in at least one vertex in $\lambda_i[x_i, \text{tail}[d]]$. Let u be the first vertex in $\lambda_i[x_i, \text{tail}[d]]$ after $\text{head}[d]$ in λ_j.

Now by looking at cycle $\lambda_i[u, \text{head}[d]] \circ \lambda_j[\text{head}[d], u]$ shown in Figure 5.5, we prove that λ_i and λ_j can be both shortest paths. Indeed, if λ_i is a shortest path, then $\lambda_j[x_j, \text{tail}[d]] \circ \lambda_i[\text{tail}[d], u] \circ \lambda_j[u, y_j]$ is shorter than λ_j because it does not contain d. Finally, if λ_j is a shortest path, then $\lambda_i[x_i, u] \circ \lambda_j[u, \text{head}[d]] \circ \lambda_i[\text{head}[d], y_i]$ is shorter than λ_i because it does not contain d. □

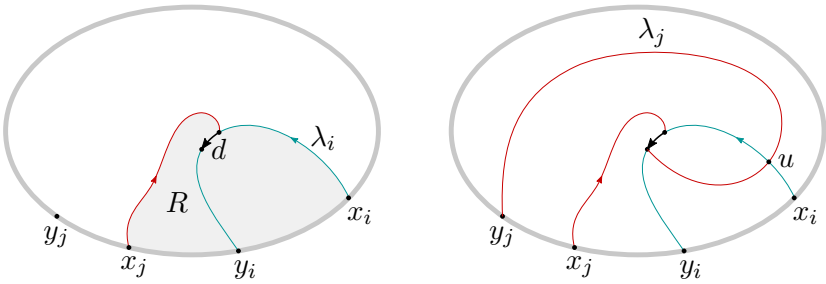


Figure 5.5: paths λ_i and λ_j, dart d, region R and vertex u used in the proof of Lemma 28.

Lemma 29. *Let $\{\lambda_i\}_{i\in[k]}$ be a set of non-crossing directed i-paths. Let $i, j \in [k]$, if $i \preceq j$, then $\lambda_i \cap \lambda_j \subseteq \lambda_\ell$, for all $\ell \in [k]$ such that $i \preceq \ell \preceq j$.*

Proof. Let us assume λ_i and λ_j have at least one common vertex and choose $\ell \in [k]$ such that $i \preceq \ell \preceq j$. Let v be a vertex in $\lambda_i \cap \lambda_j$ and let Q be the region bounded by $\overline{\lambda_j[x_j, v]}$, $\overline{\lambda_i[x_i, v]}$ and the clockwise undirected $x_j x_i$ path in f^∞ (region Q and vertex v are shown in Figure 5.6). It is clear that if $v \notin \lambda_\ell$, then $\{\lambda_i, \lambda_j, \lambda_\ell\}$ is not a set of non-crossing paths, absurdum. □

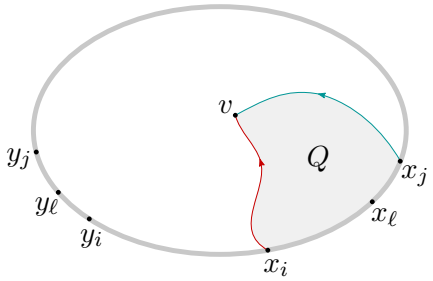


Figure 5.6: region Q and vertex v used in the proof of Lemma 29.

Now we show how to use these two lemmas for our goals. Let ρ_i be a shortest directed i-path and let ρ_j be a shortest directed j-path, for some $i, j \in [k]$, $i \neq j$. By Lemma 28, if i and j are not comparable in T_g, then ρ_i and ρ_j have no common darts. Moreover, by Lemma 29, if i is a descendant of j in T_g, then $\rho_i \cap \rho_j \subseteq \rho_{p(i)}$. By using these two facts, in order to list darts in ρ_i, then it suffices to find darts in $\rho_i \setminus \rho_{p(i)}$, for all $i \in [k] \setminus \{1\}$ (we remind that $i = 1$ is the root of T_g). To this goal we use algorithm `NCSPunion`, that builds a sequence of directed graphs $\{Y_i\}_{i\in[k]}$ such that Y_k is equal to $\bigcup_{i\in[k]} \rho_i$, where ρ_i is a shortest directed i-path, for $i \in [k]$.

We prove the correctness of algorithm `NCSPunion` in Theorem 31. At iteration i we compute $\rho_i \setminus \rho_{p(i)}$, showing that $\rho_i \setminus \rho_{p(i)} = \sigma_i \cup \text{rev}[\tau_i]$, where σ_i and τ_i are computed in Line 5 and Line 6, respectively. We observe that if ρ_i and $\rho_{p(i)}$ have no common darts, then $\sigma_i = \text{rev}[\tau_i] = \rho_i$.

Algorithm `NCSPunion`:

Input: an undirected unweighted plane graph G and k well-formed terminal pairs $\{(x_i, y_i)\}_{i\in[k]}$ on the infinite face of G

Output: a directed graph Y_k formed by the union of directed non-crossing shortest paths from x_i to y_i, for $i \in [k]$

1 Compute X_1 as in algorithm `NCSPsupergraph`;
2 Y_1 is the directed version of X_1 oriented from x_1 to y_1;
3 **for** $i = 2, \dots, k$ **do**
4 Compute X_i as in algorithm `NCSPsupergraph`;
5 σ_i is the directed path that starts in x_i and always turns left in X_i until either σ_i reaches y_i or the next dart d_i of σ_i satisfies $d_i \in Y_{i-1}$;
6 τ_i is the directed path that starts in y_i and always turns right in X_i until either τ_i reaches x_i or the next dart d'_i of τ_i satisfies $\mathrm{rev}[d'_i] \in Y_{i-1}$;
7 $Y_i = Y_{i-1} \cup \sigma_i \cup \mathrm{rev}[\tau_i]$;

To better understand Line 2 of algorithm `NCSPunion`, we recall that X_1 is an undirected 1-path, hence Y_1 is the directed version of this path.

Lemma 30. *Algorithm `NCSPunion` has $O(n)$ time complexity.*

Proof. Algorithm `NCSPunion` uses algorithm `NCSPsupergraph`, that has $O(n)$ time complexity by Lemma 26. Moreover, algorithm `NCSPunion` visits each dart of the "directed version" of X_k at most $O(1)$ times, where the *directed version of* X_k is the directed graph built from X_k by replacing each edge ab with the pair of darts $\overrightarrow{ab}$ and $\overrightarrow{ba}$. Thus, algorithm `NCSPunion` requires $O(n)$ time, since X_k is a subgraph of G. □

Theorem 31. *Graph Y_k computed by algorithm `NCSPunion` is the union of k shortest directed non-crossing i-paths, for $i \in [k]$.*

Proof. Let $\{\pi_i\}_{i\in[k]}$ be the set of paths defined in Theorem 27. For all $i \in [k]$, we denote by $\overrightarrow{\pi_i}$ the directed version of π_i, oriented from x_i to y_i.

First we define $\rho_1 = \overrightarrow{\pi_1}$ and for all $i \in [k] \setminus \{1\}$ we define

$$\rho_i = \begin{cases} \overrightarrow{\pi_i}[x_i, u_i] \circ \rho_{p(i)}[u_i, v_i] \circ \overrightarrow{\pi_i}[v_i, y_i], & \text{if } \overrightarrow{\pi_i} \text{ and } \rho_{p(i)} \text{ share at least one dart,} \\ \overrightarrow{\pi_i}, & \text{otherwise,} \end{cases} \tag{5.1}$$

where we assume that if $\overrightarrow{\pi_i}$ and $\rho_{p(i)}$ have at least one common dart, then u_i and v_i are the tail of the first common dart and the head of the last common dart, respectively, where the order is with respect to $\overrightarrow{\pi_i}$. The definition of ρ_i as in (5.1) is shown in Figure 5.7. Now we split the proof into three parts: first we prove that $\{\rho_i\}_{i\in[k]}$ is a set of shortest paths (we need it to apply Lemma 28); second we prove that $\{\rho_i\}_{i\in[k]}$ is a set of non-crossing paths (we need it to apply Lemma 29); third we prove that $Y_k = \bigcup_{i\in[k]} \rho_i$ (we prove it by Lemma 28 and Lemma 29).

$\{\rho_i\}_{i\in[k]}$ is a set of shortest paths: we proceed by induction on i. The base case is trivial because π_1 is a shortest path by definition. Let us assume that ρ_j is a shortest j-path, for $j < i$, we have to prove that ρ_i is a shortest i-path. If $\overrightarrow{\pi_i}$ and $\rho_{p(i)}$ have no common darts, then $\rho_i = \overrightarrow{\pi_i}$ by (5.1), thus the thesis holds because $\{\pi_i\}_{i\in[k]}$ is a set of shortest paths. Hence let us assume that $\overrightarrow{\pi_i}$ and $\rho_{p(i)}$ have at least one common darts, then it suffices, by definition of ρ_i, that $|\pi_i[u_i, v_i]| = |\rho_{p(i)}[u_i, v_i]|$. It is true by induction.

$\{\rho_i\}_{i\in[k]}$ is a set of non-crossing paths: we proceed by induction on i. The base case is trivial because there is only one path. Let us assume that $\{\rho_j\}_{j\in[i-1]}$ is a set of non-crossing paths, we have to prove that ρ_i does not cross ρ_j, for any $j < i$.

By construction of ρ_i, ρ_i cannot cross $\rho_{p(i)}$. Thus if ρ_i and ρ_j are crossing and j is not an ancestor of i, then either $\rho_{p(i)}$ and ρ_j are crossing or π_i and π_j are crossing; that is absurdum in both cases by induction and Theorem 27. Moreover if ℓ is an ancestor of i such that $\ell \neq p(i)$, then ρ_i does not cross ρ_ℓ otherwise ρ_ℓ would cross $\rho_{p(i)}$, absurdum by induction. Hence $\{\rho_i\}_{i\in[k]}$ is a set of non-crossing paths.

Y_k is the union of ρ_i's: now we prove that $Y_k = \bigcup_{i\in[k]} \rho_i$. In particular we show that $\rho_1 = \overrightarrow{\pi_1}$ and for all $i \in [k] \setminus \{1\}$

$$\rho_i = \begin{cases} \sigma_i \circ \rho_{p(i)}[u_i, v_i] \circ \mathrm{rev}[\tau_i], & \text{if } \overrightarrow{\pi_i} \text{ and } \rho_{p(i)} \text{ share at least one dart,} \\ \overrightarrow{\pi_i}, & \text{otherwise.} \end{cases} \tag{5.2}$$

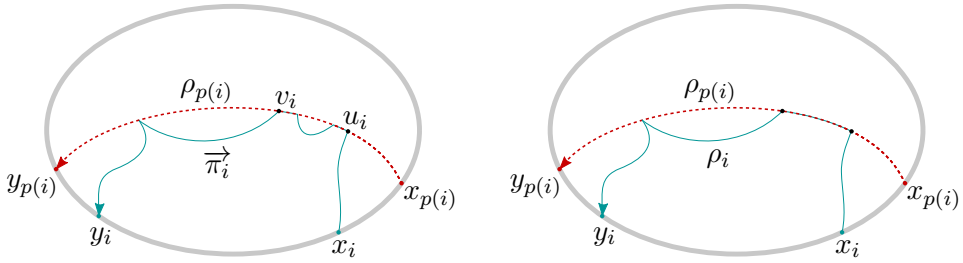
Again, we proceed by induction on i. The base case is trivial, thus we assume that (5.1) is equivalent to (5.2) for all $i < \ell$. We have to prove that (5.1) is equivalent to (5.2) for $i = \ell$.

If $\overrightarrow{\pi_\ell}$ does not intersect any dart of $\rho_{p(\ell)}$, then (5.1) is equivalent to (5.2). Thus we assume that $\overrightarrow{\pi_\ell}$ and $\rho_{p(\ell)}$ have at least one common dart. By (5.1) and (5.2) and by definition of σ_i and τ_i in Line 5 and Line 6, respectively, it suffices to prove that $d_i \in \rho_{p(i)}$ and $\mathrm{rev}[d'_i] \in \rho_{p(i)}$.

Now, by induction we know that $d_i \in \rho_\ell$ for some $\ell < i$, we have to show that $d_i \in \rho_{p(i)}$. By Lemma 28 and being $\{\rho_j\}_{j\in[k]}$ a set of shortest paths, it holds that ℓ is an ancestor or a descendant of i. Being the x_j's visited clockwise by starting from x_1, then ℓ is an ancestor of i. Finally, by Lemma 29 and being $\{\rho_j\}_{j\in[k]}$ a set of non-crossing paths, it holds that $\rho_i \cap \rho_\ell \subseteq \rho_{p(i)}$. Being $p(i) < i$, then $d_i \in \rho_{p(i)}$ as we claimed. By a similar argument, it holds that $\mathrm{rev}[d'_i] \in \rho_{p(i)}$. □

Now we can state and prove the main theorem of this chapter.

Theorem 32. *Given an undirected plane graph G with small integer weights and a set of well-formed terminal pairs $\{(x_i, y_i)\}$ on the infinite face f^∞ of G we can compute $U = \bigcup_{i\in[k]} p_i$ and the lengths of all p_i, for $i \in [k]$, where p_i is a shortest i-path and $\{p_i\}_{i\in[k]}$ is a set of non-crossing paths, in $O(n + L)$ time, where L is the sum of all edge weights.*



(a) path $\overrightarrow{\pi_i}$ and $\rho_{p(i)}$ (b) path ρ_i

Figure 5.7: proof of Theorem 31, explanation of (5.1).

Proof. By Theorem 31, the required graph U is the undirected version $\overline{Y_k}$ of the graph computed by algorithm `NCSPunion`, that has $O(n+L)$ time complexity by Lemma 30; indeed, we can split an edge of weight r in r unweighted edges. Moreover, we compute the length of p_i, for all $i \in [k]$, in $O(n+L)$ time by Theorem 19. □

Chapter 6

Max flow vitality of edges and vertices in undirected planar graphs

We give efficient algorithms to compute an additive guaranteed approximation of the vitality of edges and vertices with respect to the *st*-max flow in undirected planar graphs. We show that in the general case high vitality values are well approximated in time close to the time currently required to compute *st*-max flow $O(n \log\log n)$. We also give improved, and sometimes optimal, results in the case of integer capacities. All our algorithms work in $O(n)$ space.

6.1 Introduction

Let G be a planar graph and let s and t be two fixed vertices. In this chapter we propose fast algorithms for computing an additive guaranteed approximation of the vitality with respect to the *st*-max flow of all edges and vertices whose capacity is less than an arbitrary threshold c. Later we explain that these results can be used to obtain a useful approximation of vitality for general distribution of capacities and in the case of power-law distribution. We stress that in applications we are usually interested in finding edges and or vertices with high vitality, i.e., edges or vertices whose removal involves important decrease on the max flow value.

Our main results are summarized in the following two theorems. We recall that $c : E(G) \to \mathbb{R}^+$ is the edge capacity function, we define the capacity $c(v)$ of a vertex v as the sum of the capacities of all edges incident on v. Moreover, for each $x \in E(G) \cup V(G)$ we denote by $vit(x)$ its vitality with respect to the *st*-max flow.

Theorem 33. *Let G be a planar graph with positive edge capacities. Then for any $c, \delta > 0$, we compute a value $vit^\delta(e) \in (vit(e) - \delta, vit(e)]$ for all $e \in E(G)$ satisfying $c(e) \leq c$, in $O(\frac{c}{\delta} n + n \log\log n)$ time.*

Theorem 34. *Let G be a planar graph with positive edge capacities. Then for any $c, \delta > 0$, we compute a value $vit^{\delta}(v) \in (vit(v) - \delta, vit(v)]$ for all $v \in V(G)$ satisfying $c(v) \leq c$, in $O(\frac{c}{\delta}n + n \log n)$ time.*

All our algorithms work in $O(n)$ space. To explain the result stated in Theorem 33, we note that in the general case capacities are not bounded by any function of n. Thus, in order to obtain useful approximations, c/δ would seem to be very high. Despite this we explain that in many cases we can assume c/δ constant, implying that the time complexity of Theorem 33 is equal to the best current time bound for computing the st-max flow. The following remark is crucial, where $c_{max} = \max_{e \in E(G)} c(e)$.

Remark 35 (Bounding capacities)**.** *We can bound all edge capacities higher than MF to MF, obtaining a new bounded edge capacity function. This change has no impact on the st-max flow value or the vitality of any edge/vertex. Thus w.l.o.g., we can assume that $c_{max} \leq MF$.*

By using Remark 35 we can explain why c/δ can be assumed constant in Theorem 33, we study separately the case of general distribution of capacities and the case of power-law distribution.

- *General distribution* (after bounding capacities as in Remark 35). If we set $c = c_{max}$ and $\delta = c/k$, for some constant k, then we obtain edge capacities with an additive error less than MF/k, because of Remark 35. In many applications this error is acceptable even for small values of k, e.g., $k = 10, 50, 100$. In this way we obtain small percentage error of vitality for edges with high vitality—edges whose vitality is comparable with MF—while edges with small vitality—edges whose vitality is smaller than MF/k—are badly approximated. We stress that we are usually interested in high capacity edges, and that with these choices the time complexity is $O(n \log \log n)$, that is the time currently required for the computation of the st-max flow.
- *Power-law distribution* (after bounding capacities as in Remark 35). The previous method cannot be applied to power-law distribution because the largest part of edges has capacity lower than MF/k, even for high value of k. Thus we have to separate edges with high capacity and edges with low capacity. Let $c = \frac{c_{max}}{\ell}$ for some constant ℓ and let $H_c = \{e \in E(G) \mid c(e) > c\}$. By power-law distribution, $|H_c|$ is small even for high value of ℓ, and thus we compute the exact vitality of edges in H_c by using Corollary 2. For edges with capacity less than c, we set $\delta = c/k$, for some constant k. By Remark 35 we compute the vitality of these edges with an additive error less than $\frac{MF}{k\ell}$. Again, the overall time complexity is equal or close to the time currently required for the computation of the st-max flow.

To apply the same argument to vertex vitality in Theorem 34 we need some observations. If G's vertices have maximum degree d, then, after bounding capacities as in Remark 35, it holds $\max_{v \in V(G)} c(v) \leq dMF$. Otherwise, we note that a real-world planar graph is expected to have few vertices with high degree. The exact vitality of these vertices can be computed by Corollary 2 by removing the edges adjacent to a vertex one by one, or by using the following our result.

Theorem 36. *Let G be a planar graph with positive edge capacities. Then for any $S \subseteq V(G)$, we compute $vit(v)$ for all $v \in S$ in $O(|S|n + n \log \log n)$ time.*

If we denote by $E_S = \sum_{v \in S} deg(v)$, then the result in Theorem 36 is more efficient than the result given in Corollary 2 if either $|S| < \log n$ and $E_S > |S| \log n$ or $|S| \geq \log n$ and $E_S > \frac{|S| n^{1/3}}{\log^{8/3}}$.

Small integer case. In the case of integer capacity values that do not exceed a small constant, or in the more general case in which capacity values are integers with bounded sum, by using Theorem 32 we obtain the two following corollaries.

Corollary 37. *Let G be a planar graph with integer edge capacities and let L be the sum of all the edge capacities. Then*

- *for any $H \subseteq E(G) \cup V(G)$, we compute $vit(x)$ for all $x \in H$, in $O(|H|n + L)$ time;*
- *for any $c \in \mathbb{N}$, we compute $vit(e)$ for all $e \in E(G)$ satisfying $c(e) \leq c$, in $O(cn + L)$ time.*

Corollary 38. *Let G be a planar graph with unit edge capacities. Let $n_{>d}$ be the number of vertices whose degree is greater than d. We compute the vitality of all edges in $O(n)$ time and the vitality of all vertices in $O(\min\{n^{3/2}, n(n_{>d} + d + \log n)\})$ time.*

Finally, the results by Kowalik and Kurowski [92] about shortest paths of bounded length in unweighted planar graphs allows us to get the following corollary.

Corollary 39. *Let G be a planar graph with unit edge capacities where only a constant number of vertices have degree greater than a fixed constant d. Then we compute the vitality of all vertices in $O(n)$ time.*

Our approach. We adopt Itai and Shiloach's approach [79], that first computes a modified version D of a dual graph of G, then reduces the computation of the max flow to the computation of shortest non-crossing paths between pairs of vertices on the infinite face of D. We first study the effect on D of an edge or a vertex removal in G, showing that computing the vitality of an edge or a vertex can be reduced to computing some distances in D (see Proposition 42 and Proposition 43).

Then we determine required distances by solving SSSP instances. To decrease the cost we use a *divide and conquer* strategy: we slice D in regions delimited by some of the non-crossing shortest paths computed above. We choose shortest non-crossing paths with similar lengths, so that we compute an additive approximation of each distance by looking into a single region instead of examining the whole graph D (see Lemma 46).

Finally we have all the machinery to compute an approximation of required distances of Proposition 42 and Proposition 43 and obtain edges and vertices vitalities.

6.2 Deleting an edge or a vertex

In this section we show our main theoretical results (Proposition 42 and Proposition 43) that allow us to compute edge and vertex vitality. In Subsection 6.2.1 we show the effects in G^* and D of removing an edge or a vertex from G. In Subsection 6.2.2 we deal with crossings between π and st-separating cycles and in Subsection 6.2.3 we state the two main propositions about edge and vertex vitality.

6.2.1 Effects on G^* and D of deleting an edge or a vertex of G

We observe that removing an edge e from G corresponds to contracting endpoints of e^* into one vertex in G^*. With respect to D, if $e^* \notin \pi$, then the removal of e corresponds to the contraction into one vertex of endpoints of e^D. If $e^* \in \pi$, then both copies of e^* have to be contracted. In Figure 6.1 we show the effects of removing edge eg from graph G in Figure 3.3.

Let v be a vertex of G. Removing v corresponds to contracting vertices of face f_v^* in G^* into a single vertex. In D, if f_v^* and π have no common vertices, then all vertices of f_v^D are contracted into one. Otherwise f_v^* intersects π on vertices $\bigcup_{i\in I}\{v_i^*\}$ for some non empty set $I \subseteq [k]$. Then all vertices of f_v^D are contracted into one vertex, all vertices of $\bigcup_{i\in I}\{x_i\}$ not belonging to f_v^D are contracted into another vertex and all vertices of $\bigcup_{i\in I}\{y_i\}$ not belonging to f_v^D are contracted into a third vertex. For convenience, we define $q^x_{f_v^D} = (\bigcup_{i\in I_v}\{x_i\}) \setminus V(f_v^D)$ and $q^y_{f_v^D} = (\bigcup_{i\in I_v}\{y_i\}) \setminus V(f_v^D)$. To better understand these definitions, see Figure 6.2. In Figure 6.3 it is shown what happens when we remove vertex g of graph G in Figure 3.3.

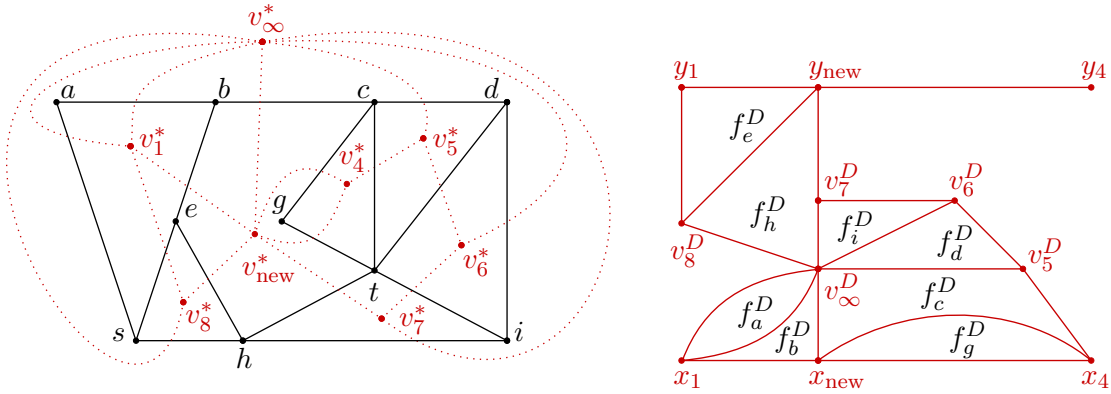


Figure 6.1: starting from graph G in Figure 3.3, we show on the left graph $G - eg$ and $(G - eg)^*$, and graph D_{G-eg} on the right.

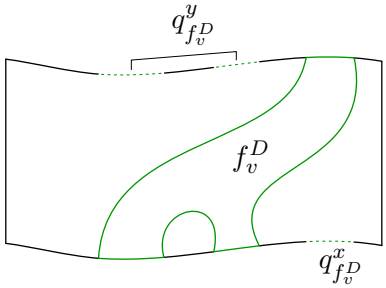


Figure 6.2: a face f_v^D, for some $v \in V(G)$, and sets $q^x_{f_v^D}$ and $q^y_{f_v^D}$. Removing v from G corresponds in D to contracting vertices of f_v^D, $q^x_{f_v^D}$ and $q^y_{f_v^D}$ in three distinct vertices.

6.2.2 Single-crossing st-separating cycles

As explained in Subsection 3.4.1, Itai and Shiloach [79] consider only shortest st-separating cycles that cross π exactly once, that correspond in D to paths from x_i to y_i, for some $i \in [k]$. In our

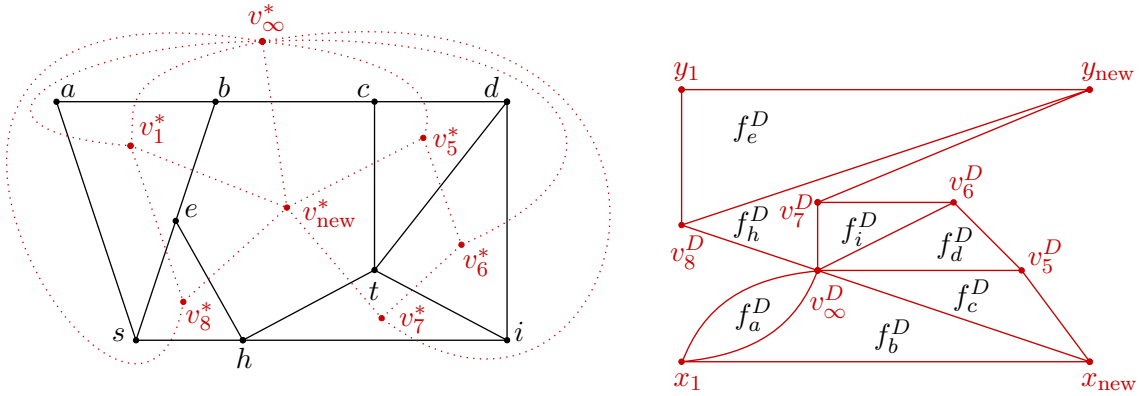


Figure 6.3: starting from graph G in Figure 3.3, we show on the left graph $G - g$ and $(G - g)^*$, and graph D_{G-g} on the right.

approach, we contract vertices of an edge or a face of G^*. Despite this we can still consider only st-separating cycles that cross π exactly once. The proof of this is the goal of this subsection.

Lemma 40. *Let γ be a simple st-separating cycle and let S be either an edge or a face of G^*. Let $r = |V(\gamma) \cap V(S)|$. After contracting vertices of S into one vertex, then γ becomes the union of r simple cycles and exactly one of them is an st-separating cycle.*

Proof. Since an edge can be seen as a degenerate face, we prove the statement only in the case in which S is a face f. Let $v^* \in V(\gamma)$ and let $u^*_1, u^*_2, \ldots, u^*_r$ be the vertices in $V(\gamma) \cap V(f)$ ordered in clockwise order starting from v^*. For convenience, let $u^*_{r+1} = u^*_1$. For $i \in [r]$, let q_i be the clockwise $u^*_i u^*_{i+1}$ path on γ. After contracting the vertices in f into one, q_i becomes a cycle. Each q_i's joined with the counterclockwise $u^*_i u^*_{i+1}$ path on the border cycle of f defines a region R_i of G^*. We remark that if q_i is composed by a single edge e^*, then q_i becomes a self-loop and region R_i is a composed only by e^*.

Cycle γ splits graph G^* into two regions: a region internal to γ called R_{in} and an external region called R_{out}. W.l.o.g., we assume that $s \in R_{in}$ and $t \in R_{out}$. Now we split the proof into two cases: $f \subseteq R_{in}$ and $f \subseteq R_{out}$.

Case $f \subseteq R_{in}$: by above, it holds that $R_1, \ldots, R_r \subseteq R_{in}$ (see Figure 6.4 on the left). Being γ an st-separating cycle, then there exists a unique $j \in [r]$ such that $s \in R_j$. Thus, after contracting vertices in f into one, p_j becomes the unique st-separating cycle, while all others R_i's become cycles that split G^* into two regions, and each region contains neither s nor t (see Figure 6.4 on the right).

Case $f \subseteq R_{out}$: by above there exists a unique $j \in [r]$ such that $R_i \subseteq R_{out}$ for all $i \neq j$ and $R_{in} \subseteq R_j$ (see Figure 6.5 on the left). W.l.o.g., we assume that $j = r$. After contracting the vertices in f into one, all regions $R_1, \ldots, R_{r-1}$ become regions inside R_r because of the embedding (see Figure 6.5 on the right). We recall that $s \in R_{in}$, thus there are two cases: if $t \in R_i$ for some $i \in [r-1]$, then p_i becomes the unique st-separating cycle; otherwise, $t \in R_{out}$, and thus p_r becomes the unique st-separating cycle. □

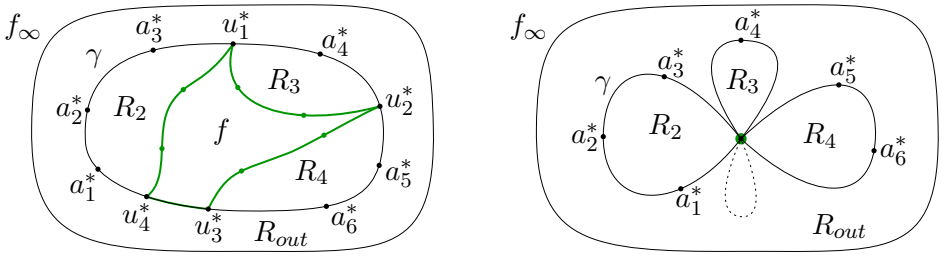


Figure 6.4: on the left, cycle γ and face f belonging to R_{in}. On the right, cycle γ after contracting all vertices in f into one, the dashed edge represents a self-loop.

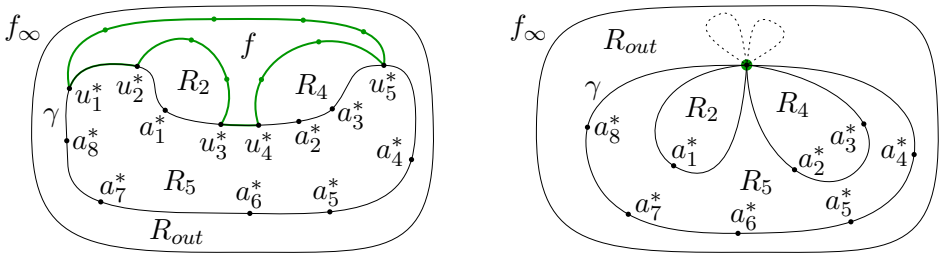


Figure 6.5: on the left, cycle γ and face f belonging to R_{out}. On the right, cycle γ after contracting all vertices in f into one, dashed edges represent self-loops.

Let Γ be the set of all st-separating cycles in G^*, and let Γ_1 be the set of all st-separating cycles in G^* that cross π exactly once. Given $\gamma \in \Gamma$ and either an edge or a face S of G^*, thanks to Lemma 40 we can define $\Delta_S(\gamma)$ as "the length of the unique st-separating cycle contained in γ after contracting vertices in S into one". Being MF equal to the length of a minimum st-separating cycle, the following relations hold:

$$\text{for any } e \in E(G),\ vit(e) = MF - \min_{\gamma \in \Gamma} \Delta_{e^*}(\gamma),$$

$$\text{for any } v \in V(G) \setminus \{s,t\},\ vit(v) = MF - \min_{\gamma \in \Gamma} \Delta_{f_v^*}(\gamma).$$

Now we show that in the above equations we can replace set Γ with set Γ_1.

Lemma 41. *Let $e \in E(G)$ and $v \in V(G) \setminus \{s,t\}$. It holds that $vit(e) = MF - \min_{\gamma \in \Gamma_1} \Delta_{e^*}(\gamma)$ and $vit(v) = MF - \min_{\gamma \in \Gamma_1} \Delta_{f_v^*}(\gamma)$.*

Proof. We recall that removing an edge e from G corresponds to contracting endpoints of e^* into one vertex, while removing a vertex v from G corresponds to contracting all the vertices in face f_v^* into one vertex. So we prove the thesis only in the more general case of vertex removal. For convenience, we denote f_v^* by f. Let $\gamma \in \Gamma$ be such that $vit(f) = MF - \Delta_f(\gamma)$ and assume that $\gamma \notin \Gamma_1$. If $V(\gamma) \cap V(f) = \emptyset$, then $vit(f) = 0$, hence it suffices to remove crossings between γ and π, see [79]. Thus let us assume that $V(\gamma) \cap V(f) \neq \emptyset$.

By Lemma 40, there exist unique $a^*, b^* \in V(f) \cap V(\gamma)$ such that the clockwise a^*b^* path p on γ becomes an st-separating cycle after the contraction of vertices in f into one. Then we remove crossings between p and π in order to obtain a path p' not longer than p as above. Finally, let

$\gamma' = p \circ q$, where q is the clockwise a^*b^* path on f. It holds that $\gamma' \in \Gamma_1$ and $\Delta_f(\gamma') \leq \Delta_f(\gamma)$, the thesis follows. □

6.2.3 Vitality vs. distances in D

The main results of this subsection are Proposition 42 and Proposition 43. The first proposition shows which distances in D are needed to obtain edge vitality and in the latter proposition we do the same for vertex vitality. In Subsection 6.2.1 we proved that removing an edge or a vertex from G corresponds to contracting in single vertices some sets of vertices of D. The main result of Proposition 42 and Proposition 43 is that we can consider these vertices individually.

Let e be an edge of G. The removal of e from G corresponds to the contraction of endpoints of e^* into one vertex in G^*. Thus if an st-separating cycle γ of G^* contains e^*, then the removal of e from G reduces the length of γ by $\omega(e^*)$. Thus e has strictly positive vitality if and only if there exists an st-separating cycle γ in G^* whose length is strictly less than $MF + \omega(e^*)$ and $e^* \in \gamma$. This is the main idea to compute the vitality of all edges. Now we have to translate it to D.

We observe that capacities of edges in G become lengths (or weights) in D. For this reason, we define $\omega(e^D) = c(e)$, for all edges $e \in G$ satisfying $e^* \notin \pi$ and $\omega(e_x^D) = \omega(e_y^D) = c(e)$ for all edges $e \in G$ satisfying $e^* \in \pi$.

For $i \in [k]$, we define $d_i = \text{dist}_D(x_i, y_i)$. We observe that $MF = \min_{i\in[k]} d_i$. For a subset S of $V(D)$ and any $i \in [k]$ we define $d_i(S) = \min\{d_i, \text{dist}_D(x_i, S) + \text{dist}_D(y_i, S)\}$. We observe that $d_i(S)$ represents the distance in D from x_i to y_i if all vertices of S are contracted into one.

For each $x \in V(G) \cup E(G)$ we define MF_x as the max flow in graph $G - x$. By definition, $vit(x) = MF - MF_x$ and, trivially, x has strictly positive vitality if and only if $MF_x < MF$.

Proposition 42. *For each edge e of G, if $e^* \notin \pi$, then $MF_e = \min_{i\in[k]}\{d_i(e^D)\}$. If $e^* \in \pi$, then $MF_e = \min_{i\in[k]} \big\{ \min\{d_i(e_x^D), d_i(e_y^D)\}\big\}$.*

Proof. Let e be an edge of G. If $vit(e) = 0$, then $MF_e = MF$ and the thesis trivially holds. Hence let us assume $vit(e) > 0$, then by Lemma 41 there exists $\gamma \in \Gamma_1$ such that $\omega(\gamma) < MF + \omega(e^*)$ and $e^* \in \gamma$. If $e^* \notin \pi$, then e corresponds in D to edge e^D, thus the thesis holds. If $e^* \in \pi$, then we note that every path in D containing both e_x^D and e_y^D corresponds in G^* to an st-separating cycle that passes through e^* twice, thus its length is equal or greater than $MF + 2c(e)$. Thus we consider only paths that contain e_x^D or e_y^D but not both. The thesis follows. □

Now we deal with vertex vitality. Note that if f_v^* and π have some common vertices, then one among $q^x_{f_v^D}$ and $q^y_{f_v^D}$ could be empty. For convenience, we set $d_i(\emptyset) = +\infty$, for all $i \in [k]$.

Proposition 43. *For each vertex v of G, if f_v^* and π have no common vertices, then $MF_v =$*

$\min_{i\in[k]}\{d_i(f)\}$, *where* $f = f_v^D$, *otherwise*

$$MF_v = \min \left\{ \begin{array}{l} \min_{i\in[k]}\{d_i(f)\}, \\ \min_{i\in[k]}\{d_i(q_f^x)\}, \\ \min_{i\in[k]}\{d_i(q_f^y)\}, \\ dist_D(f, q_f^x), \\ dist_D(f, q_f^y) \end{array} \right\}. \tag{6.1}$$

Proof. If f_v^* and π have no common vertices, then the proof is analogous to the edge case. Thus let us assume that f_v^* and π have common vertices. Let D' be the graph obtained from D by adding a vertex u, v, z connected with all vertices of q_f^x, of q_f^y, of f, respectively, with zero weight edges; for convenience we assume that q_f^x and q_f^y are both not empty. By Lemma 41 and discussion in Subsection 6.2.1, $MF_v = \omega(p)$, where p is a shortest x_iy_i path in D', varying $i \in [k]$.

Note that after contracting vertices in f into one vertex there exists an x_iy_i path whose length is $dist_D(f, x_i)$, for all $x_i \in q_f^x$. In particular, there exists an x_iy_i path whose length is $dist_D(f, q_f^x)$, for some i satisfying $x_i \in q_f^x$. The same argument applies for q_f^y. This implies that if $vit(v) = 0$, then Equation (6.1) is correct. Hence we assume that $vit(v) > 0$, so at least one among u, v and z belongs to p.

If $u \in p$ and $v, z \notin p$ (resp., $v \in p$ and $u, z \notin p$), then $\omega(p) = \min_{i\in[k]}\{d_i(q_f^x)\}$ (resp., $\omega(p) = \min_{i\in[k]}\{d_i(q_f^y)\}$) . If $z \in p$ and $u, v \notin p$ then $\omega(p) = \min_{i\in[k]}\{d_i(f)\}$. We have analyzed all cases in which p contains exactly one vertex among u, v and z. To complete the proof, we prove that, for any $i \in [k]$, every x_iy_i path that contains at least two vertices among u, v and z also contains a subpath whose length is at least $\min\{\text{dist}_D(f, q_f^x), \text{dist}_D(f, q_f^y)\}$.

Let ℓ be an x_iy_i path, for some $i \in [k]$. If $u, z \in \ell$, then there exists a subpath ℓ' of ℓ from a vertex x_j of q_f^x to a vertex r of f. If we add to ℓ' the two zero weighted edges rz and zy_j we obtain a x_jy_j path whose length is at least $\text{dist}_D(f, q_f^x)$. We can use a symmetric strategy if $v, z \in \ell$.

It remains only the case in which $u, v \in \ell$. If q_f^x and q_f^y are both non-empty, then f splits D and D' into two or more parts and no parts contain vertices of both q_f^x and q_f^y (see Figure 6.2). Thus if $u, v \in \ell$, then ℓ passes through at least one vertex of f, implying that ℓ has a subpath from a vertex of f to a vertex of q_f^x, or q_f^y. As above, this path can be transformed in a x_jy_j path shorter than ℓ whose length is at least $\min\{\text{dist}_D(f, q_f^x), \text{dist}_D(f, q_f^y)\}$, for some $j \in [k]$. □

6.3 Slicing graph D preserving approximated distances

In this section we explain our *divide and conquer* strategy. We slice graph D along shortest x_iy_i's paths. If these paths have lengths that differ at most δ, then we have a δ additive approximation of distances required in Proposition 42 and Proposition 43 by looking into a single slice instead of the whole graph D. This result is stated in Lemma 46. These slices can share boundary vertices and edges, implying that their dimension might be $O(n^2)$. In Lemma 47 we compute an implicit representation of these slices in linear time.

From now on, we mainly work on graph D, thus we omit the superscript D unless we refer to G or G^*. To work in D we need a shortest x_iy_i path and its length, for all $i \in [k]$. In the following theorem we show time complexities for obtaining elements in D.

Theorem 44 ([54, 80]). *If G is a planar graph with positive edge capacities,*

- *we compute $U = \bigcup_{i\in[k]} p_i$ and $\omega(p_i)$ for all $i \in [k]$, where p_i is a shortest x_iy_i path in D and $\{p_i\}_{i\in[k]}$ is a set of pairwise non-crossing single-touch paths, in $O(n \log\log n)$ time—see [80] for computing U and Theorem 19 for computing $\omega(p_i)$'s;*
- *for every $I \subseteq [k]$, we compute $\bigcup_{i\in I} p_i$ in $O(n)$ time—see [54] by noting that U is a forest and the paths can be found by using nearest common ancestor queries.*

From now on, for each $i \in [k]$ we fix a shortest x_iy_i path p_i, and we assume that $\{p_i\}_{i\in[k]}$ is a set of pairwise single-touch non-crossing shortest paths. Let $U = \bigcup_{i\in[k]} p_i$, see Figure 6.6(a). Each p_i's splits D into two parts as shown in the following definition and in Figure 6.6(b).

Definition 45. *For each $i \in [k]$, we define Left_i as the subgraph of D bounded by the cycle $\pi_y[y_1, y_i] \circ p_i \circ \pi_x[x_i, x_1] \circ l$, where l is the leftmost x_1y_1 path in D. Similarly, we define Right_i as the subgraph of D bounded by the cycle $\pi_y[y_i, y_k] \circ r \circ \pi_x[x_k, x_i] \circ p_i$, where r is the rightmost x_ky_k path in D.*

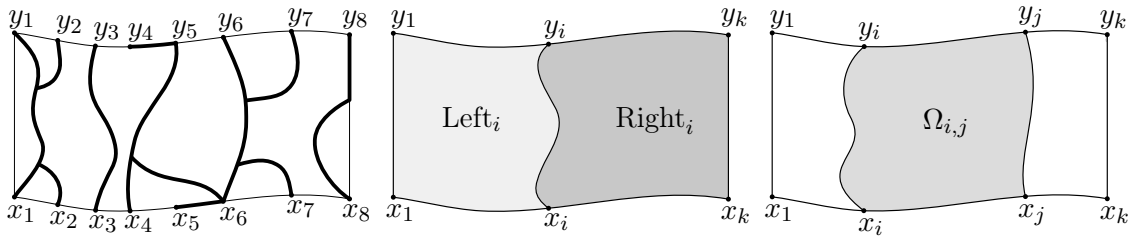


Figure 6.6: in (a) graph U in bold, in (b) subgraphs Left_i and Right_i are highlighted, in (c) subgraph $\Omega_{i,j}$, for some $i < j$.

Based on Definition 45, for each $i, j \in [k]$, with $i < j$, we define $\Omega_{i,j} = \text{Right}_i \cap \text{Left}_j$, see Figure 6.6(c). We classify (x_i, y_i)'s pairs according to the difference between d_i and MF. Each class contains pairs for which this difference is about r times δ; where $\delta > 0$ is an arbitrarily fixed value.

For each $r \in \mathbb{N}$, we define $L_r = (\ell_1^r, \ldots, \ell_{z_r}^r)$ as the ordered list of indices in $[k]$ such that $d_j \in [MF + \delta r, MF + \delta(r+1))$, for all $j \in L_r$, and $\ell_j^r < \ell_{j+1}^r$ for all $j \in [z_r - 1]$. It is possible that $L_r = \emptyset$ for some $r > 0$ (it holds that $L_0 \neq \emptyset$). If no confusion arises, we omit the superscript r; thus we write ℓ_i in place of ℓ_i^r.

The following lemma is the key of our slicing strategy. In particular, Lemma 46 can be applied for computing distances required in Proposition 42 and Proposition 43, since the vertex set of a face or an edge of D is always contained in a slice. An application is in Figure 6.8.

Lemma 46. *Let $r > 0$ and let $L_r = (\ell_1, \ell_2, \ldots, \ell_z)$. Let S be a set of vertices of D such that $S \subseteq \Omega_{\ell_i,\ell_{i+1}}$ for some $i \in [z-1]$. Then*

$$\min_{\ell \in L_r} d_\ell(S) > \min\{d_{\ell_i}(S), d_{\ell_{i+1}}(S)\} - \delta.$$

Moreover, if $S \subseteq Left_{\ell_1}$ (resp., $S \subseteq Right_{\ell_z}$) then $\min_{\ell\in L_r} d_\ell(S) > d_{\ell_1}(S) - \delta$ (resp., $\min_{\ell\in L_r} d_\ell(S) > d_{\ell_z}(S) - \delta$).

Proof. We need the following crucial statement.

a) Let $i < j \in L_r$. Let L be a set of vertices in Left_i and let R be set of vertices in Right_j. Then $d_i(L) < d_j(L) + \delta$ and $d_j(R) < d_i(R) + \delta$.

Proof of a): we prove that $d_i(L) < d_j(L)+\delta$. By symmetry, it also proves that $d_j(R) < d_i(R)+\delta$. Let us assume by contradiction that $d_i(L) \geq d_j(L) + \delta$.

Let α (resp., ϵ, μ, ν) be a path from x_i (resp., y_i, x_j, y_j) to z_α (resp., z_ϵ, z_μ, z_ν) whose length is $d(x_i, L)$ (resp. $d(y_i, L)$, $d(x_j, L)$, $d(y_j, L)$), see Figure 6.7 on the left. Being $x_j, y_j \in \text{Right}_i$ and $L \subseteq \text{Left}_i$, then μ and ν cross p_i. Let v be the vertex that appears first in $p_i \cap \mu$ starting from x_j on μ and let u be the vertex that appears first in $p_i \cap \nu$ starting from y_j on ν. An example of these paths is in Figure 6.7 on the left. Let $\zeta = p_i[y_i, u]$, $\theta = p_i[u, v]$, $\beta = p_i[x_i, v]$, $\kappa = \mu[x_j, v]$, $\iota = \nu[y_j, u]$, $\eta = \nu[u, z_\nu]$ and $\gamma = \mu[v, z_\mu]$, see Figure 6.7 on the right.

Now $\omega(\beta) + \omega(\gamma) \geq \omega(\alpha)$, otherwise α would not be a shortest path from x_i to L. Similarly $\omega(\zeta) + \omega(\eta) \geq \omega(\epsilon)$. Moreover, being $\omega(\zeta) + \omega(\theta) + \omega(\beta) = d_i$, then $\omega(\theta) \leq d_i - \omega(\alpha) + \omega(\gamma) - \omega(\epsilon) + \omega(\eta)$. Being $d_i(L) \geq d_j(L) + \delta$, then $\omega(\alpha) + \omega(\epsilon) \geq \omega(\mu) + \omega(\nu) + \delta$, this implies $\omega(\alpha) + \omega(\epsilon) \geq \omega(\kappa) + \omega(\gamma) + \omega(\iota) + \omega(\eta) + \delta$.

It holds that $\omega(\theta)+\omega(\kappa)+\omega(\iota) \leq d_i-\omega(\alpha)+\omega(\gamma)-\omega(\epsilon)+\omega(\eta)+\omega(\alpha)+\omega(\epsilon)-\omega(\gamma)-\omega(\eta)-\delta = d_i - \delta < d_j$ because $i, j \in L_r$ imply $|d_i - d_j| < \delta$. Thus $\kappa \circ \theta \circ \iota$ is a path from x_j to y_j strictly shorter than d_j, absurdum. End proof of a).

Being $S \subseteq \text{Right}_{\ell_j}$ for all $j < i$ and $S \subseteq \text{Left}_{\ell_{j'}}$ for all $j' > i + 1$, then the first part of the thesis follows from a). The second part follows also from a) by observing that if $S \subseteq \text{Left}_{\ell_1}$, then $S \subseteq \text{Left}_\ell$ for all $\ell \in L_r$. □

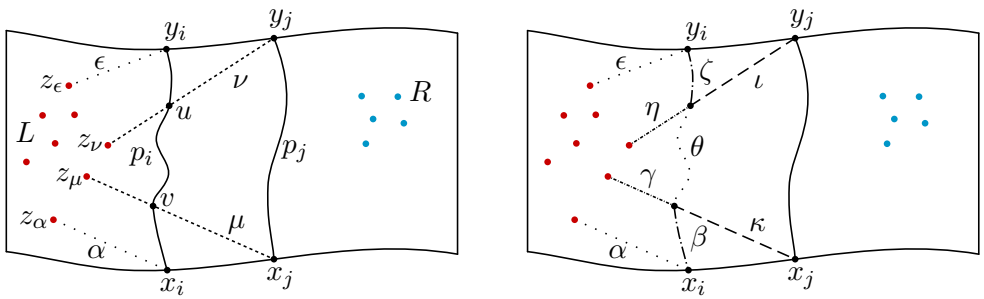


Figure 6.7: example of paths and subpaths used in the proof of a).

To compute distances in D we have to solve some SSSP instances in some $\Omega_{i,j}$'s subsets. These subsets can share boundary edges, thus the sum of their edges might be $O(n^2)$. We note

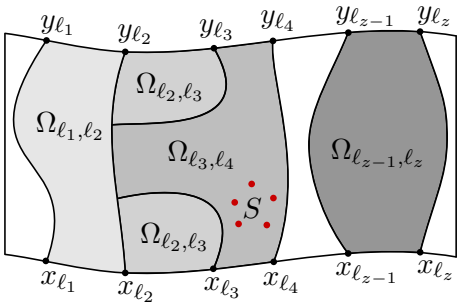


Figure 6.8: by Lemma 46, it holds that $\min_{\ell \in L_r} d_\ell(S) \geq \min\{d_{\ell_3}(S), d_{\ell_4}(S)\} - \delta$.

that, by the single-touch property, if an edge e belongs to $\Omega_{i,j}$ and $\Omega_{j,\ell}$ for some $i < j < \ell \in [k]$, then $e \in p_j$.

To overcome this problem we introduce subsets $\widetilde{\Omega}_{i,j}$ in the following way: for any $i < j \in [k]$, if $p_i \cap p_j$ is a non-empty path q, then we define $\widetilde{\Omega}_{i,j}$ as $\Omega_{i,j}$ in which we replace path q with an edge with the same length; note that the single-touch property implies that all vertices in q but its extremal have degree two. Otherwise, we define $\widetilde{\Omega}_{i,j} = \Omega_{i,j}$. Note that distances between vertices in $\widetilde{\Omega}_{i,j}$ are the same as in $\Omega_{i,j}$. Now we show how to compute all $\widetilde{\Omega}_{i,j}$'s in $O(n)$ time.

Lemma 47. *Let $A = (a_1, a_2, \ldots, a_z)$ be any increasing sequence of indices in $[k]$. It holds that $\sum_{i \in [z-1]} |E(\widetilde{\Omega}_{a_i,a_{i+1}})| = O(n)$. Moreover, given U, we compute $\widetilde{\Omega}_{a_i,a_{i+1}}$, for all $i \in [z-1]$, in $O(n)$ total time.*

Proof. For convenience, we denote by Ω_i the set $\Omega_{a_i,a_{i+1}}$, for all $i \in [z-1]$. We note that if $e \in \Omega_i \cap \Omega_{i+1}$, then $e \in p_{i+1}$. Thus, if e belongs to more than two Ω_i's, then e belongs to exactly two $\widetilde{\Omega}_i$'s because it is contracted in all other Ω_i's by definition of the $\widetilde{\Omega}_i$'s. Thus $\sum_{i \in [z-1]} |E(\widetilde{\Omega}_i)| = O(n) + O(z) = O(n)$ because $z \leq k \leq n$.

To obtain all the $\widetilde{\Omega}_i$'s, we compute $U_z = \bigcup_{a \in A} p_a$ in $O(n)$ time by Theorem 44. Then we preprocess all trees in U_z in $O(n)$ time by using Gabow and Tarjan's result [54] in order to obtain the intersection path $p_{a_i} \cap p_{a_{i+1}}$ and its length in $O(1)$ time. Finally, we build $\widetilde{\Omega}_i$ in $O(|E(\widetilde{\Omega}_i)|)$, for all $i \in [z-1]$, with a BFS visit of Ω_i that excludes vertices in $p_{a_i} \cap p_{a_{i+1}}$. □

6.4 Computing edge vitality

Now we can give our main result about edge vitality stated in Theorem 33. We need the following preliminary lemma that is an easy consequence of Lemma 46 and Lemma 47.

Lemma 48. *Let $r \in \mathbb{N}$, given U, we compute a value $\alpha_r(e) \in [\min_{i \in L_r}\{d_i(e)\}, \min_{i \in L_r}\{d_i(e)\} + \delta)$ for all $e \in E(D)$ in $O(n)$ time.*

Proof. We compute $U_r = \bigcup_{i \in L_r} p_i$ in $O(n)$ time by Theorem 44. Let $e \in E(D)$. If $e \in U_r$, we set $\alpha_r(e) = MF + \delta(r+1) - \omega(e)$. If $e \notin U_r$, then either $e \in \widetilde{\Omega}_{\ell_i,\ell_{i+1}}$ for some $i \in [z_r - 1]$, or $e \in \text{Left}_{\ell_1}$, or $e \in \text{Right}_{\ell_z}$.

For all $e \subseteq \text{Left}_{\ell_1}$, we set $\alpha_r(e) = d_{\ell_1}(e)$, similarly, for all $e \subseteq \text{Right}_{\ell_z}$, we set $\alpha_r(e) = d_{\ell_z}(e)$. Finally, if $e \in \widetilde{\Omega}_{\ell_i,\ell_{i+1}}$, then we set $\alpha_r(e) = \min\{d_{\ell_i}(e), d_{\ell_{i+1}}(e)\}$. All these choices satisfy the required estimation by Lemma 46.

To compute required distances, it suffices to solve two SSSP instances with sources x_i and y_i to vertices in $\widetilde{\Omega}_{\ell_{i-1},\ell_i} \cup \widetilde{\Omega}_{\ell_i,\ell_{i+1}}$, for each $\ell_i \in L_r$ (we handle extremal pairs as in Lemma 46). In total we spend $O(n)$ time by Lemma 47 by using algorithm in [75] for SSSP instances. □

Theorem 33. *Let G be a planar graph with positive edge capacities. Then for any $c, \delta > 0$, we compute a value $vit^\delta(e) \in (vit(e) - \delta, vit(e)]$ for all $e \in E(G)$ satisfying $c(e) \leq c$, in $O(\frac{c}{\delta}n + n \log\log n)$ time.*

Proof. We compute U in $O(n \log\log n)$ time by Theorem 44. If $d_i > MF + c(e)$, then $d_i(e^D) > MF$, so we are only interested in computing (approximate) values of $d_i(e^D)$ for all $i \in [k]$ satisfying $d_i < MF + c$. By Lemma 48, for each $r \in \{0, 1, \ldots, \lceil \frac{c}{\delta} \rceil\}$, we compute $\alpha_r(e^D) \in [\min_{i \in L_r} d_i(e^D), \min_{i \in L_r} d_i(e^D) + \delta)$, for all $e^D \in E(D)$, in $O(n)$ time. Then, for each $e^D \in E(D)$, we compute $\alpha(e^D) = \min_{r \in \{0,1,\ldots,\frac{c}{\delta}\}} \alpha_r(e^D)$; it holds that $\alpha(e^D) \in [\min_{i \in [k]}\{d_i(e^D)\}, \min_{i \in [k]}\{d_i(e^D)\} + \delta)$. Then, by Proposition 42, for each $e \in E(G)$ satisfying $c(e) \leq c$, we compute a value $vit^\delta(e) \in (vit(e) - \delta, vit(e)]$ in $O(1)$ time. □

6.5 Computing vertex vitality

In this section we show how to compute vertex vitality by computing an additive approximation of distances required in Proposition 43.

Let us denote by F_D the set of faces of D. By Proposition 43, for each face $f \in F_D$ we need $\min_{i \in [k]}\{d_i(f)\}$, this is discussed in Lemma 49. For faces $f \in F^y = \{f \in F_D \mid f$ and π_x have common vertices$\}$ we need also $\min_{i \in [k]}\{d_i(q_f^y)\}$ and $\text{dist}_D(f_v, q_f^y)$. Similarly, for faces $f \in F^x = \{f \in F_D \mid f$ and π_y have common vertices$\}$ we need also $\min_{i \in [k]}\{d_i(q_f^x)\}$ and $\text{dist}_D(f_v, q_f^x)$.

We observe that there is symmetry between q_f^x and q_f^y. Thus we restrict some definitions and results to the "y case" and then we use the same results for the "x case". In this way, we have to show only how to compute $\text{dist}_D(f, q_f^y)$ (it is done in Subsection 6.5.1) and $\min_{i \in [k]}\{d_i(q_f^y)\}$ (see Subsection 6.5.2) for each face $f \in F_D$ that intersects π_x on vertices, i.e., $f \in F^y$.

By using the same procedure of Lemma 48, we can also computing $d_i(f)$ for $f \in F_D$. Thus we can state the following lemma.

Lemma 49. *Let $r \in \mathbb{N}$, given U, we compute a value $\alpha_r(f) \in [\min_{i \in L_r}\{d_i(f)\}, \min_{i \in L_r}\{d_i(f)\} + \delta)$ for all $f \in F_D$ in $O(n)$ time.*

6.5.1 Computing $\text{dist}_D(f, q_f^y)$

Our only result of this subsection is stated in Lemma 51. To obtain it, we use the following result that easily derives from Klein's algorithm about the multiple source shortest path problem [87].

Theorem 50 ([87])**.** *Given an n vertices undirected plane graph G with positive edge lengths, given r pairs $\{(a_i, b_i)\}_{i \in [r]}$ where the b_i's are on the boundary of the infinite face and the a_i's are anywhere, it is possible to compute $dist_G(a_i, b_i)$, for all $i \in [r]$, in $O(r \log n + n \log n)$ time and $O(n)$ space.*

Lemma 51. *We compute $dist_D(f, q_f^y)$, for all $f \in F^y$, in $O(n \log n)$ time.*

Proof. For each $i \in [k]$ let $F_i \subseteq F^y$ be the set of faces such that $x_i \in f$, for all $f \in F_i$. We observe that if $|F_i| = m$, then $deg_D(x_i) \geq m + 1$, where $deg_D(x_i)$ is the degree of x_i in D.

Let D' be the graph obtained by adding a new vertex u_f for each face $f \in F^y$ and connecting u_f to all vertices of f by an edge of length L, where $L = \sum_{e \in E(D)} \omega(e)$ (see Figure 6.9 for an example of construction of graph D'). Thus $\text{dist}_D(y_i, f) = \text{dist}_{D'}(y_i, u_f) - L$.

We compute $d_{D'}(y_i, u_f)$, for $i \in [k]$ and $f \in F_i$, by using the result stated in Theorem 50. Being $|V(D')| = O(n)$, we spend $O\big(\log n \sum_{i \in [k]} |F_i| + n \log n\big) \leq O\big(\log n \sum_{i \in [k]} (deg_D(x_i) - 1) + O(n \log n)\big) = O(n \log n + n \log n) = O(n \log n)$ time. Finally, for all $f \in F^y$

$$\text{dist}_D(f, q_f^y) = \min_{\{i \in [k] \,|\, x_i \in f\}} \text{dist}_D(y_i, f) = \min_{\{i \in [k] \,|\, x_i \in f\}} \text{dist}_{D'}(y_i, u_f) - L.$$

Thus we need time $\sum_{\{i \in [k] \,|\, x_i \in f\}} O(1) \leq O(\sum_{f \in F^y} |V(f)|) \leq O(\sum_{f \in F_D} |V(f)|) = O(n)$. □

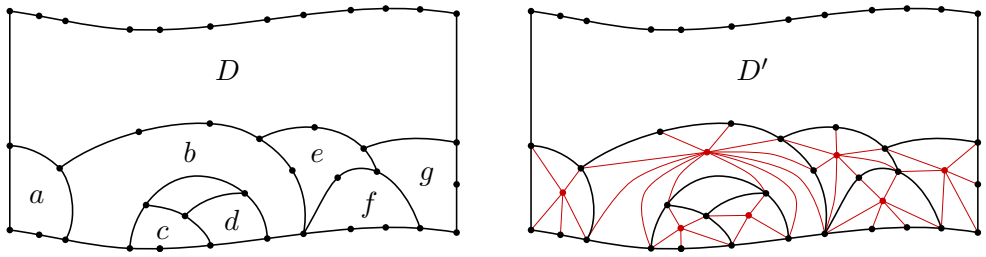


Figure 6.9: graph D, faces in F^y and graph D' used in the proof of Lemma 51.

6.5.2 Computing $d_i(q_f^y)$

We note that for computing the $d_i(q_f^y)$'s we cannot directly use Lemma 46 as we have done for the $d_i(e)$'s and the $d_i(f)$'s. Indeed, it is possible that vertices in q_f^y are not contained in any slice $\Omega_{i,j}$, with i, j consecutive indices in L_r. To overcome this, we have to introduce a partial order on faces of D.

For all $f \in F^y$, we define f^- and f^+ as the minimum and maximum indices in $[k]$, respectively, such that $x_{f^-}, x_{f^+} \in V(f)$. Now we introduce the concept of *maximal face*. Let $f \in F^y$ and let p_f and q_f be the two subpaths of the border cycle of f from x_{f^-} to x_{f^+}. We say that $g \sqsubset f$ if g is contained in the region R bounded by $\pi_x[x_{f^-}, x_{f^+}] \circ p_f$, this implies that g is also contained in the region R' bounded by $\pi_x[x_{f^-}, x_{f^+}] \circ q_f$, thus the definition does not depend on the choice of p_f and q_f. Finally, we say that f is *maximal* if there does not exist any face $g \in F^y$ satisfying $f \sqsubset g$, and we define $F_{max} = \{f \in F^y \,|\, f \text{ is maximal}\}$, see the left part of Figure 6.10. We find F_{max} in $O(n)$ time.

Given $r \in \mathbb{N}$ and $f \in F^y$, we define f_r^+ as the smallest index in L_r such that $f^+ < f_r^+$ (if $f^+ > \ell_{z_r}^r$, then we define $f_r^+ = \ell_{z_r}^r$). Similarly, we define f_r^- as the largest index in L_r such that $f_r^- > f^-$ (if $f^- < \ell_1^r$, then we define $f_r^- = \ell_1^r$), see the right part of Figure 6.10.

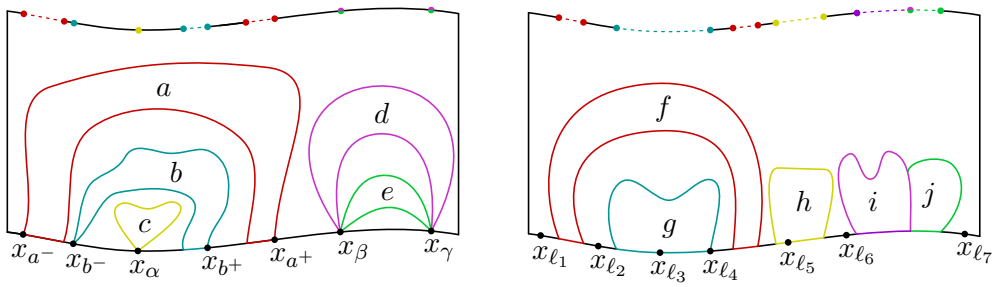


Figure 6.10: on the left $c \sqsubset b \sqsubset a$ and $e \sqsubset d$; a and d are the only maximal faces; it holds that $c^- = c^+ = \alpha$, $[d^-, d^+] = [e^-, e^+] = [\beta, \gamma]$. On the right let $L_r = (\ell_1, \ell_2, \ldots, \ell_7)$, it holds that: $[f_r^-, f_r^+] = [\ell_1, \ell_5]$, $[g_r^-, g_r^+] = [\ell_2, \ell_5]$, $[h_r^-, h_r^+] = [\ell_4, \ell_6]$, $[i_r^-, i_r^+] = [j_r^-, j_r^+] = [\ell_6, \ell_7]$.

Now we deal with computing $d_i(q_f^y)$, for all $f \in F^y$. By following Equation (6.1), we can restrict only to the easier case in which f satisfies $d_i(q_f^y) < \text{dist}_D(f, q_f^y)$; indeed, if f does not satisfy it, then we are not interested in the value of $d_i(q_f^y)$.

Lemma 52. *Let $r \in \mathbb{N}$. Given $dist_D(f, q_f^y)$ and given U, for all $f \in F^y$ satisfying $\min_{i \in L_r} d_i(q_f^y) < dist_D(f, q_f^y)$ we compute a value $\beta_r(f) \in [\min_{i \in L_r} d_i(q_f^y), \min_{i \in L_r} d_i(q_f^y) + \delta)$ in $O(n)$ total time.*

Proof. Let $f \in F^y$. We observe that if $i \in [f^-, f^+]$, then every path from x_i to q_f^y passes through either x_{f^-} or x_{f^+}. Thus, for each $i \in [f^-, f^+]$, it holds that $d_i(q_f^y) \geq \text{dist}_D(f, q_f^y)$. Hence for any $f \in F^y$ satisfying $\min_{i \in L_r} d_i(q_f^y) < \text{dist}_D(f, q_f^y)$ it holds that

$$\min_{i \in L_r} d_i(q_f^y) = \min_{i \in L_r, i \notin [f^-, f^+]} \{d_i(q_f^y)\}, \tag{6.2}$$

being $q_f^y \subseteq \text{Right}_{f_r^-}$ and $q_f^y \subseteq \text{Left}_{f_r^+}$, then Lemma 46 and Equation (6.2) imply

$$\min_{i \in L_r} d_i(q_f^y) = \min_{i \in L_r, i \notin [f^-, f^+]} \{d_i(q_f^y)\} \geq \min\{d_{f_r^-}(q_f^y), d_{f_r^+}(q_f^y)\} - \delta. \tag{6.3}$$

To complete the proof, we need to show how to compute $d_{f_r^-}(q_f^y)$ and $d_{f_r^+}(q_f^y)$, for all $f \in F^y$ satisfying $\min_{i \in L_r} d_i(q_f^y) < \text{dist}_D(f, q_f^y)$ in $O(n)$ time. In the following statement we prove it by removing the request that each face $f \in F^y$ has to satisfy $\min_{i \in L_r} d_i(q_f^y) < \text{dist}_D(f, q_f^y)$.

b) We compute $d_{f_r^-}(q_f^y)$ and $d_{f_r^+}(q_f^y)$, for all $f \in F^y$, in $O(n)$ time.

Proof of b): we recall that $d_i(q_f^y) = \text{dist}_D(x_i, q_f^y) + \text{dist}_D(y_i, q_f^y)$, for all $i \in [k]$ and $f \in F^y$. Being $q_f^y \subseteq V(\pi_y)$ we compute $\text{dist}_D(y_i, q_f^y)$ in $O(|V(q_f^y)|)$ time. Thus we have to compute only $\text{dist}_D(x_i, q_f^y)$, for required $i \in L_r$ and $f \in F^y$.

For each $f \in F^y$, let $R_f = \Omega_{f_r^-, f_r^+}$, and let $\mathcal{R} = \bigcup_{f \in F_{max}} R_f$. We observe that, given two maximal faces f and g, it is possible that $R_f = R_g$. This happens if and only if $f_r^- = g_r^-$ and $f_r^+ = g_r^+$ (see face i and face j in Figure 6.11). We overcome this abundance by introducing $\widetilde{F}$ as a minimal set of faces such that $\mathcal{R} = \bigcup_{f \in \widetilde{F}} R_f$ and $R_f \neq R_g$, for all distinct $f, g \in \widetilde{F}$ (see Figure 6.11 for an example of $\widetilde{F}$).

For each $f \in \widetilde{F}$, it holds that $\pi_y[f_r^-, f_r^+] \subseteq R_f$. Thus, by the above argument, if $g \in F^y$ and $R_g \subseteq R_f$, then $q_g^y \subseteq R_f$. We solve 4 SSSP instances in R_f with sources x_j, for all $j \in \{f_r^-, f_r^+, f^-, f^+\}$ (possibly, $f_r^- = f^-$ and/or $f_r^+ = f^+$ and/or $f^- = f^+$). Now we prove that this suffices to compute $d_{f_r^-}(q_f^y)$ and $d_{f_r^+}(q_f^y)$, for all $f \in F^y$. In particular we show that, after solving the SSSP instances, we compute $d_{g_r^-}(q_g^y)$ and $d_{g_r^+}(q_g^y)$ in $O(|V(g)|)$ time, for each $g \in F^y$.

Let $g \in F^y$, and let $f \in \widetilde{F}$ be such that $g \subseteq R_f$. There are two cases: either $g_r^- = f_r^-$ and $g_r^+ = f_r^+$, or $g_r^- \neq f_r^-$ and/or $g_r^+ \neq f_r^+$.

If the first case occurs, then we compute $\mathrm{dist}_D(x_{g_r^-}, q_g^y) = \mathrm{dist}_D(x_{f_r^-}, q_g^y)$ and $\mathrm{dist}_D(x_{g_r^+}, q_g^y) = \mathrm{dist}_D(x_{f_r^+}, q_g^y)$ in $O(|V(g)|)$ time, because $q_g^y \subseteq R_f$ and $|V(q_g^y)| < |V(g)|$. Otherwise, w.l.o.g., we assume that $g_r^- \neq f_r^-$ (if $g_r^+ \neq f_r^+$, then the proof is similar). By definitions of $\widetilde{F}$, Ω_g, and Ω_f, it holds that $g \sqsubset f$. Thus $g_r^- \in [f^-, f^+]$, therefore every path from $x_{g_r^-}$ to q_g^y passes through either f^- or f^+ (see g_3 and f_5 in Figure 6.11). By this discussion, it follows that

$$\begin{aligned}\mathrm{dist}_D(x_{g_r^-}, q_g^y) &= \min\left\{\begin{array}{l}\mathrm{dist}_D(x_{g_r^-}, x_{f^-}) + \mathrm{dist}_D(x_{f^-}, q_f^y)\\ \mathrm{dist}_D(x_{g_r^-}, x_{f^+}) + \mathrm{dist}_D(x_{f^+}, q_f^y)\end{array}\right\}\\ &= \min\left\{\begin{array}{l}|\pi[x_{g_r^-}, x_{f^-}]| + \mathrm{dist}_D(x_{f^-}, q_f^y)\\ |\pi[x_{g_r^-}, x_{f^+}]| + \mathrm{dist}_D(x_{f^+}, q_f^y)\end{array}\right\}.\end{aligned}$$

We compute all these distances by the solutions of previous SSSP instances in $O(|V(q_g^y)|)$ time, and thus we compute $\mathrm{dist}_D(x_{g_r^-}, q_g^y)$ in $O(|V(q_g^y)|)$ time. By symmetry, the same cost is required to compute $\mathrm{dist}_D(x_{g_r^+}, q_g^y)$.

We have proved that, after solving the described SSSP instances, we compute $d_{f_r^-}(q_f^y)$ and $d_{f_r^+}(q_f^y)$, for all $f \in F^y$, in $O(|V(f)|)$ time for each $f \in F^y$. Being $\sum_{f \in F^y} |V(f)| = O(n)$, it remains to show that we can solve all the previous SSSP instances in $O(n)$ time. We want to use Lemma 47 (we recall that, for our purposes, distances in $\Omega_{f_r^-, f_r^+}$ are the same in $\widetilde{\Omega}_{f_r^-, f_r^+}$).

Let us fix $i \in [h]$ and let $a = f_i$, $b = f_{i+1}$, $c = f_{i+2}$ and $d = f_{i+3}$. We cannot use directly Lemma 47 because it is possible that $a_r^- < b_r^+$ (see in Figure 6.11 $a = f_3$ and $b = f_4$, thus $b_r^- = \ell_4 < \ell_5 = a_r^+$) and thus we might have not an increasing set of indices. But, by definition of $\widetilde{F}$, it holds that $a_r^+ \leq d_r^-$, indeed, $a_r^+ \in [b_r^-, c_r^+]$ otherwise $R_b = R_c$; these relations do not depend on i. Similarly, $d_r^- \geq a_r^+$. Thus we solve first the SSSP instances in R_{f_i}, for all $i \in [h]$ such that $i \equiv 0 \bmod 3$; then for $i \equiv 1 \bmod 3$ and finally for $i \equiv 2 \bmod 3$. By Lemma 47 it costs $O(n)$ time. End proof of b).

Finally, by Equation (6.3), we set $\beta_r(f) = \min\{d_{f_r^-}(q_f^y), d_{f_r^+}(q_f^y)\}$, for all $f \in F^y$ satisfying $\beta_r(f) < \mathrm{dist}_D(f, q_f^y)$ and we ignore faces in F^y that do not satisfy it. □

6.5.3 Computational complexity of vertex vitality

Now we give our theorems about vertex vitality. To prove Theorem 34 we follow the same approach used in Theorem 33, by referring to Proposition 43 in place of Proposition 42.

We recall that the result stated in Theorem 36 is more efficient than the result in Corollary 2 if either $|S| < \log n$ and $E_S > |S| \log n$ or $|S| \geq \log n$ and $E_S > \frac{|S| n^{1/3}}{\log^{8/3}}$, where $E_S = \sum_{v \in S} deg(v)$.

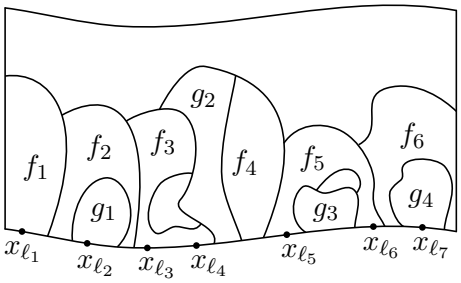


Figure 6.11: assume that $L_r = (\ell_1, \ldots, \ell_7)$. A possible $\widetilde{F}$ is $\widetilde{F} = \{f_1, \ldots, f_6\}$. Moreover, g_1, g_3, g_4 are not in F_{max}, $g_2 \in F_{max}$ and $R_{g_2} = R_{f_4}$ thus $g_2 \notin \widetilde{F}$.

Theorem 34. *Let G be a planar graph with positive edge capacities. Then for any $c, \delta > 0$, we compute a value $vit^{\delta}(v) \in (vit(v) - \delta, vit(v)]$ for all $v \in V(G)$ satisfying $c(v) \leq c$, in $O(\frac{c}{\delta}n + n \log n)$ time.*

Proof. We compute D and U in $O(n \log\log n)$ time by Theorem 44. If $c(v) < c$, then $\omega(f_v^D) < c$. For convenience, we denote f_v^D by f. By Lemma 51, we compute $\text{dist}_D(f, q_f^y)$ (resp., $\text{dist}_D(f, q_f^x)$) in $O(n \log n)$ time, for all $f \in F^y$ (resp., for all $f \in F^x$). Now we have to show how to obtain $\min_{i\in[k]}\{d_i(f)\}$, $\min_{i\in[k]}\{d_i(q_f^y)\}$ and $\min_{i\in[k]}\{d_i(q_f^x)\}$ that we may compute with an error depending on δ.

We note that $\min_{i\in[k]} d_i(f) = \min_{i\in[k], d_i < MF+\omega(f)} d_i(f)$. Indeed, if $d_i(f) = MF - z$, for some $z > 0$, then d_i is at most $MF - z + \omega(f)$. For the same reason, $\min_{i\in[k]} d_i(q_f^x) = \min_{i\in[k], d_i < MF+\omega(f)} d_i(q_f^x)$ and, similarly, $\min_{i\in[k]} d_i(q_f^y) = \min_{i\in[k], d_i < MF+\omega(f)} d_i(q_f^y)$.

By using Lemma 48, for each $r \in \{0, 1, \ldots, \lceil \frac{c}{\delta} \rceil\}$, we compute a value $\alpha_r(f) \in [\min_{i\in L_r} d_i(f), \min_{i\in L_r} d_i(f) + \delta)$, for all $f \in F_D$, in $O(n)$ time. Then, for each $f \in F_D$, we compute $\alpha(f) = \min_{r\in\{0,1,\ldots,\frac{c}{\delta}\}} \alpha_r(f)$. By above, for any $f \in F_D$ satisfying $\omega(f) < c$, it holds that $\alpha(f)$ satisfies $\alpha(f) \in [\min_{i\in[k]}\{d_i(f)\}, \min_{i\in[k]}\{d_i(f)\} + \delta)$.

With a similar strategy, by replacing Lemma 48 with Lemma 52, for each $f \in F^y$ satisfying $\omega(f) < c$ and $\min_{i\in L_r} d_i(q_f^y) < \text{dist}_D(f, q_f^y)$, we compute a value $\beta(f) \in [\min_{i\in[k]}\{d_i(q_f^y)\}, \min_{i\in[k]}\{d_i(q_f^y)\} + \delta)$. The same results hold for the "x case": for each $f \in F^x$ satisfying $\omega(f) < c$ and $\min_{i\in L_r} d_i(q_f^x) < \text{dist}_D(f, q_f^y)$, we compute a value $\gamma(f) \in [\min_{i\in[k]}\{d_i(q_f^x)\}, \min_{i\in[k]}\{d_i(q_f^x)\} + \delta)$.

Then, by Proposition 43, for each $v \in V(G)$ satisfying $c(v) \leq c$ ($\omega(f) < c$) we compute a value $vit^{\delta}(v)$ satisfying $vit^{\delta}(v) \in (vit(v) - \delta, vit(v)]$ in $O(1)$ time by using $\text{dist}_D(f, q_f^x)$, $\text{dist}_D(f, q_f^y)$, $\alpha(f)$, $\beta(f)$ and $\gamma(f)$. □

Theorem 36. *Let G be a planar graph with positive edge capacities. Then for any $S \subseteq V(G)$, we compute $vit(v)$ for all $v \in S$ in $O(|S|n + n \log\log n)$ time.*

Proof. We compute D and U in $O(n \log\log n)$ time by Theorem 44. For convenience, we denote f_v^D by f. To compute $\min_{i\in[k]}\{d_i(f)\}$, $\text{dist}_D(f, q_f^x)$ and $\text{dist}_D(f, q_f^y)$ we put a vertex u_f in face f and we connect it to all vertices of f with zero weighted edges. Then we solve an SSSP instance with source u_f and we compute $\min_{i\in[k]}\{d_i(f)\}$, $\text{dist}_D(f, q_f^x)$ and $\text{dist}_D(f, q_f^y)$ in $O(n)$. With

a similar strategy, we compute $\min_{i\in[k]}\{d_i(q_f^y)\}$ and $\min_{i\in[k]}\{d_i(q_f^x)\}$ in $O(n)$ time. Finally, by Proposition 43, for each $v \in S$, we compute $vit(v)$ in $O(1)$ time. □

6.6 Small integer capacities and unit capacities

If the edge capacities are integer, then we compute the max flow in $O(n+L)$ time thanks to the algorithm by Eisenstat and Klein [48], and U in $O(n+L)$ time thanks to Theorem 32, where L is the sum of all the edge capacities.

Corollary 37. *Let G be a planar graph with integer edge capacities and let L be the sum of all the edge capacities. Then*

- *for any $H \subseteq E(G) \cup V(G)$, we compute $vit(x)$ for all $x \in H$, in $O(|H|n + L)$ time;*
- *for any $c \in \mathbb{N}$, we compute $vit(e)$ for all $e \in E(G)$ satisfying $c(e) \leq c$, in $O(cn + L)$ time.*

Proof. Note that, being all the edge capacities integer, then every edge or vertex vitality is an integer. Thus, by taking $\delta = 1$ in Theorem 33 and Theorem 34, we obtain all the vitalities without error. The two statements follow from the proof of Theorem 33 and Theorem 34 by taking $\delta = 1$ and by computing U in $O(n + L)$ time instead of $O(n \log\log n)$ time by Theorem 32. □

Corollary 38. *Let G be a planar graph with unit edge capacities. Let $n_{>d}$ be the number of vertices whose degree is greater than d. We compute the vitality of all edges in $O(n)$ time and the vitality of all vertices in $O(\min\{n^{3/2}, n(n_{>d} + d + \log n)\})$ time.*

Proof. The complexity of edge vitality is implied by Corollary 37 by taking $c = 1$ and because $L = O(n)$. Being the vitality integers, then we compute the vitality of all vertices in $O((n_{>d} + d)n + n \log n)$ time by Theorem 34.

To compute the vitality of all vertices in $O(n^{3/2})$ time we note that in a planar graph, by Euler formula, there are at most $6\sqrt{n}$ vertices whose degree is greater than $\sqrt{n}$. Thus it suffices to take $d = \sqrt{n}$, that implies $n_{>d} \leq 6\sqrt{n}$ and $O((n_{>d} + d)n + n \log n) = O(n^{3/2})$. □

Corollary 39. *Let G be a planar graph with unit edge capacities where only a constant number of vertices have degree greater than a fixed constant d. Then we compute the vitality of all vertices in $O(n)$ time.*

Proof. By above discussion, Corollary 38 and the proof of Theorem 34, it suffices to show that we can compute $\text{dist}_D(f_v, q_f^y)$ in $O(n)$ worst-time for all $f \in F^y$. For each $i \in [k]$ let $F_i \subseteq F^y$ be the set of faces such that $x_i \in f$, for all $f \in F_i$. Note that if $|F_i| = m$, then $deg_D(x_i) = m + 1$.

Let d be the maximum degree of G. We use the algorithm by Kowalik and Kurowski [92] that with a preprocessing of $O(n)$ time establishes in $O(1)$ time if the distance between two vertices of D is at most d and, if so, computes it in $O(1)$ time. We need $d_D(y_i, z)$, for all $i \in [k]$ and $z \in V(f)$, for all $f \in F_i$. In total we spend $\sum_{i\in[k]} \left(\sum_{f\in F_i} |V(f)|\right) \leq \sum_{i\in[k]} |F_i|d = \sum_{i\in[k]}(deg_D(x_i) + 1)d = O(n)$ time. □

Chapter 7

Path Covering with Forests Number of non-crossing shortest paths

We prove that if P is a set of non-crossing shortest paths of a plane graph G whose extremal vertices lie on the same face of G, then $\mathrm{PCFN}(P) \leq 4$, and this bound is tight. We also give a linear time algorithm for computing the (at most) four covering forests. These results complement the results shown in Chapter 4 about non-crossing shortest paths, where lengths of each shortest path are computed in total linear time; now each path can be listed in time proportional to its length.

7.1 Introduction

In Chapter 4 and Chapter 5 we dealt with non-crossing shortest paths in plane graphs showing some theoretical and algorithmic results. In this chapter we give the more powerful structural result on this topic. We prove that the Path Covering with Forests Number of P is at most four for every set P of non-crossing shortest paths whose extremal vertices lie on the same face of a plane graph. We also prove that this bound is tight and we describe a linear time algorithm to list the (at most) four covering forests. In this way we conclude the argument started in Chapter 4 about non-crossing shortest paths, and definitely solve the problem of listing paths and computing their distances: by having explicitly the four covering forests and by using the result by Gabow and Tarjan [54] about lowest common ancestor queries on forests, we can compute the length of each path in total linear time and we can list a path in time proportional to its length.

We first explain that this setting, i.e., non-crossing paths in plane graphs, is not too restrictive. Removing the non-crossing property makes $\mathrm{PCFN}(P)$ dependent on $|P|$. We briefly prove this with an example. In Figure 7.1 there are six pairwise crossing paths in a grid graph G having the extremal vertices on the same face of G. Note that any set of three paths forms a cycle, hence, each forest can contain at most two paths. So the Path Covering with Forests Number of

these six paths is three. It is trivial to generalize this example to a set P of single-touch shortest paths in a plane graph whose extremal vertices lie on the same face so that $\mathrm{PCFN}(P) = |P|/2$.

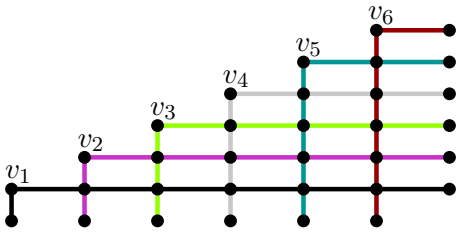


Figure 7.1: a set P of pairwise crossing paths satisfying $\mathrm{PCFN}(P) = |P|/2$. If edges incident on v_i, for $i \in [6]$, have weight less than 1 and other edges have weight 1, then colored paths are unique shortest paths between their extremal vertices.

Our approach. We translate the Path Covering with Forests Number into a problem of *forest labeling*, i.e., a labeling assigning labels to paths such that then union of paths with a same label is a forest. The number of distinct labels corresponds to the upper bound of Path Covering with Forests Number. A crucial result is that we can establish whether a labeling is a forest labeling by restricting the check only to the faces of the graph resulting from the union of the paths. At this point the main result is reached by three steps: first we prove that the Path Covering with Forests Number is constant by introducing a simple algorithm `FifteenForests` able to give a forest labeling which uses at most 15 labels (see Theorem 74 and Corollary 75); then we refine this algorithm obtaining algorithm `FourForests` able to give a forest labeling which uses at most 4 labels; finally we show that this result is tight by exhibiting a set of non-crossing shortest paths in a plane graph whose Path Covering with Forests Number is at least 4 (we recall that in Figure 2.2 there is a set of non-crossing shortest paths P such that $\mathrm{PCFN}(P) = 3$). This proves our main result.

Now we briefly explain the strategy behind our algorithms. We recursively organize all the paths in levels with respect to the genealogy tree and intersections between paths. In this way, at each iteration our algorithms have to assign labels only to paths intersecting a fixed path p.

7.2 The problem and a labeling approach

In this section we give a formal definition of the problem and we introduce a labeling approach.

Definition 53. *Given a set of paths P we say that P is a* set of non-crossing shortest paths (NCSP) *if there exists a plane graph G such that*

- *for each $p \in P$, the extremal vertices of p are in f^∞;*
- *for each $p \in P$, p is a shortest path in G and P is a set of single-touch paths;*
- *for each $p, q \in P$, p and q are non-crossing in G.*

We observe that the single-touch property can be always required for a set of shortest paths, and is also known as *consistent property* in the literature of path systems [27]. Indeed, the

single-touch property is implied by ensuring the uniqueness of the shortest path in G, that can be obtained through a tiny perturbation of edges' weights. We stress that in this chapter we use the single-touch property rather than the property of being shortest paths. Indeed, it is easy to describe a set P of k non-crossing shortest paths in a plane graphs whose Path Covering with Forests Number is $k-1$ if the single-touch property is not required.

From now on, if no confusion arises, given a NCSP P, G is the plane graph in Definition 53. We study the Path Covering with Forests Number of a NCSP P by using a labeling function that assigns labels to paths in P. So we say that a function $\mathcal{L} : Q \mapsto [k]$ is a *path labeling of* P if $\mathcal{L}$ assigns one value of $[k]$ to each path in Q, for some $k \in \mathbb{N}$ and $Q \subseteq P$. If no confusion arises, then we omit the dependence on P.

Definition 54. *Given a NCSP P, given a path labeling $\mathcal{L}$ of P, $\mathcal{L} : P \mapsto [k]$, we say that $\mathcal{L}$ is a* forest labeling *if $\bigcup_{\{p \in P \,|\, \mathcal{L}(p)=i\}} p$ is a forest for each $i \in [k]$.*

Given a path labeling $\mathcal{L}$ of P and a subset of paths $Q \subseteq P$ we define $\mathcal{L}(Q) = \bigcup_{q \in Q} \mathcal{L}(q)$. Moreover, given an edge e, we define $\mathcal{L}(e) = \mathcal{L}(\{q \in P \,|\, e \in E(q)\})$, hence, $\mathcal{L}(e)$ may contain more labels.

7.3 Restricting to faces

Our goal is to find a path labeling such that every set of paths covering a cycle does not share the same label. In this section we prove that we can restrict our check to U's faces, where U is the graph obtained by the union of all paths in P. In a first step we prove that given a NCSP P its genealogy tree can be binarized (see Subsection 7.3.1). After this simplification, we can establish whether a path labeling is a forest labeling by checking how it works on the faces of U (see Subsection 7.3.2 and in particular Theorem 60).

7.3.1 Binarization of the genealogy tree

In order to simplify the treatment, given a NCSP P, we can assume that its genealogy tree T_g is a binary tree in the following way. For two paths $p, q \in P$, we say that $p \lhd q$ if $x_1, x_q, y_q, x_p, y_p, y_1$ appear in this order on f^∞; we recall that 1 is the root of the genealogy tree. Given $p \in P$ having r children with $r \geq 3$, we order its set of children $(q_1, \ldots, q_r)$, so that $q_j \lhd q_{j+1}$ for $j \in [r-1]$. If we add a terminal pair $(x_{p'}, y_{p'})$ so that $x_{p'} = x_{q_2}$ and $y_{p'} = y_{q_r}$, then p has only two children q_1 and p'. By repeating this procedure, we obtain the binarization of T_g. Note that the number of terminal pairs becomes at most doubles. For convenience, for each path $p \in P$, if we binarize T_g, then we denote by p_r and p_ℓ the right child and left child, respectively, of p.

We observe that, being U connected, then there exists an $x_{p'}y_{p'}$ path that does not cross other paths. Moreover, this path is a shortest path in U but it might be not a shortest path in G. We do not care about this because it is an auxiliary path. By repeating this reasoning for all $i \in [k]$ having more than two children, we can assume that T_g is binary. Being U connected, then the binarization can be obtained in $O(|P|)$ time because we have only to add some terminal pairs.

Definition 55. *Given a set of paths P we say that P is a* binary set of non-crossing shortest paths (BNCSP) *if*

- *P is a NCSP;*
- *each path $p \in P$ has zero or two children in the genealogy tree;*
- *for each $p \in P$, if p has two children in the genealogy, then p, p_r, p_ℓ form a face.*

We observe that the last requirement in Definition 55 can be always obtained by modifying f^∞ contracting vertices and changing edge lengths in order to maintain the shortest path property.

In Figure 7.2 the binarization of T_g in Figure 3.2 according to Definition 55.

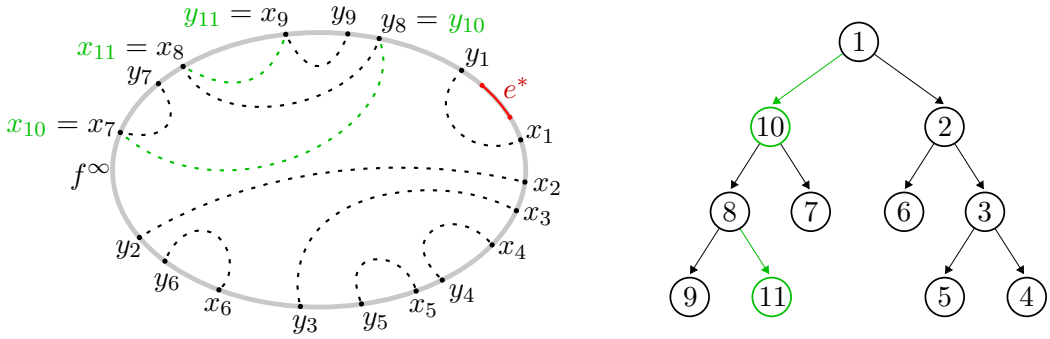


Figure 7.2: how to binarize the genealogy tree in Figure 3.2 according to Definition 55 by adding two terminal pairs (in green).

7.3.2 Solving faces

The goal of this subsection is to prove that, given a path labeling $\mathcal{L}$, if every face of U is *solved by* $\mathcal{L}$ (see Definition 59) then $\mathcal{L}$ is a forest labeling, as stated in Theorem 60. This result allows us to greatly simplify the discussion.

From now on, we assume that our input is a BNCSP P and we denote by F the set of faces of U, where we recall that $U = \bigcup_{p \in P} p$. In the following definition we specify some paths, subpaths and edges related to a face.

Definition 56. *Given a face $f \in F$ we define:*

- *q^f the* upper path of f *as the minimum path with respect to $\preceq$ satisfying $f \subseteq Int_{q^f}$;*
- *$\partial f \setminus q^f$ the* lower boundary of f*;*
- *e_r^f, e_ℓ^f the extremal edges of the lower boundary of f such that $e_r^f \in q_r^f$ and $e_\ell^f \in q_\ell^f$.*

The next two lemmas will be used in Theorem 60's proof.

Lemma 57. *Let $p \in P$ and let f be a face in F. Then the intersection between p and ∂f is a path.*

Proof. If the intersection between p and ∂f is empty or consists of a single vertex, then the thesis holds. Thus we assume that there exist two vertices $a, b \in V(p \cap \partial f)$. We have to prove that $p[a, b] \subseteq \partial f$.

Let λ be the ab path on ∂f so that the region R bounded by $\lambda \circ p[a, b]$ does not contain f. We have to prove that $\lambda = p[a, b]$. Let us assume by contradiction that there exists $e \in \lambda$ satisfying $e \notin p$, and let $q \in P$ contain e. Being R a closed region, then the non-crossing property implies $a, b \in q$. Thus the single-touch property assures $q[a, b] = p[a, b]$ and hence $e \notin q$, absurdum. □

Lemma 58. *Let f be a face in F and let q be the upper path of f. Then q intersects ∂f in at least one edge.*

Proof. Let us assume by contradiction that q does not intersect ∂f in at least one edge. Let $R = \{r_1, \ldots, r_h\}$ be the minimum set of paths such that $\partial f \subseteq \bigcup_{r \in R} E(r)$, clearly $q \notin R$. By definition of upper path, $f \subseteq \mathrm{Ext}_r$ for all $r \in Q$. Lemma 57 and minimality of R imply that $(r \cap \partial f) \cap (r' \cap \partial f) = \emptyset$, for all $r \neq r'$. Thus $r \not\preceq r'$ for all $r \neq r' \in R$, otherwise at least one $r \in R$ should satisfy $f \subseteq \mathrm{Int}_r$.

Being q the upper path of f, it holds that q is the lowest common ancestor of r and r' for all $r \neq r' \in R$, i.e., each $r \in R$ is a child of q. Finally, the single-touch property implies that $h \geq 3$, otherwise two paths would form the cycle ∂f, thus q has at least 3 children, absurdum. □

Definition 59. *Given a face f in F and a path labeling $\mathcal{L}$ of P, we say that f is* solved by $\mathcal{L}$ *if* $\mathcal{L}(e_r^f) \cap \mathcal{L}(e_\ell^f) = \emptyset$.

Theorem 60. *Let P be a BNCSP, and let $\mathcal{L}$ be a path labeling of P. If every face of F is solved by $\mathcal{L}$, then $\mathcal{L}$ is a forest labeling.*

Proof. Let us assume by contradiction that every face of F is solved by $\mathcal{L}$ and that there exists a simple cycle H so that $\bigcap_{e \in E(H)} \mathcal{L}(e) \neq \emptyset$. Let R_H be the region bounded by H. If H is a face, then we have a trivial absurdum because H is solved by $\mathcal{L}$. Otherwise, let g be a face in R_H, and let q be the upper path of g. By Lemma 58 and being H a cycle, it holds that q intersects $V(H)$. Let c be the maximal subpath of H that is in Int_q.

To finish the proof it suffices to show that there exists a face h such that its lower boundary is a subpath of c. Indeed, if so, then $\mathcal{L}(e_r^h) \neq \mathcal{L}(e_\ell^h)$ because h is solved by $\mathcal{L}$, therefore $\bigcap_{e \in E(H)} \mathcal{L}(e) = \emptyset$, absurdum.

Let $D = \{f \in F \mid f \subseteq \mathrm{Int}_q \cap R_H \text{ and } \partial f \cap c \neq \emptyset\}$ and for each face $d \in D$ let $\Delta_d = \{f \in D \mid f \subseteq \mathrm{Int}_{q^d}\}$, we recall that q^d is the upper path of d. We have to find a face $h \in D$ satisfying $|\Delta_h| = 1$, i.e., $\Delta_h = \{h\}$; indeed, this implies that the lower boundary of h is a subpath of c.

Now, let $d \in D$. If $|\Delta_d| = 1$, then the proof is completed, otherwise we observe that for each $d' \in \Delta_d$, it holds that $\Delta_{d'} < \Delta_d$ because $d \notin \Delta_{d'}$ and $\Delta_d > 0$ because $d \in \Delta_d$. Hence there exists a face $h \in D$ satisfying $|\Delta_h| = 1$. □

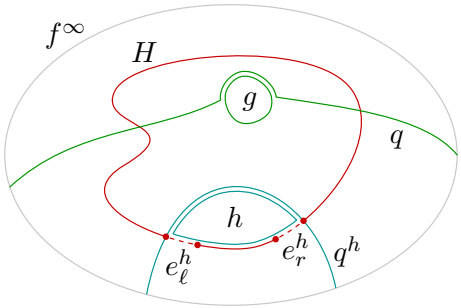


Figure 7.3: faces g, h, paths q, q^h and cycle H used in the proof of Theorem 60. Edges e^h_r and e^h_ℓ are dotted.

7.4 A first easy upper bound of the Path Covering with Forests Number

In this section we show that the Path Covering with Forests Number of a NCSP P is at most 15. In particular, we present algorithm `FifteenForests` that produces a forest labeling of P which uses at most 15 labels. We stress that every result of this section will be used in Section 7.5 to build algorithm `FourForests` which gives an improved upper bound.

In Subsection 7.4.1 we describe the outline of algorithm `FifteenForests`, in particular we explain in which order we visit paths in P by introducing the sets **Max** and **Touch**. In Subsection 7.4.2 we classify the faces related to a path in **Max** in three types. In Subsection 7.4.3 we deal with faces of the first type and Subsection 7.4.4 with faces of second and third type. Finally, in Subsection 7.4.5 we prove the correctness of algorithm `FifteenForests`.

7.4.1 Outline of the algorithm

From now on, unless otherwise stated, P denotes a BNCSP. Algorithm `FifteenForests` finds a forest labeling of P which uses at most 15 labels.

Definition 61. *Given $p \in P$, we define $Touch_p = \{q \in P \mid q \preceq p \wedge q$ and p share at least one vertex$\}$ and $Max_p = \{q \in P \mid q \preceq p \wedge q$ is a maximal path w.r.t. $\preceq$ in $P \setminus Touch_p\}$.*

Roughly speaking, Touch_p consists of all paths in Int_p that "touch" p, and Max_p consists of all paths in Int_p that are maximal w.r.t. $\preceq$ after the removal of paths in Touch_p.

By applying recursively Definition 61, we can organize path in P in levels. We start by setting $\mathbf{Max}_0 = \{p_1\}$, and then we define sets $\mathbf{Max}_i$ and $\mathbf{Touch}_i$ recursively as follows:

$$\begin{cases} p \in \mathbf{Touch}_i, & \text{if and only if } p \in \text{Touch}_m \text{ for some } m \in \mathbf{Max}_{i-1}, \\ m \in \mathbf{Max}_i, & \text{if and only if } m \in \text{Max}_p \text{ for some } p \in \mathbf{Touch}_{i-1}. \end{cases}$$

We define also $\mathbf{Max} = \bigcup_{i\in\mathbb{N}} \mathbf{Max}_i$ and $\mathbf{Touch} = \bigcup_{i\in\mathbb{N}} \mathbf{Touch}_i$. For convenience, let $N \in \mathbb{N}$ satisfy $\mathbf{Max} = \bigcup_{i\in N} \mathbf{Max}_i$ and $\mathbf{Touch} = \bigcup_{i\in N} \mathbf{Touch}_i$; in few words N is the last level. Note that $p \in \text{Touch}_p$ for all $p \in P$, thus $\mathbf{Max} \subseteq \mathbf{Touch}$. These sets are explained in Figure 7.4.

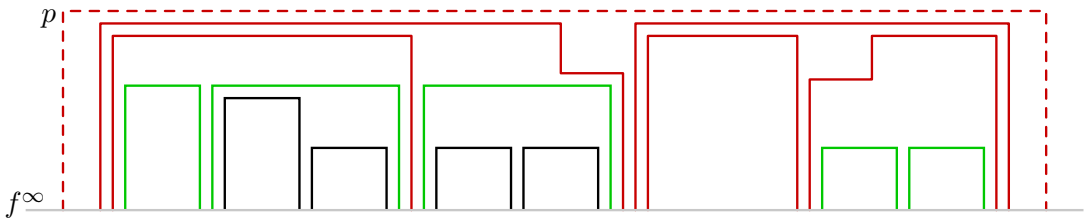


Figure 7.4: p is the dotted path, paths in Touch_p are red (note that $p \in \text{Touch}_p$), paths in Max_p are green. Thus if $p \in \mathbf{Max}_{i-1}$, then red paths are in $\mathbf{Touch}_i$, green paths in $\mathbf{Max}_i$ and black paths in $\mathbf{Touch}_{i+1}$.

`FifteenForests:`

Input: a NCSP P

Output: a forest labeling $\mathcal{L} : P \mapsto [15]$

1 Transform P from a NCSP to a BNCSP;
2 For each path $p \in P$ define the global variable $\mathcal{L}(p)$ initialized to `NULL`;
3 For each path $p \in \mathbf{Max}$ define the global variable $triple(p)$ initialized to `NULL`;
4 $triple(p_1) \leftarrow T_1$;
5 **for** $i = 0, \ldots, N$ **do**
6 **for each** $p \in \boldsymbol{Max}_i$ **do**
7 `LabelTouch`(p); `// assign` $\mathcal{L}(q) \in triple(p)$ `for all` $q \in \text{Touch}_p$
8 `TripleMax`(p); `// assign` $triple(q)$ `for all` $q \in \text{Max}_p$
9 **return** $\mathcal{L}$;

Before explaining the details of algorithm `FifteenForests` we state an important remark about the two global variables assigned to paths in P. We prefer to use global variables in order to avoid having too much entries in our functions.

Remark 62. *In our algorithms we use global variables:*

- *for each path $p \in \boldsymbol{Max}$ we define the global variable $triple(p)$ that assumes as value a non-ordered triple of labels;*
- *for each path $p \in P$ we define the global variable $\mathcal{L}(p)$ which denotes the label assigned to p; $\mathcal{L}$ is the output of our main algorithms (algorithm `FifteenForests` and algorithm `FourForests`);*
- *both variables are initialized to `NULL` and do not change once assigned.*

We define also five triples of labels: $T_1 = \{1, 2, 3\}$, $T_2 = \{4, 5, 6\}$, $T_3 = \{7, 8, 9\}$, $T_4 = \{10, 11, 12\}$ and $T_5 = \{13, 14, 15\}$, and let $\mathcal{T} = \{T_1, T_2, T_3, T_4, T_5\}$.

Now we can describe how algorithm `FifteenForests` works. Algorithm `FifteenForests` has N iterations and is based on function `LabelTouch`(p) and function `TripleMax`(p). The former assigns one label of $triple(p)$ to $\mathcal{L}(q)$ for all $q \in \text{Touch}_p$, the latter assigns one triple in $\mathcal{T}$ to $triple(q)$

for all $q \in \text{Max}_p$. The assignments are set so that at iteration i all faces in $\text{Touch}_p \cup \text{Max}_p$ are solved by $\mathcal{L}$, for all $p \in \mathbf{Max}_i$.

We note that the partial order $\preceq$ is respected: if $p \preceq q$, then algorithm `FifteenForests` labels p after q is labeled. Let's start with a preliminary definition.

7.4.2 Face types

Given $p \in \mathbf{Max}$, we classify all faces whose upper path is in Touch_p in three types as shown in Definition 63 and in Figure 7.5. We observe that, given a face f of $\text{Touch}_p \cup \text{Max}_p$, the upper path q of f is in Touch_p because the children of each path in Max_p are not in $\text{Touch}_p \cup \text{Max}_p$. This fact is crucial to classify the faces in the following definition.

Definition 63. *Let $p \in \boldsymbol{Max}$. Let f be a face of $Touch_p \cup Max_p$ and let q be the upper path of f. We say that*

- *f is* of type I for p *if $q_r, q_\ell \in Touch_p$;*
- *f is* of type II for p *if $q_r, q_\ell \in Max_p$. For convenience, we denote q_r and q_ℓ by m^f_r and m^f_ℓ, respectively;*
- *f is* of type III for p *if $q_r \in Touch_p$ and $q_\ell \in Max_p$ (or $q_r \in Max_p$ and $q_\ell \in Touch_p$).*

If no confusion arises, then we omit the reference to p.

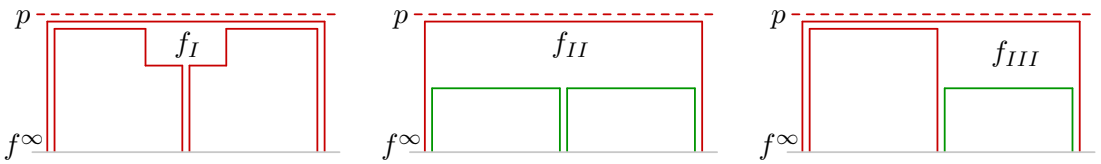


Figure 7.5: f_I, f_{II}, f_{III} are faces of type I, II, III, respectively, for p.

7.4.3 Dealing with faces of type I

The main result of this subsection is given in Proposition 68, that allows us to assign $\mathcal{L}(q)$ for all paths q in Touch_p with the three labels in $triple(p)$ so that every face of type I for p is solved by $\mathcal{L}$, for an arbitrary $p \in \mathbf{Max}$. We obtain such a labeling by using function `RightLabel` and function `LabelTouch`. Let's start with a definition, explained in Figure 7.6, about paths in Touch_p.

Definition 64. *Let $p \in \boldsymbol{Max}$. We define $TreeTouch_p$ as the subtree of T_g induced by $Touch_p$. Given $q \in Touch_p$ we define $R^p(q)$ as the set of* right descendants of q w.r.t. p *and it is composed by all elements in $TreeTouch_p$ that are in the path from q to the rightmost leaf of the subtree of $TreeTouch_p$ rooted at q. Similarly, we define $L^p(q)$ as the set of* left descendants of q w.r.t. p *and it is composed by all elements in $TreeTouch_p$ that are in the path from q to the leftmost leaf of the subtree of $TreeTouch_p$ rooted at q. When no confusion arises, we denote $R^p(q)$ and $L^p(q)$ by $R(q)$ and $L(q)$, respectively.*

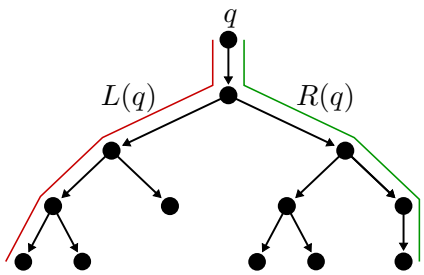


Figure 7.6: the subtree of TreeTouch$_p$ rooted at q. In red vertices belonging to $L(q)$ and in green those to $R(q)$. Note that q and its child are both in $L(q) \cap R(q)$.

In order to solve faces of type I for p we deal with intersecting paths. Given two paths p and q with $q \preceq p$, roughly speaking, it holds that if q intersects p on vertices, then q splits p into three subpaths: the right subpath containing x_p, the subpath $p \cap q$ (this is a path by the single-touch property), and the left subpath containing y_p. Note that the right subpath or the left subpath might be composed by one vertex. From this fact we obtain the following remark, that can be formally proved by using Jordan's Curve Theorem [83] and by observing that $p \circ \gamma_p$ is a closed curve for every arbitrary path p in P.

Remark 65. *Let $p, q, q' \in P$ satisfy $q \preceq p$, $q' \preceq p$ and $p \cap q \neq \emptyset$. Then q splits p into three subpaths: $p = r \circ p \cap q \circ \ell$, with $x_p \in r$ and $y_p \in \ell$. The following statements hold:*

- *if $q' \cap r \neq \emptyset$, then $q \lhd q'$;*
- *if $q' \cap \ell \neq \emptyset$, then $q' \lhd q$.*

We want to apply the previous result to faces of type I for p. Given a face f of type I for p, we observe that e_r^f and e_ℓ^f both have exactly one extremal vertex in p by their definition. Thus if a path q contains either e_r^f or e_ℓ^f, then $q \in \text{Touch}_p$. The following lemma, whose proof is strictly based on Remark 65, explains which paths contain e_r^f and which paths contains e_r^f. This result is the key to label all paths in Touch$_p$ with three labels in order to solve all faces of type I for p (see Lemma 67 and Proposition 68).

For an edge e we define $P(e) = \{p \in P \mid e \in p\}$.

Lemma 66. *Let $p \in \boldsymbol{Max}$. Let f be a face of type I for p and let q be the upper path of f, then $P(e_r^f) \subseteq L(q_r)$ and $P(e_\ell^f) \subseteq R(q_\ell)$.*

Proof. We prove that $P(e_r^f) \subseteq L(q_r)$, by symmetry, it also proves that $P(e_\ell^f) \subseteq R(q_\ell)$. First of all, $e_r^f \in q_r$ by Definition 56. Let z be the extremal vertex of e_r^f not belonging to p. Then the subpath λ of q_r from z to y_{q_r} does not intersect p on vertices because of the single-touch property.

Let us assume that q_r has two children c_r and c_ℓ belonging to Touch$_p$, with $c_\ell \lhd c_r$. Indeed, if q_r has no children belonging to Touch$_p$, then the thesis is trivial, and if q_r has exactly one child belonging to Touch$_p$, then it belongs to $L(q_r)$ by definition.

Being $\lambda \cap p = \emptyset$, then e_r^f belongs to c_ℓ; indeed, if e_r^f belongs to c_r, then Remark 65 would imply $c_\ell \cap q_r = \lambda$, thus $c_\ell \notin \text{Touch}_p$, absurdum. By repeating recursively this reasoning, the thesis follows. □

Now we introduce recursive function `RightLabel`(p, q, σ) whose entries are p, q, σ, where p is a path in **Max**, q is a path in Touch$_p$ and σ is an permutation of *triple*(p). By calling `RightLabel`(p, q, σ), we label all paths in Touch$_p$ belonging to Int$_q$. It holds that q is labeled with the first label in σ. Then if q has one child in Touch$_p$, then σ is passed to its child without permutations. If q has two children in Touch$_p$, then σ is passed to its children with permutations. An example of how these permutations change is given in Figure 7.7.

`RightLabel`$(p, q, (c_1, c_2, c_3))$:

Input: a path p in **Max**, a path q in *Touch*$_p$ and a permutation (c_1, c_2, c_3) of *triple*(p)
Output: assign $\mathcal{L}(q') \in$ *triple*(p) for all $q' \in$ Touch$_p$ such that $q' \preceq q$

1 $\mathcal{L}(q) \leftarrow c_1$;
2 **if** *q has one child q' in Touch$_p$* **then**
3 `RightLabel`$(p, q', (c_1, c_2, c_3))$;
4 **if** *q has two children in Touch$_p$* **then**
5 `RightLabel`$(p, q_r, (c_1, c_3, c_2))$;
6 `RightLabel`$(p, q_\ell, (c_2, c_1, c_3))$;

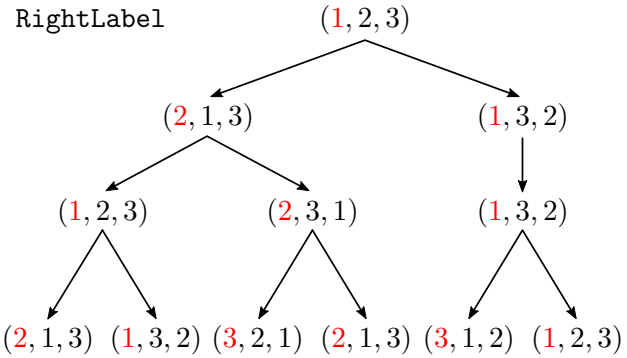


Figure 7.7: assume that *triple*$(p) = \{1, 2, 3\}$. Example of permutations of *triple*(p) during the call `RightLabel`$(p, q, (1, 2, 3))$, where q is the vertex on the top. The first element of the permutation, in red, denotes also the label assigned to the path.

The recursive calls of function `RightLabel` change the permutation of labels in order to obtain the following lemma. In Subsection 7.5.2 we will introduce its symmetric version called function `LeftLabel`.

Lemma 67. *Let $p \in$ **Max**, let (c_1, c_2, c_3) be any ordering of triple(p) and let $q \in$ Touch$_p$. If we call `RightLabel`$(p, q, (c_1, c_2, c_3))$, then $\mathcal{L}(R(q)) = \{c_1\}$ and $\mathcal{L}(L(q)) \subseteq \{c_1, c_2\}$.*

Proof. To prove that $\mathcal{L}(R(q)) = \{c_1\}$, we observe that the first element of the permutation does not change for all the calls in $R(q)$ because of Line 3 and Line 5.

Similarly, to prove that $\mathcal{L}(L(q)) \subseteq \{c_1, c_2\}$, we observe that the first and the second element of the permutation are the same (possibly swapped) for all the calls in $L(q)$ because of Line 3 and Line 6. □

By using function `RightLabel`, we introduce the following compact function.

`LabelTouch`(p):

Input: a path p in **Max**

Output: assign $\mathcal{L}(q) \in$ *triple*(p) for all q in Touch_p so that every face of type I for p is solved by $\mathcal{L}$

```
1 Let (c1, c2, c3) be an arbitrary ordering of triple(p);
2 RightLabel(p, p, (c1, c2, c3));          // assign L(q) ∈ triple(p) for all q ∈ Touch_p
```

Proposition 68. *Let $p \in$ **Max**, if we call `LabelTouch`(p), then $\mathcal{L}(q) \in$ triple(p) for all $q \in \text{Touch}_p$ and every face of type I for p is solved by $\mathcal{L}$.*

Proof. It's clear that if we call `LabelTouch`(p), then all paths in Touch_p are labeled with labels in *triple*(p). Now let f be a face of type I for p and let q be the upper path of f. By definition of face of type I, $q \in \text{Touch}_p$. We have to prove that f is solved by $\mathcal{L}$. Note that function `LabelTouch` calls recursively function `RightLabel`, thus by calling `LabelTouch`(p) we arrive to call `RightLabel`$(p, q, (c_1, c_2, c_3))$, for some (c_1, c_2, c_3) permutation of *triple*(p).

The call `RightLabel`$(p, q, (c_1, c_2, c_3))$ implies the call `RightLabel`$(p, q_r, (c_1, c_3, c_2))$ and the call `RightLabel`$(p, q_\ell, (c_2, c_1, c_3))$. Lemma 67 implies $\mathcal{L}(L(q_r)) \subseteq \{c_1, c_3\}$ and $\mathcal{L}(R(q_\ell)) = \{c_2\}$. Being $\mathcal{L}(e_r^f) \subseteq \mathcal{L}(L(q_r))$ and $\mathcal{L}(e_\ell^f) \subseteq \mathcal{L}(R(q_\ell))$ by Lemma 66, then $\mathcal{L}(e_r^f) \cap \mathcal{L}(e_\ell^f) = \emptyset$, hence f is solved by $\mathcal{L}$. □

7.4.4 Dealing with faces of type II and type III

In this subsection we build function `TripleMax`(p) that assigns *triple*(q) for all $q \in \text{Max}_p$, where $p \in$ **Max**. The consequences of the execution of `TripleMax`(p) are explained in Proposition 73 and they are concerned paths in Touch_p and Max_p that *interfere with* (see Definition 69) the same face.

To solve the faces of type I for p we worked only with paths in Touch_p, because given f of type I for p, then q contains e_r^f or e_ℓ^f only if $q \in \text{Touch}_p$. To deal with faces of type II and type III we have to work also with paths in Max_p. For this reason we introduce the concept of *interfere with*.

Definition 69. *Let $f \in F$, we say that $q \in P$* interferes with *f if q contains either e_r^f or e_ℓ^f.*

The structure of faces of type II for p is easy. Indeed, given a face f of type II for p, then m_r^f and m_ℓ^f interfere with f and it does not exist any $q \in \text{Max}_p \cup \text{Touch}_p$ interfering with f so that $q \notin \{m_r^f, m_\ell^f\}$. Clearly, some paths in $\text{Touch}_{m_r^f}$ and $\text{Touch}_{m_\ell^f}$ may interfere with f, but they are labeled by `LabelTouch`(m_r^f) and `LabelTouch`(m_ℓ^f) (see algorithm `FifteenForests`). Thus to solve the face f it suffices to set *triple*$(m_r^f) \cap$ *triple*$(m_\ell^f) = \emptyset$.

Dealing with faces of type III is more complex. For convenience, we define $\text{Max}_p^{II} = \{m \in \text{Max}_p \mid f_m \text{ is a face of type II for } p\}$ and $\text{Max}_p^{III} = \{m \in \text{Max}_p \mid f_m \text{ is a face of type III for } p\}$.

To explain the following definition we note that, given a path $m \in \mathbf{Max} \setminus \{p_1\}$, m forms a face f with its parent and its sibling (the other child of the parent). In order to solve f, we deal with the extremal edge of the lower boundary of f belonging to m. Clearly, this does not happen for p_1, for which we choose as arbitrary edge the edge adjacent to x_1.

Definition 70. *Given $m \in \boldsymbol{Max} \setminus \{p_1\}$ we define $f_m \in F$ the face whose upper path is the parent of m in T_g. Moreover, we define $edge^m$ the edge among $e_r^{f_m}$ and $e_\ell^{f_m}$ belonging to m. Finally, we define $edge^{p_1}$ as the edge in p_1 adjacent to x_1.*

Note that, given $m \in \mathrm{Max}_p^{III}$, $edge^m$ belongs only to m—and so, possibly, to some paths in Touch_m—and there are not other paths in $\mathrm{Touch}_p \cup \mathrm{Max}_p$ containing $edge^m$. But there may exist a path $m' \in \mathrm{Max}_p^{II} \cup \mathrm{Max}_p^{III}$ distinct from m interfering with f. Thus we introduce the relation $\dashrightarrow$ and we study its structure in Lemma 72. We observe that $\dashrightarrow$ is not transitive.

Definition 71. *Let $p \in \boldsymbol{Max}$. Given $m, m' \in Max_p$ we write $m \dashrightarrow m'$ if m interferes with $f_{m'}$.*

Figure 7.8 explains Definition 71, note that $m_3 \dashrightarrow m_5$ even if m_3 and m_5 are vertex disjoint.

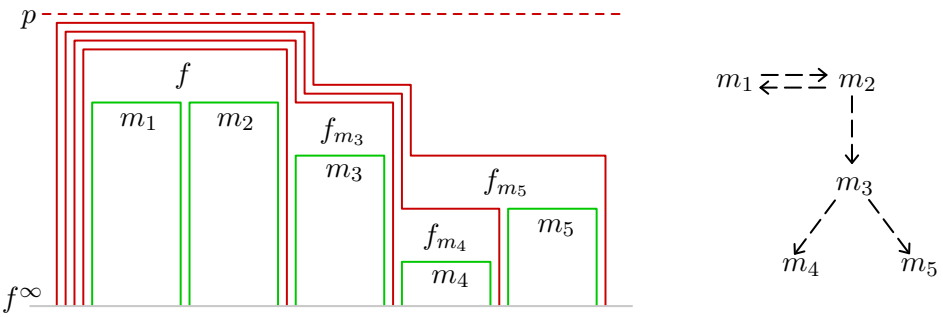


Figure 7.8: on the left in red paths in Touch_p and in green paths in Max_p. On the right the graph $(\mathrm{Max}_p, \dashrightarrow)$. Note that $(\mathrm{Max}_p^{III}, \dashrightarrow)$ is a rooted tree, f is a face of type II for p while f_{m_3} and f_{m_5} are faces of type III for p.

The main consequence of the following lemma is that $(\mathrm{Max}_p^{III}, \dashrightarrow)$ is bipartite. This fact will be used in function `TripleMax`.

Lemma 72. *For each $p \in \boldsymbol{Max}$, the graph $(Max_p^{III}, \dashrightarrow)$ is a forest of rooted trees.*

Proof. For each $m \in \mathrm{Max}_p^{III}$ we denote by q_m its parent in T_g. We split the proof into three parts.

c) let $m, m' \in \mathrm{Max}_p$. If $m \dashrightarrow m'$, then $q_m \preceq q_{m'}$ and q_m interferes with $f_{m'}$;
d) every vertex in $(\mathrm{Max}_p, \dashrightarrow)$ has at most one incoming dart;
e) in $(\mathrm{Max}_p^{III}, \dashrightarrow)$ there are no cycles.

Proof of c): It holds that $f_{m'} \subseteq \mathrm{Int}_{q_{m'}}$, thus the lower boundary of $f_{m'}$ is in $\mathrm{Int}_{q_{m'}} \setminus q_{m'}$. This implies that q_m is in $\mathrm{Int}_{q_{m'}} \setminus q_{m'}$ because otherwise m could not satisfy $m \dashrightarrow m'$. Therefore, $q_m \preceq q_{m'}$ as we claimed. Moreover, q_m interferes with $f_{m'}$ for the same reasoning. End proof of c).

Proof of d): Let us assume by contradiction that there exists $m \in \mathrm{Max}$ having two incoming darts in $(\mathrm{Max}_p, \dashrightarrow)$. Then, by definition of $\dashrightarrow$, there exist $m', m'' \in \mathrm{Max}_p$ sharing the extremal

edge e of the lower boundary of f_m not contained in m. Being $m' \circ \gamma_{m'}$ and $m'' \circ \gamma_{m''}$ two closed curves, then every face f containing e is either in Int_m or in $\text{Int}_{m'}$. Thus f is not a face of type I, nor II nor III for p, absurdum. End proof of d).

Proof of e): Let us assume by contradiction that there exist r elements in Max_p^{III}, $m_1, m_2, \ldots, m_r$, $r \geq 2$ such that $m_i \dashrightarrow m_{i+1}$, for all $i \in [r-1]$ and $m_r \dashrightarrow m_1$. Then c) implies $q_{m_1} \preceq q_{m_2} \preceq \ldots \preceq q_{m_r} \preceq q_{m_1}$, and thus $q_{m_1} = q_{m_2} = \ldots = q_{m_r}$. Thus q_{m_1} has at least r children. If $r \geq 3$, then it is absurdum because T_g is a binary tree. Else, $r = 2$ and thus $f_{m_1} = f_{m_2}$ is a face of type II because m_1 and m_2 share the same parent in T_g, implying $m_1 \notin \text{Max}_p^{III}$ and $m_2 \notin \text{Max}_p^{III}$, absurdum. End proof of e).

Now, we observe that the thesis is a consequences of d) and e). □

`TripleMax`(p):

Input: a path p in **Max**

Output: assign $triple(q) \in \mathcal{T}$ for all $q \in \text{Max}_p$

1 Let $\{X, Y, W, Z\} = \mathcal{T} \setminus \{triple(p)\}$;

2 **for each** *face f of type II for p* **do**

3 $\quad triple(m_r^f) \leftarrow X$;

4 $\quad triple(m_\ell^f) \leftarrow Y$;

5 Bipartite the forest $(\text{Max}_p^{III}, \dashrightarrow)$ into the two classes A and B so that $u \not\dashrightarrow v$ for all u, v in the same class;

6 **for each** $m \in A$ **do**

7 $\quad triple(m) \leftarrow W$

8 **for each** $m \in B$ **do**

9 $\quad triple(m) \leftarrow Z$

The following lemma is crucial to prove the correctness of algorithm `FifteenForests` and explains the main consequences of function `TripleMax`.

Proposition 73. *Let $p \in$ **Max** and let f be a face of type II or type III for p. If we call `TripleMax`(p) then*

73.(1) let $m \in Max_p$ interfere with f, then $triple(m) \cap triple(p) = \emptyset$;

73.(2) let $m, m' \in Max_p$ interfere with f, then $triple(m) \cap triple(m') = \emptyset$.

Proof. The first statements is implied by Line 1 and all lines in which a value of *triple* is assigned. To prove the second statement let f and g be a face of type II and type III for p, respectively. By definition, the unique paths in Max_p that interfere with f are m_r^f and m_ℓ^f. The for cycle in Line 2 implies $triple(m_r^f) = X$ and $triple(m_\ell^f) = Y$, and the thesis applies in this case. Now let m, m' interfere with g. Hence either $m \dashrightarrow m'$ or $m' \dashrightarrow m$ and, w.l.o.g., we assume that $m' \dashrightarrow m$. Thus $g = f_m$ and there are two cases: either $m \in \text{Max}_p^{II}$ or $m \in \text{Max}_p^{III}$. If the former

case applies, then $triple(m) \in \{X, Y\}$ because of the for cycle in Line 2 and $triple(m) \in \{W, Z\}$ because of Line 7 and Line 9, so the thesis holds in this case. If the latter case applies, then Line 5, Line 7 and Line 9 imply $triple(m) = W$ and $triple(m') = Z$, or vice-versa, and thus the thesis holds. We stress that we can bipartite the graph $(\text{Max}_p^{III}, \dashrightarrow)$ because Lemma 72 states that it is a forest. □

7.4.5 Correctness of algorithm `FifteenForests`

In this subsection we prove the correctness of algorithm `FifteenForests` shown in Subsection 7.4.1. As a consequence we have Corollary 75 which state that every NCSP has Path Covering with Forests Number at most 15.

Theorem 74. *Given a NCSP P, algorithm `FifteenForests` produces a forest labeling $\mathcal{L}$ of P which uses at most 15 labels.*

Proof. Thanks to Theorem 60 we only need to prove that every faces in $\bigcup_{q \in P} q$ is solved by $\mathcal{L}$. Let f be a face, then there exists $p \in \mathbf{Max}$ such that f is a face of type I, or type II or type III for p.

If f is a face of type I for p, then f is solved by $\mathcal{L}$ because of Proposition 68 and the call `LabelTouch`(p).

If f is a face of type II for p, then $e_r^f \in m_r^f$ and $e_\ell^f \in m_\ell^f$. Moreover, if a path $q \in P$ interferes with f, then either $q \in \text{Touch}_{m_r^f}$ or $q \in \text{Touch}_{m_\ell^f}$. Thus f is solved by $\mathcal{L}$ because of 73.(2) and the calls `LabelTouch`(m_r^f) and `LabelTouch`(m_ℓ^f).

If f is a face of type III for p, then either $e_r^f \in m^f$ or $e_\ell^f \in m^f$, where m^f is the unique child of q belonging to Max_p. W.l.o.g., we assume that $e_r^f \in m^f$. We observe that e_ℓ^f may belong to some paths in Touch_p and at most one path in Max_p. Every path q in Touch_p satisfies $\mathcal{L}(q) \in triple(p)$, thus thanks to 73.(1) and the call `LabelTouch`(m^f) we can ignore paths in Touch_p to determine whether f is solved by $\mathcal{L}$. Hence let us assume that there exists $m' \in \text{Max}_p$ satisfying $e_\ell^f \in m'$. Then f is solved by $\mathcal{L}$ because of both the statements of Proposition 73 and the calls `LabelTouch`(m^f) and `LabelTouch`(m'). □

Corollary 75. *Let P be a set of non-crossing shortest paths in a plane graph G whose extremal vertices lie on the infinite face of G. Then the Path Covering with Forests Number of P is at most 15.*

7.5 The Path Covering with Forests Number is at most four

In this section we show that the Path Covering with Forests Number of a NCSP P is at most 4. In particular, we present algorithm `FourForests` that produces a forest labeling of P which uses at most 4 labels. We strictly use all results in Section 7.4.

In Subsection 7.5.1 we describe the outline of algorithm `FourForests`. In Subsection 7.5.2 we deal with faces of the first type and Subsection 7.5.3 with faces of second and third type. Finally, in Subsection 7.5.4 we exhibit algorithm `FourForests` and we prove its correctness.

7.5.1 Outline of the algorithm

Algorithm `FourForests` is a refining of algorithm `FifteenForests`, thus they share the same structure; the only differences are that function `LabelTouch` and function `TripleMax` are replaced with function `LabelSpecialTouch` and function `TripleSpecialMax`, respectively. We introduce $special(p) \in triple(p)$ for all $p \in \mathbf{Max}$ as a special label of $triple(p)$. Function `LabelSpecialTouch`(p) assigns one label of $triple(p)$ for all $q \in \text{Touch}_p$ so that all paths containing $edge^p$ (see Definition 70) are labeled with $special(p)$. Function `TripleSpecialMax`(p) assigns $triple(q)$ and $special(q)$ for all $q \in \text{Max}_p$. The assignments are set so that at iteration i all faces in $\text{Touch}_p \cup \text{Max}_p$ are solved by $\mathcal{L}$, for all $p \in \mathbf{Max}_i$.

`FourForests`:

Input: a NCSP P

Output: a forest labeling $\mathcal{L} : P \mapsto [4]$

1 Transform P from a NCSP to a BNCSP;
2 For each path $p \in P$ define the global variable $\mathcal{L}(p)$ initialized to `NULL`;
3 For each path $p \in \mathbf{Max}$ define the global variables $triple(p), special(p)$ both initialized to `NULL`;
4 $triple(p_1) \leftarrow \{1, 2, 3\}$;
5 $special(p_1) \leftarrow 1$;
6 **for** $i = 0, \ldots, N$ **do**
7 **for each** $p \in \boldsymbol{Max}_i$ **do**
8 `LabelSpecialTouch`(p); `// assign` $\mathcal{L}(q) \in triple(p)$ `for all` $q \in \text{Touch}_p$
9 `TripleSpecialMax`(p); `// assign` $triple(q), special(q)$ `for all` $q \in \text{Max}_p$
10 **return** $\mathcal{L}$;

7.5.2 Dealing with faces of type I

The main goal of this subsection is to build function `LabelSpecialTouch`(p) which assigns $\mathcal{L}(q) \in triple(p)$ for all $q \in \text{Touch}_p$ so that all paths containing $edge^p$ are labeled with $special(p)$ and all faces of type I for p are solved by $\mathcal{L}$ (see Proposition 78), where $p \in \mathbf{Max}$. We obtain function `LabelSpecialTouch` by joining function `RightLabel` and function `LeftLabel`.

Function `LeftLabel` is equal to function `RightLabel` except for the last line in which q_r and q_ℓ are swapped. Therefore, Lemma 76 is a consequence of Lemma 67.

Lemma 76. *Let $p \in \boldsymbol{Max}$, let (c_1, c_2, c_3) be any ordering of $triple(p)$ and let $q \in \text{Touch}_p$. If we call `LeftLabel`$(p, q, (c_1, c_2, c_3))$, then $\mathcal{L}(R(q)) \subseteq \{c_1, c_2\}$ and $\mathcal{L}(L(q)) = \{c_1\}$.*

We can use function `RightLabel` and function `LeftLabel` to label all elements in Touch_p in order to solve all faces of type I for p and to assure that all paths containing $edge^p$ are labeled with $special(p)$. This is made with function `SpecialLabel`$(p, q, (c_r, c_\ell))$ whose entries are $p, q, (c_r, c_\ell)$, where p is a path in $\mathbf{Max}$, q is a path in Touch_p, and c_r, c_ℓ are two labels in $triple(p)$.

`LeftLabel`$(p, q, (c_1, c_2, c_3))$:

Input: a path p in **Max**, a path q in $Touch_p$ and a permutation (c_1, c_2, c_3) of $triple(p)$
Output: assign $\mathcal{L}(q') \in triple(p)$ for all $q' \in \text{Touch}_p$ such that $q' \preceq q$

```
1 L(q) ← c_1;
2 if q has one child q' in Touch_p then
3     LeftLabel(p, q', (c_1, c_2, c_3));
4 if q has two children in Touch_p then
5     LeftLabel(p, q_ℓ, (c_1, c_3, c_2));
6     LeftLabel(p, q_r, (c_2, c_1, c_3));
```

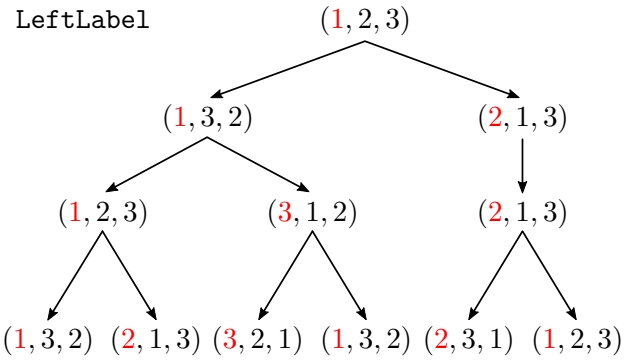


Figure 7.9: assume that $triple(p) = \{1, 2, 3\}$. Example of permutations of $triple(p)$ during the call `LeftLabel`$(p, q, (1, 2, 3))$, where q is the vertex on the top. The first element of the permutation, in red, denotes also the label assigned to the path.

We call `SpecialLabel`$(p, q, (c_r, c_\ell))$ if and only if q contains $edge^p$, and the child of q containing $edge^p$ is labeled with $special(p)$; moreover, if the right (resp. left) child of q does not contain $edge^p$, then we label it with c_r (resp., c_ℓ). If both children of q do not contain $edge^p$, then we label all left descendants of q with $special(p)$—in a way, we assume that if the right child of q does not contain $edge^p$, then the left child does. Clearly, if q has one child in Touch_p, then we assume that it contains $edge^p$.

It holds that $c_r, c_\ell \in triple(p) \setminus \{special(p)\}$ but they are not necessarily distinct. Let Q be the set of all elements in Touch_p that contain $edge^p$. We observe that Q forms a path in TreeTouch_p. In few words, function `SpecialLabel` calls `RightLabel` for paths on the left of Q and `LeftLabel` for paths on the right of Q (see Figure 7.10). The recursive calls of `SpecialLabel` turn the labels in order to obtain the following lemma.

Lemma 77. *Let $p \in$ **Max**, let $c_r, c_\ell \in triple(p) \setminus \{special(p)\}$ be not necessarily distinct and let $q \in Touch_p$ satisfy $edge^p \in q$. If we call `SpecialLabel`$(p, q, (c_r, c_\ell))$, then $\mathcal{L}(R(q)) \subseteq \{c_r, special(p)\}$ and $\mathcal{L}(L(q)) \subseteq \{c_\ell, special(p)\}$.*

Proof. If q has only one child in TreeTouch_p, then function `SpecialLabel` does the same call to its child, thus we can assume that q has two children in TreeTouch_p. W.l.o.g., we assume that $edge^p \in q_r$, indeed, if $edge^p \notin q_r$, then the proof is symmetric. Thus function `SpecialLabel`

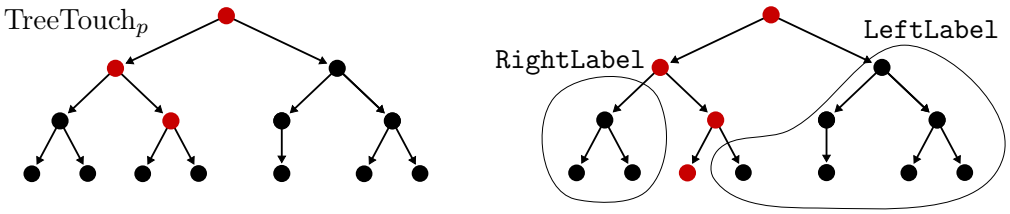


Figure 7.10: on the left TreeTouch$_p$ in which red vertices correspond to paths containing $edge^p$. On the right, red vertices correspond to paths labeled with $special(p)$. We show for which vertices `SpecialLabel` calls `RightLabel` and for which `LeftLabel`.

`SpecialLabel`$(p, q, (c_r, c_\ell))$:

Input: a path $p \in$ **Max**, a path $q \in Touch_p$, two labels $c_r, c_\ell \in triple(p) \setminus \{special(p)\}$

Output: assign $\mathcal{L}(q') \in triple(p)$ for all $q' \in$ Touch$_p$ such that $q' \preceq q$ so that $\mathcal{L}(q'') = special(p)$ for all $q'' \in$ Touch$_p$ containing $edge^p$

1 $\mathcal{L}(q) \leftarrow special(p)$;
2 **if** *q has one child q' in $Touch_p$* **then** `SpecialLabel`$(p, q', (c_r, c_\ell))$;
3 **if** *q has two children in $Touch_p$* **then**
4 **if** $edge^p \in q_r$ **then**
5 `SpecialLabel`$(p, q_r, (c_r, triple(p) \setminus \{c_\ell, special(p)\}))$;
6 `RightLabel`$(p, q_\ell, (c_\ell, special(p), triple(p) \setminus \{c_\ell, special(p)\}))$;
7 **else**
8 `SpecialLabel`$(p, q_\ell, (triple(p) \setminus \{c_r, special(p)\}, c_\ell))$;
9 `LeftLabel`$(p, q_r, (c_r, special(p), triple(p) \setminus \{c_r, special(p)\}))$;

---

does the call `SpecialLabel`$(p, q_r, (c_r, triple(p) \setminus \{c_\ell, special(p)\}))$ and the call `RightLabel`$(p, q_\ell, (c_\ell, special(p), triple(p) \setminus \{c_\ell, special(p)\})$. The second call implies that $\mathcal{L}(L(q)) \subseteq \{c_\ell, special(p)\}$ by Lemma 67. It remains to show that $\mathcal{L}(R(q)) \subseteq \{c_r, special(p)\}$.

Let w be the first element w.r.t. $\preceq$ in $R(q)$ such that w_r does not contain $edge^p$, i.e., $edge^p \notin w_r$ and $edge^p \in z$ for all $z \in R(q) \setminus R(w_r)$. Thus all elements in $R(q)$ before (w.r.t. $\preceq$) w_r are labeled with $special(p)$ because of Line 1. Moreover, by Line 7 we call `LeftLabel`$(p, w_r, (c_r, special(p), triple(p) \setminus \{c_r, special(p)\}))$; indeed, we note that the third element (i.e., c_r) in all these recursive calls of `SpecialLabel` does not change because of Line 5. Finally, Lemma 76 implies that $\mathcal{L}(R(w_r)) \subseteq \{c_r, special(p)\}$, thus $\mathcal{L}(R(q)) \subseteq \{c_r, special(p)\}$ as we claimed. □

Proposition 78. *Let $p \in$ **Max**, if we call `LabelSpecialTouch`(p), then every face of type I for p is solved by $\mathcal{L}$, $\mathcal{L}(q) \in triple(p)$ for all paths in $Touch_p$ and $\mathcal{L}(q') = special(p)$ for all paths in $Touch_p$ containing $edge^p$.*

Proof. By the recursion of function `LabelSpecialTouch`, it is clear that all paths in Touch$_p$ are labeled with labels in $triple(p)$ and all paths in Touch^p containing $edge^p$ are labeled with $special(p)$. It remains to prove that every face of type I for p is solved by $\mathcal{L}$. We need the following preliminary claim.

f) Let $c_r, c_\ell \in \mathit{triple}(p)$ be not necessarily distinct and let f be a face of type I for p such that the upper path q of f is in Touch_p. If we call $\texttt{SpecialLabel}(p, q, (c_r, c_\ell))$, then f is solved by $\mathcal{L}$.

Proof of f): It holds that q contains edge^p and one of its children contains edge^p (we have assumed that if the right child does not contain edge^p, then the left child does). W.l.o.g., q_r contains edge^p. Thus $\mathcal{L}(q_r) = \mathit{special}(p)$, $\mathcal{L}(q_\ell) = c_\ell$, we call $\texttt{SpecialLabel}(p, q_r, (c_r, \mathit{triple}(p) \setminus \{c_\ell, \mathit{special}(p)\}))$ and $\texttt{RightLabel}(p, q_\ell, (c_\ell, \mathit{special}(p), \mathit{triple}(p) \setminus \{c_\ell, \mathit{special}(p)\}))$.

By Lemma 66, $\mathcal{L}(e_r^f) \in \mathcal{L}(L(q_r))$ and $\mathcal{L}(e_\ell^f) \in \mathcal{L}(R(q_\ell))$. By Lemma 67 and by calling $\texttt{RightLabel}(p, q_\ell, (c_\ell, \mathit{special}(p), \mathit{triple}(p) \setminus \{c_\ell, \mathit{special}(p)\}))$, it holds that $\mathcal{L}(R(q_\ell)) \subseteq \{c_\ell\}$. Moreover, by calling $\texttt{SpecialLabel}(p, q_r, (c_r, \mathit{triple}(p) \setminus \{c_\ell, \mathit{special}(p)\}))$ and by Lemma 77, $\mathcal{L}(L(q_r)) \subseteq \{(\mathit{triple}(p) \setminus \{c_\ell, \mathit{special}(p)\}), \mathit{special}(p)\} = \mathit{triple}(p) \setminus \{c_\ell\}$. Hence f is solved by $\mathcal{L}$ because $\mathcal{L}(e_r^f) \cap \mathcal{L}(e_\ell^f) = \emptyset$. End proof of f).

Now let f be a face of type I for p and let q be its upper path. Because of the call $\texttt{LabelSpecialTouch}(p)$, we call either $\texttt{RightLabel}(p, q, (c_1, c_2))$ or $\texttt{LeftLabel}(p, q, (c_1, c_2))$ or $\texttt{SpecialLabel}(p, q, (c_r, c_\ell))$ for some distinct $c_1, c_2 \in \mathit{triple}(p)$ and not necessarily distinct $c_r, c_\ell \in \mathit{triple}(p) \setminus \{\mathit{special}(p)\}$. If one among the first two cases applies, then f is solved by $\mathcal{L}$ by applying Lemma 67 or Lemma 76 and the same reasoning of Proposition 68's proof. If the last case applies, then f is solved by $\mathcal{L}$ because of f). □

`LabelSpecialTouch`(p):

Input: a path p in **Max**

Output: assign $\mathcal{L}(q) \in \mathit{triple}(p)$ for all $q \in \text{Touch}_p$ so that every face of type I for p is solved by $\mathcal{L}$ and $\mathcal{L}(q') = \mathit{special}(p)$ for all $q' \in \text{Touch}_p$ containing edge^p

1 Let c_r, c_ℓ in $\mathit{triple}(p) \setminus \{\mathit{special}(p)\}$ be not necessarily distinct;

2 `SpecialLabel` $(p, p, (c_r, c_\ell))$; `// assign` $\mathcal{L}(q) \in \mathit{triple}(p)$ `for all` $q \in \text{Touch}_p$

The following corollary is a consequence of Lemma 67, Lemma 76 and Lemma 77 and is crucial in the proof of Lemma 80 in the next subsection.

Corollary 79. *Let $p \in$ **Max**. If we call `LabelSpecialTouch`(p), then $|\mathcal{L}(R(q))| \leq 2$ and $|\mathcal{L}(L(q))| \leq 2$ for all $q \in \mathit{Touch}_p$.*

7.5.3 Dealing with faces of type II and type III

In algorithm `FifteenForests` we use many labels, and we assign $\mathit{triple}(q)$ for each $q \in \text{Max}_p$ so that $\mathit{triple}(q) \cap \mathit{triple}(p) = \emptyset$, ignoring paths in Touch_p because they are labeled with labels in $\mathit{triple}(p)$, where $p \in$ **Max**. In this section, arguing with algorithm `FourForests`, we have only four labels available, and the previous request is impossible to satisfy; indeed, two distinct triples in $\{1, 2, 3, 4\}$ have non empty intersection. Thus we have to consider also paths in Touch_p. In function `TripleSpecialMax` we assign $\mathit{triple}(q)$ for each $q \in \text{Max}_p$ by visiting each tree T in

$(\mathrm{Max}_p^{III}, \dashrightarrow)$, and in the following lemma we show that all paths in Touch_p that interfere with a face related to T have at most two labels.

Lemma 80. *Let $p \in \boldsymbol{Max}$, let T be a tree in the forest $(Max_p^{III}, \dashrightarrow)$ and let $\Delta = C\big(\{q \in Touch_p \,|\, q$ interferes with f_m for some $m \in V(T)\}\big)$. If we call `LabelSpecialTouch`(p), then $|\Delta| \leq 2$.*

Proof. For convenience, let $Q = \{q \in \mathrm{Touch}_p \,|\, q \text{ interferes with } f_m \text{ for some } m \in V(T)\}$. Thanks to Corollary 79, it suffices to prove that either $Q \subseteq R(q)$ or $Q \subseteq L(q)$ for some $q \in \mathrm{Touch}_p$. For each $m \in V(T)$ we denote by q_m its parent in T_g. Let m^{root} be the root of T and, w.l.o.g., we assume that m^{root} is the right child in T_g of $q_{m^{\mathrm{root}}}$. For the sake of clarity the proofs of g), h) and i) are at the end of the main proof.

g) let $m, m' \in V(T)$ satisfy $m \dashrightarrow m'$. If m is the right child of q_m in T_g, then m' is the right child of $q_{m'}$ in T_g.

Because of g) and being m^{root} the right child of $q_{m^{\mathrm{root}}}$, given any $m \in V(T)$, then m is the right child of its parent q_m in T_g. For each $m \in V(T)$ we define P_m as the path in $V(T)$ from the root m^{root} to m. Moreover, for each $m \in V(T)$, let $Q_m = \{q \in \mathrm{Touch}_p \,|\, q \text{ interferes with } f_m\}$ and $Q_{P_m} = \bigcup_{m' \in P_m} Q_{m'}$. We need other two statements:

h) for each $m \in V(T)$ it holds that $Q_m \subseteq R(q_m)$,

i) for each $m \in V(T)$ it holds that $Q_{P_m} \subseteq R(q_m)$.

Let $w^{\mathrm{root}} \in \mathrm{Touch}_p$ be the child of $q_{m^{\mathrm{root}}}$ different from m^{root}. Being m^{root} in Max_p^{III} then w^{root} interferes with $f_{m^{\mathrm{root}}}$, hence, $w^{\mathrm{root}} \in \bigcap_{m \in V(T)} Q_{P_m}$. By i), $w^{\mathrm{root}} \in R(q_m)$ for every $m \in V(T)$, consequently we can order all elements in $V(T)$ in $(m_1, m_2, \ldots, m_{|V(T)|})$ so that $q_{m_1} \in R(q_{m_2}) \subseteq R(q_{m_3}) \subseteq \ldots \subseteq R(q_{m_{|V(T)|}})$; otherwise there would be a vertex in $\mathrm{TreeTouch}_p$ with two incoming darts because of the definition of right descendant in Definition 64. It is easy to see that $m_{|V(T)|}$ is a leaf in T. Finally, $Q = \bigcup_{m \in V(T)} Q_m \subseteq \bigcup_{m \in V(T)} Q_{P_m} \subseteq R(q_{m_{|V(T)|}})$ by above reasoning and i), and the thesis holds. To complete the proof, now we prove the previous three claims.

Proof of g): let w be the child of q_m different from m and let w' be the child of $q_{m'}$ different from m'. By the single-touch property, $q_{m'}$ splits w' into three parts: $w' = r \circ q_{m'} \cap w' \circ \ell$, with $x_{w'} \in r$ and $y_{w'} \in \ell$. Let e_r (resp., e_ℓ) be the extremal edge of r (resp., ℓ) that does not contain $x_{w'}$ (resp., $y_{w'}$). All these paths and edges are shown in Figure 7.11.

Being $m \dashrightarrow m'$, then either $e_r \in m$ or $e_\ell \in m$; indeed, one among e_r and e_ℓ is an extremal edge of the lower boundary of $f_{m'}$. By Remark 65 applied to w' and $q_{m'}$, the thesis holds if $e_r \in m$. We note that if $e_\ell \in m$, then w_m does not intersect p because of Remark 65 and because m is the right child of q_m in T_g. This is absurdum because $w_m \in \mathrm{Touch}_p$. Thus, by above, $e_r \in m$ and the thesis is proved. End proof of g).

Proof of h): being m the right child of q_m, if $q_{m'} \in \mathrm{Touch}_p$ interferes with f_m, then $e_r^{f_m}$ belongs to $q_{m'}$. Thus all paths in Q_m contain $e_r^{f_m}$ and they can be ordered w.r.t. $\preceq$. Hence $Q_m =$

$\{q_1, q_2, \ldots, q_z\}$ so that $q_{i+1} \preceq q_i$ for all $i \in [z-1]$. Now we can proceed by induction on i. We note that q_1 is the child of q_m in T_g different from m and $q_z \in R(q_m)$ because q_z is the only child in TreeTouch$_p$. Therefore, we have proved the base case. The induction case is trivial if q_i has only one child in TreeTouch$_p$, otherwise it follows from the same reasoning of g). End proof of h).

Proof of i): let $(m_1, m_2, \ldots, m_w)$ be the ordered sequence of vertices in P_m. Then $Q_{m_i} \subseteq R(q_{m_i})$ because of h). Moreover, being $m_i \dashrightarrow m_{i+1}$, then $q_{m_i} \in Q_{m_{i+1}}$ (this is formally proved in c) in Lemma 72). Hence for any $i \in [w]$, $Q_{m_i} \in R(q_{m_w})$ by applying repeatedly as stated, and the thesis follows. End proof of i).

The proof is now complete. □

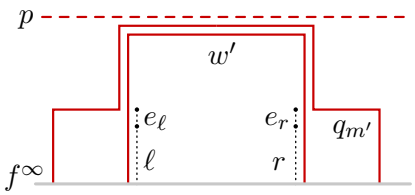


Figure 7.11: paths and edges used in the proof of g). In red continuous line paths w' and $q_{m'}$, in black dotted lines path r and ℓ, and we point out edges e_r and e_ℓ.

The following result explains the effects of function `TripleSpecialMax`. We recall that, given $m \in \mathbf{Max} \setminus \{p_1\}$, $f_m \in F$ denotes the face whose upper path is the parent of m in T_g.

Proposition 81. *Let $p \in \boldsymbol{Max}$, let f be a face of type II for p, let g be a face of III for p and let $m \in Max_p^{III}$ satisfy $g = f_m$. If we call `LabelSpecialTouch`(p) and `TripleMax`(p) then*

81.(1) $special(m_r^f) \neq special(m_\ell^f)$;

81.(2) let $q \in Touch_p$ interfere with g. Then $special(m) \neq \mathcal{L}(q)$;

81.(3) let $m' \in Max_p \setminus \{m\}$ interfere with g. Then $special(m) \notin triple(m')$.

Proof. The first statement is a consequence of the for cycle in Line 1. Let T be the tree in $(\mathrm{Max}_p^{III}, \dashrightarrow)$ containing m and let Δ be as defined in Line 8. The second statement is implied by the for cycle in Line 7, indeed, $\mathcal{L}(q) \in \Delta$ and $special(m) \notin \Delta$ because of Line 11.

It remains to prove the third statement. There are two cases: $m' \in \mathrm{Max}_p^{II}$ and $m' \in \mathrm{Max}_p^{III}$. If the former case applies, then m is the root of T. Thus $triple(m) = triple(p)$ because of Line 5 or Line 6 and $special(m) \notin triple(p)$ because of Line 10 that assigns the class A to the root of T. If the latter case applies, then $m' \dashrightarrow m$. Hence, because of Line 10, $m \in A$ and $m' \in B$, or vice-versa. Finally, $special(m) \notin triple(m')$ by construction of the for cycles in Line 12 and Line 15. □

7.5.4 Correctness of algorithm `FourForests`

We prove in Theorem 82 the correctness of algorithm `FourForests` shown in Subsection 7.5.1; the proof is analogous to Theorem 74's proof. As a consequence we have Corollary 83 which states that every NCSP has Path Covering with Forests Number at most 4.

`TripleSpecialMax`(p):

Input: a path p in **Max**

Output: an assignment of $triple(q)$ and $special(q)$ for all $q \in \mathrm{Max}_p$

for each *face* f *of type II for* p **do**
  Let $label_r, label_\ell \in triple(p)$ be distinct;
  $special(m_r^f) \leftarrow label_r$;
  $special(m_\ell^f) \leftarrow label_\ell$;
  $triple(m_r^f) \leftarrow triple(p)$;
  $triple(m_\ell^f) \leftarrow triple(p)$;
for each *tree* T *in the forest* $(Max_p^{III}, \dashrightarrow)$ **do**
  Let $\Delta = C\big(\{q \in \mathrm{Touch}_p \mid q \text{ interferes with } f_m \text{ for some } m \in V(T)\}\big)$;
  If $|\Delta| = 1$, then add to Δ another label of $triple(p)$;
  Bipartite the tree T into the two classes A and B so that the root of T is in A and $u \not\dashrightarrow v$ for all u, v in the same class;
  Let $label_A = \{1,2,3,4\} \setminus \{triple(p)\}$ and $label_B = triple(p) \setminus \Delta$;
  for each $m \in A$ **do**
    $special(m) \leftarrow label_A$;
    $triple(m) \leftarrow \Delta \cup \{label_A\}$
  for each $m \in B$ **do**
    $special(m) \leftarrow label_B$;
    $triple(m) \leftarrow triple(p)$

Theorem 82. *Given a NCSP P, algorithm* `FourForests` *produces a forest labeling $\mathcal{L}$ of P which uses at most 4 labels.*

Proof. Thanks to Theorem 60 we only need to prove that every faces in $\bigcup_{p \in P} p$ is solved by $\mathcal{L}$. Let f be a face, then there exists $p \in$ **Max** such that f is a face of type I, or type II or type III for p.

If f is a face of type I for p, then f is solved by $\mathcal{L}$ because of Proposition 78 and the call `LabelSpecialTouch`(p).

If f is a face of type II for p, then $e_r^f \in m_r^f$ and $e_\ell^f \in m_\ell^f$. Moreover, if a path $q \in P$ interferes with f, then either $q \in \mathrm{Touch}_{m_r^f}$ or $q \in \mathrm{Touch}_{m_\ell^f}$. Thus f is solved by $\mathcal{L}$ because of 81.(1) and the calls `LabelSpecialTouch`(m_r^f) and `LabelSpecialTouch`(m_ℓ^f).

If f is a face of type III for p, then either $e_r^f \in m^f$ or $e_\ell^f \in m^f$, where m^f is the unique child of q belonging to Max_p. W.l.o.g., we assume that $e_r^f \in m^f$. We observe that e_ℓ^f may belong to some paths in Touch_p and at least one in Max_p. If there does not exist any path $m' \in \mathrm{Max}_p$ satisfying $e_\ell^f \in m'$, then f is solved by $\mathcal{L}$ because of 81.(2) and the call `LabelSpecialTouch`(m^f). Otherwise, let $m' \in \mathrm{Max}_p$ satisfy $e_\ell^f \in m'$. Then f is solved by $\mathcal{L}$ because of 81.(2), 81.(3) and the calls `LabelSpecialTouch`(m^f) and `LabelSpecialTouch`(m'). □

Corollary 83. *Let P be a set of non-crossing shortest paths in a plane graph G whose extremal vertices lie on the infinite face of G. Then the Path Covering with Forests Number of P is at most 4.*

7.6 Four forests are necessary

In this section we prove that in the general case covering a NCSP P may require four forests. As a consequence, the result in Theorem 82 is tight.

Theorem 84. *There exists a NCSP P such that $PCFN(P) = 4$.*

Proof. The NCSP P of this proof is built recursively.

A) For each k there exists a NCSP P_k such that:

- $P_k = \text{Touch}_{p_1}$, where p_1 is the path corresponding to the root of the genealogy tree;
- the genealogy tree of P_k is a complete binary tree composed by $2^k - 1$ paths;
- if f is a face of type I for p, then the lower boundary of f has exactly two edges, i.e., the lower boundary consists of e_r^f and e_ℓ^f;
- for each face f of type I for p_1, it holds that $e_r^f \in q'$, for all $q' \in L(q_r^f)$, and $e_\ell^f \in q''$, for all $q'' \in R(q_\ell^f)$ (we recall that q_r^f and q_ℓ^f are the right child and the left child, respectively, of the upper path of f, see Definition 56).

The construction of P_k can be obtained by generalizing P_3 in Figure 7.12. We consider P_{13}.

For each leaf w of P_6 let $R(w) = \{r_1^w, \dots, r_8^w\}$ be ordered so that $r_{i+1}^w \preceq r_i^w$, for all $i \in [7]$. Similarly, let $L(w) = \{\ell_1^w, \dots, \ell_8^w\}$ be ordered so that $\ell_{i+1}^w \preceq \ell_i^w$, for all $i \in [7]$ (note that $r_1^w = \ell_1^w = w$). For all leaf w of P_6, we add three paths q_1^w, q_2^w, q_3^w whose extremal vertices are in the subpath of the infinite face between $y_{\ell_1^w}$ and $y_{\ell_8^w}$ according to Figure 7.13. Similarly, we add three paths p_1^w, p_2^w, p_3^w whose extremal vertices are in the subpath of the infinite face between $x_{r_1^w}$ and $x_{r_8^w}$ according to a symmetric version of Figure 7.13.

In this way we obtain a NCSP P composed by $6 \cdot 2^5 + 2^{13} - 1 = 8383$ paths. Let us assume by contradiction that there exists a forest labeling $\mathcal{L} : P \mapsto [3]$. We say that a face f is *unsolved by* $\mathcal{L}$ if $\bigcap_{e \in E(\partial f)} \mathcal{L}(e) \neq \emptyset$. To finish the proof it suffices to prove that there exists a face f in $\bigcup_{p \in P} p$ unsolved by $\mathcal{L}$.

B) There exist three paths $q, q', q'' \in P_{13}$ such that $q \preceq q' \preceq q''$, $|\mathcal{L}(\{q, q', q''\})| = 3$ and q is a leaf of P_4.

C) Let f, f_1, f_2 be the faces in P_{13} whose upper paths are q, q_r, q_ℓ, respectively. If e is an extremal edge of the lower boundary of a face g in $\{f, f_1, f_2\}$ then $|\mathcal{L}(e)| < 3$, otherwise g would be unsolved by $\mathcal{L}$ because of B).

D) Let $q_{r,\ell}$ be the left child of q_r and let $q_{\ell,r}$ be the right child of q_ℓ. By C), for some distinct $c_1, c_2 \in [3]$ it happens

- $\mathcal{L}(R(q_{r,\ell})) = \{c_1\}$ and $\mathcal{L}(L(q_{r,\ell})) = \{c_1, c_2\}$, or

- $\mathcal{L}(L(q_{r,\ell})) = \{c_1, c_2\}$ and $\mathcal{L}(L(q_{r,\ell})) = \{c_1\}$, or
- $\mathcal{L}(R(q_{\ell,r})) = \{c_1\}$ and $\mathcal{L}(L(q_{\ell,r})) = \{c_1, c_2\}$, or
- $\mathcal{L}(L(q_{\ell,r})) = \{c_1, c_2\}$ and $\mathcal{L}(L(q_{\ell,r})) = \{c_1\}$,

indeed, if no one of the previous one applies, then

- $|\mathcal{L}(R(q_{r,\ell}))| = 1$ and $|\mathcal{L}(L(q_{r,\ell}))| = 1$, which in turn implies $\mathcal{L}(R(q_{r,\ell})) = \mathcal{L}(L(q_{r,\ell})) = \mathcal{L}(q_{r,\ell})$ and thus the face f_2 is unsolved by $\mathcal{L}$, absurdum, or
- $|\mathcal{L}(R(q_{\ell,r}))| = 1$ and $|\mathcal{L}(L(q_{\ell,r}))| = 1$, similar to the previous case, or
- $|\mathcal{L}(R(q_{r,\ell}))| = |\mathcal{L}(L(q_{r,\ell}))| = |\mathcal{L}(R(q_{r,\ell}))| = |\mathcal{L}(L(q_{r,\ell}))| = 2$, and thus f is unsolved by $\mathcal{L}$ because of B) and C), absurdum.

E) By D), there exists a leaf w of P_6 such that either $\mathcal{L}(R(w)) = \{c\}$ and $\mathcal{L}(L(w)) = \{c, c'\}$ for some distinct $c, c' \in [3]$, or $\mathcal{L}(L(w)) = \{c\}$ and $\mathcal{L}(R(w)) = \{c, c'\}$. Let us assume that the latter case applies.

F) Now, A) and B) imply that every path w' in P_{13} such that $w' \preceq w$ is labeled according to function `LeftLabel`$(p_1, w, (c, c', c''))$, where $c'' = \{1, 2, 3\} \setminus \{c, c'\}$. Hence, $\mathcal{L}(\{\ell_2^w, \ell_4^w, \ell_6^w, \ell_8^w\}) = \{c'\}$ and $\mathcal{L}(\{\ell_2^w, \ell_4^w, \ell_6^w, \ell_8^w\}) = \{c\}$.

G) Let g_1, g_2, g_3 be the faces generated by paths q_1^w, q_2^w, q_3^w as depicted in Figure 7.13. By F), if $\mathcal{L}(q_1^w) \in \{c, c'\}$, then f_1 is unsolved by $\mathcal{L}$, thus $\mathcal{L}(q_1^w) = c''$. Similarly, $\mathcal{L}(q_2^w) = c''$ and $\mathcal{L}(q_3^w) = c''$. Finally, $c'' \in \bigcap_{e \in E(g_3)} \mathcal{L}(e)$, and thus g_3 is unsolved by $\mathcal{L}$, absurdum. □

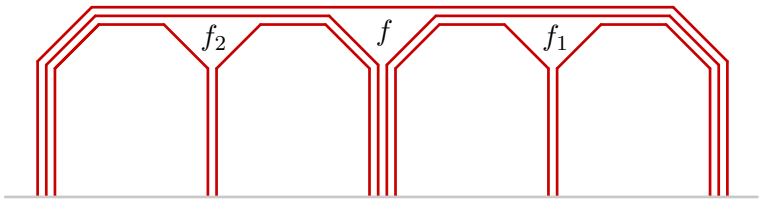


Figure 7.12: the NCSP P_3 and faces f, f_1, f_2.

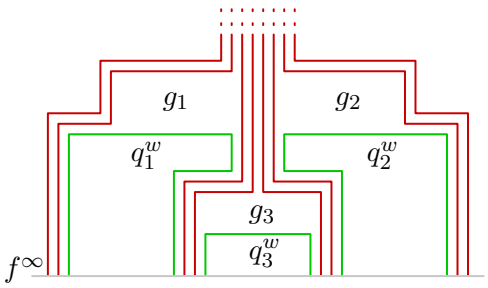


Figure 7.13: paths q_1^w, q_2^w and q_3^w,

7.7 Covering with four forests in linear time

We proved that given a NCSP P algorithms `FourForests` gives us a forest labeling $\mathcal{L}$ of P which uses at most 4 labels. In this section we prove that the four forests induced by $\mathcal{L}$ can be obtained in $O(|E(U)|)$ time. To reach this goal we use a multiple labeling $\widetilde{C} : E(U) \to 2^{[4]}$ which assigns a list of at most four labels to each edge of U.

Theorem 85. *Let P be a NCSP and let $\mathcal{L}$ be a forest labeling of P which uses at most 4 labels. We can obtain a multiple labeling $\widetilde{C} : E(U) \to 2^{[4]}$ satisfying $\mathcal{L}(q) \in \widetilde{C}(e)$ for all $q \in P$ and $e \in q$ in $O(|E(U)|)$ time.*

Proof. We obtain $\widetilde{C}$ through $\mathcal{L}$ given by algorithm `FourForests`. For convenience and readability, we proceed step by step.

A) In Chapter 4, in particular Remark 20, we give an implicit representation of non-crossing shortest paths, and we describe the intersection between each path and its children in the genealogy tree. We can also list each path p in $O(\max\{\ell, \ell \log\log(\frac{k}{\ell})\})$ where ℓ is the number of edges in p and $k = |P|$. But if we know the edges in p, then we can list the edges of a child q of p without visiting again the edges in $q \cap p$ because of the implicit and compact representation. In algorithm `FourForests`, we visit the paths by passing from parent to children and so on, thus the listing can be made in total linear time, because we do not visit already visited edges.

B) Each edge e is visited in at most two iterations of cycle in Line 6 of algorithm `FourForests`. Indeed, let $p \in \mathbf{Max}$ be such that $e \in \bigcup_{q \in \mathrm{Touch}_p \cup \mathrm{Max}_p} q$ and $e \notin \bigcup_{q \in \mathrm{Touch}_{p'} \cup \mathrm{Max}_{p'}} q$ for all $p \preceq p'$. It holds that there exist at most two paths m_1, m_2 in Max_p containing e. In this way e appears only in the iterations i and $i+1$ assuming that $p \in \mathbf{Max}_i$.

C) By B) to finish the proof it suffices to show that, for each $p \in \mathbf{Max}$, `LabelSpecialTouch`(p) and `TripleSpecialMax`(p) can both be executed in $O(|E(\bigcup_{q \in \mathrm{Max}_p \cup \mathrm{Touch}_p} q)|)$. This trivially holds for `TripleSpecialMax`(p) that can be executed by visiting each path in $\mathrm{Max}_p \cup \mathrm{Touch}_p$ a constant number of times. Hence we focus on `LabelSpecialTouch`(p) which works only with paths in Touch_p.

D) Let $p \in \mathrm{Max}$, we update $\widetilde{C}(e)$ for all $e \in q$ as soon as $\mathcal{L}(q)$ is assigned, for each $q \in \mathrm{Touch}_p$. In this way each path is visited exactly one time, but some paths may share edges, hence we have to pass on the same edge at most a constant number of times.

E) Let $q, q_1, q_2 \in \mathrm{Touch}_p$, so that $q \preceq q_1 \preceq q_2$. The single-touch and non-crossing properties imply that $q_2 \cap q \subseteq q_1 \cap q$, thus as soon as $\mathcal{L}(q)$ is assigned, then there are four vertices v_1, v_2, v_3, v_4 not necessarily distinct and three labels $c_1, c_2, c_3 \in \mathit{triple}(p)$ such that

- the edges of the subpath of q from x_q to v_1 are labeled by $\widetilde{C}$ with c_1;
- the edges of the subpath of q from v_1 to v_2 are labeled by $\widetilde{C}$ with c_1, c_2;
- the edges of the subpath of q from v_2 to v_3 are labeled by $\widetilde{C}$ with c_1, c_2, c_3;
- the edges of the subpath of q from v_3 to v_4 are labeled by $\widetilde{C}$ with c_1, c_2;
- the edges of the subpath of q from v_4 to y_q are labeled by $\widetilde{C}$ with c_1.

In this way, by knowing v_1, v_2, v_3, v_4, we can update $\widetilde{C}(e)$ for all $e \in q'$, where q' is a child of q, with the value $\mathcal{L}(q')$ without passing on edges of q already labeled with $\mathcal{L}(q')$, see also the discussion in A). Thus we can update $\widetilde{C}$ so that each edge in $\bigcup_{q \in \text{Touch}_p} q$ is visited at most three times, i.e., one for each label in $\mathit{triple}(p)$. □

Chapter 8

Two new characterizations of path graphs

We start from the characterization of path graphs by Monma and Wei [102] and we reduce it to some 2-coloring subproblems, obtaining the first characterization that directly leads to a polynomial recognition algorithm. Then we exhibit a list of minimal forbidden 2-edge colored subgraphs in each of the attachedness graph.

8.1 Introduction

In this chapter we describe two new characterizations of path graphs. Our first characterization is the only one, to the best of our knowledge, that directly implies a polynomial recognition algorithm, and we obtain it by starting from Monma and Wei's characterization [102]. The second characterization follows from the first one and consists in a list of minimal forbidden subgraphs and a list of minimal forbidden induced subgraphs on the attachedness graph. Now we describe in detail the two new characterizations.

In the characterization by Monma and Wei [102], described in Section 3.5, the graph is decomposed recursively by clique separators and in every decomposition step one has to solve a coloring problem (see Theorem 9); the solution of the coloring problem is used to represent the graph as the intersection graph of paths in a tree. The difficulty with their coloring problem led them to not prove that it can be solved in polynomial time. In our first characterization we simplify Monma and Wei's characterization by reducing it to some 2-coloring subproblems that are clearly solvable in polynomial time (see Section 8.3, in particular Subsection 8.3.1). Thus this characterization also describes a polynomial recognition algorithm. Moreover, it has two consequences.

• The obstructions to our 2-coloring subproblems are well-known, this allows us to give our second characterization, which consists in a list of minimal forbidden subgraphs and a list of minimal forbidden induced subgraphs of the attachedness graphs of the input graph (see

Section 8.4, in particular Theorem 102).

• Our first characterization is used in Chapter 9 to describe a recognition algorithm that specializes for path graphs and directed path graphs. We refer to Chapter 9 for further details.

8.2 A *strong* coloring

In this section we briefly resume the characterization in [102] about path graph already explained in Section 3.5. We also give it in a recursive fashion (see Corollary 87).

Given a clique separator C of G, in the following definition we say when G is *strong C-colorable*, that corresponds to the conditions stated in Theorem 9. We use the term "strong" because in Section 8.3 we introduce a weaker coloring and we prove that they are equivalent.

Definition 86. *Let C be a clique separator of G, we say that G is* strong C-colorable *if there exists $f : \Gamma_C \to [s]$ such that:*

86.(1) if $\gamma \leftrightarrow \gamma'$, then $f(\gamma) \neq f(\gamma')$;

86.(2) if $\{\gamma, \gamma', \gamma''\}$ is a neighboring triple, then $|f(\{\gamma, \gamma', \gamma''\})| \leq 2$.

We refer to a coloring f satisfying the conditions of Definition 86 as a *strong C-coloring*. In the following corollary, we translate Theorem 9 from a recursive fashion to a local one that is more useful to our purposes by using Definition 86. We recall that a graph with no clique separators (i.e., an atom) is a path graph if and only if it is chordal.

Corollary 87. *A chordal graph G is a path graph if and only if G is strong C-colorable, for all clique separator C of G.*

By Corollary 87, deciding whether a graph G is a path graph is as difficult as deciding whether G is strong C-colorable for each separator C. It is thus natural to wonder whether there are obstructions to strong C-colorability and, if so, how do such obstructions look like on the attachedness graph of G. One such obstruction is easily recognized (see [102]): let $\{\gamma, \gamma', \gamma''\} \subseteq \Gamma_C$ be a neighboring triple and suppose that γ, γ' and γ'' are pairwise antipodal (hence $\{\gamma, \gamma', \gamma''\}$ induces a triangle on the C-antipodal graph of G). We refer to any such triple as a *full antipodal triple*. It is clear that if Γ_C contains a full antipodal triple, then G is not strong C-colorable because the two conditions in Definition 86 cannot be both satisfied. For later reference we formalize this easy fact in a lemma.

Lemma 88. *Let C be a clique separator of G. If G is strong C-colorable, then Γ_C has no full antipodal triples.*

8.3 A *weak* coloring

In this section we introduce a *weak coloring* that is used in Theorem 93 to give our first characterization of path graphs. This characterization simplifies Monma and Wei's one [102] from

an algorithmic point of view justifying the terms "strong" and "weak". This simplification is explained in Subsection 8.3.1. In Subsection 8.3.2 we prove Theorem 93.

Dominance is a reflexive and transitive relation. Hence $(\Gamma_C, \leq)$ is a preorder. We assume that such a preorder is in fact a partial order. The latter assumption is not restrictive as shown implicitly by Schäffer [127] and explicitly as follows. Let $\sim$ be the equivalence relation on Γ_C defined by $\gamma \sim \gamma' \Leftrightarrow (\gamma \leq \gamma' \wedge \gamma' \leq \gamma)$, i.e., $\sim$ is the standard equivalence relation associated with a preorder. If $\gamma \sim \gamma'$ for some two $\gamma, \gamma' \in \Gamma_C$, then for any $\gamma'' \in \Gamma_C$ it holds that $\gamma \leftrightarrow \gamma''$ if and only if $\gamma' \leftrightarrow \gamma''$. Analogously, if $\gamma \sim \gamma'$ for some two $\gamma, \gamma' \in \Gamma_C$, then for any $\gamma'' \in \Gamma_C$ it holds that $\gamma \leq \gamma''$ if and only if $\gamma' \leq \gamma''$. Moreover, if γ is a neighboring of v, for some $v \in C$, and $\gamma \sim \gamma'$, then γ' is a neighboring of v.

The following lemma shows that it is not restrictive to assume that $\Gamma_C = \Gamma_C / \sim$.

Lemma 89. *Let C be a clique separator of G. If there exists $f : \Gamma_C / \sim \to [m]$ satisfying 86.(1) and 86.(2), then G is strong C-colorable.*

Proof. Let $\widetilde{f} : \Gamma_C \to [m]$ be defined by $\widetilde{f}(\gamma) = f([\gamma]_\sim)$. It holds that $\widetilde{f}$ satisfies 86.(1) and 86.(2), hence $\widetilde{f}$ is a strong C-coloring. □

After the lemma, we assume that $(\Gamma_C, \leq)$ is a partial order for every clique separator C of G. In other words, we assume $\Gamma_C = \Gamma_C / \sim$.

Given a clique separator C of G, we define $\text{Upper}_C = \{u \in \Gamma_C \mid u \not\leq \gamma, \text{ for all } \gamma \in \Gamma_C\}$ the set of *upper bounds* of Γ_C with respect to $\leq$. From now on we fix $(u_1, u_2, \ldots, u_\ell)$ an ordering of Upper_C and for all $i, j \in [\ell]$ and $i < j$ we define

$$D_i^C = \{\gamma \in \Gamma_C \mid \gamma \leq u_i \text{ and } \gamma \not\leq u_j, \ \forall j \in [\ell] \setminus \{i\}\}; \tag{8.1}$$

$$D_{i,j}^C = \{\gamma \in \Gamma_C \mid \gamma \leq u_i, \gamma \leq u_j \text{ and } \gamma \not\leq u_k, \ \forall k \in [\ell] \setminus \{i, j\}\}; \tag{8.2}$$

$$\mathcal{D}^C = \left\{D_i^C \mid i \in [\ell]\right\} \cup \left\{D_{i,j}^C \mid i, j \in [\ell], i < j\right\}. \tag{8.3}$$

In few words, D_i^C consists of those elements of Γ_C dominated only by u_i and no other upper bounds, while $D_{i,j}^C$ consists of those elements of Γ_C dominated only by u_i and u_j and no other upper bounds. Referring to Figure 8.1, $\text{Upper}_C = \{u_1, u_2\} = \{\gamma_3, \gamma_5\}$, if we fix the ordering $(u_1, u_2) = (\gamma_3, \gamma_5)$, then $D_1^C = \{\gamma_2, \gamma_3\}$, $D_2^C = \{\gamma_1, \gamma_5\}$ and $D_{1,2}^C = \{\gamma_4\}$. If no confusion arises, we omit the superscript C.

Remark 90. *If Upper_C has no full antipodal triples, then for each $\gamma \in \Gamma_C$ there are at most two different $u, u' \in \text{Upper}_C$ such that $\gamma \leq u$ and $\gamma \leq u'$. Thus the D_i's and the $D_{i,j}$'s form a partition of Γ_C.*

Before giving the definition of weak coloring we need some preliminary definitions. Let $\text{Cross}_C = \{\gamma \in \Gamma_C \mid \ \gamma \in D, \text{ for some } D \in \mathcal{D}, \text{ and } \gamma \leftrightarrow \gamma', \text{ for some } \gamma' \notin D\}$. In few words, Cross_C is composed by all elements in D, varying $D \in \mathcal{D}$, that are antipodal to at least one element not in D, i.e., Cross_C is composed by all elements that "cross" the partition through antipodality.

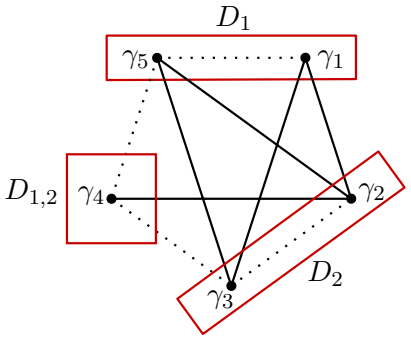


Figure 8.1: connected components of Figure 3.7, and sets D_1, D_2 and $D_{1,2}$ according to the ordering of $\text{Upper}_C = \{u_1, u_2\} = \{\gamma_3, \gamma_5\}$ as $(u_1, u_2) = (\gamma_3, \gamma_5)$.

Definition 91. *Let C be a clique separator of G. Let $(u_1, u_2, \ldots, u_\ell)$ be any ordering of Upper_C. We say that $f : \text{Upper}_C \cup \text{Cross}_C \to [\ell]$ is a* partial C-coloring *if f satisfies the following:*

91.(1) for all $i \in [\ell]$, $f(u_i) = i$;

91.(2) for all $i \in [\ell]$, for all $\gamma \in D_i$, if $\exists \gamma \notin D_i$ such that $\gamma \leftrightarrow \gamma'$, then $f(\gamma) = i$;

91.(3) for all $i < j \in [\ell]$, for all $\gamma \in D_{i,j}$, if $\exists \gamma' \notin D_{i,j}$ such that $\gamma \leftrightarrow \gamma'$, then

$$\begin{cases} f(\gamma) = i, & \text{if } \gamma' \in D_j, \\ f(\gamma) = j, & \text{if } \gamma' \in D_i. \end{cases} \tag{8.4}$$

We comment Definition 91. The first condition says how we color the upper bounds of Γ_C; actually, because of 86.(1), we only need that two antipodals upper bounds have different colors, i.e., $u, u' \in \text{Upper}_C$ and $u \leftrightarrow u'$ imply $f(u) \neq f(u')$ for all strong C-coloring f. We prefer to set $f(u_i) = i$ in order to use color i, j, k instead of $f(u_i), f(u_j), f(u_k)$ and so on; furthermore, condition 91.(1) implies the uniqueness of a partial C-coloring. Finally, it is easy to see that conditions 91.(2) and 91.(3) hold for every strong C-coloring satisfying 91.(1).

Note that there exists always a coloring satisfying conditions 91.(1) and 91.(2), while condition 91.(3) cannot be satisfied if there exist $\gamma \in D_{i,j}$, $\gamma' \in D_i$ and $\gamma'' \in D_j$ such that $\gamma \leftrightarrow \gamma'$ and $\gamma \leftrightarrow \gamma''$, for some $i < j \in [\ell]$. This fact leads to a kind of obstruction (see Section 8.4).

We are ready to give the definition of weak-coloring.

Definition 92. *Let C be a clique separator of G. Let $(u_1, u_2, \ldots, u_\ell)$ be any ordering of Upper_C. We say that G is* weak C-colorable *if there exists $f : \Gamma_C \to [\ell + 1]$ such that f restricted to $\text{Upper}_C \cup \text{Cross}_C$ is a partial C-coloring and*

92.(1) for all $i \in [\ell]$, $f(D_i) \subseteq \{i, \ell + 1\}$;

92.(2) for all $i < j \in [\ell]$, $f(D_{i,j}) \subseteq \{i, j\}$;

92.(3) for all $D \in \mathcal{D}$, if $\gamma, \gamma' \in D$ and $\gamma \leftrightarrow \gamma'$, then $f(\gamma) \neq f(\gamma')$.

We refer to a coloring f satisfying the conditions of Definition 92 as a *weak C-coloring*.

We comment Definition 92. It is easy to see that if we extend a partial C-coloring, then conditions 86.(1) and 86.(2) imply conditions 92.(2) and 92.(3). The condition 92.(1) is more

restrictive than the necessary. Indeed, conditions 86.(1) and 86.(2) should imply $f(D_i) \subseteq \{i, c_i\}$ (a possible choice of c_i is $\ell + i$), but the stiff structure given by the absence of full antipodal triples should imply that all elements colored by c_i's are pairwise not antipodal, and thus we can use the same color for all of them (as Proposition 97's proof shows).

In the following theorem we give our first characterization of path graphs, and we prove it in Subsection 8.3.2.

Theorem 93. *A chordal graph G is a path graph if and only if Γ_C has no full antipodal triples and G is weak C-colorable, for all clique separator C of G.*

8.3.1 Weak coloring equals to 2-coloring subproblems

Now we explain why our coloring problem shown in Theorem 93 simplifies the one stated in Corollary 87 (equivalent to Theorem 9 by Monma and Wei [102]) from an algorithmic point of view. Note that the two conditions of strong C-coloring are in conflict with each other. Indeed, if too few colors are used, then the first condition is violated, otherwise, if one uses too many colors, then the second condition is violated. For this reason their characterization does not describe directly a polynomial algorithm. Despite this, Schäffer [127] succeed in implementing a sophisticated backtracking polynomial algorithm that starts from their characterization.

Our characterization in Theorem 93 requires the absence of full antipodal triples and the check of six conditions: 91.(1), 91.(2), 91.(3) of Definition 91 and 92.(1), 92.(2), 92.(3) of Definition 92. First we note that conditions 91.(1), 91.(2), and 92.(1), 92.(2) are *always* satisfiable, i.e., there exists always a coloring $f : \Gamma_C \to [\ell + 1]$ that satisfies them. Moreover, checking for the absence of full antipodal triples and condition 91.(3) are polynomial problems, because they are antipodal paths/cycles of length 3. Finally, condition 92.(3) consists in 2-coloring problems restricted on elements in D, for $D \in \mathcal{D}$. In other words, after polynomial checks, we succeed in reducing the coloring problem by Monma and Wei to some 2-coloring subproblems. This allows us to exhibit a list of forbidden subgraphs in the attachedness graph (see Section 8.4).

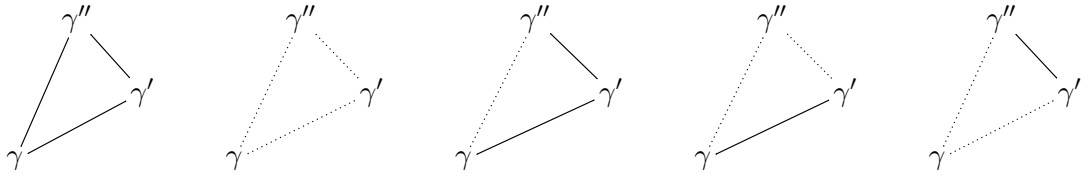
8.3.2 Proof of Theorem 93

This subsection is split into three parts: in Subsection 8.3.2 we give some useful results about dominance and antipodality, in Subsection 8.3.2 and Subsection 8.3.2 we prove the "if part" and the "only if part" of Theorem 93, respectively. In particular, Theorem 93 is implied by Proposition 97 and by Proposition 98.

Preliminary results

It is convenient to have a handy pictorial representation to deal with the relations $\leq$ and $\leftrightarrow$. Two elements γ', $\gamma'' \in \Gamma_C$ such that $\gamma' \leq \gamma''$ are drawn placing γ' below γ''—here "below" means that viewing the sheet as a portion of the Cartesian plane with origin placed in left bottom corner, the ordinate of γ' is smaller than the ordinate of γ''—and joining them by a dotted line, while, if γ', $\gamma'' \in \Gamma_C$ such that $\gamma' \leftrightarrow \gamma''$, then their are joined by a continuous line wherever they are

placed. For, instance, the following diagrams, represent all possible cases involving three pairwise attached elements of Γ_C (there is not a case that is impossible because of the transitivity of $\leq$).



(8.5)

(i) (ii) (iii) (iv) (v)

Lemma 94. *Let C be a clique separator of G, the following hold:*

94.(a) $\gamma \leq \gamma' \Rightarrow \gamma$ *and* γ *are neighboring of* v*, for all* $v \in V(\gamma) \cap C$*;*

94.(b) $\gamma \leftrightarrow \gamma' \Rightarrow \gamma$ *and* γ *are neighboring of* v*, for all* $v \in V(\gamma) \cap V(\gamma') \cap C$*;*

94.(c) let $\gamma, \gamma', \gamma'' \in \Gamma_C$*, if one among* (ii),(iii),(iv),(v) *of* (8.5) *applies, then* $\gamma, \gamma', \gamma''$ *are neighboring of* v *for all* $v \in V(\gamma) \cap V(\gamma') \cap C$*.*

Proof. The first two statements follow from definitions of $\leq$ and $\leftrightarrow$. The last statement holds for cases (ii) and (v) by applying 94.(a) on $\gamma \leq \gamma'$ and $\gamma \leq \gamma''$. For cases (iii) and (iv), 94.(b) implies that γ and γ' are neighboring of v for all $v \in V(\gamma) \cap V(\gamma') \cap C$, finally, 94.(a) applied to γ and γ'' implies the thesis. □

The following lemma shows that the absence of full antipodal triples gives a stiff structure of antipodality graph with respect to partition $\mathcal{D}$. Indeed, for some $D, D' \in \mathcal{D}$, it holds that γ cannot be antipodal to γ', for $\gamma \in D$ and $\gamma' \in D'$.

Lemma 95. *Let C be a clique separator of G and let $i < j \in [\ell]$. If Upper_C has no full antipodal triples, then the following hold:*

95.(a) $\gamma \leq \gamma'$ *and* $\gamma \in D_{i,j} \Rightarrow \gamma' \in D_{i,j} \cup D_i \cup D_j$*;*

95.(b) $\gamma \leftrightarrow \gamma'$ *and* $\gamma \in D_{i,j} \Rightarrow \gamma' \in D_{i,j} \cup D_i \cup D_j$*;*

95.(c) $\gamma \leftrightarrow \gamma'$*,* $\gamma \in D_i$ *and* $\gamma' \notin D_i \Rightarrow \gamma \leftrightarrow u_k$ *for* $\gamma' \leq u_k$ *and* $k \neq i$*.*

Proof. Statements 95.(a) and 95.(b) follow from the absence of full antipodal triples, transitivity of $\leq$, 94.(a) and 94.(b). Indeed, if one among 95.(a) and 95.(b) is denied, then, by using 94.(a) and 94.(b), we find a full antipodal triple $\{u_i, u_j, u_k\} \in \text{Upper}_C$, absurdum by hypothesis.

To prove 95.(c), we observe that $\gamma' \notin D_i$ implies that there exists $u_k \in \text{Upper}_C$ such that $\gamma' \leq u_k$ and $k \neq i$. Such k is unique, indeed, if there exists $k' \neq k$ such that $\gamma' \leq u_{k'}$, then $\{u_k, u_{k'}, u_i\}$ is a neighboring set by 94.(a) and 94.(b), and thus it is full antipodal triple, absurdum by hypothesis. Finally, $\gamma, \gamma', u_i, u_k$ is a neighboring set because of 94.(a) and 94.(b), and thus $u_i \leftrightarrow u_k$ as claimed. □

Lemma 95 implies that every partial C-coloring sets the color of $\gamma \in \Gamma_C$ if and only if $\gamma \in \text{Cross}_C$. By its definition, the color is univocally determined, i.e., if there exists a partial C-coloring, then it is unique.

Strong coloring implies weak coloring

The following lemma shows that a strong C-coloring f can be modified in order to satisfy condition 91.(1), and, if so, then f satisfies conditions 91.(2), 91.(3), 92.(2), 92.(3). This is the first step to prove that definition of strong C-coloring implies the definition of weak C-coloring.

Lemma 96. *Let C be a clique separator and let $f : \Gamma_C \to [r]$ be a strong C-coloring. Then there exists $g : \Gamma_C \to [r']$, with $r' \geq r$, satisfying 91.(1). Moreover, conditions 91.(2), 91.(3), 92.(2), 92.(3) hold for every strong C-coloring satisfying 91.(1).*

Proof. Let's start with the first part of the claim. By Lemma 88, there are no full antipodal triples in Γ_C. If $f(u_i) \neq f(u_j)$ for all $i \neq j \in [\ell]$, then the thesis is true. Thus let us assume that there exists $i \neq j \in [\ell]$ such that $f(u_i) = f(u_j)$. We need a preliminary result that explains how to obtain a strong C-coloring g satisfying $g(u_i) \neq g(u_j)$ starting from f.

j) Let $i, j \in [\ell]$ be such that $f(u_i) = f(u_j)$. Let $\Omega_i = \{\gamma \in \Gamma_C \mid \gamma \leq u_i \text{ and } f(\gamma) = f(u_i)\}$. Let

$$g(\gamma) = \begin{cases} r+1, & \text{if } \gamma \in \Omega_i, \\ f(\gamma), & \text{otherwise,} \end{cases} \tag{8.6}$$

then g is a strong C-coloring and $g(u_i) \neq g(u_j)$.

Proof of j): It is clear that g satisfies 86.(1). Let us assume by contradiction that g does not satisfy 86.(2). Thus let $\gamma, \gamma', \gamma'' \in \Gamma_C$ be such that $|g(\{\gamma, \gamma', \gamma''\})| = 3$ and $\gamma, \gamma', \gamma''$ are neighboring of v for some $v \in C$. W.l.o.g., by (8.6), we assume that $\gamma \in \Omega_i$, $\gamma', \gamma'' \notin \Omega_i$, $f(\gamma') = f(u_i)$ and $f(\gamma'') \neq f(u_i)$. Indeed, if one of these conditions do not hold, then $|g(\{\gamma, \gamma', \gamma''\})| < 3$ because f is a strong C-coloring.

Being $v \in V(\gamma)$ and $\gamma \in \Omega_i$, then $v \in V(u_i)$ by 94.(a). Being γ' neighboring of v, then either $\gamma' \leq u_i$ or $u_i \leftrightarrow \gamma'$. If $\gamma' \leq u_i$, then $f(\gamma') = f(u_i)$ implies $\gamma' \in \Omega_i$, absurdum. If $u_i \leftrightarrow \gamma'$, then $f(\gamma') = f(u_i)$ implies that f is not a strong C-coloring, absurdum. Finally, $g(u_i) \neq g(u_j)$ because $g(u_i) = r + 1$ while $g(u_j) \leq r$. End proof of j).

By repeatedly applying j), we obtain a strong C-coloring g satisfying 91.(1). To complete the proof, we have to prove that g satifies 91.(2), 91.(3), 92.(2), 92.(3).

Let us assume by contradiction that 91.(2) does not hold. Then let $\gamma \in D_i$, for some $i \in [\ell]$, and let $\gamma' \notin D_i$ be such that $\gamma \leftrightarrow \gamma'$ and assume that $g(\gamma) \neq i$. Being $\gamma' \notin D_i$, then there exists $u_j \in \text{Upper}_C$ such that $\gamma' \leq u_j$ and $i \neq j$. By 94.(b), γ, γ' are neighboring of v, for some $v \in C$, moreover, by 94.(a), u_i, u_j are neighboring of v. Thus γ, u_i, u_j are neighboring of v, implying that $\gamma \leftrightarrow u_j$ because $\gamma \in D_i$. Finally, 86.(1) imply $g(\gamma) \neq j$, that implies $|g(\{\gamma, u_i, u_j\}| = 3$, absurdum because g is a strong C-coloring.

Let us assume that 91.(3) does not hold. Then there exist $\gamma \in D_{i,j}$, $\gamma' \in D_j$ such that $\gamma \leftrightarrow \gamma'$ and $g(\gamma) \neq i$ (the case in which $\gamma' \in D_i$ and $g(\gamma) \neq j$ is similar). As above, by 94.(b) and 94.(a), it holds that $\gamma, \gamma', u_i, u_j$ are neighboring of v for some $v \in C$. Moreover, $\gamma \leftrightarrow u_j$ and thus 91.(2) implies $g(\gamma') = j$. Being g a strong C-coloring and being $\gamma \leftrightarrow \gamma'$, 86.(1) implies $g(\gamma) \neq j$. Thus $|g(\{\gamma, u_i, u_j\})| = 3$, absurdum.

If 92.(2) does not hold for a $\gamma \in D_{i,j}$, for some $i < j \in [\ell]$, then γ, u_i, u_j would deny 86.(2). Indeed, γ, u_i, u_j are neighboring of v, for each $v \in V(\gamma) \cap C$, by 94.(a). Thus γ, u_i, u_j deny 86.(2) because of 91.(1), absurdum.

Finally, 92.(3) is implied by 86.(2). □

The following proposition proves the "if part" of Theorem 93.

Proposition 97. *Let C be a clique separator of G. If G is strong C-colorable, then Γ_C has no full antipodal triples and G is weak C-colorable.*

Proof. Let $f : \Gamma_C \to [r]$ be a strong C-coloring. We have to find $g : \Gamma_C \to [\ell + 1]$ such that g is a weak C-coloring and prove that there are no full antipodal triples in Upper_C. First we observe that 86.(1) and 86.(2) implies that there are no full antipodal triples in Γ_C.

By Lemma 96, we can assume that f satisfies conditions 91.(1), 91.(2), 91.(3), 92.(2), 92.(3) and that $r \geq \ell$.

For all $i \in [\ell]$ let $\Omega_i = \{\gamma \in D_i \mid f(\gamma) \neq i\}$. Now, let $i \in [\ell]$. For all $\gamma, \gamma' \in \Omega_i$ it holds that $\gamma \not\nleftrightarrow \gamma'$, indeed, if not, then $f(\gamma) \neq f(\gamma')$ because of 86.(1), and this implies $|f(\{\gamma, \gamma', u_i\})| = 3$, absurdum because f a strong C-coloring. Thus if we define

$$g(\gamma) = \begin{cases} \ell + i, & \text{if } \gamma \in \Omega_i, \\ f(\gamma), & \text{otherwise,} \end{cases}$$

then $g : \Gamma_C \to [2\ell]$ is a strong C-coloring because f is a strong C-coloring and we used the same color for a class of non-antipodal elements.

For all distinct $i, j \in [\ell]$, for all $\gamma \in \Omega_i$ and $\gamma' \in \Omega_j$, it holds that $\gamma \not\nleftrightarrow \gamma'$. Indeed, if $\gamma \leftrightarrow \gamma'$, then, by assuming $\gamma \in D_i$ and $\gamma' \in D_j$, condition 91.(2) implies $f(\gamma) = i$ and $f(\gamma') = j$ and thus $\gamma \notin \Omega_i$ and $\gamma' \notin \Omega_j$, absurdum. Finally, let $\Omega = \bigcup_{i \in [\ell]} \Omega_i$ and let

$$h(\gamma) = \begin{cases} \ell + 1, & \text{if } \gamma \in \Omega, \\ g(\gamma), & \text{otherwise,} \end{cases}$$

it is clear that g satisfies 92.(1); moreover, 92.(1) and 92.(2) imply $g : \Gamma_C \to [\ell + 1]$. By the same above reasoning, g is a strong C-coloring and this finishes the proof. □

Weak coloring implies strong coloring

Now we prove the "only if part" of Theorem 93.

Proposition 98. *Let C be a clique separator of G. If Γ_C has no full antipodal triples and G is weak C-colorable, then G is strong C-colorable.*

Proof. First we observe that if there is a full antipodal triple, then Γ_C is not strong C-colorable neither weak C-colorable. Thus we assume that there are no full antipodal triples. By Remark 90, it holds that $\Gamma_C = \mathcal{D}$.

Let f be a weak C-coloring of G. We have to prove that 86.(1) and 86.(2) are satisfied. Let $\gamma, \gamma' \in \Gamma_C$ be such that $\gamma \leftrightarrow \gamma'$. We want to prove that $f(\gamma) \neq f(\gamma')$. Let $i < j \in [\ell]$. Because of the absence of full antipodal triples, then there are four cases: $\gamma, \gamma' \in D_i$ (case 1), $\gamma, \gamma' \in D_{i,j}$ (case 2), $\gamma \in D_i$ and $\gamma' \in D_{i,j}$ (case 3), $\gamma \in D_i$ and $\gamma \in D_j$ (case 4) (there are no other cases because of 95.(b)).

If case 1 or case 2 happens, then $f(\gamma) \neq f(\gamma')$, by 92.(3). If case 3 happens, then $f(\gamma) \neq f(\gamma')$, by 91.(2) and 91.(3). Finally, if case 4 happens, then $i = f(\gamma) \neq f(\gamma') = j$, by 91.(2). Thus 86.(1) is satisfied.

Now it remains to prove that 86.(2) holds. Let us assume by contradiction that there exist $\gamma, \gamma', \gamma'' \in \Gamma_C$ such that they form a neighboring triple and $|f(\{\gamma, \gamma', \gamma''\})| = 3$. In (8.5) there are all possibilities of relations between these three elements. Case (i) cannot apply because it is a full antipodal triple. Thus we have to prove that in cases (ii),(iii),(iv),(v) $|f(\{\gamma, \gamma', \gamma''\})| < 3$.

Before we examine each case, we need the following result.

k) In any case among (ii),(iii),(iv),(v) if there are $\eta \in D_i$ and $\eta' \in D_j$, then $f(\eta) = i$ and $f(\eta') = j$. Similarly, if there are $\mu \in D_{i,j}$ and $\mu' \in D_i$ (resp., $\mu' \in D_j$), then $f(\mu') = i$ (resp., $f(\mu') = j$).

Proof of k): We note that any case among (ii),(iii),(iv),(v) is a neighboring triple. Thus condition 94.(a) and 94.(b) imply $\eta \leftrightarrow u_j$ and $\eta' \leftrightarrow u_i$, hence $f(\eta) = i$ and $f(\eta') = j$ because of 91.(2). The second part of the claim holds with a similar reasoning. End proof of k).

Let's start with cases (ii) and (v). If $\gamma \in D_i$, for some $i \in [\ell]$, then $\gamma', \gamma'' \in D_i$ because of transitivity of $\leq$, thus $f(\{\gamma, \gamma', \gamma''\}) \subseteq \{i, \ell + 1\}$ by 92.(1). Else, $\gamma \in D_{i,j}$, for some $i < j \in [\ell]$, then $\gamma', \gamma'' \in D_i \cup D_j \cup D_{i,j}$ because of 95.(a), thus $f(\{\gamma, \gamma', \gamma''\}) \subseteq \{i, j\}$ by 92.(1), 92.(2) and k).

Now we deal with case (iv). For short, for any $i \in [\ell]$, we define $\overline{D}_i = (\bigcup_{j<i} D_{j,i}) \cup (\bigcup_{j>i} D_{i,j})$. If $\gamma'' \in D_{i,j}$, for some $i < j \in [\ell]$, then $\gamma, \gamma' \in D_{i,j}$ because of transitivity of $\leq$, thus $f(\{\gamma, \gamma', \gamma''\}) \subseteq \{i, j\}$ by 92.(2). Else, $\gamma'' \in D_i$, for some $i \in [\ell]$, then $\gamma, \gamma' \in D_i \cup \overline{D}_i$ because of 95.(b). There are two sub-cases: either $\gamma, \gamma', \gamma'' \in D_i$, or at least one among γ, γ' is in $\overline{D}_i$. If the first sub-case happens, then the proof is completed by 92.(1). Otherwise, w.l.o.g., let us assume that $\gamma \in D_{i,j}$ for some $j > i$. Thus $u_j \leftrightarrow \gamma''$, and, by 91.(2), $f(\gamma'') = i$, implying that $f(\{\gamma, \gamma', \gamma''\}) \subseteq \{i, j\}$.

It remains to check case (iii). If $\gamma'' \in D_{i,j}$, for some $i < j \in [\ell]$, then $\gamma \in D_{i,j}$. By 95.(b), $\gamma' \in D_i \cup D_j \cup D_{i,j}$. Thus $f(\{\gamma, \gamma', \gamma''\}) \subseteq \{i, j\}$ because of 91.(2), 91.(3) and 92.(3). Else, $\gamma'' \in D_i$, for some $i \in [\ell]$, then there are two sub-cases: either $\gamma \in D_i$, or $\gamma \in D_{i,j}$, for some $i < j \in [\ell]$ (the sub-case $\gamma \in D_{j,i}$ is similar). If the first sub-case happens, then $|f(\{\gamma, \gamma', \gamma''\})| < 3$ because $f(\gamma'') = f(\gamma) = i$ by 91.(2). If the second sub-case happens, then $\gamma' \in D_i \cup D_j \cup D_{i,j}$ by above. Thus $f(\{\gamma, \gamma', \gamma''\}) \subseteq \{i, j\}$ by 92.(1), 92.(2) and k). □

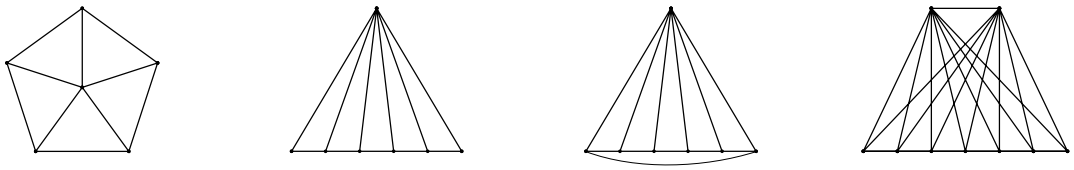
8.4 Forbidden subgraphs in attachedness graphs

In Subsection 8.3.1 we described exactly which are the obstructions to the weak coloring, thus we can list all the obstructions of path graphs in the form of subgraphs of the C-attachedness graphs

of a chordal graph G, and in Subsection 8.4.2 we compare our obstructions with obstructions by Lévêque *et al.* [98]. Recall that the C-attachedness graph of G is the graph $(\Gamma_C, \bowtie)$ with reflexive pairs neglected—whose edges are therefore distinct pairs $\gamma\gamma'$, $\gamma, \gamma' \in \Gamma_C$ such that $\gamma \bowtie \gamma'$. Also recall that the C-antipodality and the C-dominance graph of G factor $(\Gamma_C, \bowtie)$. Such a factorization yields a 2-edge coloring of $(\Gamma_C, \bowtie)$ which models the interactions between $\leftrightarrow$ and $\leq$.

In the following definition we present an uncolored version of our obstructions to path graphs.

Definition 99. – *For an integer m such that $m \geq 3$, the* m-wheel *is the graph on $[m+1]$ where the vertices in $[m]$ induces a cycle and vertex $m+1$ is adjacent to all the other vertices (see Figure 8.2.a).*
– *For an integer m such that $m \geq 4$, the* m-fan *is the graph on $[m]$ such that $[m-1]$ induces a path having extremal vertices 1 and $m-1$ and vertex m is adjacent to all the other vertices (see Figure 8.2.b).*
– *The* m-chorded fan *is the graph obtained from the m-fan by adding an edge between vertices 1 and $m-1$. Note that the m-chorded fan is isomorphic to the $m-1$-wheel (see Figure 8.2.c).*
– *For an integer m such that $m \geq 4$, the* m-double fan *is the graph on $[m]$ such that $[m]$ induces a cycle and vertices $m-1$ and m are adjacent to all other vertices (see Figure 8.2.d).*



(a) (b) (c) (d)

Figure 8.2: (a) 5-wheel; (b) 7-fan; (c) 7-chored-fan; (d) 9-double fan.

Figure 8.3 lists certain special 2-edge-colored graphs, obtained as 2-edge-colored versions of the aforesaid graphs, needed in the characterization of path graphs (Theorem 102). The two colors are represented by dotted or continuous lines, respectively.

It is convenient to settle a specific notation and terminology to present the results. An *isomorphism of edge-colored graphs* is a graph isomorphism which preserves edge colors. All of the 2-edge-colored graphs in Figure 8.3 are pairwise non isomorphic as edge-colored graphs. We denote by $\mathcal{F}$ the collection they form—$\mathcal{F}$ stands for "forbidden". Hence

$$\mathcal{F} = \left\{ W^{(0)}_{2k+1},\ W^{(1)}_{2k+1},\ F_{2n+1},\ \widetilde{F}_{2n+1},\ DF_{2n+1} \mid k \geq 1,\ n \geq 2 \right\}.$$

Also let

$$\mathcal{F}_0 = \left\{ W^{(0)}_{2k+1}, W^{(1)}_{2k+1}, F_{2n+1} \right\}.$$

Triangles of attachedness graphs play a special role. A triangle which is induced by a neighboring triple the C-attachedness graph of G is called a *full triangle*, otherwise it is called *empty*. A triangle whose all edges are antipodal is an *antipodal triangle*. Not every triangle in C-attachedness

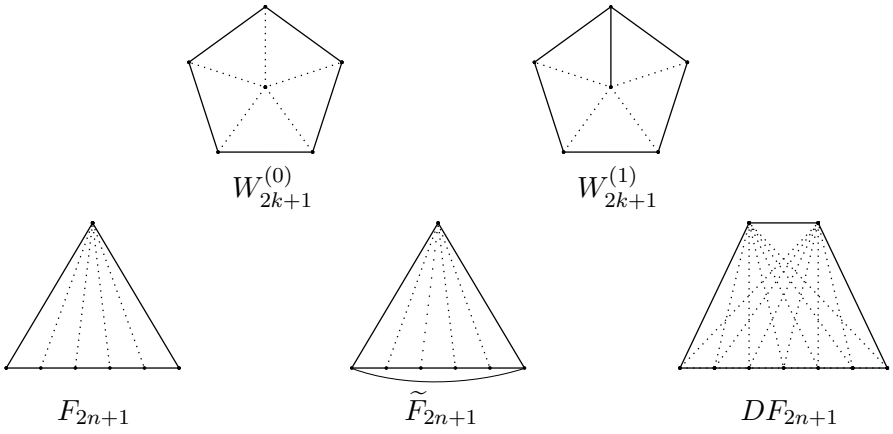


Figure 8.3: 2-edge-colored graphs occurring in Theorem 102, $k \geq 1$ and $n \geq 2$.

graph of G is full, indeed, an antipodal triangle might be empty (recall the discussion right after Lemma 94).

Unfortunately there is not a way to establish whether an antipodal triangle is full or empty, let's see an example. Let G be the graph F_2 in Figure 8.5, G has only one separator, C, say, let H and M be its C-antipodality and C-attachedness graphs. Hence $M = H \cong K_3$ and the triangle spans a neighboring triple. However, if we denote by z the universal vertex of G, then $G - z$ is separated by $C \setminus z$. Again, let $C' = C \setminus z$ be the only clique separator, it holds that $M = H \cong K_3$ but the triangle does not span a neighboring triple.

We know that full antipodal triples are obstructions to strong C-colorability. Therefore, full antipodal triangles are obstructions to membership in the class of path graphs and they should be added to $\mathcal{F}$. However, since full antipodal triangles are not just edge colored triangles (because they have also the property of being full), we must treat such triangles separately in our statements. To overcome this (somehow unaesthetic and noising) ambiguity we use a standard trick.

For a graph G let G^+ be the graph defined as follows. Let $V(G) = V = \{v_1, v_2, \ldots, v_n\}$ and V^+ be a copy of V, $V^+ = \{v_1^+, v_2^+, \ldots, v_n^+\}$. Let

$$G^+ = \left(V \cup V^+, E(G) \cup \{v_i v_i^+\}_{i=1}^n\right).$$

Lemma 100. *Let G be a graph. Then G is a path graph if and only if G^+ is a path graph.*

Proof. Since G is an induced subgraph of G^+, G is a path graph if G^+ is such. Let T be a clique path tree of G. For all $v \in V(G)$, let $\mathcal{K}_v$ be the set of all cliques of G containing v. By Theorem 7, $\mathcal{K}_v$ induces a path in T, let $\tilde{C}_v \in \mathcal{K}_v$ be an extremal vertex of this path. Thus it suffices to join vv^+ to $\tilde{C}_v$ for all $v \in V(G)$ to yield a clique path tree for G^+. □

The reason for having introduced graph G^+ relies on the fact that, for each clique separator C of G^+, full antipodal triangles of G appear in C-attachedness graph of G^+ as small wheels as shown next.

Lemma 101. *Let C be a clique separator of G^+ and let M be the C-attachedness graph of G^+. Then M has no full antipodal triangles and has no induced copies of $W^{(0)}_{2k+1}$ if and only if M has no induced copies of $W^{(0)}_{2k+1}$.*

Proof. One direction is trivial. For the other direction it suffices to prove that if M has no induced copies of $W^{(0)}_3$, then M has no full antipodal triangles. We prove the contrapositive: if M has a full antipodal triangle, then M has an induced copy of $W^{(0)}_3$. Observe first that $C \cap \{v^+ \mid v \in V(G)\} = \emptyset$. For if not, then C is necessarily of the form $\{v, v^+\}$ for some $v \in V(G)$ (note that in this case v is a cut vertex); in this case however M would contain no antipodal edges at all and thus no full antipodal triangles. Hence $v^+ \notin C$ for each $v \in C$. Observe that for each $v \in C$ the graph $\gamma^+ = (\{v, v^+\}, \{vv^+\})$ is $\leq$-dominated by every other neighboring subgraph γ of v. Let $\{\gamma, \gamma', \gamma''\}$ be the set of vertices of a full antipodal triangle in M. Hence, there is some $z \in V(G)$ such that γ, γ' and γ'' are neighboring subgraphs of z. If γ_z is the subgraph of G induced by $\{z, z^+\} \cup C$, then $\{\gamma_z, \gamma, \gamma', \gamma''\}$ induces a copy of $W^{(0)}_3$ in M. □

In the following theorem we claim our characterization by forbidden subgraphs in the attachedness graphs. Note that the graphs in $\mathcal{F}$ are induced obstructions, while the graphs in $\mathcal{F}_0$ are not necessarily induced. Moreover, statements 102.(3) and 102.(5) are equivalent to 102.(2) and 102.(4), respectively, by using G^+ in place of G thanks to Lemma 100 and Lemma 101.

Theorem 102. *Let G be a chordal graph. Then the following statements are equivalent:*

102.(1) *G is a path graph;*

102.(2) *for each clique separator C of G, the C-attachedness graph of G has no full antipodal triangles and has no subgraphs isomorphic to any of the graphs in $\mathcal{F}_0$;*

102.(3) *for each clique separator C of G, the C-attachedness graph of G^+ has no subgraphs isomorphic to any of the graphs in $\mathcal{F}_0$;*

102.(4) *for each clique separator C of G, the C-attachedness graph of G has no full antipodal triangles and has no induced subgraphs isomorphic to any of the graphs in $\mathcal{F}$;*

102.(5) *for each clique separator C of G, the C-attachedness graph of G^+ has no induced subgraphs isomorphic to any of the graphs in $\mathcal{F}$.*

The equivalences 102.(2)⇔102.(3) and 102.(4)⇔102.(5) in the theorem above follows straightforwardly by Lemma 100 and Lemma 101. The remaining implications in Theorem 102 (the core of the characterization), will be the content of the next subsection.

8.4.1 Proof of Theorem 102

We prove Theorem 102 according to the schema 102.(1) $\overset{\text{Lemma 103, 104}}{\Longleftrightarrow}$ 102.(2) $\overset{\text{Lemma 105}}{\Longleftrightarrow}$ 102.(4), indeed, we remember that the equivalences 102.(2)⇔102.(3) and 102.(4)⇔102.(5) are implied by Lemma 100 and Lemma 101. In particular Lemma 103 implies that every member of $\mathcal{F}_0$ and every full antipodal triangle is an obstruction, Lemma 104 explains that $\mathcal{F}_0$ joined with a full

antipodal triangle is a minimal set of obstruction and, finally, Lemma 105 shows the equivalence of containing a member of $\mathcal{F}_0$ as subgraphs and a member of $\mathcal{F}$ as induced subgraph.

In what follows G is chordal graph which is not an atom.

Lemma 103. *If G is a path graph, then, for each clique separator C, the C-attachedness graph of G has neither full antipodal triangles nor copies of any of the graphs in $\mathcal{F}_0$ as subgraphs.*

Proof. Let C be a clique separator. Let us denote by M the C-attachedness graph of G. Being G a path graph, then C contains no full antipodal triangles by Lemma 88. Suppose by contradiction that M contains, as a subgraph, a copy S of F_{2n+1} or $W^{(0)}_{2k+1}$ or $W^{(1)}_{2k+1}$. In all cases, S contains a subgraph F_0 on $\{\theta_0, \theta_1 \ldots, \theta_{2t}\}$, for some integer t, fulfilling the following conditions:

- $\theta_i\theta_{i+1}$ is an antipodal edge of M, i.e., $\theta_i \leftrightarrow \theta_{i+1}$, for $i = 1, \ldots, 2t-1$;
- $\theta_0\theta_i$ is a dominance edge of M, i.e., either $\theta_i \leq \theta_0$ or $\theta_0 \leq \theta_i$, for all $i = 1, \ldots, 2t$.

We claim that:

l) If f is any strong C-coloring of G, then $f(\theta_1) \neq f(\theta_{2t})$ and $f(\theta_0) \in \{f(\theta_1), f(\theta_{2t})\}$.

Proof of l): By Lemma 94 all triangles $\{\theta_0, \theta_i, \theta_{i+1}\}$ are full, for $i = 1, \ldots, 2t-1$. Hence, being f a strong C-coloring, 86.(2) implies that $|f(\{\theta_0, \theta_i, \theta_{i+1}\})| = 2$, for $i = 1, \ldots, 2t-1$. Thus if $f(\theta_0) = f(\theta_1)$, then $f(\theta_2) \neq f(\theta_0)$, $f(\theta_3) = f(\theta_0), \ldots, f(\theta_{2t}) \neq f(\theta_0)$. Instead, if $f(\theta_0) \neq f(\theta_1)$, then $f(\theta_2) = f(\theta_0)$, $f(\theta_3) \neq f(\theta_0), \ldots, f(\theta_{2t}) = f(\theta_0)$. Hence, in both cases, we proved the claim. End proof of l).

We now use l) to prove a contradiction to the strong C-colorability of G. Suppose first that $S \cong F_{2n+1}$, for some n, then let $V(S) = \{\eta, \gamma_1, \ldots, \gamma_{2n}\}$ where η is the maximum degree vertex of S. Let F' be the subgraph induced by $V(S) = \{\eta, \gamma_2, \ldots, \gamma_{2n-1}\}$. Hence $F' \cong F_0$. By l), γ_2 and γ_{2n-1} have opposite colors and $f(\gamma_\eta) \in \{f(\gamma_2), f(\gamma_{2n-1})\}$. Moreover, the triangles induced by $\{\eta, \gamma_1, \gamma_2\}$ and $\{\eta, \gamma_{2n-1}, \gamma_{2n}\}$ are both full by Lemma 94 and at least one of them cannot be 2-colored under f.

Suppose now that $S \cong W^{(0)}_{2k+1}$ or $S \cong W^{(1)}_{2k+1}$ for some k. Let $V(S) = \{\eta, \gamma_1, \ldots, \gamma_{2k+1}\}$ where η is still the maximum degree vertex of S (if $S \cong W^{(1)}_{2k+1}$, then let γ_1 be the only vertex such that $\gamma_1\eta$ is an antipodal edge) and let F'' be the subgraph induced by $V(S) = \{\eta, \gamma_1, \ldots, \gamma_{2k}\}$. Clearly, $F'' \cong F_0$, as well. By l), γ_1 and γ_{2k} have opposite colors and $f(\eta) \in \{f(\gamma_1), f(\gamma_{2k})\}$. It holds that $f(\gamma_{2k+1}) \notin \{f(\gamma_{2k}), f(\gamma_1)\}$ because $\gamma_{2k+1} \leftrightarrow \gamma_{2k}$ and $\gamma_{2k+1} \leftrightarrow \gamma_1$. Moreover, the triangles induced by $\{\eta, \gamma_1, \gamma_{2k+1}\}$ and $\{\eta, \gamma_{2k}, \gamma_{2k+1}\}$ are both full by Lemma 94 and at least one of them cannot be 2-colored under f. In any case a contradiction to the strong C-colorability of is achieved. □

Lemma 104. *If for each clique separator C, the C-attachedness graph of G has neither full antipodal triangles nor copies of any of the graphs in $\mathcal{F}_0$ as subgraphs, then G is path graph.*

Proof. By Corollary 87, G is a path graph if and only if G is strong C-colorable for each clique separator C. We prove the contrapositive statement: if G is not strong C-colorable for some

clique separator C, then the C-attachedness graph M of G contains full antipodal triangles or some copy of a graph of $\mathcal{F}_0$ as subgraphs. Since each graph in $\mathcal{F}$ contains some graph of $\mathcal{F}_0$ as subgraph, we show the statement with $\mathcal{F}$ in place of $\mathcal{F}_0$. Denote by H the C-antipodality graph and remember that $\mathcal{D}$ is a partition of elements of Γ_C if there are not full antipodal triple. For $D \in \mathcal{D}$ denote by H_D the subgraph of H induced by D.

By Theorem 93, G is not a path graph if one of the following apply: C contains a full antipodal triple, C does not admits a partial C-coloring, the C partial coloring defined on $\mathrm{Upper}_C \cup \mathrm{Cross}_C$ cannot be extended to a weak C-coloring on Γ_C. Let $(u_1, \dots, u_\ell)$ be any ordering of Upper_C.

If C contains a full antipodal triple then this full antipodal triple is also a full antipodal triangle. If C does not admit a partial C-coloring, then 91.(3) is not be satisfiable; indeed, 91.(1) and 91.(2) are always satisfiable. Hence there exist $\gamma \in D_{i,j}$, $\gamma' \in D_i$ and $\gamma'' \in D_j$, for some $i < j \in [\ell]$, such that $\gamma \leftrightarrow \gamma'$ and $\gamma \leftrightarrow \gamma''$. Now there are two cases: $\gamma' \leftrightarrow \gamma''$ or $\gamma' \not\leftrightarrow \gamma''$. If the first case applies, then $\{\gamma, \gamma', \gamma'', u_i\}$ induces a copy of $W_3^{(1)}$ in M (refer to Lemma 94 and Lemma 95 to determine all colored edges in M). Else, the second case applies and $\{\gamma, \gamma', \gamma'', u_i, u_j\}$ induces a copy of DF_5.

Thus it remains to study only the case in which C has no full antipodal triangles, C admits a partial C-coloring $g : \mathrm{Upper}_C \cup \mathrm{Cross}_C \to [\ell]$ and g cannot be extended to a weak C-coloring on Γ_C. Being 92.(1) and 92.(2) be always satisfiable by an extension of g (as proved in Lemma 96), then there exists $D \in \mathcal{D}$ such that every extension of g does not satifies 92.(3) on D.

Conditions 92.(1), 92.(2) and 92.(3) implies that H_D is 2-colored. Only three cases can occur:

- H_D is non bipartite. In this case no 2-coloring g of H exists;
- H_D is bipartite but it contains a path P with an even number of vertices whose extremal vertices have the same color under g;
- H_D is bipartite but it contains a path P with an odd number of vertices whose extremal vertices have different color under g;

In the first case, H_D contains an odd cycle Q, on $2k+1$ vertices, say, as subgraph. Hence, for $u \in \mathrm{Upper}_C \cap D$, the subgraph induced by $Q \cup \{u\}$ in H contains a copy of $W_{2k+1}^{(0)}$ as a subgraph.

In the second case let $\Theta = \{\theta_1, \dots, \theta_{2k}\}$ be the set of vertices of P. Suppose first that $D = D_i$ for some $i \in [\ell]$. By definition of g there are $\gamma, \gamma' \notin D_i$ such that $\gamma \leftrightarrow \theta_1$ and $\gamma' \leftrightarrow \theta_{2k}$. It holds that $\gamma \leftrightarrow u_i$ and $\gamma' \leftrightarrow u_i$ by the transitivity of $\leq$ and the definition of D_i. Now, let N be the subgraph induced by $\Theta \cup \{\gamma, \gamma', u_i\}$. If $\gamma = \gamma'$ then N contains $W_{2k+1}^{(1)}$ as subgraph. If $\gamma \neq \gamma'$, then N contains either F_{2n+1} or $\widetilde{F}_{2n+1}$ according to whether $\gamma \leftrightarrow \gamma'$ or not. If $D = D_{i,j}$, then we obtain the same results with a similar reasoning.

The third case can apply only to $D = D_{i,j}$ for some i, $j \in [\ell]$, because all the elements of $\mathrm{Cross}_C \cap D_i$ have the same color i under g. Let $\Theta = \{\theta_1, \dots, \theta_{2k+1}\}$ be the set of vertices of P. By the definition of g there are $\gamma \in D_i$ and $\gamma' \in D_j$ such that $\gamma \leftrightarrow \theta_1$ and $\gamma' \leftrightarrow \theta_{2k+1}$. Then $\Theta \cup \{\gamma, \gamma', u_i, u_j\}$ induces a subgraph in $H_{i,j}$ that contains DF_{2n+1} as subgraph. □

Lemma 105. *Let C be a clique separator of G. If the C-attachedness graph of G has no full antipodal triangles, then it has a copy of a graph in $\mathcal{F}_0$ as a subgraph if and only if it has a copy of a graph in $\mathcal{F}$ as an induced subgraph.*

Proof. Since any graph in $\mathcal{F}_0$ is contained as subgraph in one of the graph in $\mathcal{F}$, then one direction is trivial. Let us prove the other direction. Let H and M be the C-antipodality and C-attachedness graph of G. We have to prove that if M contains some copy of a graph of $\mathcal{F}_0$, then M contains an induced copy of some graph of $\mathcal{F}$. Let S be a graph of $\mathcal{F}_0$. For a cycle Q of S it is convenient to distinguish between chords that are edges of the antipodality graphs, which we call a-chords, from those that are edges of the dominance graph, which we call d-chords.

Now, let Q be an antipodal odd cycle of S on $2k+1$ vertices for some integer $k \geq 2$, i.e., the vertex set of Q is $\{\gamma_0, \ldots, \gamma_{2k}\}$ and the edges are $\{\gamma_0\gamma_1, \ldots, \gamma_{2k-1}\gamma_{2k}, \gamma_0\gamma_{2k}\}$, where all the edges of Q are antipodal edges. Suppose that Q has either no a-chords, that is Q is induced in H, or Q has precisely the a-chord $\gamma_1\gamma_{2k}$. We will show that each graph in $\mathcal{F}_0$ contains such a cycle with possible d-chords with an extremal in γ_0. The following fact about such a Q is crucial to prove the lemma and it implies that if Q has at least one d-cords with an extremal in γ_0, then Q induces in M a copy of F_{2k+1}, DF_{2k+1} or $\widetilde{F}_{2k+1}$.

m) If $\gamma_0\gamma_j$ is a d-chord of Q with, say, $\gamma_j \leq \gamma_0$, $j \notin \{1, 2k\}$, then Q has d-chords $\gamma_0\gamma_l$ with $\gamma_l \leq \gamma_0$, for all $l \notin \{1, 2k\}$. Moreover,

- if Q is induced in H and Q has some other d-chord, then Q possesses either all d-chords $\gamma_1\gamma_j$ with $\gamma_j \leq \gamma_1$, $j \notin \{0, 2\}$, or, symmetrically, all the d-chords $\gamma_{2k}\gamma_j$, with $\gamma_j \leq \gamma_{2k}$, $j \notin \{0, 2k-1\}$;
- if $\gamma_1\gamma_q$ is an a-chord of Q, then Q has no other d-chords.

Proof of m): In the first place, observe that $\gamma_{j-1} \leftrightarrow \gamma_j$ and $\gamma_{j+1} \leftrightarrow \gamma_j$ trivially imply $\gamma_{j-1} \bowtie \gamma_j$ and $\gamma_{j+1} \bowtie \gamma_j$ hence, by Lemma 94, it holds that $\gamma_0 \bowtie \gamma_{j-1}$ and $\gamma_0 \bowtie \gamma_{j+1}$. Thus $\gamma_0\gamma_{j-1}$ and $\gamma_0\gamma_{j+1}$ are d-chords of Q, because the unique possible a-chord is $\gamma_1\gamma_{2k}$. Necessarily $\gamma_{j-1} \leq \gamma_0$ for, if not, then $\gamma_j \leq \gamma_0 \leq \gamma_{j-1}$ would imply $\gamma_j \leq \gamma_{j-1}$ contradicting that $\gamma_{j-1} \leftrightarrow \gamma_j$. By the same reasons, $\gamma_{j+1} \leq \gamma_0$. A repeated application of this argument to $j-1$ and $j+1$ in place of j proves the first part of the claim (see Figure 8.4(a)).

The first part of the claim is clearly invariant under automorphisms of Q. Consequently, we deduce that if Q has another d-chord $\gamma_h\gamma_\ell$ with $\gamma_\ell \leq \gamma_h$ and $h \notin \{1, 2k\}$, then Q has also d-chords $\gamma_h\gamma_1$ and $\gamma_h\gamma_{2k}$. But this is impossible because it would imply $\gamma_{2k} \leq \gamma_h \leq \gamma_0$ while we know that $\gamma_0 \leftrightarrow \gamma_{2k}$. Hence all the other possible d-chords of Q have one extremal in $\{\gamma_1, \gamma_{2k}\}$. On the other hand Q cannot possess d-chords $\gamma_1\gamma_h$ and $\gamma_{2k}\gamma_\ell$ for some $h, \ell \in [2k]$ because, by the first part of the claim, it would possess the d-chord $\gamma_1\gamma_{2k}$ and this would imply $\gamma_{2k} \leq \gamma_1$ and $\gamma_1 \leq \gamma_{2k}$ and consequently the contradiction $\gamma_1 = \gamma_{2k}$ (see Figure 8.4(b) and Figure 8.4(c)).

It remains to prove that if $\gamma_1\gamma_{2k}$ is an a-chord of Q, then Q has no other d-chords with one extremal in $\{\gamma_1, \gamma_{2k}\}$ (hence no other d-chords at all, as Figure 8.4(d)) shows). Suppose that Q has a d-chord with one extremal in $\{\gamma_1, \gamma_{2k}\}$, γ_1 say. Then Q has the d-chord $\gamma_1\gamma_{2k-1}$ by above. Since $\gamma_{2k-1} \leq \gamma_0$, $\gamma_{2k-1} \leq \gamma_1$ and $\gamma_{2k-1} \leftrightarrow \gamma_{2k}$, by Lemma 94 it follows that $\{\gamma_0, \gamma_1, \gamma_{2k}\}$ induces a full antipodal triangle in M, contradicting that M has no such triangles. End proof of m).

We can now complete the proof of the lemma. Let S be a copy in M of any of the three graphs in $\mathcal{F}_0$, and let S have n vertices $\gamma_0, \gamma_1, \ldots, \gamma_{n-1}$. Observe that S possesses an odd cycle

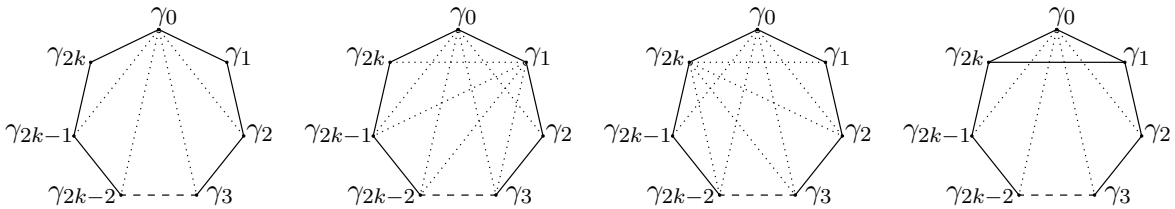


Figure 8.4: graphs in the proof of m). Note that the graph in (a) is isomorphic to F_{2k+1}, while the graphs in (b) and (c) are isomorphic to DF_{2k+1}, and the graph in (d) is isomorphic to $\widetilde{F}_{2k+1}$.

R on at least $n-1$ vertices; more precisely, the wheels have an odd cycle on $n-1$ vertices and the fan on n vertices. Let γ_0 be the highest degree vertex in S and let H_R and M_R be the graphs induced by R in H and M, respectively. Let Q be a cycle with minimum possible order among the odd cycles of order at least 5 contained in H_R. Hence either Q is an odd hole of H or Q is an odd cycle of H with exactly one a-chord which belongs to a triangle having the other two edges in Q, otherwise the minimality is denied. Clearly, the dominance edges of S induced by $V(Q)$ are d-chords of Q. Suppose first that Q has no extra d-chords other than those. In this case we are done because, $V(Q) \cup \{\gamma_0\}$ (possibly $\theta \in V(Q)$ when S is a fan) induces either a wheel or a fan or a chorded fan. We may therefore assume that Q possesses some extra d-chord (a dominance edge of M_R which is not in S). Possibly after relabeling, Q is of the form described in m) and Q possesses all the d-chords $\gamma_0\gamma_i$, $i \in [n-1]$ by m). If Q possesses no other d-chords we are done, because $V(Q)$ induces either a chorded fan or a fan according to whether or not Q possesses the unique a-chord $\gamma_1\gamma_t$. If Q possesses some other d-chord, still by m), then Q possesses either all d-chords $\gamma_1\gamma_j$ with $\gamma_j \leq \gamma_1$, $j \notin \{0, 2\}$, or all the d-chords $\gamma_t\gamma_j$, with $\gamma_j \leq \gamma_t$, $j \notin \{0, t-1\}$. In this case $V(Q)$ induces a double fan in M. The proof is completed. □

8.4.2 Comparison with Lévêque, Maffray, and Preissmann's characterization

We give a brief comparison of our characterization with Lévêque, Maffray, and Preissmann's characterization [98], whose list of minimal forbidden subgraphs of path graphs is given in Figure 8.5.

Table 8.1 gives a kind of dictionary between the two characterizations. The table reads as follows. For each row of the table, if a chordal graph G contains an induced copy of one of the subgraphs in the leftmost column (according to the characterization in [98]), then each of the graphs in the rightmost column occurs as an induced copy in the C-attachedness graph of G^+ for some clique separator C (according to our characterization). From the table it is apparent a sort of coarsening of the obstructions.

We do not prove how we obtain Table 8.1 because it can be proved by enumeration, but we report few observations. First of all, it is not necessary to build graph G^+ but it suffices to build the attacchedness graph of G, for G equal to every obstruction, and observe that a full antipodal triangle corresponds to $W_3^{(0)}$ in the attacchedness graph of G^+ (see Lemma 101's proof). Obstructions F_i for $i \in \{1, 2, 3, 4, 6, 7, 13, 14, 15\}$ have exactly one clique separator and

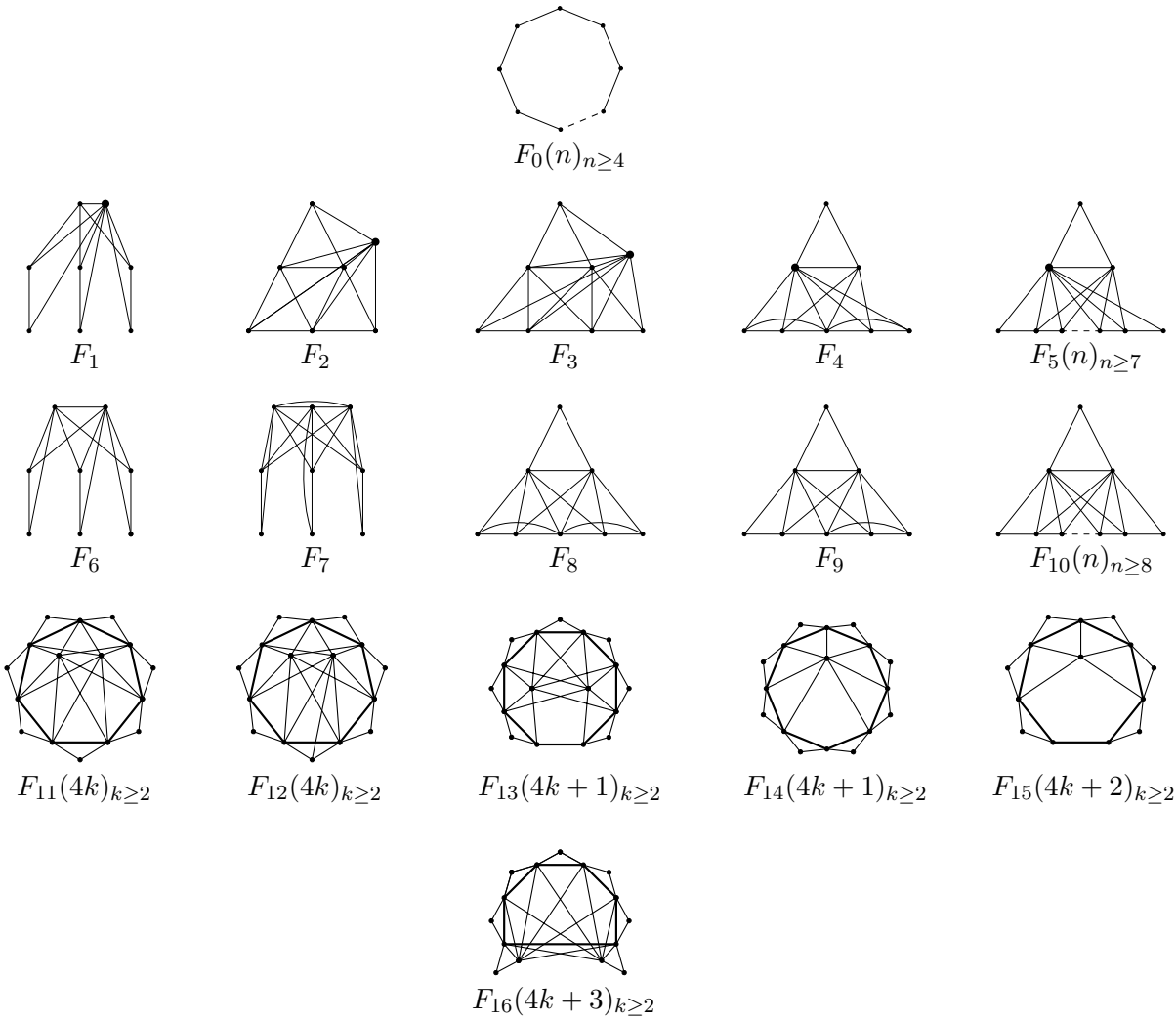


Figure 8.5: Lévêque, Maffray and Preissmann's exhaustive list of minimal non path graphs [98] (bold edges form a clique).

thus there is one to one correspondence between obstructions in [98] and ours. Obstructions F_j for $j \in \{8, 9, 11, 16\}$ have exactly two clique separators that are symmetric, thus they generate the same obstruction in $\mathcal{F}$. The same applies on obstructions $F_5(n)$ and $F_{10}(n)$, where the number of clique separators grows with n but all clique separators generate similar attacchedness graphs that have the same obstruction.

We have to give particular attention only on $F_{12}(4k)$ because it has two clique separators that generate two different attacchedness graphs, moreover, we need to distinguish the case $k = 2$ and the case $k > 2$, as we reported in Table 8.1.

Finally we remark that the obstructions in our characterization are 2-edge colored subgraphs and that they have to be forbidden in each graph of the collection of the attachedness graph of G^+, while in Lévêque, Maffray, and Preissmann's characterization the obstructions are forbidden

in the input graph itself.

Family	Obstruction
$F_1, F_2, \ldots, F_9, F_{10}$	$W_3^{(0)}$
$F_{11}(4k)_{k\geq 2}$	$W_{2k-1}^{(0)}$
$F_{12}(4k)_{k\geq 2}$	$W_3^{(0)}$, $W_3^{(1)}$, (for $k=2$), F_{2k-1}, $W_{2k-1}^{(1)}$ (for $k>2$)
$F_{13}(4k+1)_{k\geq 2}$	DF_{2k-1}
$F_{14}(4k+1)_{k\geq 2}$	$\widetilde{F}_{2k+1}$
$F_{15}(4k+2)_{k\geq 2}$	F_{2k+1}
$F_{16}(4k+3)_{k\geq 2}$	F_{2k+1}

Table 8.1: A dictionary between Lévêque, Maffray and Preissmann's characterization and Statement 102.(4) in Theorem 102. Note that F_0 is the obstruction to chordality.

Chapter 9

A new algorithm to recognize path graphs and directed path graphs

We present a new recognition algorithm for path graphs and directed path graphs. It has the same time complexity as the faster recognition algorithms known so far but it does not require complex data structures and has an easy and intuitive implementation.

9.1 Introduction

In this chapter we present a recognition algorithm that specializes for path graphs and directed path graphs, it has $O(p(m+n))$ time complexity (we recall that p denotes the number of cliques). Our algorithm is based on the characterization of directed path graphs given by Monma and Wei [102], and the characterization of path graph described in Theorem 93.

On path graph side, our algorithm requires a treatment that is shorter than both algorithms proposed by Schäffer [127] and Chaplick [36], its implementation is easier and it does not need complex data structures, while Chaplick's algorithm is based on PQR-trees and Schäffer's one is a complex backtracking algorithm.

On directed path graph side, to the best of our knowledge, our algorithm is the only one that does not use results by Chaplick *et al.* [37], in which it is given a linear algorithm able to establish whether a path graph is a directed path graph too (as stated in Theorem 11).

In this way, we do not improve time complexity but we unify and strictly simplify the study of path graphs and directed path graphs from an algorithmic point of view. Main results are in Theorem 115 and Theorem 117.

9.2 Recognition algorithm for path graphs

In this section we introduce algorithm `RecognizePathGraphs`, that is able to recognize path graph. In Subsection 9.2.1 we show the algorithm and we prove its correctness. In Subsec-

tion 9.2.2 we show its time complexity.

For convenience in the following corollary we resume all the conditions of the characterization of path graphs described in Theorem 93, which is based on Definition 91 and Definition 92.

Corollary 106. *A chordal graph G is a path graph if and only if G is an atom or for a clique separator C each graph $\gamma \in \Gamma_C$ is a path graph, $Upper_C$ has no full antipodal triangles and there exists $f : \Gamma_C \to [r+1]$ such that:*

106.(1) for all $i \in [r]$, $f(u_i) = i$;

106.(2) for all $i \in [r]$, $f(D_i) \subseteq \{i, r+1\}$;

106.(3) for all $i < j \in [r]$, $f(D_{i,j}) \subseteq \{i, j\}$;

106.(4) for all $i \in [r]$, for all $\gamma \in D_i$, if $\exists u \in Upper_C$ such that $\gamma \leftrightarrow u$, then $f(\gamma) = i$;

106.(5) for all $i < j \in [r]$, for all $\gamma \in D_{i,j}$ such that $\exists \gamma' \in D_k$, for $k \in \{i, j\}$, satisfying $\gamma \leftrightarrow \gamma'$, then $f(\gamma) = \{i, j\} \setminus \{k\}$;

106.(6) for all $D \in \mathcal{D}^C$, for all $\gamma, \gamma' \in D$ such that $\gamma \leftrightarrow \gamma'$, $f(\gamma) \neq f(\gamma')$.

9.2.1 The algorithm and its correctness

Now we present algorithm `RecognizePathGraphs`. Note that it is an implementation of Corollary 106 with very small changes. W.l.o.g., we assume that G is connected, indeed, a graph G is a path graph if and only if all its connected components are path graphs. Moreover, we can obtain the clique path tree of G by merging arbitrarily the clique path tree of each connected component.

Algorithm `RecognizePathGraphs`

Input: a graph G

Output: if G is a path graph, then return a clique path tree of G; else return `FALSE`

Step 1 Test if G is chordal. If not, then return `FALSE`.

Step 2 If G has at most two cliques, then return a clique path tree of G. Else find a clique separator C, let $\Gamma_C = \{\gamma_1, \ldots, \gamma_s\}$, $\gamma_i = G[V_i \cup C]$, be the set of connected components separated by C.

Step 3 Recursively test the graphs $\gamma \in \Gamma_C$. If any one is not a path graph, then return `FALSE`, otherwise, return a clique path tree T_γ for each $\gamma \in \Gamma_C$.

Step 4 Compute $\Gamma_C/\sim$ and initialize $f(\gamma) =$ `NULL`, for all $\gamma \in \Gamma_C/\sim$. Compute Upper_C and fix an order of its element, i.e., $\text{Upper}_C = (u_1, \ldots, u_r)$, and set $f(u_i) = i$, for all $i \in [r]$. If a full antipodal triangle in Upper_C is found, then return `FALSE`. Compute D_i, for all $i \in [r]$, and $D_{i,j}$, for $i < j \in [r]$.

Step 5 For all $i \in [r]$, if there exist $\gamma \in D_i$ and $\gamma' \in \text{Upper}_C$ such that $\gamma \leftrightarrow \gamma'$, then $f(\gamma) = i$.

Step 6 For all $i \in [r]$, extend f to all elements of D_i so that $f(D_i) \subseteq \{i, r+1\}$ and $f(\gamma) \neq f(\gamma')$ for all $\gamma, \gamma' \in D_i$ satisfying $\gamma \not\leftrightarrow \gamma'$. If it is not possible, then return `FALSE`.

Step 7 For all $i < j \in [r]$

- if there exist $\gamma \in D_{i,j}$, $\gamma' \in D_i$, $\gamma'' \in D_j$, such that $\gamma \leftrightarrow \gamma'$ and $\gamma \leftrightarrow \gamma''$, then return `FALSE`;
- if there exist $\gamma \in D_{i,j}$ and $\gamma' \in D_j$ such that $\gamma \leftrightarrow \gamma'$, then $f(\gamma) = i$;
- if there exist $\gamma \in D_{i,j}$ and $\gamma' \in D_i$ such that $\gamma \leftrightarrow \gamma'$, then $f(\gamma) = j$.

Step 8 For all $i < j \in [r]$, extend f to all elements of $D_{i,j}$ so that $f(D_{i,j}) \subseteq \{i, j\}$ and $f(\gamma) \neq f(\gamma')$ for all $\gamma, \gamma' \in D_{i,j}$ satisfying $\gamma \not\leftrightarrow \gamma'$. If it is not possible, then return `FALSE`.

Step 9 Convert the coloring $f : \Gamma_C/\sim \to [r+1]$ in a clique path tree of Γ_C.

Theorem 107. *Given a graph G, algorithm `RecognizePathGraphs` can establish whether G is a path graph. If it so, algorithm `RecognizePathGraphs` returns a clique path tree of G.*

Proof. The first three steps of algorithm `RecognizePathGraphs` are implied by the first part of Corollary 106. By following Corollary 106, we have to check that there are no full antipodal triangles in Upper_C (this is made in Step 4), and we have to find $f : \Gamma_C \to [r+1]$ satisfying 106.(1),...,106.(6), where $r = |\text{Upper}_C|$. The latter part is done in Step 5, Step 6, Step 7 and Step 8. In particular Step 5 is implied by 106.(4), Step 7 by 106.(5), Step 6 and Step 8 both by 106.(6). Finally, Step 9 completes the recursion started in Step 3. At the end of recursion we have to build the clique path three on Γ_C. □

9.2.2 Implementation details and time complexity

In this subsection we analyze all steps of algorithm `RecognizePathGraphs`. We want to explain them in details and compute the computational complexity of the algorithm. Some of these steps are already discussed in [127], anyway, we describe them to have a complete treatment.

Step 1 and Step 2

We can recognize chordal graphs in $O(m+n)$ time by using either an algorithm due to Rose *et al.* [125], or an algorithm due to Tarjan and Yannakakis [133]. Both recognition algorithms can be extended to an algorithm that also produces a clique tree in $O(m+n)$ time [108]. In particular, we can list all cliques of G. It holds that a clique in $\mathbf{C}$ is a separator clique if and only if it is not a leaf of the clique tree.

Step 3

This step can be done by calling recursively algorithm `RecognizePathGraphs` for all $\gamma \in \Gamma_C$. Obviously, Step 1 has to be done only for G, indeed, the property to be chordal is inherited by subgraphs.

From now on, we focus exclusively on the recursive part of algorithm `RecognizePathGraphs`. Thus we suppose that G is a chordal graph that is not an atom and G is separated by a clique

separator C. Let $\Gamma_C = \{\gamma_1, \ldots, \gamma_s\}$ be the set of connected components and we assume that all elements in Γ_C are path graphs. It holds that $s \geq 2$ because C is a clique separator.

Step 4 and Step 5

We have to compute $\Gamma_C/\sim$, this problem is already solved in [127]. Thus we first report some definitions and results in [127].

For each $\gamma \in \Gamma_C$, let T_γ be the clique path tree of γ. Let n_γ be the unique neighbour of C in T_γ (its uniqueness is proved in [102], in particular, it is proved that C is a leaf of T_γ). Moreover, let $W(\gamma) = V(n_\gamma) \cap V(C)$.

Definition 108 ([127]). *Let $\gamma \in \Gamma_C$ and $v \in V(C)$. We define $F(\gamma, v)$ as the node of T_γ representing the clique containing v that is furthest from C. We observe that if $v \notin W(\gamma)$, then $F(\gamma, v) = \emptyset$.*

By using Definition 108, one can obtain the following lemma.

Lemma 109 ([127]). *Let $\gamma, \gamma' \in \Gamma_C$. It holds that*

$$\gamma' \leq \gamma \Leftrightarrow \gamma \cap \gamma' \neq \emptyset \text{ and } F(\gamma, v) = F(\gamma, v') \text{ for all } v, v' \in W(\gamma').$$

Moreover, computing if $\gamma' \leq \gamma$ and/or $\gamma' \leftrightarrow \gamma$ costs $\min(|W(\gamma)|, |W(\gamma')|)$.

Remark 110 ([127]). *To compute $F(\gamma, v)$ we do one breath-first traversal of T_γ starting at n_γ. Each time we visit a new node C', for each $v \in V(C')$, if $v \in V(C)$, we update $F(\gamma, v)$. This costs constant time for each pair (C', v) such that C' is a clique in γ and $v \in V(C')$. There are at most $m + n$ such pairs.*

Now we return to the discussion of computing $\Gamma_C/\sim$. First, we sort all $\gamma \in \Gamma_C$ so that the γ_i precedes γ_j, if $|W(\gamma_i)| \geq |W(\gamma_j)|$: we compute $W(\gamma)$ for all $\gamma \in \Gamma_C$ (it costs $|W(\gamma)|$ for each γ), then the sort can be executed in $O(s)$ time by using bucket sort (we remember that $s \leq n$).

Now, let $\gamma, \gamma', \gamma'' \in \Gamma_C$ satisfy $|W(\gamma)| = |W(\gamma')| = |W(\gamma'')|$ and $v \in W(\gamma) \cap W(\gamma') \cap W(\gamma'')$, for any $v \in C$. By Lemma 109, we check if $\gamma \sim \gamma'$ in $O(|W(\gamma)|)$ time. If $\gamma \not\sim \gamma'$, then either ($\gamma'' \sim \gamma$ or $\gamma'' \sim \gamma'$) or $\{\gamma, \gamma', \gamma''\}$ is a full antipodal triangle (this follows from definition of $\leftrightarrow$ and W). Hence we can compute $\Gamma_C/\sim$ in $O(\sum_{\gamma \in \Gamma_C} |W(\gamma)|)$ time, indeed, each element in Γ_C need to be checked with at most two other elements in Γ_C.

To argue with the second part of Step 4, let $u(v) = \{\gamma \in \mathrm{Upper}_C \mid v \in W(\gamma)\}$. Note that $|u(v)| \leq 2$ for all $v \in V(C)$, otherwise $u(v) \supseteq \{u_i, u_j, u_k\}$ for some $v \in V(C)$, thus $\{u_i, u_j, u_k\}$ is a full antipodal triangle. Hence, by Lemma 109, $\gamma \in \mathrm{Upper}_C$ if and only if there does not exist any $u_i \in \mathrm{Upper}_C$ such that $F(u_i, v) = F(u_i, v') \neq \emptyset$ for all $v, v' \in W(\gamma)$. Hence, to establish whether γ is in Upper_C it is sufficient to look at $u(v)$ for all $v \in W(\gamma)$. By using a similar argument, we can compute D, for $D \in \mathcal{D}$. Also Step 5 can be done with a similar procedure. Hence these steps can be performed in $O(\sum_{\gamma \in \Gamma_C} |W(\gamma)|)$ time.

Step 6 and Step 8

We define $f_i(\cdot)$ as $f(\cdot)$ after Step i requiring that in Step i algorithm `RecognizePathGraphs` does not return `FALSE`. We will use f_5, f_7 and f_9.

Lemma 111. *The following statements hold:*

111.(1) let $i \in [r]$ and let $\gamma, \gamma' \in D_i$ be such that $\gamma' \leq \gamma$. If $f_5(\gamma) =$ `NULL`, then $f_5(\gamma') =$ `NULL`;

111.(2) let $i < j \in [r]$ and let $\gamma, \gamma' \in D_{i,j}$ be such that $\gamma' \leq \gamma$. If $f_7(\gamma) =$ `NULL`, then $f_7(\gamma') =$ `NULL`.

Proof. We prove both statements separately.

111.(1) Assume by contradiction that $f_5(\gamma') \neq$ `NULL` and $f_5(\gamma) =$ `NULL`. By Step 5 there exists $i \neq k \in [r]$ such that $u_k \leftrightarrow \gamma'$, and thus u_k, γ' are neighboring of v, for some $v \in C$. Thus γ' is a neighboring of v by transitivity of $\leq$. Hence it holds either $\gamma \leq u_k$, or $u_k \leq \gamma$, or $u_k \leftrightarrow \gamma$. Now, $u_k \not\leq \gamma$ because $u_k \in \text{Upper}_C$, $\gamma \not\leq u_k$ because $\gamma \in D_i$ and $k \neq i$. Thus $u_k \leftrightarrow \gamma$ and Step 5 implies $f_5(\gamma) = i = f_5(\gamma')$, absurdum.

111.(2) Assume by contradiction that $f_7(\gamma') \neq$ `NULL` and $f_7(\gamma) =$ `NULL`. W.l.o.g., let us assume that $f_7(\gamma') = i$. By Step 7, there exists $\eta \in D_j$ such that $\eta \leftrightarrow \gamma'$. Thus $\eta \bowtie \gamma$ by transitivity of $\leq$. Hence, as before, it holds either $\gamma \leq \eta$, or $\eta \leq \gamma$, or $\eta \leftrightarrow \gamma$. Now, $\eta \not\leq \gamma$, otherwise $\eta \in D_{i,j}$, $\gamma \not\leq \eta$, otherwise $\gamma \leq \eta$ by transitivity of $\leq$. Thus $\eta \leftrightarrow \gamma$ and Step 7 implies $f_7(\gamma) = i = f_7(\gamma')$, absurdum. □

Lemma 112. *Step 6 and Step 8 have $O(\sum_{\gamma \in \Gamma_C} |W(\gamma)|)$ time complexity.*

Proof. It suffices to prove that Step 8 for $D = D_{i,j}$ can be executed in $O(\sum_{\gamma \in D_{i,j}} |W(\gamma)|)$. Indeed, the same procedure can be applied for Step 6 for $D = D_i$.

Let Colored be the set of elements of $D_{i,j}$ already colored before Step 8, i.e., Colored $= \{\gamma \in D_{i,j} \mid f_7 \neq$ `NULL`$\}$. We first check that there are not $\gamma, \gamma' \in$ Colored such that $\gamma \leftrightarrow \gamma'$ and $f(\gamma) = f(\gamma')$.

We sort elements of $D_{i,j} = (\gamma^1, \ldots, \gamma^{|D_{i,j}|})$ so that $|W(\gamma^k)| \geq |W(\gamma^{k+1})|$, for $k \in [|D| - 1]$ (this can be done in Step 4 without adding cost). We visit elements in Colored by following this order. For any $v \in W(D_{i,j})$ we define $\ell_i(v)$ as the lowest $\gamma \in D_{i,j}$ w.r.t $\leq$ among all visited element satisfying $f_7(\gamma) = i$. Similarly, we define $\ell_j(v)$. We initialize $\ell_i(v) = \ell_j(v) = \emptyset$ for all $v \in W(D_{i,j})$.

Let $\gamma \in$ Colored, and, w.l.o.g., let us assume that $f_7(\gamma) = i$. Then γ is not antipodal to previous visited elements colored with i if $\ell_i(u) = \ell_i(v)$ for all $u, v \in W(\gamma)$. Indeed, either $\ell_i(v) = \emptyset$ for all $v \in W(\gamma)$ and thus $\gamma \not\bowtie \gamma'$ for all visited γ' colored with i, or $\ell_i(v) = \gamma'$ for all $v \in W(\gamma)$ and hence there cannot exist γ'' already visited satisfying $\gamma'' \leftrightarrow \gamma$ because it would imply $\ell_i(v) = \gamma''$ for some $v \in \gamma''$.

Now we deal with uncolored elements. We define Uncolored as the set of all elements of $D_{i,j}$ that have not an assigned color. We say that $\gamma \in$ Uncolored is *solved* if there exist $u, v \in W(\gamma)$ such that $\ell_i(u) \neq \ell_i(v)$ or $\ell_j(u) \neq \ell_j(v)$. Note that if γ is solved, then either we can set

(uniquely) its color or we have to return FALSE; both cases are implied by Lemma 111 and the above reasoning done for Colored.

For all $v \in W(D_{i,j})$ we write $\gamma \in M_v$ if $v \in W(\gamma)$, $\gamma \in$ Uncolored and $|W(\gamma)|$ is maximal among all $\gamma' \in$ Uncolored satisfying $v \in W(\gamma')$. Note that $|M_v| \leq 2$ for all $v \in W(D_{i,j})$, otherwise there would be a full antipodal triangle.

Now we describe how to set the color of elements in Uncolored. We search (in any order) $v \in W(D_{i,j})$ such that M_v is solved. We observe that if we visit M_v before M_u, M_v is not solved, M_u is solved and $W(M_u) \cap W(M_v) \neq \emptyset$, then M_v becomes solved after the (uniquely) choose of the color of M_u. Moreover, if for all $v \in W(D_{i,j})$ M_v is not solved, then for all $\gamma \in$ Uncolored and $\gamma \in D_{i,j} \setminus$ Uncolored it holds $\gamma \not\leftrightarrow \gamma'$ because of definition of $M_i(v)$'s. Thus the 2-coloration of $D_{i,j}$ required in Step 8 (if it exists) it is not unique, and thus we can choose $v \in W(D_{i,j})$ and the color of M_v arbitrarily.

In this way we visit each $\gamma \in D_{i,j}$ at most two times and each time we spend at most $O(|W(\gamma)|)$ time. Finally, we can update $\ell_i(v)$'s and $\ell_j(v)$'s each time we visit an element γ of Colored or we set the color of γ' in Uncolored in $O(|W(\gamma)|)$ and $O(|W(\gamma')|)$ time, respectively. We update M_v's in $O(1)$ time if we represent M_v as a vector; we build these vectors varying $v \in W(D_{i,j})$ in $O(\sum_{\gamma \in D_{i,j}} |W(\gamma)|)$ time. The thesis follows. □

Step 7

In the proof of the following lemma we show how to do Step 7.

Lemma 113. *Step 7 has $O(\sum_{\gamma \in \mathcal{D}} |W(\gamma)|)$ time complexity.*

Proof. For all $k \in [r]$ let $\ell_k(v)$ be as defined in the proof of Lemma 112, thus we can assume that we know $\ell_k(v)$ for all $k \in [r]$ and $v \in u_k$ after Step 6. We need the following result.

n) let $k \in \{i, j\}$, then $\exists \gamma' \in D_k$ such that $\gamma \leftrightarrow \gamma'$ if and only if $\gamma \leftrightarrow \ell_k(v)$, for any $v \in W(\gamma)$.

Proof of n): The "only if part" is trivial. For the "if part", let us assume by contradiction that $\gamma \not\leftrightarrow \ell_k(v)$ for all $v \in W(\gamma)$ and that there exists $\gamma' \in D_k$ such that $\gamma \leftrightarrow \gamma'$. Let $v \in W(\gamma)$, if $\gamma \not\leftrightarrow \ell_k(v)$, then $\gamma \leq \ell_k(v)$, because of definitions of $\ell_k(v)$ and D_k. By generality of v, $\gamma \geq \ell_k(v')$ for all $v' \in W(\gamma)$. Now, let $z \in W(\gamma) \cap W(\gamma')$. It holds that $\ell_k(z) \leq \gamma'$, by definition of $\ell_k(z)$, hence $\gamma \leq \gamma'$, absurdum. End proof of n).

Now let $i < j \in [r]$. To do Step 7 for $D_{i,j}$, for all $\gamma \in D_{i,j}$, by n) it suffices to choose an arbitrary $v \in W(\gamma)$ and check if $\gamma \leftrightarrow \ell_i(v)$ and $\gamma \leftrightarrow \ell_j(v)$. By Lemma 109, it costs $O(|W(\gamma)|)$ time. The thesis follows. □

Step 9

Let C be the clique separator of G fixed at Step 2. In [102] (proof of Proposition 9) it is shown how to build a clique path tree on the cliques in Γ_C starting from a coloring satisfying 86.(1) and 86.(2) in $O(|\Gamma_C|)$ time. Finally, by Theorem 107, f_9 satisfies 86.(1) and 86.(2), and Lemma 89 implies that Step 9 has $O(|\Gamma_C|)$ time complexity.

Time complexity

In Theorem 115 we show that algorithm `RecognizePathGraphs` has $O(p(m+n))$ time complexity by summarizing the results of previous subsections and by using the following lemma.

Lemma 114 ([127]). *For every clique separator C of G, then $\sum_{\gamma\in\Gamma_C}(|W(\gamma)|) \leq m+n$.*

Theorem 115. *Algorithm `RecognizePathGraphs` can establish whether a graph G is a path graph in $O(p(m+n))$ time.*

Proof. By Theorem 107, it suffices to prove that algorithm `RecognizePathGraphs` can be executed in $O(p(m+n))$ time. Step 1 and Step 2 have $O(m+n)$ time complexity, while Step 3 has p times the complexity of steps 4,...,9. Let C be the clique separator in a recursive call of Step 3. Steps $4,\ldots,8$ can be executed in $O(\sum_{\gamma\in\Gamma_C}|W(\gamma)|)$ time and Step 9 has $O(|\Gamma_C|)$ time complexity. By Lemma 114 and being $|\Gamma_C| \leq n$ for all clique separators C, the thesis follows. □

9.3 Recognition algorithm for directed path graphs

In this section we present algorithm `RecognizeDirectedPathGraphs` that is able to recognize directed path graphs. It is based on Theorem 10 and on algorithm `RecognizePathGraphs`, both algorithms have the same time complexity.

Thanks to Theorem 10, fixed a clique separator C, we have to check if H_{Γ_C} is 2-colorable; where $H_X = (X, \{\gamma\gamma' \mid \gamma,\gamma' \in X \text{ and } \gamma \leftrightarrow \gamma'\})$ for each $X \subseteq \Gamma_C$.

If H_{Γ_C} is 2-colorable, then there are no antipodal triangles in Γ_C. Moreover, the absence of antipodal triangles implies the absence of full antipodal triangles. Thus, if there are no antipodal triangles in Γ_C, then D_i and $D_{i,j}$'s form a partition of Γ_C, as proved in Chapter 8.

We can 2-color elements of Γ_C in the following order: first we 2-color Upper_C, then, for all $D \in \mathcal{D}$, we color all elements in D that are antipodal to at least one element not in D, finally, for all $D \in \mathcal{D}$, we color all elements in D that are antipodal to other elements in D. This procedure is correct even if there are more connected component in H_{Γ_C}. Indeed, we note that if $u, u' \in \text{Upper}_C$ are in two distinct connected components of H_{Upper_C}, then u, u' are also in two distinct connected components of H_{Γ_C} because of definition of Upper_C and relations $\leq$ and $\leftrightarrow$. Moreover, fixed $D \in \mathcal{D}$, elements in D that are antipodal to elements not in D have the color uniquely determined if the colors of Upper_C are already fixed. Thus we have proved the following theorem (note that 116.(1) implies that there are no full antipodal triangles in Upper_C).

Theorem 116. *A chordal graph G is a directed path graph if and only if G is an atom or for a clique separator C each graph $\gamma \in \Gamma_C$ is a directed path graph and there exists $f : \Gamma_C \to \{0,1\}$ such that:*

116.(1) for all $u, u' \in \text{Upper}_C$, if $u \leftrightarrow u'$, then $f(u) \neq f(u')$;

116.(2) for all $i \in [r]$, for all $\gamma \in D_i$ if $\exists u \in \text{Upper}_C$ such that $\gamma \leftrightarrow u$, then $f(\gamma) = f(u_i)$;

116.(3) for all $i < j \in [r]$, for all $\gamma \in D_{i,j}$ such that $\exists \gamma' \in D_k$, for $k \in \{i,j\}$, satisfying $\gamma \leftrightarrow \gamma'$, then $f(\gamma) = \{0,1\} \setminus f(u_k)$;

116.(4) for all $D \in \mathcal{D}^C$, for all $\gamma, \gamma \in D$ such that $\gamma \leftrightarrow \gamma'$, $f(\gamma) \neq f(\gamma')$.

Theorem 116 implies algorithm `RecognizeDirectedPathGraphs`. For a proof of its correctness, we remind to Theorem 107's proof, by using Theorem 10 in place of Theorem 9.

Algorithm `RecognizeDirectedPathGraphs`

Input: a graph G

Output: if G is a directed path graph, then return a directed clique path tree of G; else return `FALSE`

Step 1.D Test if G is chordal. If not, then return `FALSE`.

Step 2.D If G has at most two cliques, then return a directed clique path tree of G. Else find a clique separator C, let $\Gamma_C = \{\gamma_1, \ldots, \gamma_s\}$, $\gamma_i = G(V_i \cup C)$, be the set of connected components separated by C.

Step 3.D Recursively test the graphs $\gamma \in \Gamma_C$. If any one is not a directed path graph, then return `FALSE`, otherwise, return a directed clique path tree T_γ for each $\gamma \in \Gamma_C$.

Step 4.D Compute $\Gamma_C/\sim$ and initialize $f(\gamma) =$ `NULL`, for all $\gamma \in \Gamma_C/\sim$. Function f has values in $\{0,1\}$. Compute Upper_C and fix an order of its element, i.e., $\mathrm{Upper}_C = (u_1, \ldots, u_r)$. Assign values of f to elements in Upper_C so that antipodal elements have different color. If it is not possible, then return `FALSE`. Compute D_i, for all $i \in [r]$, and $D_{i,j}$, for $i < j \in [r]$.

Step 5.D For all $i \in [r]$

- if there exist $\gamma \in D_i$, $u, u' \in \mathrm{Upper}_C$ such that $\gamma \leftrightarrow u$, $\gamma \leftrightarrow u'$, $f(u) \neq f(u')$, then return `FALSE`;
- if there exist $\gamma \in D_i$ and $u \in \mathrm{Upper}_C$ such that $\gamma \leftrightarrow u$, then $f(\gamma) \neq f(u)$.

Step 6.D For all $i \in [r]$, extend f to all elements of D_i so that $f(\gamma) \neq f(\gamma')$ for all $\gamma, \gamma' \in D_i$ satisfying $\gamma \not\leftrightarrow \gamma'$. If it is not possible, then return `FALSE`.

Step 7.D For all $i < j \in [r]$

- if there exist $\gamma \in D_{i,j}$, $\gamma' \in D_i$, $\gamma'' \in D_j$, such that $\gamma \leftrightarrow \gamma'$ and $\gamma \leftrightarrow \gamma''$, then return `FALSE`;
- if there exist $\gamma \in D_{i,j}$ and $\gamma' \in D_j$ such that $\gamma \leftrightarrow \gamma'$, then $f(\gamma) = f(u_i)$;
- if there exist $\gamma \in D_{i,j}$ and $\gamma' \in D_i$ such that $\gamma \leftrightarrow \gamma'$, then $f(\gamma) = f(u_j)$.

Step 8.D For all $i < j \in [r]$, extend f to all elements of $D_{i,j}$ so that $f(\gamma) \neq f(\gamma')$ for all $\gamma, \gamma' \in D_{i,j}$ satisfying $\gamma \not\leftrightarrow \gamma'$. If it is not possible, then return `FALSE`.

Step 9.D Convert the coloring $f : \Gamma_C \to \{0,1\}$ in a directed clique path tree of Γ_C.

Every step of algorithm `RecognizeDirectedPathGraphs` has the same time complexity of the corresponding step of algorithm `RecognizePathGraphs`. We do not need a proof of it except for Step 4.D (the part regarding the 2-coloring of Upper_C) and Step 9.D.

For the complexity of Step 9.D we proceed as in algorithm `RecognizePathGraphs`. Let C be the clique separator of G fixed at Step 3.D. In [102] (proof of Proposition 9) it is shown how to build a directed clique path tree on the cliques in Γ_C starting from a 2-coloring of H_{Γ_C} in $O(|\Gamma_C|)$ time. This is also the complexity of Step 9.D.

To 2-color Upper_C in Step 4.D, note that $u, u' \in \text{Upper}_C$ are antipodal if and only if $W(u) \cap W(u') \neq \emptyset$. Thus Step 4.D requires $O(\sum_{u \in \text{Upper}_C} |W(u)|)$ time by using the same technique explained in Subsection 9.2.2.

Consequently, the following theorem applies.

Theorem 117. *Algorithm* `RecognizeDirectedPathGraphs` *can establish whether a graph G is a directed path graph in $O(p(m+n))$ time.*

Conclusions

In this thesis we argue mainly with non-crossing shortest paths in plane graphs and path graphs from a theoretical and algorithmic point of view.

All existing algorithms for the NCSP problem [130, 131] obtain the union of non-crossing shortest paths but are not able neither to compute paths' lengths nor to list a path in time proportional to its length. In [131] both problems are solved only in the restricted case when the union of non-crossing shortest paths is a forest but, in the general case, the union may contain cycles. Thanks to this thesis, these drawbacks are overcome. We give a linear time algorithm able to compute lengths of all the non-crossing shortest paths in total linear time and we succeed in covering the union of non-crossing paths with four forests so that every path is contained in at least one forest. In this way we can list a path in time proportional to its length by using the result by Gabow and Tarjan [54]. Moreover, we describe several structural properties of non-crossing shortest paths as shortcuts and their Path Covering with Forests Number, and we also solve the NCSP problem in the unweighted case in linear time. All these results can be easily applied in a geometric setting, where it is asked to search for paths in polygons instead of plane graphs.

We use non-crossing shortest paths also to compute an additive guaranteed approximation of the vitality of all edges and vertices with bounded capacity with respect to the st-max flow in undirected planar graphs. This allows us to determine the edges and vertices with highest vitality in time equal or close to the time currently required for the computation of the st-max flow. Moreover, we obtain some optimal results in the integer capacity case.

We also present two new characterizations of path graphs. To the best of our knowledge, our first characterization is the only one that directly describes a polynomial algorithm. In the second one we give a list of local minimal forbidden subgraphs. We use our first characterization to describe a recognition algorithm that specializes for path graphs and directed path graphs. This algorithm does not improve the time complexity of existing recognition algorithms but we unify and simplify both recognition problems from an algorithmic point of view.

Open problems and future work. Regarding non-crossing shortest paths in plane graphs, we left open the problem of finding the union of non-crossing shortest paths in $o(n \log \log k)$ time also in the weighted case; such a result would also imply solving planar max flow. All our results can be extended to the case of terminal pairs lying on two distinct faces, by the same argument shown in [131]; we wish to investigate the case in which terminal pairs lying on a

bounded number of faces or the case where each terminal pair contains only one vertex on the infinite face of the plane graph.

About edge and vertex vitalities, it is still open the problem of computing the exact vitality of all edges and vertices of an undirected planar graph within the same time bound as computing the max flow value, as in the *st*-planar case.

Our approach used to characterize path graphs may be extended to rooted path graphs, for which a list of minimal forbidden subgraphs is unknown, even if some partial results were found [33, 65, 66].

With the introduction of the Path Covering with Forests Number, we propose an original nuance on the classical covering problem and we would like to deal with the Path Covering with Forests Number in a more general context. The first generalization is to ask whether the Path Covering with Forests Number of a set of non-crossing single-touch shortest paths in a plane graph is bounded by a constant, even if the extremal vertices of the paths are not required to lie on the same face. We conjecture that such a constant exists for general plane graph thanks also to the following remark, whose proof is omitted.

Remark 118. *Let P be a set of non-crossing single-touch shortest paths in a plane graph G and let v be a vertex of G. Then all paths of P containing v form a tree.*

Conjecture 119. *There exists $\ell \in \mathbb{N}$ such that $PCFN(P) \leq \ell$, for any set P of non-crossing single-touch shortest paths in a plane graph.*

The Path Covering with Forests Number can be studied also for a set of paths P beyond planar graphs, as in k-planar graphs [112], k-quasi-planar graphs [1], RAC graphs [44], fan-crossing-free graphs [39], fan-planar graphs [84], k-gap-planar graphs [14]; a complete survey about these graph classes can be found in [76].

It would be interesting to investigate the Path Covering with Forests Number in the case of crossing paths in plane graphs or related graph classes. As shown in Section 7.1, for a set of crossing paths P, its Path Covering with Forests Number may depend on $|P|$. To deal with these cases it is necessary to understand "how much" the paths cross each others. In [126] several notions about crossing, crossing number and variants are explained, thus the Path Covering with Forests Number can be studied with respect to these measures. We note that in [126] the notions are given for graphs, but they can be extended to sets of paths.

Clearly, the Path Covering with Forests Number can be generalized to other covering families as done for the arboricity or the classical covering problem. Thus we can introduce, for example, the Path Covering with Stars Number, the Path Covering with Caterpillars Number, Path Covering with Planar Graphs Number and so on. We hope that more results on Path Covering with Forests Number or its variants for particular graphs and paths classes could lead to more efficient algorithms for shortest paths and distance problems.

Bibliography

[1] P. K. Agarwal, B. Aronov, J. Pach, R. Pollack, and M. Sharir, *Quasi-Planar Graphs Have a Linear Number of Edges*, Combinatorica, 17 (1997), pp. 1–9.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows*, (1988).

[3] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, and M. Reddy, *Applications of Network Optimization*, Handbooks in Operations Research and Management Science, 7 (1995), pp. 1–83.

[4] J. Akiyama, G. Exoo, and F. Harary, *Covering and packing in graphs IV: Linear arboricity*, Networks, 11 (1981), pp. 69–72.

[5] J. Akiyama and M. Kano, *Path Factors of a Graph, Graphs and Applications (Boulder, Colo., 1982), 1-21*, Wiley-Interscience Publication, Wiley, New York, (1985).

[6] D. L. Alderson, G. G. Brown, W. M. Carlyle, and L. A. Cox Jr, *Sometimes There Is No "Most-Vital" Arc: Assessing and Improving the Operational Resilience of Systems*, Military Operations Research, 18 (2013), pp. 21–37.

[7] I. Algor and N. Alon, *The Star Arboricity of Graphs*, Discrete Mathematics, 75 (1989), pp. 11–22.

[8] N. Alon, *The Linear Arboricity of Graphs*, Israel Journal of Mathematics, 62 (1988), pp. 311–325.

[9] N. Alon, C. McDiarmid, and B. A. Reed, *Star arboricity*, Combinatorica, 12 (1992), pp. 375–380.

[10] D. S. Altner, Ö. Ergun, and N. A. Uhan, *The Maximum Flow Network Interdiction Problem: Valid Inequalities, Integrality Gaps, and Approximability*, Operations Research Letters, 38 (2010), pp. 33–38.

[11] N. Apollonio and L. Balzotti, *A New Characterization of Path Graphs*, CoRR, abs/1911.09069 (2019).

[12] G. Ausiello, P. G. Franciosa, I. Lari, and A. Ribichini, *Max flow vitality in general and st-planar graphs*, Networks, 74 (2019), pp. 70–78.

[13] ——, *Max-flow vitality in undirected unweighted planar graphs*, CoRR, abs/2011.02375 (2020).

[14] S. W. Bae, J. Baffier, J. Chun, P. Eades, K. Eickmeyer, L. Grilli, S. Hong, M. Korman, F. Montecchiani, I. Rutter, and C. D. Tóth, *Gap-planar graphs*, Theoretical Computer Science, 745 (2018), pp. 36–52.

[15] Z. K. Baker and M. B. Gokhale, *On the Acceleration of Shortest Path Calculations in Transportation Networks*, in IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2007, 2007, pp. 23–34.

[16] L. Balzotti, *A New Algorithm to Recognize Path Graphs and Directed Path Graphs*, CoRR, abs/2012.08476v2 (2020).

[17] ——, *Non-Crossing Shortest Paths are Covered with Exactly Four Forests*, CoRR, (2022).

[18] L. Balzotti and P. G. Franciosa, *Non-Crossing Shortest Paths in Undirected Unweighted Planar Graphs in Linear Time*, Computer Science – Theory and Applications, CSR 2022. Lecture Notes in Computer Science, 13296 (2022), pp. 77–95.

[19] ——, *How Vulnerable is an Undirected Planar Graph with respect to Max Flow*, in Algorithms and Complexity: 13th International Conference, CIAC 2023, Proceedings, Springer, 2023.

[20] ——, *Non-Crossing Shortest Paths Lengths in Planar Graphs in Linear Time*, in Algorithms and Complexity: 13th International Conference, CIAC 2023, 2023, Proceedings, Springer, 2023.

[21] A. Bassa, J. Burns, J. Campbell, A. Deshpande, J. Farley, M. Halsey, S.-Y. Ho, D. Kleitman, S. Michalakis, P.-O. Persson, P. Pylyavskyy, L. Rademacher, A. Riehl, M. Rios, J. Samuel, B. E. Tenner, A. Vijayasarathy, and L. Zhao, *Partitioning a Planar Graph of Girth 10 into a Forest and a Matching*, Studies in Applied Mathematics, 124 (2010), pp. 213–228.

[22] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner, *Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm*, ACM Journal of Experimental Algorithmics, 15 (2010).

[23] L. W. Beineke, *A Survey of Packings and Coverings of Graphs*, in The many facets of graph theory, Springer, 1969, pp. 45–53.

[24] M. Bentert, A. Nichterlein, M. Renken, and P. Zschoche, *Using a Geometric Lens to Find k Disjoint Shortest Paths*, in 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, vol. 198 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 26:1–26:14.

[25] K. Bérczi and Y. Kobayashi, *The Directed Disjoint Shortest Paths Problem*, in 25th Annual European Symposium on Algorithms, ESA 2017, vol. 87 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 13:1–13:13.

[26] S. N. Bhatt and F. T. Leighton, *A Framework for Solving VLSI Graph Layout Problems*, Journal of Computer and System Sciences, 28 (1984), pp. 300–343.

[27] G. Bodwin, *On the Structure of Unique Shortest Paths in Graphs*, in Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, SIAM, 2019, pp. 2071–2089.

[28] K. S. Booth and G. S. Lueker, *Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms*, Journal of Computer and System Sciences, 13 (1976), pp. 335–379.

[29] O. V. Borodin, A. O. Ivanova, A. V. Kostochka, and N. N. Sheikh, *Planar graphs decomposable into a forest and a matching*, Discrete Mathematics, 309 (2009), pp. 277–279.

[30] O. V. Borodin, A. V. Kostochka, N. N. Sheikh, and G. Yu, *Decomposing a planar graph with girth 9 into a forest and a matching*, European Journal of Combinatorics, 29 (2008), pp. 1235–1241.

[31] G. Borradaile and P. N. Klein, *An $O(n \log n)$ Algorithm for Maximum* st*-Flow in a Directed Planar Graph*, Journal of the ACM, 56 (2009), pp. 9:1–9:30.

[32] S. Cabello, *Many Distances in Planar Graphs*, Algorithmica, 62 (2012), pp. 361–381.

[33] K. Cameron, C. T. Hoàng, and B. Lévêque, *Asteroids in rooted and directed path graphs*, Electronic Notes in Discrete Mathematics, 32 (2009), pp. 67–74.

[34] ——, *Characterizing Directed Path Graphs by Forbidden Asteroids*, Journal of Graph Theory, 68 (2011), pp. 103–112.

[35] P. A. Catlin, J. W. Grossman, A. M. Hobbs, and H. Lai, *Fractional arboricity, strength, and principal partitions in graphs and matroids*, Discrete Applied Mathematics, 40 (1992), pp. 285–302.

[36] S. Chaplick, *PQR-Trees and Undirected Path Graphs*, University of Toronto, 2008.

[37] S. Chaplick, M. Gutierrez, B. Lévêque, and S. B. Tondato, *From Path Graphs to Directed Path Graphs*, in Graph Theoretic Concepts in Computer Science - 36th International Workshop, WG 2010, vol. 6410 of Lecture Notes in Computer Science, 2010, pp. 256–265.

[38] D. Z. Chen and J. Xu, *Shortest Path Queries in Planar Graphs*, in Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, ACM, 2000, pp. 469–478.

[39] O. Cheong, S. Har-Peled, H. Kim, and H. Kim, *On the Number of Edges of Fan-Crossing Free Graphs*, Algorithmica, 73 (2015), pp. 673–695.

[40] D. G. Corneil, S. Olariu, and L. Stewart, *The Ultimate Interval Graph Recognition Algorithm?*, in Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM/SIAM, 1998, pp. 175–180.

[41] E. Dahlhaus and G. Bailey, *Recognition of Path Graphs in Linear Time*, in 5th Italian Conference on Theoretical Computer Science, World Scientific, 1996, pp. 201–210.

[42] D. Das, E. Kipouridis, M. P. Gutenberg, and C. Wulff-Nilsen, *A Simple Algorithm for Multiple-Source Shortest Paths in Planar Digraphs*, in 5th Symposium on Simplicity in Algorithms, SOSA, Virtual Conference, 2022, SIAM, 2022, pp. 1–11.

[43] É. C. de Verdière and A. Schrijver, *Shortest Vertex-Disjoint Two-Face Paths in Planar Graphs*, ACM Transactions on Algorithms, 7 (2011), pp. 19:1–19:12.

[44] W. Didimo, P. Eades, and G. Liotta, *Drawing graphs with right angle crossings*, Theoretical Computer Science, 412 (2011), pp. 5156–5166.

[45] P. F. Dietz, *Intersection Graph Algorithms*, tech. rep., Cornell University, 1984.

[46] H. N. Djidjev, *Efficient Algorithms for Shortest Path Queries in Planar Digraphs*, in International Workshop on Graph-Theoretic Concepts in Computer Science, Springer, 1996, pp. 151–165.

[47] T. Eilam-Tzoreff, *The Disjoint Shortest Paths Problem*, Discrete Applied Mathematics, 85 (1998), pp. 113–138.

[48] D. Eisenstat and P. N. Klein, *Linear-Time Algorithms for Max Flow and Multiple-Source Shortest Paths in Unit-Weight Planar Graphs*, in Symposium on Theory of Computing Conference, STOC'13, ACM, 2013, pp. 735–744.

[49] J. Erickson and A. Nayyeri, *Shortest Non-Crossing Walks in the Plane*, in Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, SIAM, 2011, pp. 297–208.

[50] S. D. Eriksson-Bique, J. Hershberger, V. Polishchuk, B. Speckmann, S. Suri, T. Talvitie, K. Verbeek, and H. Yildiz, *Geometric k Shortest Paths*, in Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, SIAM, 2015, pp. 1616–1625.

[51] J. Fakcharoenphol and S. Rao, *Planar graphs, negative weight edges, shortest paths, and near linear time*, Journal of Computer and System Sciences, 72 (2006), pp. 868–889.

[52] L. R. Ford and D. R. Fulkerson, *Maximal Flow Through a Network*, Canadian journal of Mathematics, 8 (1956), pp. 399–404.

[53] G. N. Frederickson, *Fast Algorithms for Shortest Paths in Planar Graphs, with Applications*, SIAM Journal on Computing, 16 (1987), pp. 1004–1022.

[54] H. N. Gabow and R. E. Tarjan, *A Linear-Time Algorithm for a Special Case of Disjoint Set Union*, Journal of Computer and System Sciences, 30 (1985), pp. 209–221.

[55] F. Gavril, *The Intersection Graphs of Subtrees in Trees are Exactly the Chordal Graphs*, Journal of Combinatorial Theory, Series B, 16 (1974), pp. 47–56.

[56] ——, *A Recognition Algorithm for the Intersection Graphs of Directed Paths in Directed Trees*, Discrete Mathematics, 13 (1975), pp. 237–249.

[57] ——, *A Recognition Algorithm for the Intersection Graphs of Paths in Trees*, Discrete Mathematics, 23 (1978), pp. 211–227.

[58] P. Gawrychowski, S. Mozes, O. Weimann, and C. Wulff-Nilsen, *Better Tradeoffs for Exact Distance Oracles in Planar Graphs*, in Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2018, pp. 515–529.

[59] A. V. Goldberg, *Point-to-Point Shortest Path Algorithms with Preprocessing*, in SOFSEM 2007: 33rd Conference on Current Trends in Theory and Practice of Computer Science, vol. 4362 of Lecture Notes in Computer Science, Springer, 2007, pp. 88–102.

[60] D. Gonçalves, *Etudes de différents problèmes de partition de graphes*, PhD thesis, Bordeaux 1, 2006.

[61] D. Gonçalves, *Caterpillar arboricity of planar graphs*, Discrete Mathematics, 307 (2007), pp. 2112–2121.

[62] ——, *Covering planar graphs with forests, one having bounded maximum degree*, Journal of Combinatorial Theory, Series B, 99 (2009), pp. 314–322.

[63] D. Gonçalves and P. Ochem, *On star and caterpillar arboricity*, Discrete Mathematics, 309 (2009), pp. 3694–3702.

[64] J. L. Gross and T. W. Tucker, *Topological Graph Theory*, Courier Corporation, 2001.

[65] M. Gutierrez, B. Lévêque, and S. B. Tondato, *Asteroidal quadruples in non rooted path graphs*, Discussiones Mathematicae Graph Theory, 35 (2015), pp. 603–614.

[66] M. Gutierrez and S. B. Tondato, *On models of directed path graphs non rooted directed path graphs*, Graphs and Combinatorics, 32 (2016), pp. 663–684.

[67] A. Gyárfás and D. West, *Multitrack Interval Graphs*, (1995).

[68] M. Habib, R. M. McConnell, C. Paul, and L. Viennot, *Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing*, Theoretical Computer Science, 234 (2000), pp. 59–84.

[69] S. L. Hakimi, *On the degrees of the vertices of a directed graph*, Journal of the Franklin Institute, 279 (1965), pp. 290–308.

[70] F. Harary, *Covering and packing in graphs, I.*, Annals of the New York Academy of Sciences, 175 (1970), pp. 198–205.

[71] F. Harary and A. J. Schwenk, *Evolution of the path number of a graph: Covering and packing in graphs, II*, in Graph theory and computing, Elsevier, 1972, pp. 39–45.

[72] R. Hassin, *Maximum Flow in (s,t) Planar Networks*, Information Processing Letters, 13 (1981), p. 107.

[73] R. Hassin and D. B. Johnson, *An $O(n\log^2 n)$ Algorithm for Maximum Flow in Undirected Planar Networks*, SIAM Journal on Computing, 14 (1985), pp. 612–624.

[74] W. He, X. Hou, K. Lih, J. Shao, W. Wang, and X. Zhu, *Edge-Partitions of Planar Graphs and Their Game Coloring Numbers*, Journal of Graph Theory, 41 (2002), pp. 307–317.

[75] M. R. Henzinger, P. N. Klein, S. Rao, and S. Subramanian, *Faster Shortest-Path Algorithms for Planar Graphs*, Journal of Computer and System Sciences, 55 (1997), pp. 3–23.

[76] S. Hong and T. Tokuyama, *Beyond Planar Graphs, Communications of NII Shonan Meetings*, Springer, 2020.

[77] W. Hsu, *A Simple Test for Interval Graphs*, in Graph-Theoretic Concepts in Computer Science, 18th International Workshop, Proceedings, vol. 657 of Lecture Notes in Computer Science, Springer, 1992, pp. 11–16.

[78] W. Hsu and R. M. McConnell, *PC trees and circular-ones arrangements*, Theoretical Computer Science, 296 (2003), pp. 99–116.

[79] A. Itai and Y. Shiloach, *Maximum Flow in Planar Networks*, SIAM Journal on Computing, 8 (1979), pp. 135–150.

[80] G. F. Italiano, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen, *Improved Algorithms for Min Cut and Max Flow in Undirected Planar Graphs*, in Proceedings of the 43rd ACM Symposium on Theory of Computing, ACM, 2011, pp. 313–322.

[81] N. Jing, Y. Huang, and E. A. Rundensteiner, *Hierarchical Optimization of Optimal Path Finding for Transportation Applications*, in CIKM '96, Proceedings of the Fifth International Conference on Information and Knowledge Management, ACM, 1996, pp. 261–268.

[82] D. B. Johnson and S. M. Venkatesan, *Partition of Planar Flow Networks*, in 24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983, IEEE Computer Society, 1983, pp. 259–263.

[83] C. Jordan, *Cours d'analyse de l'École polytechnique*, vol. 1, Gauthier-Villars et fils, 1893.

[84] M. Kaufmann and T. Ueckerdt, *The Density of Fan-Planar Graphs*, Electronic Journal of Combinatorics, 29 (2022).

[85] D. Kim and N. F. Maxemchuk, *Simple Robotic Routing in Ad Hoc Networks*, in 13th IEEE International Conference on Network Protocols (ICNP 2005), IEEE Computer Society, 2005, pp. 159–168.

[86] V. King, S. Rao, and R. E. Tarjan, *A Faster Deterministic Maximum Flow Algorithm*, Journal of Algorithms, 17 (1994), pp. 447–474.

[87] P. N. Klein, *Multiple-source shortest paths in planar graphs*, in Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2005, pp. 146–155.

[88] K. B. Knauer and T. Ueckerdt, *Three ways to cover a graph*, Discrete Mathematics, 339 (2016), pp. 745–758.

[89] Y. Kobayashi and C. Sommer, *On shortest disjoint paths in planar graphs*, Discrete Optimization, 7 (2010), pp. 234–245.

[90] N. Korte and R. H. Möhring, *An Incremental Linear-Time Algorithm for Recognizing Interval Graphs*, SIAM Journal on Computing, 18 (1989), pp. 68–81.

[91] D. Koschützki, K. A. Lehmann, L. Peeters, S. Richter, D. Tenfelde-Podehl, and O. Zlotowski, *Centrality Indices*, in Network Analysis: Methodological Foundations, vol. 3418 of Lecture Notes in Computer Science, Springer, 2004, pp. 16–61.

[92] L. Kowalik and M. Kurowski, *Short Path Queries in Planar Graphs in Constant Time*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, ACM, 2003, pp. 143–148.

[93] Y. Kusakari, D. Masubuchi, and T. Nishizeki, *Finding a Noncrossing Steiner Forest in Plane Graphs Under a 2-Face Condition*, Journal of Combinatorial Optimization, 5 (2001), pp. 249–266.

[94] J. Łącki and P. Sankowski, *Min-Cuts and Shortest Cycles in Planar Graphs in $O(n \log \log n)$ Time*, in Algorithms - ESA 2011 - 19th Annual European Symposium, Proceedings, vol. 6942 of Lecture Notes in Computer Science, Springer, 2011, pp. 155–166.

[95] F. T. Leighton, *Complexity issues in VLSI: optimal layouts for the shuffle-exchange graph and other networks*, MIT press, 1983.

[96] ——, *New Lower Bound Techniques for VLSI*, Mathematical systems theory, 17 (1984), pp. 47–70.

[97] C. Lekkekerker and J. Boland, *Representation of a finite graph by a set of intervals on the real line*, Fundamenta Mathematicae, 51 (1962), pp. 45–64.

[98] B. Lévêque, F. Maffray, and M. Preissmann, *Characterizing Path Graphs by Forbidden Induced Subgraphs*, Journal of Graph Theory, 62 (2009), pp. 369–384.

[99] W. Lochet, *A Polynomial Time Algorithm for the k-Disjoint Shortest Paths Problem*, in Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, SIAM, 2021, pp. 169–178.

[100] A. P. Masucci, K. Stanilov, and M. Batty, *Exploring the evolution of London's street network in the information space: A dual approach*, Physical Review E, 89 (2014), p. 012805.

[101] R. M. McConnell and J. P. Spinrad, *Modular decomposition and transitive orientation*, Discrete Mathematics, 201 (1999), pp. 189–241.

[102] C. L. Monma and V. K. Wei, *Intersection Graphs of Paths in a Tree*, Journal of Combinatorial Theory, Series B, 41 (1986), pp. 141–181.

[103] M. Montassier, P. O. de Mendez, A. Raspaud, and X. Zhu, *Decomposing a graph into forests*, Journal of Combinatorial Theory, Series B, 102 (2012), pp. 38–52.

[104] L. Mouatadid and R. Robere, *Path Graphs, Clique Trees, and Flowers*, CoRR, abs/1505.07702 (2015).

[105] S. Mozes and C. Sommer, *Exact Distance Oracles for Planar Graphs*, in Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2012, pp. 209–222.

[106] P. Mutzel, T. Odenthal, and M. Scharbrodt, *The Thickness of Graphs: A Survey*, Graphs and Combinatorics, 14 (1998), pp. 59–73.

[107] C. Nash-Williams, *Decomposition of finite graphs into forests*, Journal of the London Mathematical Society, 1 (1964), pp. 12–12.

[108] M. B. Novick, *Parallel Algorithms for Intersection Graphs*, tech. rep., Cornell University, 1990.

[109] Y. Nussbaum, *Improved Distance Queries in Planar Graphs*, in Algorithms and Data Structures - 12th International Symposium, WADS, vol. 6844 of Lecture Notes in Computer Science, Springer, 2011, pp. 642–653.

[110] J. Orlin, *Contentment in Graph Theory: Covering Graphs with Cliques*, in Indagationes Mathematicae (Proceedings), vol. 80, Elsevier, 1977, pp. 406–424.

[111] J. B. Orlin, *Max Flows in $O(nm)$ Time, or Better*, in Symposium on Theory of Computing Conference, STOC'13, ACM, 2013, pp. 765–774.

[112] J. Pach and G. Tóth, *Graphs Drawn with Few Crossings per Edge*, Combinatorica, 17 (1997), pp. 427–439.

[113] B. S. Panda, *The forbidden subgraph characterization of directed vertex graphs*, Discrete Mathematics, 196 (1999), pp. 239–256.

[114] E. Papadopoulou, *k-Pairs Non-Crossing Shortest Paths in a Simple Polygon*, International Journal of Computational Geometry and Applications, 9 (1999), pp. 533–552.

[115] C. Payan, *Graphes équilibrés et Arboricité Rationnelle*, European Journal of Combinatorics, 7 (1986), pp. 263–270.

[116] C. A. Phillips, *The Network Inhibition Problem*, in Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, ACM, 1993, pp. 776–785.

[117] J. Picard and M. Queyranne, *A Network Flow Solution to some Nonlinear 0-1 Programming Problems, with Applications to Graph Theory*, Networks, 12 (1982), pp. 141–159.

[118] T. Pisanski, P. Potocnik, J. Chen, J. L. Gross, T. W. Tucker, A. Vince, D. Archdeacon, S. Negami, and A. T. White, *Topological Graph Theory*, in Handbook of Graph Theory, Discrete Mathematics and Its Applications, Chapman & Hall / Taylor & Francis, 2003, pp. 610–786.

[119] V. Polishchuk and J. S. B. Mitchell, *Thick Non-Crossing Paths and Minimum-Cost Flows in Polygonal Domains*, in Proceedings of the 23rd ACM Symposium on Computational Geometry, ACM, 2007, pp. 56–65.

[120] B. Raney and K. Nagel, *Iterative route planning for large-scale modular transportation simulations*, Future Generation Computer Systems, 20 (2004), pp. 1101–1118.

[121] H. D. Ratliff, G. T. Sicilia, and S. Lubore, *Finding the* n *Most Vital Links in Flow Networks*, Management Science, 21 (1975), pp. 531–539.

[122] D. Rautenbach and L. Volkmann, *Some structural results on linear arboricity*, Australasian Journal of Combinatorics, 17 (1998), pp. 267–274.

[123] J. H. Reif, *Minimum s-t Cut of a Planar Undirected Network in* $O(n \log^2(n))$ *time*, SIAM Journal on Computing, 12 (1983), pp. 71–81.

[124] P. Renz, *Intersection Representations of Graphs by Arcs*, Pacific Journal of Mathematics, 34 (1970), pp. 501–510.

[125] D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic Aspects of Vertex Elimination on Graphs*, SIAM Journal on computing, 5 (1976), pp. 266–283.

[126] M. Schaefer, *The Graph Crossing Number and its Variants: A Survey*, Electronic Journal of Combinatorics, (2012), pp. DS21–Apr.

[127] A. Schäffer, *A Faster Algorithm to Recognize Undirected Path Graphs*, Discrete Applied Mathematics, 43 (1993), pp. 261–295.

[128] A. Schrijver, *Finding k Disjoint Paths in a Directed Planar Graph*, SIAM Journal on Computing, 23 (1994), pp. 780–788.

[129] S. Schwartz, *An overview of graph covering and partitioning*, Discrete Mathematics, 345 (2022), p. 112884.

[130] A. J. Steiger, *Single-Face Non-Crossing Shortest Paths in Planar Graphs*, M.S. thesis, University of Illinois at Urbana-Champaign, 2017. [Online], Available: http://hdl.handle.net/2142/98345, (2017).

[131] J. Takahashi, H. Suzuki, and T. Nishizeki, *Shortest Noncrossing Paths in Plane Graphs*, Algorithmica, 16 (1996), pp. 339–357.

[132] R. E. Tarjan, *Decomposition by Clique Separators*, Discrete Mathematics, 55 (1985), pp. 221–232.

[133] R. E. Tarjan and M. Yannakakis, *Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs*, SIAM Journal on computing, 13 (1984), pp. 566–579.

[134] W. T. Tutte, *On the Problem of Decomposing a Graph into n Connected Factors*, Journal of the London Mathematical Society, 1 (1961), pp. 221–230.

[135] P. van Emde Boas, *Preserving Order in a Forest in Less Than Logarithmic Time and Linear Space*, Information Processing Letters, 6 (1977), pp. 80–82.

[136] D. Wagner and K. Weihe, *A Linear-Time Algorithm for Edge-Disjoint Paths in Planar Graphs*, Combinatorica, 15 (1995), pp. 135–150.

[137] Y. Wang and Q. Zhang, *Decomposing a planar graph with girth at least 8 into a forest and a matching*, Discrete Mathematics, 311 (2011), pp. 844–849.

[138] R. D. Wollmer, *Some Methods for Determining the Most Vital Link in a Railway Network*, Rand Corporation, 1963.

[139] R. K. Wood, *Deterministic Network Interdiction*, Mathematical and Computer Modelling, 17 (1993), pp. 1–18.

[140] J. Wu, *On the Linear Arboricity of Planar Graphs*, Journal of Graph Theory, 31 (1999), pp. 129–134.

[141] A. Ziliaskopoulos, D. Kotzinos, and H. S. Mahmassani, *Design and implementation of parallel time-dependent least time path algorithms for intelligent transportation systems applications*, Transportation Research Part C: Emerging Technologies, 5 (1997), pp. 95–107.