Theoretical Computer Science Department
Faculty of Mathematics and Computer Science
Jagiellonian University

New combinatorial structures for several algorithmic problems

Grzegorz Guśpiel

PhD dissertation

Supervisor:
prof. dr hab. Paweł Idziak

Co-supervisor:
dr Grzegorz Gutowski

Kraków, May 2019

Acknowledgments

First, I would like to express my gratitude to Professor Paweł Idziak for putting both his heart and countless hours into our meetings, during which he helped me improve my writing skills. I am also grateful for his feedback regarding presenting at seminars. It would have been easier to just let some of my mistakes pass, but he wanted me to be at my best and I greatly appreciate his efforts.

I am deeply indebted to Grzegorz Gutowski and Tomasz Krawczyk, who took me to a research workshop while I was a master's student and invited me to work with them. This was a very warm welcome to my first attempts at research and I very much enjoyed solving problems together. I also thank Grzegorz for his invaluable assistance with writing this thesis.

I thank Piotr Micek for being a great partner for discussions about combinatorics and for table tennis matches. I am very thankful for the opportunity to learn from Jarosław Grytczuk and Jakub Kozik, too. I would like to express my appreciation to Lech Duraj for generously offering me his advice on teaching, to Adam Polak for being my primary point of reference that inspired me to work harder, and to all colleagues from the Theoretical Computer Science Department for the exceptional work atmosphere.

I am also grateful to Bartłomiej Bosek for introducing me to the permutation inversion problem and reading my first drafts, to Grzegorz Gutowski and Paweł Rzążewski for valuable suggestions regarding my **NP**-hardness proof for CROSSING-MINIMIZING MATCHING, and to Vladan Majerech for pointing out a mistake in the first version of my permutation inversion algorithm.

I thank the Polish Ministry of Science and Higher Education for supporting parts of the research in this thesis, under the project number DI2013 000443.

Most importantly, I thank my parents and family for raising me with love and for their enormous care about my education and development.

Contents

Introduction

Combinatorial structures play a fundamental role to design, understand and analyze algorithms and their complexity. They are very often used as:

- data structures, to reduce the running time of an algorithm,
- tools to prove lower bounds for the computational complexity of considered problems,
- universal structures encoding possible local behavior for graphs with prescribed properties,

and in many, many other situations. Our choice above is obviously not representative of all the applications of combinatorial structures in computer science, but the results presented in this thesis fall into these three categories.

In Chapter 1, we introduce a new data structure for the problem of inverting permutations in-place. The best known deterministic algorithm for such in-place inversion is relatively simple and runs in $O(n^2)$ time. The application of our structure allows to reduce the running time to $O\big(n^{3/2}\big)$.

The main result of Chapter 2 is an algorithm for extending visibility representations of planar graphs. The time complexity of the initial version of our algorithm is $O(n^2)$ due to the need to solve 2-SAT instances of that size. To improve the running time, we develop a data structure based on a set of persistent AVL trees. The combination of our data structure and dominance drawings of planar graphs allows us to significantly reduce the size of the 2-SAT formulas and consequently speed up the algorithm to $O\big(n \log^2 n\big)$.

Chapter 3 retains a bit of the graph drawing flavor of Chapter 2, as we discuss finding a perfect matching in a bipartite graph (already drawn in the plane) that minimizes the number of crossing edges. We use some new combinatorial structures to prove the **NP**-hardness of this problem.

Finally, Chapter 4 is devoted to the problem of finding smallest possible universal targets for homomorphisms of edge-colored graphs. These universal targets, for a given class of graphs, have combinatorial properties that allow to elegantly capture every local behavior possible in all the graphs from the class. Our line of research was motivated by the work of Alon and Marshall [3], and started with the paper [47]. In Chapter 4 we develop new combinatorial tools that allow us to construct asymptotically optimal universal targets for rational graph classes.

Preliminaries

All results of this dissertation are set in the context of graph theory. For definitions of standard graph-theoretical concepts, we refer the reader to [30]. We also use certain specific notions and notational conventions listed below.

Set notation. For every positive integer k, the set $\{1, \ldots, k\}$ is denoted by $[k]$.

Graphs. We consider *undirected* and *directed graphs*. In both cases, we use the notation $G = (V(G), E(G))$, where $V(G)$ is the finite nonempty *vertex set*, and $E(G)$ the *edge set* of a graph G. If we do not specify whether a graph is directed or not, we mean an undirected graph. Directed graphs are sometimes called *digraphs*. Both undirected and directed graphs are assumed to be *simple*, i.e. to have no loops and no multiple edges. Sporadically we make an exception to this rule, and then we explicitly let the reader know that this is the case.

Notation for edges. An undirected edge between vertices u and v is written as $\{u, v\}$, whereas a directed edge from u to v is written as (u, v).

Degree of a vertex. For a vertex v of an undirected graph G, the number $\deg_G(v)$ is its *degree*. For a vertex v of a directed graph G, we use $\deg^{\text{in}}_G(v)$ and $\deg^{\text{out}}_G(v)$ instead, to denote the *indegree* and *outdegree* of v, respectively. The subscript G may be dropped if the graph is clear from the context.

Subgraphs. For two graphs G' and G, directed or not, we say that G' is a *subgraph* of G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. If additionally $E(G') = E(G)$ when restricted to vertices in $V(G')$, we say that G' is an *induced subgraph* of G.

Distance between vertices. The *distance* from vertex u to vertex v is the number of edges in the shortest path from u to v. The distance from a vertex u to a subgraph G' of a graph G is the smallest distance from u to a vertex of G'.

Graph orientations. An *oriented graph* $\vec{G}$ is a directed graph such that for any pair of vertices $u, v \in V(\vec{G})$ either $(u, v) \notin E(\vec{G})$ or $(v, u) \notin E(\vec{G})$. An *orientation* $\vec{G}$ of a graph G is an oriented graph obtained by assigning a direction to each edge of G. Such an orientation is called a *d-orientation* if every vertex of $\vec{G}$ has indegree at most d.

Graph classes. By a *class of graphs* we mean a nonempty set of graphs closed under isomorphisms.

Pseudocode conventions. In algorithm listings, we model our notation after [22]. In particular, we use "$=$" for assignment and "==" for equality testing.

1

An in-place, subquadratic algorithm for permutation inversion

In the permutation inversion problem, the algorithm is given a positive integer n and a permutation π of the set $V = \{1, ..., n\}$ presented in an array t, where $t[i] = \pi(i)$ for all $i \in V$. The goal is to perform a sequence of modifications of t, so that eventually $t[i] = \pi^{-1}(i)$ for all $i \in V$. In this chapter we focus on algorithms that use $O(\log n)$ bits of additional memory. Algorithms with this tiny bound on additional memory are called in-place algorithms. The best known in-place deterministic algorithm for the permutation inversion problem runs in $O(n^2)$. By applying a new data structure, we reduce the running time to $O\big(n^{3/2}\big)$.

1.1 Previous work

The in-place permutation inversion problem was considered in Knuth [57], where two solutions are described, by Huang and by Boothroyd. These algorithms, however, are allowed to store any value from the range $[-n, ..., n]$ in the array t. This seems to bypass the heart of the problem. In fact, the signs of the values in t can be used as a vector of n bits. The problem is trivial when such a vector is allowed. Algorithms described in this chapter are allowed to store values only from the range $[1, ..., n]$ in t (which may lead to t temporarily not representing a permutation).

First, we describe an $O(n^2)$ time in-place algorithm. Observe that it is straightforward to reverse a single cycle of π:

Listing 1.1 Pseudocode for reversing a cycle

```
REVERSE-CYCLE(start)
1  cur = t[start]
2  prev = start
3  while cur ≠ start
4        next = t[cur]
5        t[cur] = prev
6        prev = cur
7        cur = next
8  t[start] = prev
```

For a cycle in the permutation, let its *leader* be the smallest element in this cycle. The following code finds in $O(n)$ time the leader of the cycle (containing a given element $start$):

Listing 1.2 Pseudocode for finding the leader of a cycle

```
CYCLE-LEADER(start)
1  cur = t[start]
2  smallest = start
3  while cur ≠ start
4        smallest = min(smallest, cur)
5        cur = t[cur]
6  return smallest
```

To obtain the inverse of π, it suffices to reverse each of the cycles exactly once:

Listing 1.3 A quadratic in-place algorithm for permutation inversion

```
1  for i = 1 to n
2        if CYCLE-LEADER(i) == i
3              REVERSE-CYCLE(i)
```

In 1995, Fich, Munro and Poblete [35] published a paper on a similar topic: the permutation π is given by means of an oracle and the goal is to permute the contents of an array according to π. They provided an algorithm with running time $O\big(n \log^2 n\big)$ that uses $O\big(\log^2 n\big)$ bits of additional memory. The concept of a cycle leader comes from this paper.

In 2015, the methods of [35] were extended to the permutation inversion problem by El-Zein, Munro and Robertson [33, 70], who gave an algorithm with running time $O(n \log n)$ that uses $O\big(\log^2 n\big)$ bits of additional memory.

It is interesting to note that the quadratic algorithm can be easily modified to achieve $O(n \log n)$ expected running time. In [35], the authors point out, attributing the idea to a personal communication with Impagliazzo, that one can use a randomly chosen hash function h and choose the cycle leader to be the element i with the smallest value $h(i)$. This idea can be applied to cycle inversion as well. When visiting element i in the main loop, we start reversing its cycle and either complete this operation or encounter an element j such that $h(j) < h(i)$, in which case we revert the operation. This way, assuming no hash collisions, every cycle is reversed once and if h is chosen randomly, the average time spent in the main loop at any single element is $O(\log n)$. Thus, we obtain an in-place algorithm with expected running time $O(n \log n)$.

To our knowledge, we present the first subquadratic deterministic in-place algorithm. We compare it with previously known algorithms in Table 1.1. The description of our algorithm is contained in [46] as well as in this chapter. The running time of this algorithm is $O\big(n^{3/2}\big)$. Such time reduction is possible due to an alternative representation of permutation cycles.

time complexity	required bits of additional memory	source
$O(n)$	$O(\log n)$ with $t[i] \in [-n, ..., n]$	Huang [57]
$O(n)$	$O(\log n)$ with $t[i] \in [-n, ..., n]$	Boothroyd [57]
$O(n^2)$	$O(\log n)$	folklore (Listing 1.3)
$O(n \log n)$	$O(\log^2 n)$	El-Zein, Munro, Robertson [33, 70]
$O(n \log n)$ in expectation	$O(\log n)$	Impagliazzo [35]
$O(n^{3/2})$	$O(\log n)$	this thesis (Listing 1.9)

Table 1.1: A summary of permutation inversion algorithms.

1.2 A summary of the $O(n^{3/2})$ algorithm

Observe that the quadratic algorithm runs in time $O(n^{3/2})$ on any instance in which each cycle is of size at most $O(\sqrt{n})$. The complexity is worse when a large proportion of the elements belong to cycles of greater sizes. We modify the quadratic algorithm to handle large cycles differently, while keeping its behavior for small cycles.

During the course of the algorithm, with the current state of array t we associate a directed graph G_t on the vertex set V and with edge $(i, t[i])$ for each $i \in V$. The graph G_t may contain loops and when we consider cycles, loops are counted among them. Initially, G_t is a disjoint union of cycles. When we modify t, we can get G_t to be an arbitrary graph with outdegree of each vertex equal to 1.

We will introduce an alternative way of storing a long cycle in the array t. This alternative representation of a cycle is achieved by redirecting a small number of edges. The result is a graph with multiple connected components; each component has $O(\sqrt{n})$ elements and is a directed path leading to a cycle. We will use the lengths of these paths and cycles to encode some information. This information will allow us to revert the edge redirections and obtain the original cycle.

When we first encounter a long cycle, we reverse it and convert the reversed cycle to the alternative representation. Next, whenever in the main loop we visit another vertex of the cycle, we traverse only the component of the vertex, which takes time $O(\sqrt{n})$. Previously this step could take $O(n)$ and this is the improvement that allows us to achieve $O(n^{3/2})$ time complexity. When the loop finishes, all short cycles are reversed and all the long cycles are stored using the alternative representation with minor modifications. Next, we perform one pass over all vertices to remove the modifications and another pass to convert long cycles back from the alternative representation.

1.3 Segments and cycle detection

We call each connected component of the alternative representation to be a *segment*. A segment is a disjoint union of two directed graphs: a cycle and a path, together with an edge from the last vertex of the path to one of the vertices in the cycle. The first vertex of the path is called the *beginning* of the segment, the number of vertices in the segment is called its *size*, and the path is called its *tail*.

We use the sizes of the segments, as well as the sizes of their cycles, to store some information. Thus, whenever in the main loop variable i becomes a vertex of a segment, we need to be able to compute in-place the size of the cycle and the distance from i to the cycle. The problem is called cycle detection and at least two different algorithms are known to solve it in-place and in time proportional to the size of the segment: the Floyd's cycle-finding algorithm (also called the tortoise and hare algorithm) mentioned in Knuth [58, Section 3.1, Exercise 6], and the Brent's [14] algorithm. In Listing 1.4, we shortly present the first one. The idea is as follows. First, we initialize two variables, the *tortoise* and the *hare*, to point at element i (called $start$ in the following pseudocode). Next, we simultaneously progress both variables: the tortoise moves one step at a time and the hare moves two steps at a time, where by a step we mean setting $v = t[v]$. We stop when both variables point at the same element (which must happen after a number of steps that is linear in the number of vertices reachable from i). Next, we bring the hare back to element i and start progressing the pointers again, this time both by one step at a time. They first meet at the beginning of the cycle, i.e. the only vertex with indegree 2 in the segment, which gives us the distance from $start$ to the cycle. Finally, we use the tortoise one last time to compute the size of the cycle.

Listing 1.4 The Floyd's cycle-finding algorithm

```
TORTOISE-AND-HARE(start)
 1  tortoise = hare = start
 2  cycle_length = dist_to_cycle = 0
 3  repeat
 4      tortoise = t[tortoise]
 5      hare = t[t[hare]]
 6  until tortoise == hare
 7  hare = start
 8  repeat
 9      tortoise = t[tortoise]
10      hare = t[hare]
11      dist_to_cycle = dist_to_cycle + 1
12  until tortoise == hare
13  repeat
14      tortoise = t[tortoise]
15      cycle_length = cycle_length + 1
16  until tortoise == hare
17  return (cycle_length, dist_to_cycle)
```

In our setting, it will be important to search for cycles that are in a bounded distance from $start$ and have a bounded size. In such a case, we would like TORTOISE-AND--HARE to perform a number of steps that is linear in the sum of these two bounds.

Observe that if there is a cycle with distance at most d from $start$ and size at most s, the loop in Listing 1.4 in lines 3 – 6 finishes after at most $d + s$ steps. Therefore, if we modify TORTOISE-AND-HARE to take a bound max and return (NIL, NIL) if this loop performs more than max steps, we obtain an algorithm that correctly finds the distance from $start$ to the cycle and the size of the cycle if their sum is at most max, and

otherwise either successfully finds these values or returns (NIL, NIL). This algorithm has the important property of running in $O(max)$. We present it in Listing 1.5.

Listing 1.5 The Floyd's cycle-finding algorithm performing a bounded number of steps

```
LIMITED-TORTOISE-AND-HARE(start, max)
 1  tortoise = hare = start
 2  cycle_length = dist_to_cycle = 0
 3  repeat
 4      if max == 0
 5          return (NIL, NIL)
 6      tortoise = t[tortoise]
 7      hare = t[t[hare]]
 8      max = max − 1
 9  until tortoise == hare
10  hare = start
11  repeat
12      tortoise = t[tortoise]
13      hare = t[hare]
14      dist_to_cycle = dist_to_cycle + 1
15  until tortoise == hare
16  repeat
17      tortoise = t[tortoise]
18      cycle_length = cycle_length + 1
19  until tortoise == hare
20  return (cycle_length, dist_to_cycle)
```

1.4 The alternative representation of long cycles

In this section, we define the alternative representation of a long cycle and show how to compute it. Let $k = \lceil \sqrt{n} \rceil$. Up to isomorphism, there are $k^2 \geqslant n$ different segments of size between $k+1$ and $2k$ and of cycle length between 1 and k. We choose different segments of the above form to represent different elements of V. This way, a segment can 'store' a pointer to a vertex – we say that such a segment, or a pair of a segment size and a cycle length of the above form, *encodes* a vertex. The encoding is the following bijection from V to a subset of $\{k+1, ..., 2k\} \times \{1, ..., k\}$: $\textsc{Encode}(v) = (\lfloor (v-1)/k \rfloor + (k+1), ((v-1) \bmod k)+1)$. Its inverse is the function $\textsc{Decode}(s, c) = (s-(k+1))k+c+1$.

An intuitive understanding of the alternative representation is that the cycle is split into segments such that all except $O(\sqrt{n})$ edges are preserved (condition (C1)), the first segment begins at the leader, and every segment encodes the beginning of the next segment, except the first and the last one, which we need to handle differently. This is illustrated in Figure 1.1.

We set a threshold for a cycle to be handled using the alternative representation as follows: a cycle is called *long*, if it has at least $4k+3$ vertices, otherwise it is called *short*.

For a directed graph G and a subset X of its vertex set, let $G[X]$ denote the subgraph of G induced by X. For $X = \{x_1, ..., x_q\}$, we simply write $G[x_1, ..., x_q]$. Let c_1 be the leader of a long cycle $C = (c_1, ..., c_p)$ on vertex set $V' \subseteq V$, let R be a directed graph on V' with the outdegree of each vertex equal to 1 and let S be an integer from $[k+1, 2k]$. We say that a pair (R, S) is a *segment representation* of C if there exist $q \geqslant 2$ integers $i_1, ..., i_q$ such that $1 = i_1 < ... < i_q < p$ and:

(C1) for every $i \in \{1, ..., p-1\} \setminus \{i_2 - 1, ..., i_q - 1\}$, the graph R contains the edge (c_i, c_{i+1});

(C2) the graph $R[c_1, ..., c_{i_2-1}]$ is a segment with beginning c_1, size between $2k+2$ and $4k+1$ and cycle length y such that $(S, y) = \text{ENCODE}(c_{i_2})$;

(C3) for every $j = 2, ..., q-1$, the graph $R[c_{i_j}, c_{i_j+1}, ..., c_{i_{j+1}-1}]$ is a segment with beginning c_{i_j}, size x and cycle length y such that $(x, y) = \text{ENCODE}(c_{i_{j+1}})$;

(C4) the graph $R[c_{i_q}, ..., c_p]$ is a segment with beginning c_{i_q}, size $2k+1$ and cycle length at most k.

The graph R is called a *segmentation* of C.

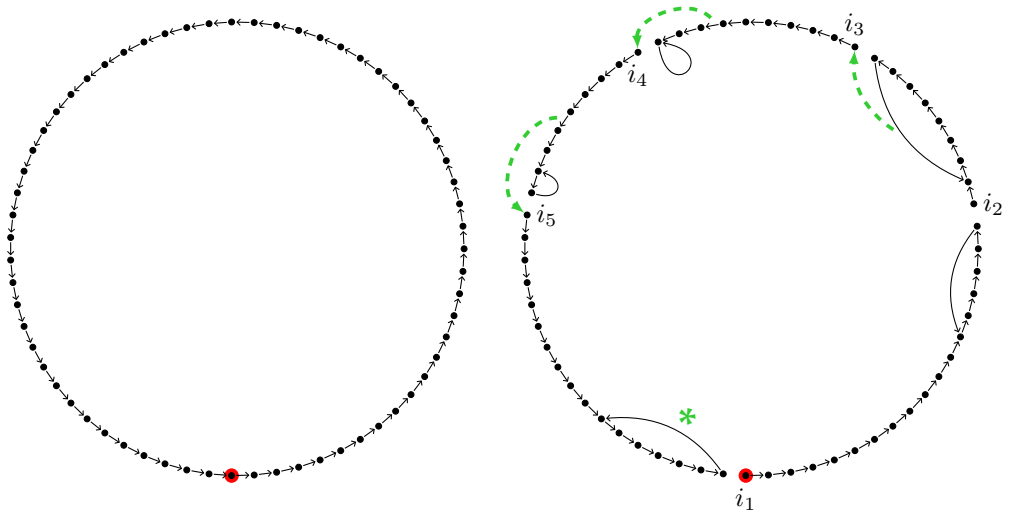


Figure 1.1: A long cycle C and a possible segmentation of C, where $k = 8$ and $q = 5$. The red color marks the cycle leader, a green line from a segment X to a vertex v indicates that X encodes v, a green asterisk indicates the segment of size $2k+1$.

The intuition behind this definition is as follows. We want the segment representation of C to store enough information to be able to restore C. Condition (C3) guarantees that every segment except the first and the last one encodes the beginning of the next one. For the first segment, the beginning of its successor can be decoded from the number S and the length of the cycle in the segment – this is condition (C2). The last segment has size $2k+1$, while the length of the cycle can be any integer from $[1, ..., k]$ (condition (C4)). The size $2k+1$ is special, as no other segment can have this size, and indicates that the segment is the last one. The fact that we are free to choose the length of the cycle in the last segment is important and will be used later in the chapter.

Recall that for a long cycle, we want to obtain a segment representation of its inverse. In Listing 1.6, we present a procedure that achieves this goal in-place and in time linear in the size of the cycle. We require that when MAKE-SEGMENTS is called, vertex $leader$ is the leader of a long cycle C that is a subgraph of G_t. We claim that when the procedure ends, the subgraph of G_t induced by the vertices of C together with the number S returned by MAKE-SEGMENTS is a segment representation of the inverse of C.

Before describing the pseudocode, we make several remarks. First, for readability, the code in Listing 1.6, and others that follow, do not necessarily meet the in-place memory requirements if understood literally, but it is straightforward to rewrite them in such a way that they do. Second, we use the notation $t^i[v]$ defined as follows: $t^0[v] = v$ and $t^i[v] = t[t^{i-1}[v]]$ for $i = 1, 2, \ldots$. Finally, we note that except creating appropriate segments, the procedure returns the value $bg\_of\_sg\_created\_first$ that is to be used later. In fact, this variable keeps the beginning of the first segment created by this procedure.

Listing 1.6 Producing a segment representation of the inverse of a cycle

```
MAKE-SEGMENTS(leader)
 1  to_encode = leader
 2  v_1 = t[leader]
 3  while leader is not among v_1, t[v_1], t^2[v_1], ..., t^{2k}[v_1]
 4      if to_encode == leader
 5          s = 2k + 1
 6          cycle_length = 1   // we can choose any length between 1 and k
 7          bg_of_sg_created_first = t^{2k}[v_1]
 8      else (s, cycle_length) = ENCODE(to_encode)
 9      let v_i = t^{i-1}[v_1] for i = 2, ..., s
10      next_v1 = t[v_s]
11      set t[v_i] = v_{i-1} for i = 2, ..., s
12      t[v_1] = v_{cycle_length}
13      to_encode = v_s
14      v_1 = next_v1
15  S = s
16  let p be the smallest i such that t^{i-1}[v_1] = leader
17  let v_i = t^{i-1}[v_1] for i = 2, ..., p
18  t[v_1] = to_encode
19  set t[v_i] = v_{i-1} for i = 2, ..., p
20  return (bg_of_sg_created_first, S)
```

In Listing 1.6, we simultaneously reverse the cycle and form new segments. We maintain positions v_1 and to_encode in this cycle such that to_encode is the predecessor of v_1. Initially, v_1 is set to $t[leader]$ and to_encode to $leader$. In each iteration of the loop, we form a new segment with beginning $t^{s-1}[v_1]$, size s and vertices $v_1, t[v_1], ..., t^{s-1}[v_1]$, and set to_encode to the beginning of this segment and v_1 to next vertex of the cycle. The first segment created in the loop is made to satisfy condition (C4), i.e. just to have the special size $2k + 1$. Every next segment formed in this loop is created with size and

cycle length to encode the vertex to_encode, as required by (C3). Line 11 is responsible for reversing the edges and line 12 for setting appropriate cycle length. The loop condition ensures that even if s is set to the maximum possible value, i.e. $2k+1$, the newly formed segment will not contain $leader$. After the loop finishes, the negation of the loop condition guarantees that the path $(v_1, t[v_1], t^2[v_1], ..., leader)$ has at most $2k+1$ vertices. In lines 18–19, this path is reversed and attached to the last created segment. We note that because long cycles are defined to have at least $4k+3$ vertices, at least two segments are created, as required in the definition of the segment representation. Thus, the last created segment has size at most $2k$ before attaching the path, and at most $4k+1$ after attaching it. Next, as the loop condition is true before the last iteration, the last created segment after the attachment of the path has at least $2k+2$ vertices. Together with the fact that its cycle length and the value S are appropriately set, we get that the condition (C2) is satisfied.

Now consider the problem of restoring the original cycle from its segment representation. We start at the leader, decode the beginning of the next segment using the value S, redirect a single edge and proceed to the next segment. This time, we use the size of the segment and the size of its cycle to decode the beginning of the next segment. We repeat this step several times until we encounter the segment of size $2k+1$. Then we redirect the last edge to the cycle leader (stored in a separate variable) and stop. According to condition (C4), the cycle in this last encountered segment may have any length between 1 and k. The code actually returns this length, as it is used by our inversion algorithm to store additional information. The full procedure to restore a long cycle is presented in Listing 1.7.

Listing 1.7 Restoring a long cycle from its segment representation

```
RESTORE-LONG-CYCLE(leader, S)
 1  (cycle_length, dist_to_cycle) = TORTOISE-AND-HARE(leader)
 2  segment_size = dist_to_cycle + cycle_length
 3  last = t^(segment_size−1)[leader]
 4  bg = t[last] = DECODE(S, cycle_length)
 5  while TRUE
 6      (cycle_length, dist_to_cycle) = TORTOISE-AND-HARE(bg)
 7      segment_size = dist_to_cycle + cycle_length
 8      last = t^(segment_size−1)[bg]
 9      if segment_size == 2k + 1
10          t[last] = leader
11          return cycle_length
12      else bg = t[last] = DECODE(segment_size, cycle_length)
```

1.5 The $O(n^{3/2})$ algorithm

Let us summarize how we change the $O(n^2)$ algorithm to obtain $O(n^{3/2})$ running time. First, when a long cycle is first visited, it is reversed and converted into its segment representation. The value S is stored in another part of the graph G_t – this step is explained

in Section 1.6, for now assume that this is somehow implemented, in appropriate time and memory complexity. Short cycles are treated as before and we add a third case to the main loop: if i belongs to a tail of a segment, we do nothing. As a result, after the main loop is complete, all short cycles are reversed and for every long cycle there is a segmentation of its inverse with the cycles in all segments reversed (notice that for every segment, its cycle σ is reversed exactly once – when i becomes the leader of σ).

It remains to reverse the cycles in all segments again and then restore the original cycle for every segment representation of a long cycle. This is achieved with two more passes over all vertices – first we reverse the cycles in segments and then we restore the long cycles. The whole algorithm is shown in Listing 1.8. We claim that it inverts the permutation in-place in $O(n^{3/2})$ time.

Listing 1.8 The in-place $O(n^{3/2})$ time algorithm to invert a permutation

```
INVERT-PERMUTATION()
 1  for i = 1 to n
 2      (cycle_length, dist_to_cycle) = LIMITED-TORTOISE-AND-HARE(i, 4k + 2)
 3      if (cycle_length, dist_to_cycle) == (NIL, NIL)
 4          // i belongs to a long cycle
 5          (bg_of_sg_created_first, S) = MAKE-SEGMENTS(i)
 6          store S
 7      else if dist_to_cycle ⩾ 1
 8          // i belongs to a tail of a segment
 9          continue
10      else // i belongs to a short cycle
11          if CYCLE-LEADER(i) == i
12              REVERSE-CYCLE(i)
13  for i = 1 to n
14      (cycle_length, dist_to_cycle) = LIMITED-TORTOISE-AND-HARE(i, 4k + 2)
15      if dist_to_cycle == 1
16          REVERSE-CYCLE(t[i])
17  for i = 1 to n
18      (cycle_length, dist_to_cycle) = LIMITED-TORTOISE-AND-HARE(i, 4k + 1)
19      if dist_to_cycle ≠ NIL and dist_to_cycle ⩾ 1
20          // i belongs to a tail of a segment and is the leader of a long cycle in π
21          retrieve S
22          RESTORE-LONG-CYCLE(i, S)
```

Let us prove the correctness of the algorithm. First, note that the call to LIMITED--TORTOISE-AND-HARE in line 2 returns (NIL, NIL) if and only if the vertex i belongs to a long cycle. This is because we only create segments of size at most $4k+1$ and long cycles are defined to have at least $4k+3$ vertices. Thanks to this observation, we know that our algorithm correctly determines whether i is a part of a tail, a short, or a long cycle, as asserted in lines 4, 8, 10. Second, note that the call to LIMITED-TORTOISE-AND-HARE in line 14 never returns (NIL, NIL), because at this point every component of G_t is either a segment of size at most $4k + 1$ or a short cycle. Third, note that the call to

LIMITED-TORTOISE-AND-HARE in line 18 does not return (NIL, NIL) if i belongs to a tail of a segment, because every segment ever created has size at most $4k+1$. Therefore, no tail is omitted by the loop in lines 17 – 22.

Now, let C be a cycle in π with vertex set V'. If C is short, our algorithm reverses it only once – when i is the leader of C. If C is long, consider all the moments when our algorithm modifies $t[v]$ for $v \in V'$. The first such moment is when during the loop in lines 1 – 12 vertex i becomes the leader of C, and then the cycle is reversed and split into segments. Next, during the same loop, all that happens is that every cycle in every segment is reversed exactly once – when i becomes the leader of the cycle. During the loop in lines 13 – 16, every cycle in every segment is reversed again, also exactly once (because for every such cycle σ there is exactly one vertex with distance to σ equal to 1). Thus, after this loop, the graph $G_t[V']$ is again a segmentation of the inverse of C. Later, the first change to $G[V']$ that occurs in the loop in lines 17 – 22 happens when i becomes the leader of C. Then, the line 22 sets $G_t[V']$ to be the inverse of C. From this moment on, the code does not modify $G_t[V']$. This completes the correctness proof.

We now prove that the running time of INVERT-PERMUTATION is $O\big(n^{3/2}\big)$. To see that the loop in lines 1 – 12 runs in $O\big(n^{3/2}\big)$, note that:

- the call to LIMITED-TORTOISE-AND-HARE in line 2 takes time $O(4k+2) = O(\sqrt{n})$,
- the call to MAKE-SEGMENTS in line 5 takes time proportional to the length of the cycle to which i belongs, potentially $O(n)$, but occurs just once for every long cycle,
- lines 11 – 12 take $O(\sqrt{n})$ time (at each iteration of the loop), as the cycle the vertex i belongs to is short.

As for the loop in lines 13 – 16, the instructions LIMITED-TORTOISE-AND-HARE$(i, 4k+2)$ and REVERSE-CYCLE$(t[i])$ take time $O(\sqrt{n})$: the former because $O(4k+2) = O(\sqrt{n})$, and the latter because the reversed cycle is a part of a segment of size $O(\sqrt{n})$. Finally, in the loop in lines 17 – 22, the call to LIMITED-TORTOISE-AND-HARE takes time $O(4k+1) = O(\sqrt{n})$, and the call to RESTORE-LONG-CYCLE is $O(n)$ but occurs exactly once for each long cycle in π.

As a final remark, we note that the code can easily be rewritten to work in-place, and thus our analysis of Listing 1.8 is complete.

1.6 Storing the value S

Recall that the length of the cycle in the segment of size $2k+1$ of a segmentation can be any integer between 1 and k. We call this cycle the *free cycle* of that segmentation and we use it to store information. We now make use of the return value of the procedure MAKE-SEGMENTS. The procedure returns a pair of integers: the beginning of the segment with the free cycle and the value S.

In our algorithm, there are two passes over all vertices. In both passes, long cycles are visited in the same order, say $C_1, ..., C_r$ (here each C_i denotes a cycle before its reversal). Let $C'_1, ..., C'_r$ be the respective inverses of $C_1, ..., C_r$. After the first pass, in place of every C_i there is a segment representation (R_i, S_i) of C'_i. The idea is to store S_i

as the length of the free cycle in R_{i-1}. This way, when restoring C'_{i-1}, we can retrieve the value S_i and use it to restore C'_i. This is done for $i \geqslant 2$, while the value S_1 is stored in a separate variable, named *first_S*. In Listing 1.9, we present the updated pseudocode of the $O(n^{3/2})$ time algorithm. The only difference compared to the previous version is the implementation of the operations **store** and **retrieve**.

Listing 1.9 The in-place $O(n^{3/2})$ time algorithm to invert a permutation

```
INVERT-PERMUTATION()
 1  storage, first_S = NIL
 2  for i = 1 to n
 3      (cycle_length, dist_to_cycle) = LIMITED-TORTOISE-AND-HARE(i, 4k + 2)
 4      if (cycle_length, dist_to_cycle) == (NIL, NIL)
 5          // i belongs to a long cycle
 6          (bg_of_sg_created_first, S) = MAKE-SEGMENTS(i)
 7          if storage == NIL
 8              first_S = S
 9          else set the length of the cycle in the segment beginning at storage to S
10          storage = bg_of_sg_created_first
11      else if dist_to_cycle ⩾ 1
12          // i belongs to a tail of a segment
13          continue
14      else // i belongs to a short cycle
15          if CYCLE-LEADER(i) == i
16              REVERSE-CYCLE(i)
17  for i = 1 to n
18      (cycle_length, dist_to_cycle) = LIMITED-TORTOISE-AND-HARE(i, 4k + 2)
19      if dist_to_cycle == 1
20          REVERSE-CYCLE(t[i])
21  S = first_S
22  for i = 1 to n
23      (cycle_length, dist_to_cycle) = LIMITED-TORTOISE-AND-HARE(i, 4k + 1)
24      if dist_to_cycle ≠ NIL and dist_to_cycle ⩾ 1
25          // i belongs to a tail of a segment and is the leader of a long cycle in π
26          S = RESTORE-LONG-CYCLE(i, S)
```

We omit the correctness, memory consumption and time consumption analysis of the code, as it is analogous to the analysis already done for Listing 1.8.

1.7 Suggestions for further research

As we said earlier, the problem can be solved in $O(n \log n)$ expected time using a randomized algorithm. Whether there is a deterministic solution running in $O(n \log^c n)$ for some constant c seems to be an interesting question.

2

The Partial Visibility Representation Extension Problem

The concept of a visibility representation of a graph is a classic one in computational geometry and graph drawing and the first studies on this concept date back to the early days of these fields (see, e.g. [73, 75] and [43] for a recent survey). In the most general setting, a visibility representation of a graph is defined as a collection of disjoint sets from an Euclidean space such that the vertices are bijectively mapped to the sets and the edges correspond to unobstructed lines of sight between two such sets. Many different classes of visibility representations have been studied via restricting the space (e.g., to be the plane), the sets (e.g., to be points [15] or line segments [16, 73]) and/or the lines of sight (e.g., to be non-crossing or axis-parallel). In this work we focus on a classic visibility representation setting in which the sets are horizontal line segments (*bars*) in the plane and the lines of sight are vertical. As such, whenever we refer to a visibility representation, we mean one of this type. The study of such representations was inspired by the problems in VLSI design [71, 72] and was conducted by different authors [32, 60, 67] under variations of the notion of visibility. Tamassia and Tollis [73] gave an elegant unification of different definitions and we follow their approach.

The results of this chapter had been obtained in collaboration with Chaplick, Gutowski, Krawczyk and Liotta and were presented at the *24th International Symposium on Graph Drawing and Network Visualization* [19]. The text of this chapter reproduces[1] the full version of these results that appeared in *Algorithmica* [20]. The only change that has been made to the journal version is in formatting, to unify this chapter with the other ones.

The main contribution to [20] by the author of this thesis is the entire speedup of the $O(n^2)$ algorithm of Section 2.4.2 for extending rectangular bar visibility representations of planar st-graphs. This contribution heavily relies on a data structure assembled in Section 2.4.3. This structure, together with dominance drawings and a known technique for building 2-SAT formulas[2], yields an algorithm that runs substantially faster than the original algorithm, i.e. its running time is $O\big(n \log^2 n\big)$. The author of this thesis also made some contributions to the proof that the Bar Visibility Representation Extension Problem for undirected graphs is **NP**-complete, presented in Section 2.5.1.

[1] The paper [20] in *Algorithmica* is *an open access article distributed under the terms of the Creative Commons CC BY (https://creativecommons.org/licenses/by/4.0/) license, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited*. There is no requirement to obtain permission to reuse this article, see *Reprints and Permissions* in https://doi.org/10.1007/s00453-017-0322-4.

[2] I.e. propagating implications through edges of a tree data structure, see [77].

For the sake of completeness, in this chapter we include not only these contributions, but the entire text of [20].

2.1 Introduction

A *horizontal bar* is an open, non-degenerate segment parallel to the x-axis of the coordinate plane. For a set Γ of pairwise disjoint horizontal bars, a *visibility ray* between two bars a and b in Γ is a vertical closed segment spanned between bars a and b that intersects a, b, and no other bar in Γ. A *visibility gap* between two bars a and b in Γ is an axis aligned, non-degenerate open rectangle spanned between bars a and b that intersects no other bar.

For a graph G, a *visibility representation* ψ is a function that assigns a distinct horizontal bar to each vertex such that these bars are pairwise disjoint and satisfy additional visibility constraints. Following Tamassia and Tollis [73], we distinguish three different visibility models:

- *Weak visibility.* In this model, for each edge $\{u, v\}$ of G, there is a visibility ray between $\psi(u)$ and $\psi(v)$ in $\psi(V(G))$.
- *Strong visibility.* In this model, two vertices u, v of G are adjacent if and only if there is a visibility ray between $\psi(u)$ and $\psi(v)$ in $\psi(V(G))$.
- *Bar visibility.* In this model, two vertices u, v of G are adjacent if and only if there is a visibility gap between $\psi(u)$ and $\psi(v)$ in $\psi(V(G))$.

The bar visibility model is also known as the ε-visibility model in the literature. See Figure 2.1 for an example that shows different representations of the cycle C_4 in three visibility models.

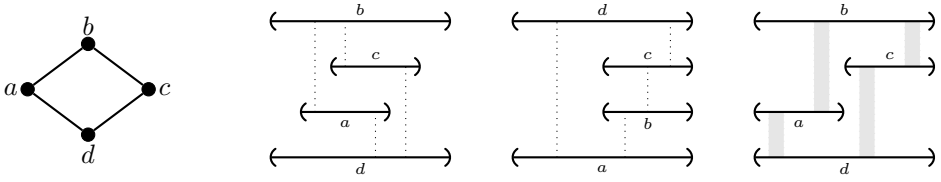


Figure 2.1: Representation of the cycle C_4 in weak, strong, and bar visibility model.

A graph that admits a visibility representation in any of these models is a planar graph, but the converse does not hold in general. Tamassia and Tollis [73] characterized the graphs that admit a visibility representation in these models as follows. A graph admits a weak visibility representation if and only if it is planar. A graph admits a bar visibility representation if and only if it has a planar embedding with all cut-vertices on the boundary of the outer face. For both of these models, Tamassia and Tollis [73] presented linear-time algorithms for the recognition of representable graphs, and for constructing the appropriate visibility representations. The situation is different for the strong visibility model. Although the planar graphs admitting a strong visibility representation are characterized in [73] (via strong st-numberings), Andreae [4] proved that the recognition of these graphs is **NP**-complete. Summing up, from a computational

point of view, the problems of recognizing graphs that admit visibility representations and of constructing such representations are well understood.

Recently, a lot of attention has been paid to the question of extending partial representations of graphs. In this setting a representation of some vertices of the graph is already fixed and the task is to find a representation of the whole graph that extends the given partial representation (see, e.g. [10, 18, 53, 54, 56, 55] for papers that study computational aspects of extending partial representations of geometric intersection graphs). Problems of this kind are often encountered in graph drawing and are sometimes computationally harder than testing for existence of an unconstrained drawing. The problem of extending partial drawings of planar graphs is a good illustration of this phenomenon. On the one hand, by Fáry's theorem [34], every planar graph can be drawn in the plane so that each vertex is represented as a point, and edges are pairwise non-crossing, straight-line segments joining the corresponding points. Moreover, such a drawing can be constructed in linear time [21, 38, 39]. On the other hand, testing whether a partial drawing of this kind (i.e., an assignment of points to some of the vertices) can be extended to a straight-line drawing of the whole graph is **NP**-hard [68]. However, an analogous problem in the model that allows the edges to be drawn as arbitrary curves instead of straight-line segments has a linear-time solution [5]. A similar phenomenon occurs when we consider contact representations of planar graphs. Every planar graph is representable as a disk contact graph [59] or a triangle contact graph [37]. Every bipartite planar graph is representable as a contact graph of horizontal and vertical segments in the plane [36, 49]. Although such representations can be constructed in polynomial time [36, 37, 63], the problems of extending partial representations of these kinds are **NP**-hard [17].

In this paper we initiate the study of extending partial visibility representations of graphs. From a practical point of view, it may be worth recalling that visibility representations are not only an appealing way of drawing graphs, but they are also typically used as an intermediate step towards constructing visualizations of networks in which all edges are oriented in a common direction and some vertices are aligned (for example to highlight critical activities in a PERT diagram). Visibility representations are also used to construct orthogonal drawings with at most two bends per edge. See, e.g. [25] for more details about these applications of visibility representations. The partial representation extension problem that we study in this paper occurs, for example, when we want to use visibility representations to incrementally draw a large network and we want to preserve the user's mental map in a visual exploration that adds a few vertices and edges per time.

Both for weak visibility and for strong visibility, the partial representation extension problems are easily found to be **NP**-hard. For weak visibility, the hardness follows easily from results on contact representations by Chaplick et al. [17]. For strong visibility, it follows trivially from results by Andreae [4]. Our contribution is the study of the partial representation extension problem for bar visibility representations. Hence, the central problem for this paper is the following:

Bar Visibility Representation Extension:

Input: (G, ψ'), where G is a graph and ψ' is a mapping assigning bars to some subset V' of $V(G)$.
Question: Does G admit a bar visibility representation ψ with $\psi|V' = \psi'$?

In Section 2.5 we show that this problem is **NP**-complete.

Theorem 2.1. *The Bar Visibility Representation Extension Problem is* **NP***-complete.*

The proof is a standard reduction from PLANARMONOTONE3SAT problem, which is known to be **NP**-complete thanks to de Berg and Khosravi [8]. The reduction uses gadgets that simulate logic gates and constructs a planar Boolean circuit that encodes the given formula.

We investigate a few natural modifications of the problem. Most notably, we provide some efficient algorithms for extension problems for directed graphs. A visibility representation introduces a natural orientation to edges of the graph – each edge is oriented from the lower bar to the upper one. The function ψ is a representation of a digraph G if, additionally to satisfying visibility constraints, it puts the bar $\psi(u)$ strictly below the bar $\psi(v)$ for each directed edge (u, v) of G. Note that a planar digraph that admits a visibility representation also admits an *upward planar drawing* (see e.g., [41]), that is, a drawing in which the edges are represented as non-crossing monotonic upward curves.

A *source* (*sink*) of a digraph is a vertex without incoming (outgoing) edges. A *planar st-graph* is a planar acyclic digraph with exactly one source s and exactly one sink t that admits a planar embedding such that s and t are on the outer face. Di Battista and Tamassia [27] proved that the following three statements are equivalent for a planar digraph G:

- G admits an upward planar drawing,
- G is a subgraph of a planar st-graph,
- G admits a weak visibility representation.

Garg and Tamassia [42] showed that the recognition of planar digraphs that admit an upward planar drawing is **NP**-complete. It follows that the recognition of planar digraphs that admit a weak visibility representation is **NP**-complete, and so is the corresponding partial representation extension problem. Nevertheless, the situation might be different for bar visibility. In Section 2.3 we prove the following lemma that characterizes planar digraphs that admit a bar visibility representation.

Lemma 2.2. *Let $st(G)$ be a graph constructed from a planar digraph G by adding two vertices s and t, the edge (s, t), an edge (s, v) for each source vertex v of G, and an edge (v, t) for each sink vertex v of G. A planar directed graph G admits a bar visibility representation if and only if the graph $st(G)$ is a planar st-graph.*

Since planar st-graphs can be recognized in linear time, planar digraphs that admit a bar visibility representation are also recognizable in linear time. The natural problem that arises is the following:

Bar Visibility Representation Extension for digraphs:

Input: (G, ψ'), where G is a directed graph and ψ' is a mapping assigning bars to some subset V' of $V(G)$.
Question: Does G admit a bar visibility representation ψ with $\psi|V' = \psi'$?

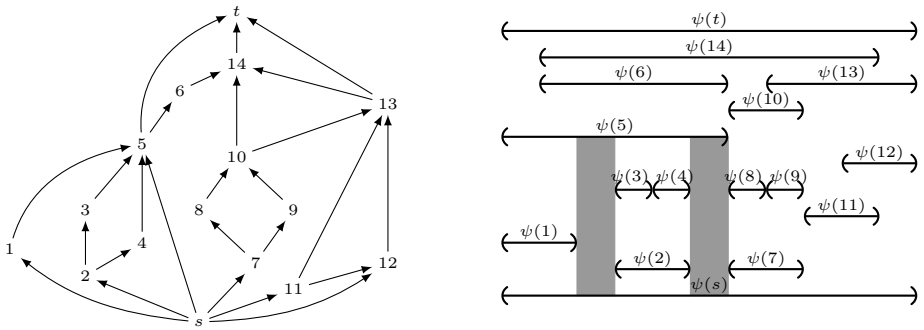


Figure 2.2: A planar st-graph G and a rectangular bar visibility representation ψ of G.

Although we do not provide a solution for this problem, we present an efficient algorithm for an important variant. A bar visibility representation ψ of a directed graph G is called *rectangular* if ψ has a unique bar $\psi(s)$ with the lowest y-coordinate, a unique bar $\psi(t)$ with the highest y-coordinate, $\psi(s)$ and $\psi(t)$ span the same x-interval, and all other bars are inside the rectangle spanned between $\psi(s)$ and $\psi(t)$. See Figure 2.2 for an example of a rectangular bar visibility representation of a planar st-graph.

Tamassia and Tollis [73] showed that a planar digraph G admits a rectangular bar visibility representation if and only if G is a planar st-graph. In Section 2.4 we give an efficient algorithm for the following problem:

Rectangular Bar Visibility Representation Extension for planar st-graphs:

Input: (G, ψ'), where G is a planar st-graph and ψ' is a mapping assigning bars to some subset V' of $V(G)$.
Question: Does G admit a rectangular bar visibility representation ψ with $\psi|V' = \psi'$?

The main result in this paper, presented in Section 2.4, is the following.

Theorem 2.3. *The Rectangular Bar Visibility Representation Extension Problem for a planar st-graph with n vertices can be solved in $O(n \log^2 n)$ time.*

Our algorithm exploits the correspondence between bar visibility representations and st-orientations of planar graphs, and utilizes the $SPQR$-decomposition of planar graphs.

In the study of planar graphs and their representations, it is important to understand the area requirements of a drawing. A common way to measure this is by the smallest integer grid in which a drawing can be realized (see, e.g. [9, 50, 51, 52, 74] for papers that specifically study the area required by visibility representations of graphs and [26] for a survey on compact drawings of graphs).

A visibility representation is a *grid representation* when all bars used in the representation have integral coordinates. Any visibility representation can be easily modified into a grid representation. However, this transformation does not preserve coordinates of the bars. In particular, it might not preserve the partial representation. We can show that the (Rectangular) Bar Visibility Representation Extension Problem is **NP**-hard on series-parallel planar st-graphs when one demands a grid representation.

Our results use different tools developed for graph representation problems. In particular, we exploit the correspondence between bar visibility representations and st-orientations of planar graphs, and utilize the $SPQR$-decomposition for planar graphs.

The rest of the paper is organized as follows. Section 2.2 contains the necessary definitions and description of the necessary tools. Section 2.3 contains a characterization of planar digraphs that admit a bar visibility representation. Section 2.4 contains the study of rectangular representations of planar st-graphs. Section 2.5 contains hardness results for grid representations and for the bar visibility representation extension problem for undirected graphs. Section 2.6 contains conclusions and some open problems.

2.2 Preliminaries

2.2.1 Notation

A *horizontal bar* is an open, non-degenerate segment parallel to the x-axis of the coordinate plane. For a horizontal bar a, functions $y(a)$, $l(a)$, $r(a)$ give the y-coordinate of a, the x-coordinate of the left end of a, and the x-coordinate of the right end of a respectively. For any bounded object Q in the plane, we use functions $X(Q)$ and $Y(Q)$ to denote the smallest possible, possibly degenerate, closed interval containing the projection of Q on the x-, and on the y-axis respectively. We denote the left end of $X(Q)$ by $l(Q)$ and the right end of $X(Q)$ by $r(Q)$. Let a and b be two horizontal bars with $y(a) < y(b)$. We say that Q is *spanned between a and b* if $X(Q) \subseteq X(a) \cap X(b)$, and $Y(Q) = [y(a), y(b)]$.

For a graph G, a visibility representation ψ in any model (see Section 2.1) is a function that assigns distinct, pairwise disjoint horizontal bars to the vertices of G. We often describe the representation ψ by providing the values of functions $y_\psi = y(\psi(v))$, $l_\psi = l(\psi(v))$, $r_\psi = r(\psi(v))$ for any vertex v of G. We drop the subscripts when the representation ψ is known from the context.

2.2.2 Planar st-graphs and their properties

Given a graph $G = (V, E)$, a *planar drawing* of G is a geometric representation of G in the plane such that: (i) each vertex $v \in V$ is drawn as a distinct point p_v; (ii) each edge $e = (u, v) \in E$ is drawn as a simple curve connecting p_u and p_v; (iii) no two edges intersect except at their common end-vertices (if they are adjacent). A graph is *planar* if it admits a planar drawing.

Let G be a connected planar graph. A planar drawing Θ of G divides the plane into topologically connected regions, called *faces*. Exactly one face of Θ is an infinite region, and is called the *external face* of Θ; the other faces are called *internal*. Each internal face is described by the counter-clockwise sequence of vertices and edges that form its boundary; the external face is described by the clockwise sequence of vertices and edges of its boundary. The description of the set of (internal and external) faces determined by a planar drawing of G is called a *planar embedding* of G.

Let G be a planar st-graph. An *st-embedding* of G is any planar embedding with s and t on the boundary of the outer face. A planar st-graph together with an st-embedding is called a *plane st-graph*. Vertices s and t of a planar (plane) st-graph are called the *poles* of G. We abuse notation and we use the term *planar (plane) uv-graph* to mean a planar (plane) st-graph with poles u and v. An *inner vertex* of G is a vertex of G other than the poles of G. A real valued function ξ from $V(G)$ is an *st-valuation* of G if for each edge (u, v) we have $\xi(u) < \xi(v)$.

Tamassia and Tollis [73] showed that the following properties are satisfied for any plane st-graph:

1. For every inner face f, the boundary of f consists of two directed paths with a common origin and a common destination.

2. The boundary of the outer face consists of two directed paths, with a common origin s and a common destination t.

3. For every inner vertex v, its incoming (outgoing) edges are consecutive around v.

To illustrate the above properties, observe in Figure 2.2 two paths on the boundary of the face $(s, 2, 3, 5, 1)$, and the alignment of incoming and outgoing edges around vertex 5.

Let G be a plane st-graph. We introduce two special objects associated with the outer face of G: the *left outer face* s^* and the *right outer face* t^*. Let $e = (u, v)$ be an edge of G. The *left face* (*right face*) of e is the face of G that is to the left (right) of e when we traverse e from u to v. If the outer face of G is to the left (right) of e then we say that the left face (right face) of e is s^* (t^*).

Using property (1) we can define the *left path* and the *right path* for each inner face of G as follows. If f is an inner face of G then the left path (right path) of f consists of edges from the boundary of f for which f is the right face (left face). For example, in Figure 2.2, the path $(s, 1, 5)$ is the left path of the face $(s, 2, 3, 5, 1)$ and the path $(s, 2, 3, 5)$ is the right path of this face.

Using property (2) we can define the left path for t^* and the right path for s^* as follows. The right path of s^* consists of edges from the boundary of the outer face that have the outer face on their left side. The left path of t^* consists of edges from the boundary of the outer face that have the outer face on their right side. The left path for s^* and the right path for t^* are not defined. For example, in Figure 2.2, the path $(s, 1, 5, t)$ is the right path of s^* and the path $(s, 12, 13, t)$ is the left path of t^*.

Using property (3) we can define the *left face* and the *right face* for each vertex of G as follows. The left face (right face) of an inner vertex v is the unique face f incident to v such that there are two edges e_1 and e_2 on the right path (left path) of f, where e_1 is an incoming edge for v and e_2 is an outgoing edge for v. If the left face (right face) of v is the outer face of G, we say that the left face (right face) of u is s^* (t^*). We also say that s^* (t^*) is the left face (right face) of s and t. For example, in Figure 2.2, the left face of vertex 5 is s^* and the right face of this vertex is the face $(s, 7, 8, 10, 14, 6, 5)$.

Let G be a plane st-graph. Let F be the set of inner faces of G together with s^* and t^*. The *dual* of G is the directed graph G^* with vertex set F and edge set consisting of all pairs (f, g) such that there exists an edge e of G with f being the left face of e and g being the right face of e. Di Battista and Tamassia [27] showed that G^* is a planar s^*t^*-graph.

Let G be a plane st-graph and let G^* be the dual of G. For two faces f and g in $V(G^*)$ we say that f *is to the left of* g, and that g *is to the right of* f, if there is a directed path from f to g in G^*.

2.2.3 $SPQR$-trees for planar st-graphs

An $SPQR$-*tree* for a planar graph G is usually used to describe all possible planar embeddings of G. In this paper we employ a specific version of $SPQR$-trees that allows us to describe all st-embeddings of a planar st-graph. Di Battista and Tamassia [28] were the first to define such $SPQR$-trees, and to prove the properties presented in this section.

Let G be a planar st-graph. A *cut-vertex* of G is a vertex whose removal disconnects G. A *separation pair* of G is a pair of vertices whose removal disconnects G. A *split pair* of G is either a separation pair or a pair of adjacent vertices. A *split component* of a split pair $\{u, v\}$ is either an edge (u, v) or a maximal subgraph C of G such that C is a planar uv-graph and $\{u, v\}$ is not a split pair of C. A *maximal split pair* $\{u, v\}$ of G is a split pair such that there is no other split pair $\{u', v'\}$ where $\{u, v\}$ is contained in some split component of $\{u', v'\}$.

An $SPQR$-*tree* T for a planar st-graph G is a recursive decomposition of G with respect to the split pairs of G. The tree T is rooted and its nodes are of four types: S for *series nodes*, P for *parallel nodes*, Q for *edge nodes*, and R for *rigid nodes*. Each node μ of T represents a planar st-graph (a subgraph of G) called the *pertinent digraph* of μ and denoted by G_μ. We use s_μ and t_μ to denote the poles of G_μ: s_μ is the source of G_μ, and t_μ is the sink of G_μ. The pertinent digraph of the root node of T is G. Each node μ of T has an associated directed multigraph $skel(\mu)$ called the *skeleton* of μ. If μ is not the root of the tree, then let λ be the parent of μ in T. The node μ is associated with an edge of the skeleton of λ, called the *virtual edge* of μ, which connects the poles of G_μ and represents G_μ in $skel(\lambda)$. The tree T is defined recursively as follows.

- *Trivial case.* If G consists of a single edge (s, t), then T is simply a Q-node μ. The skeleton $skel(\mu)$ is G.
- *Series case.* If G is a chain of biconnected components $G_1, \ldots, G_k$ for some $k \geqslant 2$ and $c_1, \ldots, c_{k-1}$ are the cut-vertices encountered in this order on any path from s to t, then the root of T is an S-node μ with children $\mu_1, \ldots, \mu_k$. Let $c_0 = s$ and $c_k = t$. The skeleton $skel(\mu)$ is the directed path $c_0, \ldots, c_k$. The pertinent digraph of μ_i is G_i, and edge (c_{i-1}, c_i) of $skel(\mu)$ is the virtual edge of μ_i.
- *Parallel case.* If $\{s, t\}$ is a split pair of G with split components $G_1, \ldots, G_k$ for some $k \geqslant 2$, then the root of T is a P-node μ with children $\mu_1, \ldots, \mu_k$. The skeleton $skel(\mu)$ has k parallel edges (s, t): $e_1, \ldots, e_k$. The pertinent digraph of μ_i is G_i, and edge e_i of $skel(\mu)$ is the virtual edge of μ_i.
- *Rigid case.* If none of the above applies, let $\{s_1, t_1\}, \ldots, \{s_k, t_k\}$ for some $k \geqslant 2$ be the maximal split pairs of G. For $i = 1, \ldots, k$, let G_i be the union of all split components of $\{s_i, t_i\}$. The root of T is an R-node μ with children $\mu_1, \ldots, \mu_k$. The skeleton $skel(\mu)$ is obtained from G by replacing each subgraph G_i with an edge $e_i = (s_i, t_i)$. The pertinent digraph of μ_i is G_i, and edge e_i of $skel(\mu)$ is the virtual edge of μ_i.

Note also that there is no additional edge between the poles of the skeleton of a series, parallel or rigid node – this is the only difference in the $SPQR$-tree definition given above and the one given in [28]. In particular, our definition ensures that we

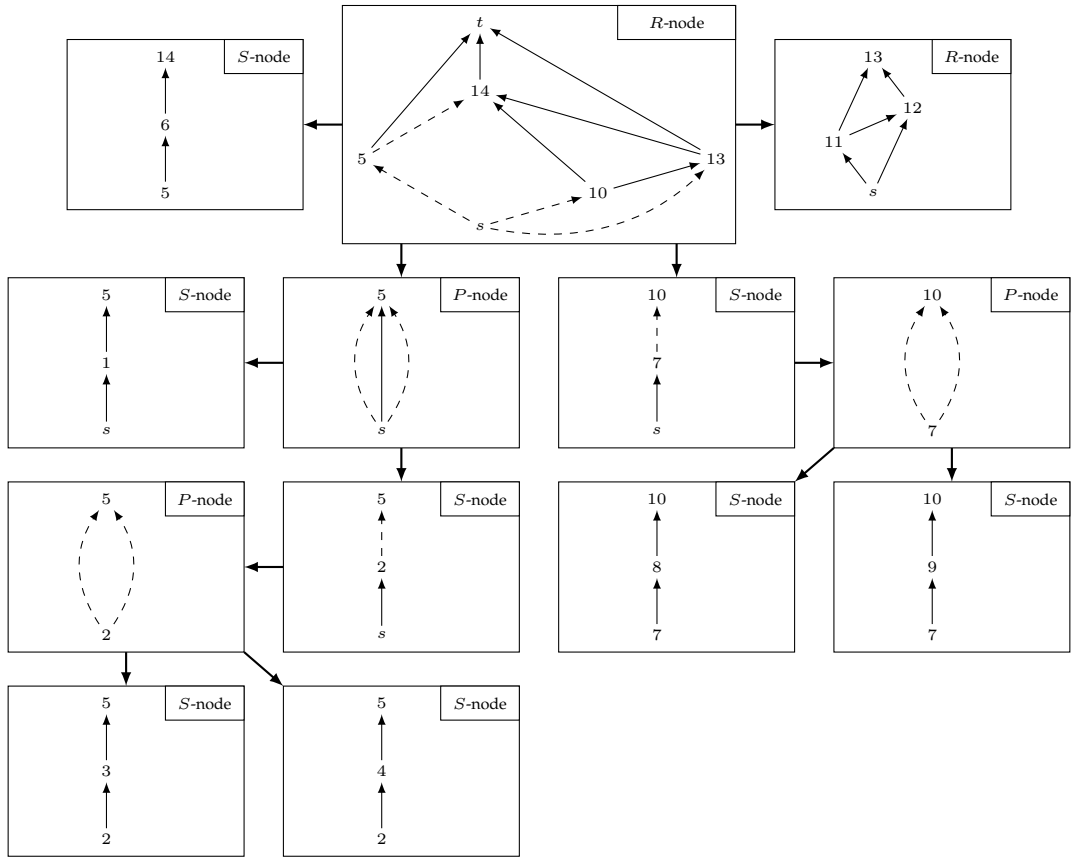


Figure 2.3: The $SPQR$-tree for the graph in Figure 2.2. The Q-nodes (leaves of the tree) have been omitted for clarity. For each S-, P-, and R-node, the skeleton is given such that each solid edge corresponds to a Q-node child and each dashed edge corresponds to a S-, P-, or R-node child.

have a one-to-one correspondence between the edges of $skel(\mu)$ and the children of μ. See Figure 2.3 for an example of an $SPQR$-decomposition of the planar st-graph presented in Figure 2.2.

Observe that the skeleton of a rigid node has only two st-embeddings, one being the flip of the other around the poles of the node. The skeleton of a parallel node with k children has $k!$ st-embeddings, one for every permutation of the edges of $skel(\mu)$. The skeleton of a series node or a edge node has only one st-embedding.

There is a correspondence between st-embeddings of a planar st-graph G and st-embeddings of the skeletons of P-nodes and R-nodes in the $SPQR$-tree T for G. Having selected an st-embedding of the skeleton of all P-nodes and all R-nodes, we can construct an embedding of G as follows. Let t be the root of T. We replace every virtual edge (u, v) in the embedding of $skel(t)$ with the recursively defined embedding of the pertinent digraph of a child of t associated with the edge (u, v). On the other hand, any st-embedding of G determines:

- one of the two possible flips of the skeleton of every R-node in T;
- a permutation of the edges in the skeleton of every P-node.

Di Battista and Tamassia [28] showed that the $SPQR$-tree T for a planar st-graph with n vertices has $O(n)$ nodes, and that the total number of edges of all skeletons is $O(n)$. Gutwenger and Mutzel [44] showed that the $SPQR$-tree can be computed in linear time.

2.2.4 NP-complete problems

Our hardness proofs use reductions from the following **NP**-complete problems:

3PARTITION:

Input: A set of positive integers $w, a_1, a_2, \ldots, a_{3m}$ such that for each $i = 1, \ldots, 3m$, we have $\frac{w}{4} < a_i < \frac{w}{2}$.
Question: Can $\{a_1, \ldots, a_{3m}\}$ be partitioned into m triples, such that the total sum of each triple is exactly w?

3PARTITION is known to be strongly **NP**-complete [40], i.e., the problem remains **NP**-complete even when the integers given in the input are encoded in unary.

PLANARMONOTONE3SAT:

Input: A *rectilinear* planar representation of a 3SAT formula in which each variable is a horizontal segment on the x-axis, each clause is a horizontal segment above or below the x-axis with straight-line vertical connections to the variables it includes. All positive clauses are above the x-axis and all negative clauses are below the x-axis. There are no clauses including both positive and negative occurrences of variables, all horizontal segments are pairwise disjoint, and each vertical connection intersects only with the two segments that it connects. See Figure 2.4 for an example.
Question: Is the formula satisfiable?

PLANARMONOTONE3SAT is known to be **NP**-complete thanks to de Berg and Khosravi [8].

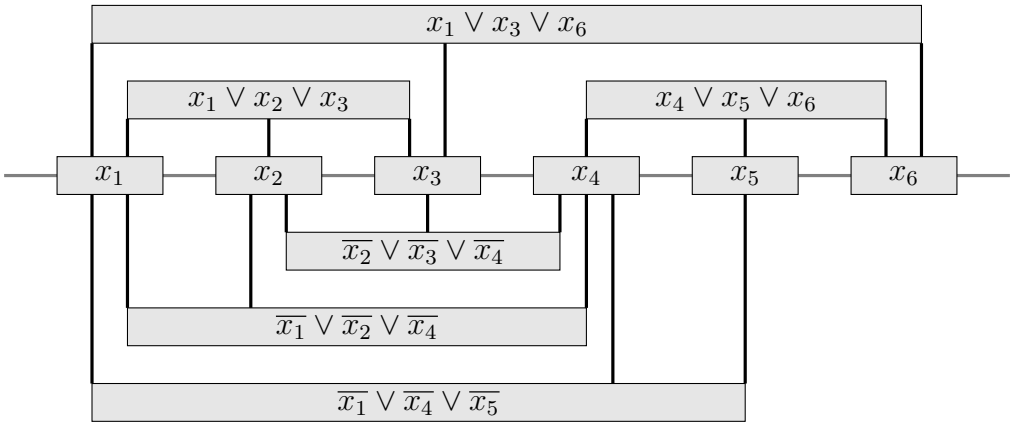


Figure 2.4: A PLANARMONOTONE3SAT formula with variables $x_1, \ldots, x_6$; positive clauses $\{x_1, x_3, x_6\}$, $\{x_1, x_2, x_3\}$, and $\{x_4, x_5, x_6\}$; and negative clauses $\{\overline{x_1}, \overline{x_4}, \overline{x_5}\}$, $\{\overline{x_1}, \overline{x_2}, \overline{x_4}\}$, and $\{\overline{x_2}, \overline{x_3}, \overline{x_4}\}$.

2.3 Bar visibility and rectangular bar visibility representations for planar digraphs

A bar visibility representation ψ of a planar st-graph is *rectangular* when $X(\psi(s)) = X(\psi(t))$ and for any vertex v we have $X(\psi(v)) \subseteq X(\psi(s))$. Tamassia and Tollis [73] observed the following connection between planar st-graphs and rectangular bar visibility representations. Any collection of pairwise disjoint bars Γ with the bottom-most bar s and the top-most bar t that satisfies $X(s) = X(t)$, and $X(a) \subseteq X(s)$ for every $a \in \Gamma$, naturally induces a planar st-graph on the set Γ – a digraph containing all edges (a, b) such that a is strictly below b and there is a visibility gap between a and b in Γ. They further showed that every planar st-graph has a rectangular bar visibility representation.

The next lemma characterizes the planar digraphs that admit a bar visibility representation. For a planar digraph G, let $st(G)$ be a graph constructed from G by adding two vertices s and t, the edge (s, t), an edge (s, v) for each source vertex v of G, and an edge (v, t) for each sink vertex v of G.

Lemma 2.2. *A planar directed graph G admits a bar visibility representation if and only if the graph $st(G)$ is a planar st-graph.*

Proof. Suppose that $st(G)$ is a planar st-graph. Tamassia and Tollis [73] showed that $st(G)$ has a rectangular bar visibility representation ψ with the bottom-most bar $\psi(s)$ and the top-most bar $\psi(t)$. Clearly, $\psi|V(G)$ is a bar visibility representation for G.

Conversely, assume that ψ is a bar visibility representation of G and Γ is the image of ψ. For every bar $a \in \Gamma$, let $A(a)$ ($B(a)$) be the interior of the set of all x such that $\psi(a)$ is the first encountered bar from Γ if we traverse downward (upward) the vertical line with the x-coordinate x. We say that a bar $a \in \Gamma$ is *visible from above (below)* if $A(a) \neq \emptyset$ ($B(a) \neq \emptyset$). Note that each $A(a)$ and $B(a)$ is a union of disjoint open intervals. Observe also that if a represents a sink (a source) of G then $A(a) = (l(a), r(a))$ ($B(a) = (l(a), r(a))$). Otherwise, ψ would not be a bar visibility representation of G.

We claim that some bars in Γ can be extended so that the new set of bars still represents G and has the property that only the bars representing the sources are visible from below and only the bars representing the sinks are visible from above. Before we

give a proof of this claim, suppose that ψ satisfies this property. We can define two bars $\psi(s)$ and $\psi(t)$ such that $X(s) = X(t)$, $X(\bigcup \Gamma) \subsetneq X(s)$, and $y_\psi(s) < y_\psi(v) < y_\psi(t)$ for every vertex v of G. This extension of ψ is a rectangular bar representation of $st(G)$. It follows that $st(G)$ is a planar st-graph.

Now we show that ψ can be modified so that the bars visible from below (above) represent the sources (sinks) of G. Let $\mathcal{X}$ denote the set of the x-coordinates of all end-points of all bars in Γ. Let ε and δ be respectively the minimum and the maximum difference between any two values in $\mathcal{X}$. Suppose that there is a bar $\psi(v)$ in Γ that is visible from below and v is not a source of G. Suppose (L, R) is an interval of $B(\psi(v))$ and observe that both L and R are in $\mathcal{X}$. Since v is not a source of G and ψ is a visibility representation of G, there is a vertex u in G such that (u, v) is an edge of G, and either $r_\psi(u) = L$ or $l_\psi(u) = R$. If $r_\psi(u) = L$, we extend $\psi(u)$ to the right so that $r_\psi(u) = R$, and if $l_\psi(u) = R$, we extend $\psi(u)$ to the left so that $l_\psi(u) = L$. Observe that such a modification only introduces additional visibility gaps between $\psi(u)$ and $\psi(v)$, and does not change any visibility gap between any other two bars. Thus, the modified ψ remains a representation of G. Moreover, all end-points of all bars are still in $\mathcal{X}$ and the total length of all bars increases by at least ε. We repeat the same procedure as long as there is a vertex v that is not a source of G and $\psi(v)$ is visible from below. The number of repetitions is bounded, as the length of any single bar never exceeds δ. In the resulting representation, each bar visible from below represents a source of G. Next, we transform ψ again, using a similar algorithm, so that each bar visible from above represents a sink of G. It is easy to see, that in the second step we do not introduce any new bars that are visible from below. Thus, in the resulting representation, all the bars visible from below are sources of G and all the bars visible from above are sinks of G. □

Lemma 2.2 gives a linear-time algorithm for the recognition of planar digraphs that admit a bar visibility representation. Recall from the discussion in Section 2.1 that the recognition of planar digraphs that admit a weak visibility representation is an **NP**-complete problem. It follows that the extension problem for digraphs in the weak visibility model is **NP**-complete. We do not know the complexity status for the extension problem for digraphs neither in the strong nor in the bar visibility model. Nevertheless, the results in Section 2.4 give hope for a polynomial-time algorithm for the extension problem for bar visibility representations of digraphs.

2.4 Rectangular bar visibility representations of planar st-graphs

In this section we solve the following problem.

Rectangular Bar Visibility Representation Extension for planar st-graphs:

Input: (G, ψ'), where G is a planar st-graph and ψ' is a mapping assigning bars to some subset V' of $V(G)$.
Question: Does G admit a rectangular bar visibility representation ψ with $\psi|V' = \psi'$?

As our algorithm is rather technical (involving many small details), we now provide a high level description of the main ideas. In the first step, our algorithm calculates y-coordinates y_ψ for our potential bars. Namely, the algorithm checks whether

$y_{\psi'} : V' \to \mathbb{R}$ is extendable to an st-valuation y_ψ of G. When such an extension does not exist, the algorithm rejects; otherwise it turns out (as shown in Lemma 2.10) that any extension of $y_{\psi'}$ can be used as y_ψ. To determine whether there is a set of x-coordinates to match this set of y-coordinates, we use a dynamic programming approach which proceeds bottom-up through the $SPQR$-tree T of the given planar st-graph G. Recall that, as discussed in Section 2.2.3, T provides a hierarchical decomposition of G according to separation pairs, i.e., each node μ of T corresponds to a separation pair (u, v) in G. Additionally, for a separation pair (u, v) corresponding to a node μ in T, the subgraph 'between' u and v is a planar st-graph, and is precisely the pertinent digraph of μ. We will see that there is a similar connection between the $SPQR$-tree and each rectangular bar visibility representation ψ of G. Namely, we describe how ψ 'decomposes' into sub-representations according to these separation pairs, i.e., for each node μ and its corresponding separation pair (u, v), there is a particular rectangular bar visibility representation ψ_μ of the pertinent digraph G_μ of μ in ψ. Moreover, we will see that each ψ_μ partitions into rectangular 'tiles' where each 'tile' is a representation of one of its children. We say that a 'tile' of a representation ψ_μ of G_μ is *valid* when it extends $\psi'|V(G_\mu)$. Now, essentially, the key to our dynamic program is to efficiently describe the set of possible valid 'tiles' of the representations of G_μ in terms of the valid 'tiles' of μ's children. It turns out that it is sufficient to consider four types of 'tiles' for each node of T in order to accomplish this. To efficiently determine the types of 'tiles' admitted by a node μ, we distinguish different cases depending on whether μ is a P-node, an S-node, a Q-node, or an R-node. As usual, for dynamic programming involving $SQPR$-trees, the R-node case is the most complex. So, we describe it first in using a quadratic-time subroutine which is more intuitive. We then describe a speed-up of this subroutine which runs in nearly linear time (this subroutine remains as the main bottleneck for our running time).

Section 2.4.1 presents structural properties of bar visibility representations in relation to an $SPQR$-decomposition. We describe how a rectangular bar visibility representation of the pertinent digraph of a node μ in the $SPQR$-decomposition is composed of rectangular bar visibility representations of the pertinent digraphs of the children of μ. In Section 2.4.2 we present an algorithm that solves this extension problem in quadratic time. In Section 2.4.3 we refine the algorithm to work in $O\big(n \log^2 n\big)$ time for a planar st-graph with n vertices.

2.4.1 Structural properties

Let Γ be a collection of pairwise disjoint bars. For a pair of bars a, b in Γ with $y(a) < y(b)$ let the *set of visibility rectangles* $R(a, b)$ be the interior of the set of points (x, y) in $\mathbb{R}^2$ that satisfy the following properties:

1. a is the first bar in Γ on a vertical line downwards from (x, y),
2. b is the first bar in Γ on a vertical line upwards from (x, y).

Figure 2.2 shows (shaded area) the set of visibility rectangles $R(s, 5)$. Note that there is a visibility gap between a and b in Γ if and only if $R(a, b)$ is non-empty. Additionally, if $R(a, b)$ is non-empty, then it is a union of pairwise disjoint open rectangles spanned between a and b.

Let G be a planar st-graph and let T be the $SPQR$-tree for G. Let ψ be a rectangular bar visibility representation of G. For every node μ of T we define the set $B_\psi(\mu)$, called the *bounding box of μ with respect to ψ*, as the closure of the following union:

$$\bigcup \{R(\psi(u), \psi(v)) : (u, v) \text{ is an edge of the pertinent digraph } G_\mu\}.$$

If ψ is clear from the context, then the set $B_\psi(\mu)$ is denoted by $B(\mu)$ and is called the *bounding box of μ*. Let $B(\psi) = X(\psi(V(G))) \times Y(\psi(V(G)))$ be the minimal closed axis-aligned rectangle that contains the representation ψ. It follows from the definition of rectangular embedding, and from the definition of bounding box, that:

1. $B(\psi) = B_\psi(\mu)$, where μ is the root of T,
2. each point in $B(\psi)$ is in the closure of at least one set of visibility rectangles $R(\psi(u), \psi(v))$ for some edge (u, v) of G,
3. each point in $B(\psi)$ is in at most one set of visibility rectangles.

The following two lemmas describe basic properties of a bounding box.

Lemma 2.4 (Q-Tiling Lemma)**.** *Let μ be a Q-node in T that corresponds to an edge (u, v) of G. For any rectangular bar visibility representation ψ of G we have:*

1. *$B(\mu)$ is a union of pairwise disjoint rectangles spanned between $\psi(u)$ and $\psi(v)$.*
2. *If $B(\mu)$ is not a single rectangle, then the parent λ of μ in T is a P-node, and u, v are the poles of the pertinent digraph G_λ.*

Proof. The first assertion is obvious. Suppose that $B(\mu)$ is a union of at least 2 rectangles. Let R_1 and R_2 be the two left-most rectangles of $B(\mu)$. Consider the rectangle S spanned between $\psi(u)$ and $\psi(v)$ and between the right side of R_1 and left side of R_2. There are some bars in $\psi(G)$ that are contained in S. The vertices corresponding to these bars together with u and v form a planar uv-graph. Hence, the split pair $\{u, v\}$ has at least two split components: the edge (u, v) and at least one other component. Thus, λ is a P-node with poles u and v. □

In Figure 2.5 observe that the set $R(s, 5)$ is a union of two rectangles. Recall the $SPQR$-decomposition presented in Figure 2.3 and that the Q-node corresponding to the edge $(s, 5)$ is a child of a P-node.

The Basic Tiling Lemma presented below describes the relation between the bounding box of an inner node μ and the bounding boxes of the children of μ in any rectangular bar visibility representation of G. In particular, the next lemma justifies the use of the name *bounding box* for the set $B(\mu)$.

Lemma 2.5 (Basic Tiling Lemma)**.** *Let μ be an inner node in T with children $\mu_1, \ldots, \mu_k$, $k \geqslant 2$. For any rectangular bar visibility representation ψ of G we have:*

1. *$\psi(v) \subseteq B(\mu)$ for every inner vertex v of G_μ.*
2. *$B(\mu)$ is a rectangle that is spanned between $\psi(s_\mu)$ and $\psi(t_\mu)$.*

3. *The sets $B(\mu_1), \ldots, B(\mu_k)$ tile the rectangle $B(\mu)$, i.e., $B(\mu_1), \ldots, B(\mu_k)$ cover $B(\mu)$ and the interiors of $B(\mu_1), \ldots, B(\mu_k)$ are pairwise disjoint.*

Proof. Observe that for an inner vertex v of G_μ, any edge of G incident to v is an edge of G_μ. The closures of the sets of visibility rectangles corresponding to all edges incident to v cover $\psi(v)$ and Property (1) follows.

To prove (2) note that for every inner vertex v of G_μ, the set

$$S_\mu(v) = X(\psi(v)) \times [y(\psi(s_\mu)), y(\psi(t_\mu))]$$

is a rectangle that is spanned between $\psi(s_\mu)$ and $\psi(t_\mu)$ and it is internally disjoint from $\psi(w)$ for any vertex w not in $V(G_\mu)$. Otherwise, there would be a visibility gap that would correspond to an edge between an inner vertex of G_μ and a vertex in $V(G) \setminus V(G_\mu)$.

If (s_μ, t_μ) is not an edge of G_μ, then

$$B(\mu) = \bigcup \{S_\mu(v) : v \text{ is an inner vertex of } G_\mu\};$$

otherwise

$$B(\mu) = \bigcup \{S_\mu(v) : v \text{ is an inner vertex of } G_\mu\} \cup R(\psi(s_\mu), \psi(t_\mu)).$$

In both cases $B(\mu)$ is a rectangle spanned between $\psi(s_\mu)$ and $\psi(t_\mu)$.

Property (3) follows immediately from the fact that the edges of $G_{\mu_1}, \ldots, G_{\mu_k}$ form a partition of the edges of G_μ. □

In the next three lemmas we extend the Basic Tiling Lemma by a more precise description of tilings of the bounding box of an inner node μ by the bounding boxes of the children of μ. We separately consider the cases that μ is a P-node, an S-node, and an R-node. Figures 2.5, and 2.6 give a graphical presentation of the Tiling Lemmas. In Lemmas 2.6, 2.7, and 2.9 we assume that $\mu_1, \ldots, \mu_k$ are the children of μ for some $k \geqslant 2$.

Lemma 2.6 (P-Tiling Lemma)**.** *Let μ be a P-node. For any rectangular bar visibility representation ψ of G we have:*

1. *If (s_μ, t_μ) is not an edge of G, then the sets $B(\mu_1), \ldots, B(\mu_k)$ are rectangles spanned between $\psi(s_\mu)$ and $\psi(t_\mu)$.*

2. *If (s_μ, t_μ) is an edge of G, then μ has exactly one child that is a Q-node, say μ_1, and:*

   - *For $i = 2, \ldots, k$, $B(\mu_i)$ is a rectangle spanned between $\psi(s_\mu)$ and $\psi(t_\mu)$.*
   - *$B(\mu_1)$ is a non-empty union of rectangles spanned between $\psi(s_\mu)$ and $\psi(t_\mu)$.*

Proof. This is an immediate consequence of the Basic Tiling Lemma and Lemma 2.4. □

When μ is an S-node or an R-node, then there is no edge (s_μ, t_μ). By the Q-Tiling Lemma and by the Basic Tiling Lemma, each set $B(\mu_i)$ is a rectangle that is spanned between the bars representing the poles of G_{μ_i}.

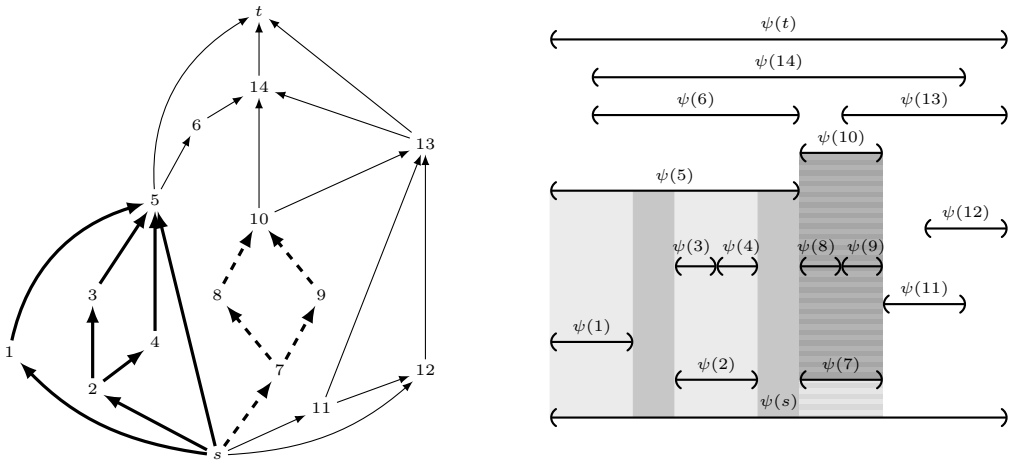


Figure 2.5: The graph G, the pertinent digraph of the P-node μ_1 (solid thick edges), the pertinent digraph of the S-node μ_2 (dashed thick edges), the representation $\psi(G)$, the tiling of μ_1 in $\psi(G)$ (solid fill), and the tiling of μ_2 in $\psi(G)$ (patterned fill).

Lemma 2.7 (S-Tiling Lemma)**.** *Let μ be an S-node. Let $c_1, \ldots, c_{k-1}$ be the cut-vertices of G_μ encountered in this order on a path from s_μ to t_μ. Let $c_0 = s_\mu$, and $c_k = t_\mu$. For any rectangular bar visibility representation ψ of G, for every $i = 1, \ldots, k-1$, we have $X(\psi(c_i)) = X(B(\mu))$. For every $i = 1, \ldots, k$, $B(\mu_i)$ is spanned between $\psi(c_{i-1})$ and $\psi(c_i)$ and $X(B(\mu_i)) = X(B(\mu))$.*

Proof. Suppose to the contrary, that the bar assigned to cut-vertex c_i is the first one that does not span the whole interval $X(B(\mu))$. This creates a gap of visibility between a vertex in the i-th biconnected component and a vertex in one of the later components. This contradicts c_i being a cut-vertex. □

The R-Tiling Lemma should describe all possible tilings of the bounding box of an R-node μ that appear in all representations of G. Since there is a one-to-one correspondence between the edges of $skel(\mu)$ and the children of μ, we abuse notation and write $B(u, v)$ to denote the bounding box of the child of μ that corresponds to the edge (u, v) of $skel(\mu)$. By the Basic Tilling Lemma, $B(u, v)$ is spanned between the bars representing u and v.

Suppose that ψ is a representation of G. The tiling $\tau = (B_\psi(\mu_1), \ldots, B_\psi(\mu_k))$ of $B_\psi(\mu)$ determines a triple $(\mathcal{E}, \xi, \chi)$, where:

- $\mathcal{E}$ is an $s_\mu t_\mu$-embedding of $skel(\mu)$,
- ξ is an st-valuation of $\mathcal{E}$,
- χ is an st-valuation of $\mathcal{E}^*$,

that are defined as follows.

Consider the following planar drawing of the planar st-graph $skel(\mu)$. Draw every vertex u in the middle of $\psi(u)$, and every edge $e = (u, v)$ as a curve that starts in the middle of $\psi(u)$, goes a little above $\psi(u)$ towards the rectangle $B_\psi(u, v)$, goes inside $B_\psi(u, v)$ towards $\psi(v)$, and a little below $\psi(v)$ to the middle of $\psi(v)$. This way we obtain a plane st-graph $\mathcal{E}$, which is an st-embedding of $skel(\mu)$. The st-valuation ξ of $\mathcal{E}$

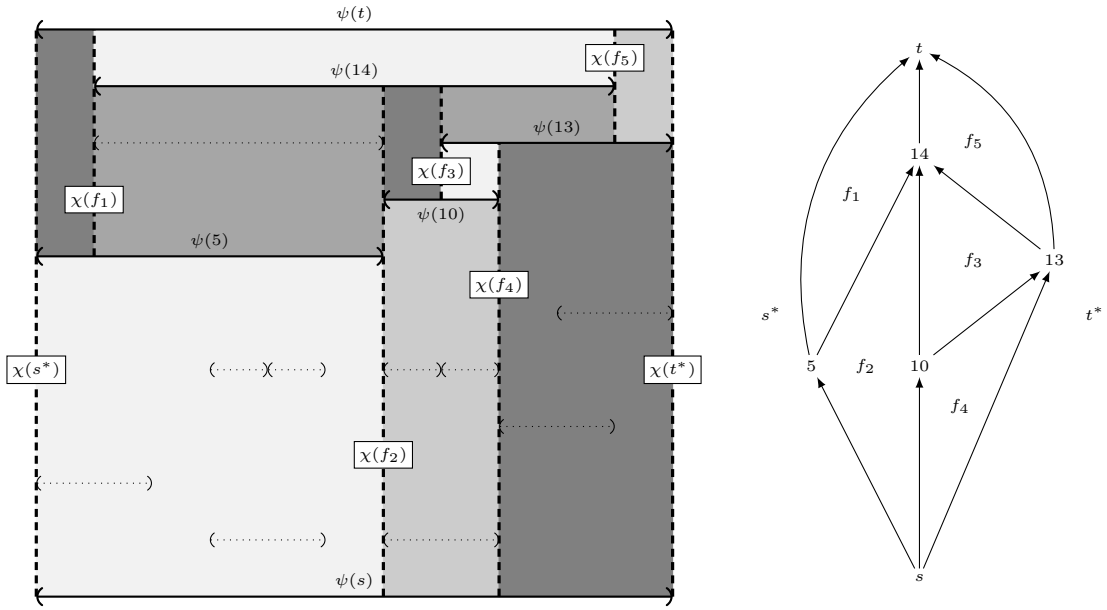


Figure 2.6: The tiling of the R-node μ, the embedding $\mathcal{E}$ of $skel(\mu)$, splitting lines (dashed) for faces of $\mathcal{E}$, and the st-valuation χ of $\mathcal{E}^*$.

is just the restriction of y_ψ to the vertices from $skel(\mu)$, i.e., $\xi = y_\psi | V(skel(\mu))$. To define the st-valuation χ of $\mathcal{E}^*$ we use the following lemma.

Lemma 2.8 (Face Condition)**.**

1. *Let f be a face in $V(\mathcal{E}^*)$ different than t^*, and $v_0, v_1, \ldots, v_n$ be the right path of f. There is a vertical line $L_r(f)$ that contains the left endpoints of $\psi(v_1), \ldots, \psi(v_{n-1})$ and the left sides of $B_\psi(v_0, v_1), \ldots, B_\psi(v_{n-1}, v_n)$.*
2. *Let f be a face in $V(\mathcal{E}^*)$ different than s^*, and $u_0, u_1, \ldots, u_m$ be the left path of f. There is a vertical line $L_l(f)$ that contains the right endpoints of $\psi(u_1), \ldots, \psi(u_{m-1})$ and the right sides of $B_\psi(u_0, u_1), \ldots, B_\psi(u_{m-1}, u_m)$.*
3. *If f is an inner face of $\mathcal{E}$ then $L_l(f) = L_r(f)$.*

Proof. To prove (1) we first show that for every $i = 1, \ldots, n-1$, we have

$$l(\psi(v_i)) = l(B_\psi(v_{i-1}, v_i)).$$

By the Basic Tiling Lemma, $B_\psi(v_{i-1}, v_i)$ is a rectangle spanned between $\psi(v_{i-1})$ and $\psi(v_i)$. It follows that $l(\psi(v_i)) \leqslant l(B_\psi(v_{i-1}, v_i))$. Suppose that $l(\psi(v_i)) < l(B_\psi(v_{i-1}, v_i))$. By the Basic Tiling Lemma again, there is a child λ of μ such that the rectangle $B_\psi(\lambda)$ has its top right corner located at the intersection of the left side of $B_\psi(v_{i-1}, v_i)$ and $\psi(v_i)$. Clearly, λ corresponds to an edge of $skel(\mu)$ that is in the embedding $\mathcal{E}$ between (v_{i-1}, v_i) and (v_i, v_{i+1}) in the clockwise order around v_i. However, there is no such edge in $\mathcal{E}$, a contradiction. Similarly, for every $i = 1, \ldots, n-1$, we have

$$l(\psi(v_i)) = l(B_\psi(v_i, v_{i+1})).$$

It follows that the left sides of the bounding boxes $B_\psi(v_0, v_1), \ldots, B_\psi(v_{n-1}, v_n)$ and the left endpoints of $\psi(v_1), \ldots, \psi(v_{n-1})$ are aligned to the same vertical line $L_r(f)$.

The proof of (2) is analogous. Property (3) is an immediate consequence of the Basic Tiling Lemma.

□

The above lemma allows us to introduce the notion of a *splitting line* for every face f in $V(\mathcal{E}^*)$; namely, the splitting line of f is: the line $L_l(f) = L_r(f)$ if f is an inner face of $\mathcal{E}$, $L_r(f)$ if f is the left outer face of $\mathcal{E}$, and $L_l(f)$ if f is the right outer face of $\mathcal{E}$. Now, let $\chi(f)$ be the x-coordinate of the splitting line for a face f in $V(\mathcal{E}^*)$. To show that $\chi(f)$ is an st-valuation of $\mathcal{E}^*$, note that for any edge (f, g) of $\mathcal{E}^*$ there is an edge (u, v) of $\mathcal{E}$ that has f on the left side and g on the right side. It follows that $\chi(f) = l(B_\psi(u, v)) < r(B_\psi(u, v)) = \chi(g)$, proving the claim. See Figure 2.6 for an illustration.

The representation ψ of G determines the triple $(\mathcal{E}, \xi, \chi)$. Note that any other representation with the same tiling $\tau = (B_\psi(\mu_1), \ldots, B_\psi(\mu_k))$ of $B(\mu)$ gives the same triple. To emphasize that the triple $(\mathcal{E}, \xi, \chi)$ is determined by tiling τ, we write $(\mathcal{E}_\tau, \xi_\tau, \chi_\tau)$.

Now, assume that $\mathcal{E}$ is an st-embedding of $skel(\mu)$, ξ is an st-valuation of $\mathcal{E}$, and χ is an st-valuation of the dual of $\mathcal{E}$. Consider the function ϕ that assigns to every vertex v of $skel(\mu)$ the bar $\phi(v)$ defined as follows:

$$\begin{aligned} y_\phi(v) &= \xi(v), \\ l_\phi(v) &= \chi(\text{left face of } v), \\ r_\phi(v) &= \chi(\text{right face of } v). \end{aligned}$$

Firstly, Tamassia and Tollis [73] showed that ϕ is a bar visibility representation of $skel(\mu)$ and that for $\tau = (B_\phi(\mu_1), \ldots, B_\phi(\mu_k))$, we have $(\mathcal{E}_\tau, \xi_\tau, \chi_\tau) = (\mathcal{E}, \xi, \chi)$.

Secondly, there is a representation ψ of G that agrees with τ on $skel(\mu)$, i.e., such that $\tau = (B_\psi(\mu_1), \ldots, B_\psi(\mu_k))$. To construct such a representation, we take any representation ψ of G, translate and scale all bars in ψ to get $B_\psi(\mu) = B_\phi(\mu)$, and represent the pertinent digraphs $G_{\mu_1}, \ldots, G_{\mu_k}$ so that the bounding box of μ_i coincides with $B_\phi(\mu_i)$ for $i = 1, \ldots, k$.

The considerations given above lead us to the following lemma.

Lemma 2.9 (R-Tiling Lemma)**.** *Let μ be an R-node. There is a one-to-one correspondence between the set*

$$\mathcal{T} = \{(B_\psi(\mu_1), \ldots, B_\psi(\mu_k)) : \psi \text{ is a rectangular bar visibility representation of } G\}$$

of all possible tilings of the bounding box of μ by the bounding boxes of $\mu_1, \ldots, \mu_k$ in all representations of G and the set

$$\mathcal{T}' = \left\{ (\mathcal{E}, \xi, \chi) : \begin{array}{l} \mathcal{E} \text{ is an } st\text{-embedding of } skel(\mu), \\ \xi \text{ is an } st\text{-valuation of } \mathcal{E}, \\ \chi \text{ is an } st\text{-valuation of the dual of } \mathcal{E}. \end{array} \right\}$$

2.4.2 Algorithm for rectangular bar visibility extension of planar st-graphs

Let G be a planar st-graph with n vertices and ψ' be a partial representation of G with the set of *fixed* vertices V'. We present a quadratic-time algorithm that tests if there exists a rectangular bar visibility representation ψ of G that extends ψ'. If such a representation exists, then the algorithm can construct it in the same time.

In the first step, our algorithm calculates y_ψ. Namely, the algorithm checks whether $y_{\psi'} : V' \to \mathbb{R}$ is extendable to an st-valuation of G. When such an extension does not exist, the algorithm rejects the instance (G, ψ'); otherwise any extension of $y_{\psi'}$ can be used as y_ψ. The correctness of this step is verified by the following lemma.

Lemma 2.10. *Suppose that ψ is a rectangular bar visibility representation of G that extends ψ'.*

1. *The function y_ψ is an st-valuation of G that extends $y_{\psi'}$,*
2. *If y is an st-valuation of G that extends $y_{\psi'}$, then the function ϕ that sends every vertex v of G into a bar so that*

$$y_\phi(v) = y(v), \qquad l_\phi(v) = l_\psi(v), \qquad r_\phi(v) = r_\psi(v)$$

is also a rectangular bar visibility representation of G that extends ψ'.

Proof. The function y_ψ extends $y_{\psi'}$, because ψ extends ψ'. It is an st-valuation of G because for an edge (u, v) of G, the bar of u is below the bar of v.

For the proof of (2), observe that for each vertex u of G we have $X(\phi(u)) = X(\psi(u))$. We claim that for any two vertices u, v of G such that the interior of $X(\psi(u)) \cap X(\psi(v))$ is non-empty we have that $y_\psi(u) < y_\psi(v)$ if and only if $y(u) < y(v)$. For the proof of this claim, let u and v be vertices of G such that the interior of $X(\psi(u)) \cap X(\psi(v))$ is non-empty. From the fact that $\psi(V(G))$ is a collection of pairwise disjoint bars, it follows that $y_\psi(u) \neq y_\psi(v)$. Without loss of generality assume that $y_\psi(u) < y_\psi(v)$. The non-empty interior of $X(\psi(u)) \cap X(\psi(v))$ means that there is a path from u to v in G. Hence $y(u) < y(v)$ as y is an st-valuation of G.

As a consequence we have that $(x_1, x_2) \times (y_\psi(u), y_\psi(v))$ is a visibility gap between bars $\psi(u)$ and $\psi(v)$ in representation ψ if and only if $(x_1, x_2) \times (y(u), y(v))$ is a visibility gap between $\phi(u)$ and $\phi(v)$ and ϕ is a rectangular bar visibility representation of G. □

Clearly, checking whether $y_{\psi'}$ is extendable to an st-valuation of G, and constructing such an extension can be done in $O(n)$-time. In the second step, the algorithm computes the $SPQR$-tree T for G, which also takes linear time.

Before we describe the last step in our algorithm, we need some preparation. For an inner node μ in T we define the sets $V'(\mu)$ and $C(\mu)$ as follows:

$$\begin{aligned} V'(\mu) &= \text{the set of fixed vertices in } V(G_\mu) \setminus \{s_\mu, t_\mu\}, \\ C(\mu) &= \begin{cases} \emptyset, \text{ if } V'(\mu) = \emptyset; \\ \text{the smallest axis aligned, closed rectangle that contains } \psi'(u) \text{ for} \\ \text{all } u \in V'(\mu), \text{ otherwise.} \end{cases} \end{aligned}$$

The set $C(\mu)$ is called the *core* of μ. For a node μ whose core is empty, our algorithm can represent G_μ in any rectangle spanned between the poles of G_μ. Thus, we focus our attention on nodes whose core is non-empty.

Assume that μ is a node whose core is non-empty. We describe the 'possible shapes' the bounding box of μ might have in a representation of G that extends ψ'. The bounding box of μ is a rectangle that is spanned between the bars corresponding to the poles of G_μ. By the Basic Tiling Lemma, if $C(\mu)$ is non-empty then $B(\mu)$ contains $C(\mu)$. For

our algorithm it is important to distinguish whether the left (right) side of $B(\mu)$ contains the left (right) side of $C(\mu)$. This criterion leads to four types of representations of μ with respect to the core of μ.

The main idea of the algorithm is to decide for each inner node μ whose core is non-empty, which of the four types of representation of μ are possible and which are not. The algorithm traverses the tree bottom-up and for each node and each type of representation it tries to construct the appropriate tiling using the information about possible representations of its children. The types chosen for different children need to fit together to obtain a tiling of the parent node. In what follows, we present our approach in more detail.

Let μ be an inner node in T. Fix $\phi' = \psi'|V'(\mu)$. It is convenient to think of ϕ' as a partial representation of the pertinent digraph G_μ obtained by restricting ψ' to the inner vertices of G_μ. In particular, ϕ' is empty if the core of μ is empty. Let x, x' be two real values. A rectangular bar visibility representation ϕ of G_μ is called an $[x, x']$-*representation of* μ if ϕ extends ϕ' and $X(\phi(s_\mu)) = X(\phi(t_\mu)) = [x, x']$.

A node μ whose core is empty has an $[x, x']$-representation for any $x < x'$. In particular, a Q-node has an $[x, x']$-representation for any $x < x'$.

We say that an $[x, x']$-representation of an inner node μ whose core is non-empty is:

- *left-loose, right-loose* (*LL-representation*), when $x < l(C(\mu))$ and $x' > r(C(\mu))$,
- *left-loose, right-fixed* (*LF-representation*), when $x < l(C(\mu))$ and $x' = r(C(\mu))$,
- *left-fixed, right-loose* (*FL-representation*), when $x = l(C(\mu))$ and $x' > r(C(\mu))$,
- *left-fixed, right-fixed* (*FF-representation*), when $x = l(C(\mu))$ and $x' = r(C(\mu))$.

The next lemma justifies this categorization of representations. It says that if a representation of a given type exists, then every representation of the same type is also realizable.

Lemma 2.11 (Stretching Lemma)**.** *Let μ be an inner node whose core is non-empty. We have that:*

- *If μ has an LL-representation, then μ has an $[x, x']$-representation for any $x < l(C(\mu))$ and any $x' > r(C(\mu))$.*
- *If μ has an LF-representation, then μ has an $[x, x']$-representation for any $x < l(C(\mu))$ and $x' = r(C(\mu))$.*
- *If μ has an FL-representation, then μ has an $[x, x']$-representation for $x = l(C(\mu))$ and any $x' > r(C(\mu))$.*

Proof. Let $x_l = l(C(\mu))$. Suppose that ϕ is some left-loose $[x_1, x']$-representation of μ with $x_1 < x_l$. For any $x_2 < x_l$ we can obtain an $[x_2, x']$-representation of μ by appropriately stretching the part of the drawing of ϕ that is to the left of x_l. If the representation is right-loose, then we can arbitrarily stretch the part of the drawing that is to the right of $r(C(\mu))$. □

The main task of the algorithm is to verify which representations are feasible for nodes that have non-empty cores. We assume that:

- μ is an inner node whose core is non-empty.
- $\mu_1, \ldots, \mu_k$ are the children of μ, $k \geqslant 2$.
- $\lambda_1, \ldots, \lambda_{k'}$ are the children of μ with $C(\lambda_i) \neq \emptyset$, $0 \leqslant k' \leqslant k$.
- $\theta(\lambda_i)$ is the set of feasible types of representations for λ_i, $\theta(\lambda_i) \subseteq \{LL, LF, FL, FF\}$.

We process the tree bottom-up and assume that $\theta(\lambda_i)$ is already computed and non-empty.

Let x and x' be two real numbers such that $x \leqslant l(C(\mu))$ and $x' \geqslant r(C(\mu))$. We provide an algorithm that tests whether an $[x, x']$-representation of μ exists. We use it to find feasible types for μ by calling it four times with appropriate values of x and x'. While searching for an $[x, x']$-representation of μ our algorithm tries to tile the rectangle $[x, x'] \times [y(s_\mu), y(t_\mu)]$ with $B(\mu_1), \ldots, B(\mu_k)$. The tiling procedure is determined by the Tiling Lemma specific for the type of μ. Note that as the core of a Q-node is empty, the algorithm splits into three cases: μ is an S-node, a P-node, and an R-node.

Case S. μ is an S-node. In this case we attempt to align the left and right sides of the bounding boxes of the children of μ to x and x' respectively. We also must have the x-intervals of the bars of the cut vertices set to $[x, x']$.

Listing 2.1 Algorithm for series node

```
1  for each cut-vertex c in G_μ
2      if c ∈ V'(μ) and X(ψ'(c)) ≠ [x, x']
3          return FALSE      // fixed cut-vertex does not span [x, x']
4  for i = 1 to k'
5      if l(C(λ_i)) > x and r(C(λ_i)) < x' and LL ∉ θ(λ_i)
6          return FALSE      // λ_i must stretch on both sides
7      if l(C(λ_i)) > x and r(C(λ_i)) == x' and LF ∉ θ(λ_i)
8          return FALSE      // λ_i must stretch only on the left side
9      if l(C(λ_i)) == x and r(C(λ_i)) < x' and FL ∉ θ(λ_i)
10         return FALSE      // λ_i must stretch only on the right side
11     if l(C(λ_i)) == x and r(C(λ_i)) == x' and FF ∉ θ(λ_i)
12         return FALSE      // λ_i must stretch on neither side
13 return TRUE
```

Claim 2.12. *There exists an $[x, x']$-representation of an S-node μ if and only if Algorithm 2.1 returns* TRUE.

Proof. Claim follows directly from the Tiling Lemma for Series Nodes and the Stretching Lemma. □

Case P. μ is a P-node. In this case we attempt to tile the rectangle $[x, x'] \times [y(s_\mu), y(t_\mu)]$ by placing the bounding boxes of the children of μ side by side from left to right. The order of children whose cores are non-empty is determined by the position of those cores. We need to find enough space to place the bounding boxes of children whose

cores are empty. Additionally, if (s_μ, t_μ) is an edge of G, then we need to leave at least one visibility gap in the tiling for that edge. Otherwise, if (s_μ, t_μ) is not an edge of G, we need to close all the gaps in the tiling. The tiling algorithm is presented as Algorithm 2.2.

Listing 2.2 Algorithm for parallel node

```
1  sort λ_i's by the value l(C(λ_i))
2  for i = 1 to k'
3       l_i = l(C(λ_i)), r_i = r(C(λ_i))          // left- and right- endpoints of cores
4  r_0 = x, l_{k'+1} = x'
5  closed = ∅                                     // set of closed gaps
6  for i = 0 to k'
7       if r_i > l_{i+1}
8            return FALSE                         // cores overlap
9       if r_i == l_{i+1}
10           closed = closed ∪ {i}                // λ_i and λ_{i+1} touch
11           if i > 0
12                θ(λ_i) = θ(λ_i) ∖ {FL, LL}      // use right-fixed rep. of λ_i
13           if i < k'
14                θ(λ_{i+1}) = θ(λ_{i+1}) ∖ {LF, LL}   // use left-fixed rep. of λ_{i+1}
15 if (k > k' or (s_μ, t_μ) ∈ E(G_μ)) and |closed| == k' + 1
16      return FALSE                              // there is no gap
17 for i = 1 to k'
18      if θ(λ_i) == ∅
19           return FALSE
20      if LL ∈ θ(λ_i)
21           closed = closed ∪ {i − 1, i}         // close both gaps
22      else if i − 1 ∉ closed and LF ∈ θ(λ_i)
23           closed = closed ∪ {i − 1}            // close left gap
24      else if FL ∈ θ(λ_i)
25           closed = closed ∪ {i}                // close right gap
26 return (s_μ, t_μ) ∈ E(G_μ) or k − k' >= k' + 1 − |closed|   // can close all gaps
```

In line 1 the children whose cores are non-empty are sorted by the left end of the core. In lines 2 to 5 the variables l_i, r_i, and an empty set $closed$ are initialized.

If there are λ_i, λ_j such that the interior of the set $X(C(\lambda_i)) \cap X(C(\lambda_j))$ is non-empty, then we prove that there is no $[x, x']$-representation of G_μ. Indeed, by the Tiling Lemma for Parallel Nodes and by $C(\lambda_i) \subseteq B(\lambda_i)$, the interior of $B(\lambda_i) \cap B(\lambda_j)$ is non-empty and hence tiling of $B(\mu)$ with $B(\mu_1), \ldots, B(\mu_k)$ is not possible. Additionally, if $r(C(\lambda_i)) = l(C(\lambda_j))$, then neither a right-loose representation of λ_i nor a left-loose representation of λ_j can be used. These checks and restrictions are performed by the algorithm in lines 6 to 14.

Let $Q_i = [r_i, l_{i+1}] \times [y(s_\mu), y(t_\mu)]$ for $i \in [0, k']$. We say that Q_i is an *open gap* (after λ_i, before λ_{i+1}) if Q_i has non-empty interior. In particular, if $x = r_0 < l_1$ ($r_{k'} < l_{k'+1} = x'$) then there is an open gap before λ_1 (after $\lambda_{k'}$). On the one hand, if there is an edge (s_μ, t_μ) or there is at least one μ_i whose core is empty, then we need at least one open

gap to construct an $[x, x']$-representation. This condition is checked by the algorithm in line 15. On the other hand, if (s_μ, t_μ) is not an edge of G then we need to close all the gaps in the tiling. There are two ways to close the gaps. Firstly, the representation of each child node whose core is empty can be placed so that it closes a gap. The second way is to use loose representations for children nodes $\lambda_1, \ldots, \lambda_{k'}$.

If $\theta(\lambda_i) = \emptyset$ for some $i = 1, \ldots, k'$, then an $[x, x']$-representation of μ does not exist. Assume that $\theta(\lambda_i)$ is non-empty for every $i = 1, \ldots, k'$. Suppose that c is a function that assigns to every λ_i a feasible type of representation from the set $\theta(\lambda_i)$. Whenever $c(\lambda_i)$ is right-loose or $c(\lambda_{i+1})$ is left-loose, we can stretch the representation of λ_i or λ_{i+1}, so that it closes the gap Q_i. In lines 17 to 25, the algorithm greedily closes as many gaps as possible. The greedy strategy processes children λ_i's from left to right and for each child: closes both adjacent gaps if it can; it prefers closing the left gap if it is not yet closed (this is the last bounding box that can close this gap) from the right gap.

If there are some open gaps left and (s_μ, t_μ) is not an edge of G, then each open gap needs to be closed by placing in this gap a representation of one or more of the children whose core is empty.

Claim 2.13. *There exists an $[x, x']$-representation of a P-node μ if and only if Algorithm 2.2 returns* TRUE.

Proof. Claim follows by the Tiling Lemma for Parallel Nodes and the Stretching Lemma. The correctness of the greedy strategy follows by a simple greedy exchange argument. □

Case R. μ is an R-node. By the Tiling Lemma for Rigid Nodes, the set of possible tilings of $B(\mu)$ by $B(\mu_1), \ldots, B(\mu_k)$ is in correspondence with the triples $(\mathcal{E}, \xi, \chi)$, where $\mathcal{E}$ is a planar embedding of $skel(\mu)$, ξ is an st-valuation of $\mathcal{E}$, and χ is an st-valuation of $\mathcal{E}^*$. To find an appropriate tiling of $B(\mu)$ (that yields an $[x, x']$-representation of μ) we search through the set of such triples. Since μ is a rigid node, there are only two planar st-embeddings of $skel(\mu)$ and we consider both of them separately. Let $\mathcal{E}$ be one of these planar embeddings. Since the y-coordinate for each vertex of G is already fixed, the st-valuation ξ is given by the y-coordinates of the vertices from $skel(\mu)$. Now, it remains to find an st-valuation χ of $\mathcal{E}^*$, i.e., to determine the x-coordinate of the splitting line for every face f of $\mathcal{E}$. First, for every face f in $V(\mathcal{E}^*)$ we compute an initial set of possible placements for the splitting line of f by taking into account the partial representation ϕ'. If f is an inner face of $\mathcal{E}$, then we have the following restrictions on $\chi(f)$:

- If u is a fixed vertex on the left path of f, then $\chi(f) = r(u)$.
- If u is a fixed vertex on the right path of f, then $\chi(f) = l(u)$.
- If λ is a child of μ whose core is non-empty, and the virtual edge of λ is on the left path of f, then $\chi(f) \geqslant r(C(\lambda))$.
- If λ is a child of μ whose core is non-empty, and the virtual edge of λ is on the right path of f, then $\chi(f) \leqslant l(C(\lambda))$.

We impose analogous conditions for the faces s^* and t^*.

Let $\mathcal{X}'(f)$ be a set of all $\chi(f)$ in $[x, x']$ that satisfy all the above conditions. If $\mathcal{X}'(f) = \emptyset$ for some face f in $V(\mathcal{E}^*)$ or $x \notin \mathcal{X}'(s^*)$ or $x' \notin \mathcal{X}'(t^*)$, then there is no $[x, x']$-representation of G_μ. Since we are looking for an $[x, x']$-representation of G_μ, we set $\mathcal{X}'(s^*) = [x, x]$ and $\mathcal{X}'(t^*) = [x', x']$ as the splitting line for s^* (t^*) must be set to x (x').

Now, we further restrict the possible values for $\chi(f)$ by taking into account the fact that χ needs to be an st-valuation of $\mathcal{E}^*$. For every two faces f and g in $V(\mathcal{E}^*)$:

- If g is to the left of f, then $\chi(f) > l(\mathcal{X}'(g))$.
- If g is to the right of f, then $\chi(f) < r(\mathcal{X}'(g))$.

Let $\mathcal{X}(f)$ be the set of all $\chi(f)$ such that $\chi(f) \in \mathcal{X}'(f)$ and that satisfy the above conditions. If $\mathcal{X}(f)$ is empty for some face f in $V(\mathcal{E}^*)$, then there is no $[x, x']$-representation of G_μ. We assume that $\mathcal{X}(f)$ is non-empty for every f in $V(\mathcal{E}^*)$. One can easily verify the following claim.

Claim 2.14.

1. *For every face f in $V(\mathcal{E}^*)$, $\mathcal{X}(f)$ is an interval in $[x, x']$,*
2. *For every two faces f and g such that f is to the left of g, we have that:*
   - *$l(\mathcal{X}(f)) \leqslant l(\mathcal{X}(g))$ and if $l(\mathcal{X}(f)) = l(\mathcal{X}(g))$ then $\mathcal{X}(g)$ is open from the left side.*
   - *$r(\mathcal{X}(f)) \leqslant r(\mathcal{X}(g))$ and if $r(\mathcal{X}(f)) = r(\mathcal{X}(g))$ then $\mathcal{X}(f)$ is open from the right side.*

A face f in $V(\mathcal{E}^*)$ is *determined* if $\mathcal{X}(f)$ is a singleton (i.e., the location of the splitting line of f is already fixed); otherwise f is *undetermined*.

In what follows, we construct a 2-CNF formula Φ that is satisfiable if and only if an $[x, x']$-representation of μ exists.

Variables of Φ. For every child λ of μ whose core is non-empty, we introduce two Boolean variables: l_λ and r_λ, which have the following interpretation:

- The true (false) value of variable l_λ means that we use left-loose (left-fixed) representation of node λ.
- The true (false) value of variable r_λ means that we use right-loose (right-fixed) representation of node λ.

For every inner face f of $\mathcal{E}$ we introduce two Boolean variables: l_f and r_f, which have the following interpretation:

- The variable l_f is true when the splitting line of f is set strictly to the right of $l(\mathcal{X}(f))$. It is false when $\chi(f) = l(\mathcal{X}(f))$.
- The variable r_f is true when the splitting line of f is set strictly to the left of $r(\mathcal{X}(f))$. It is false when $\chi(f) = r(\mathcal{X}(f))$.

In particular, if $\mathcal{X}(f)$ is open from the left (right) then l_f (r_f) is true. When f is determined then both l_f and r_f are false.

For the left outer face s^* and the right outer face t^* we introduce variables r_{s^*} and l_{t^*}. Since s^* and t^* are determined, the corresponding variables are always set to false.

Clauses of Φ. We split the clauses of Φ into four types.

Type I. Clauses of this type propagate the information about the possible representation types of the children of μ.

- For every child λ with non-empty core and for every type of representation of λ which is not feasible, we add a clause that forbids using representation of this type. For example, when there is no left-loose, right-fixed representation of λ we add a clause $\neg(l_\lambda \wedge \neg r_\lambda) = (\neg l_\lambda \vee r_\lambda)$ to Φ.

Type II. Clauses of this type enforce the meaning of variables l_f and r_f for every face f in $V(\mathcal{E}^*)$.

- For every determined inner face f, we add the clauses $(\neg l_f)$ and $(\neg r_f)$, and for s^* and t^* we add the clauses $(\neg r_{s^*})$ and $(\neg l_{t^*})$.
- For every undetermined inner face f, we add the clause (l_f) if $\mathcal{X}(f)$ is open from the left side, and (r_f) if $\mathcal{X}(f)$ is open from the right side. If $\mathcal{X}(f)$ is closed from the left and closed from the right side, we add the clause $(l_f \vee r_f)$ as the splitting line of f cannot be placed in both endpoints of $\mathcal{X}(f)$ simultaneously.

Type III. Clauses of this type enforce a 'proper tiling' of every face f in $V(\mathcal{E}^*)$ (see Face Condition Lemma). This ensures that the bounding boxes associated with the left path and the right path of f can be aligned to the splitting line of f.

- For every face f and for every node λ on the left path of f with a non-empty core:
  - We add the clause $(l_f \Rightarrow r_\lambda)$.
  - If $r(C(\lambda)) < l(\mathcal{X}(f))$, we add the clause (r_λ).
  - If $r(C(\lambda)) = l(\mathcal{X}(f))$, we add the clause $(\neg l_f \Rightarrow \neg r_\lambda)$.

The clause $(l_f \Rightarrow r_\lambda)$ asserts that whenever the splitting line of f is set to the right of $l(\mathcal{X}(f))$, then a right-loose representation of λ is necessary to align the bounding box of λ to the splitting line of f. The two remaining clauses have similar meaning.

We add analogous clauses for the nodes whose cores are non-empty and that correspond to the edges from the right path of f.

Type IV. Clauses of this type enforce that the x-coordinates of the splitting lines form an st-valuation of $\mathcal{E}^*$.

- For every pair of faces f and g in $V(\mathcal{E}^*)$ such that f is to the left of g and such that $r(\mathcal{X}(f)) \geqslant l(\mathcal{X}(g))$, we add the clause $(\neg r_f \Rightarrow l_g)$.

Such a clause forbids setting $\chi(f) = r(\mathcal{X}(f))$ and $\chi(g) = l(\mathcal{X}(g))$ – such an assignment of $\chi(f)$ and $\chi(g)$ would not be a valid st-valuation.

Claim 2.15. *Let μ be an R-node, $\mathcal{E}$ be a planar embedding of the skeleton of μ. There exists an $[x, x']$-representation of μ that corresponds to a planar embedding $\mathcal{E}$ if and only if Φ is satisfiable.*

Proof. Suppose that ϕ is an $[x, x']$-representation of μ. For every face f in $V(\mathcal{E}^*)$, the splitting line $\chi(f)$ of f in ϕ satisfies $\chi(f) \in \mathcal{X}(f)$. Thus, $\chi(f)$ determines the following assignment for l_f and r_f:

- l_f is true if and only if $\chi(f) > l(\mathcal{X}(f))$,
- r_f is true if and only if $\chi(f) < r(\mathcal{X}(f))$.

For every child λ of μ such that $C(\lambda) \neq \emptyset$ we set the variables l_λ and r_λ as follows:

- l_λ is true if and only if λ is left-loose in ϕ,
- r_λ is true if and only if λ is right-loose in ϕ.

One can easily check that this assignment satisfies Φ.

Suppose now that Φ is satisfiable. We define an $[x, x']$-representation ϕ of μ by setting a splitting line $\chi(f)$ for every face f in $V(\mathcal{E}^*)$. To conclude that ϕ is an $[x, x']$-representation it is enough to check that:

1. The function χ is an st-valuation of $\mathcal{E}^*$.
2. For every fixed vertex u we have that
$$l_\phi(u) = \chi(\text{left face of } u) \text{ and } r_\phi(u) = \chi(\text{right face of } u).$$
3. For every child λ of μ such that $C(\lambda) \neq \emptyset$ we have that
$$G_\lambda \text{ has an } [\chi(\text{left face of } \lambda), \chi(\text{right face of } \lambda)]\text{-representation.}$$

First, we define $\chi(f) = l(\mathcal{X}(f))$ when l_f is false. We also set $\chi(f) = r(\mathcal{X}(f))$ when r_f is false. Note that this definition is unambiguous as both l_f and r_f are false only for a determined face f by the fact that the clauses of Type II are satisfied. By Claim 2.14 and by the fact that the clauses of Type IV are satisfied, for any two faces f and g in $V(\mathcal{E}^*)$ for which $\chi(f)$ and $\chi(g)$ are already fixed, we have $\chi(f) < \chi(g)$ whenever f is to the left of g. Notice that $\chi(f)$ is not yet determined for inner faces f for which l_f and r_f are true. For such a face f, let $\mathcal{X}''(f)$ contain all values z such that:

- $\chi(g) < z$ whenever g is a face to the left of f and the value $\chi(g)$ is already fixed, and
- $z < \chi(h)$ whenever h is a face to the right of f and the value $\chi(h)$ is already fixed, and
- $l(\mathcal{X}(f)) < z < r(\mathcal{X}(f))$.

We claim that $\mathcal{X}''(f)$ is an open, non-empty interval. Indeed, if $\mathcal{X}''(f)$ is empty, then there are faces g and h with the values $\chi(g), \chi(h)$ fixed such that g is to the left of h and $\chi(g) \geqslant \chi(h)$, which contradicts our previous observation that if a face is to the left of another face and splitting lines are determined for both of them, then the splitting line of the first face is to the left of the splitting line of the second face.

Moreover, for any two faces f_1, f_2 such that f_1 is to the left of f_2 and neither $\chi(f_1)$ nor $\chi(f_2)$ is fixed, we have that $l(\mathcal{X}''(f_1)) \leqslant l(\mathcal{X}''(f_2))$ and $r(\mathcal{X}''(f_1)) \leqslant r(\mathcal{X}''(f_2))$. Thus, for every face f for which both l_f, r_f are true, we can choose a value $\chi(f)$ from $\mathcal{X}''(f)$ so that χ is an st-valuation of $\mathcal{E}^*$. We need to check the remaining conditions (2) and (3). Condition (2) is satisfied since, for a determined face f we have chosen $\chi(f)$ from the singleton $\mathcal{X}(f)$. Condition (3) follows from the fact that the clauses of Type I and Type III are satisfied, and by the Stretching Lemma. □

2.4.2.1 Complexity considerations

To compute the feasible representation for a node μ with k children, our algorithm works in $O(k)$-time if μ is an S-node. Algorithm 2.2 for a P-node μ needs to sort the children of μ and thus, it works in $O(k \log k)$-time. For an R-node, the number of clauses of Types I, II and III is $O(k)$. The number of clauses of Type IV is $O(k^2)$ and for some graphs, it is quadratic. Thus, the algorithm works in $O(k^2)$ time for an R-node. Since the number of all edges in all nodes of T is $O(n)$, the whole algorithm works in $O(n^2)$ time.

2.4.3 Faster algorithm

The bottleneck of the algorithm presented in Section 2.4.2 is the number of clauses of Type IV in the 2-CNF formula constructed for R-nodes. In the presented algorithm we add one clause $(\neg r_f \Rightarrow l_g)$ for any two faces f and g in $V(\mathcal{E}^*)$ such that f is to the left of g and $r(\mathcal{X}(f)) \geqslant l(\mathcal{X}(g))$. The number of such pairs of faces can be quadratic. In this section we present a different, less direct, approach that uses a smaller number of clauses to express the same set of constraints.

We can treat the planar st-graph $\mathcal{E}^*$ as a planar poset with a single minimal and a single maximal element. Using the result by Baker, Fishburn and Roberts [6] we know that such a poset has dimension at most 2. Thus, there are two numberings p and q of the vertices of $\mathcal{E}^*$ such that a face f is to the left of a face g if and only if $p(f) < p(g)$ and $q(f) < q(g)$. Such numberings correspond to dominance drawings of planar st-graphs and can be computed in linear time [29].

For each face f, we have two Boolean variables l_f and r_f, two real values $\lambda_f = l(\mathcal{X}(f))$, $\varrho_f = r(\mathcal{X}(f))$, and two integer values $p_f = p(f)$ and $q_f = q(f)$. We want to introduce a small set of 2-CNF clauses that implies $(\neg l_g \Rightarrow r_f)$ whenever $p_f < p_g$, $q_f < q_g$, and $\varrho_f \geqslant \lambda_g$.

To give an intuition for our approach, consider the simpler problem of determining for every face f the number of faces g such that $p_f < p_g$, $q_f < q_g$, and $\varrho_f \geqslant \lambda_g$. This is a three-dimensional range counting query problem and can be solved in $O\big(n \log^3 n\big)$ time using range trees, as described in Chapter 5 of [7].

It can also be solved in $O\big(n \log^2 n\big)$ time using the following sweep line algorithm. First, we setup a dynamic data structure for two-dimensional range counting queries.

Then, we process faces, one by one, in order of increasing values of p. After processing each face h, we add the point (q_h, ϱ_h) to the structure. To compute the answer for a face h, we ask the structure for the number of points (x, y) such that $x < p_h$ and $y \geqslant \lambda_h$.

Now we give an overview of our approach to the original problem of creating a small set of 2-CNF clauses. Our algorithm is a sweep line algorithm that resembles the one described above. In particular, we use a data structure similar to a two-dimensional range tree, that is simply a set of *persistent* balanced binary search trees, each with ϱ_f as the sorting key. During the course of the sweep, we create additional Boolean variables corresponding to the vertices of the trees and implications corresponding to their edges. The algorithm executes $O(n)$ queries against the structure, each of the queries takes $O\big(\log^2 n\big)$ time. As a result, both the maximal size of the structure and the total running time amount to $O\big(n \log^2 n\big)$. The algorithm produces $O\big(n \log^2 n\big)$ 2-CNF clauses that correspond to the edges of the search trees and the control flow of the queries.

For an overview of persistent data structures, refer to [31]. However, we only need the ideas presented in [64], which are summarized in this paragraph. The tree structure used in our algorithm is a modification of the AVL tree. A node α of the tree stores a pointer to its left child $left(\alpha)$, a pointer to its right child $right(\alpha)$ and the sorting key $key(\alpha)$. The parent links are purposefully not stored – AVL trees can easily be implemented without them and we must not store them for the persistent approach to work. The difference from regular AVL trees is that no node is ever modified. Let A be the set of all vertices that the insertion procedure would modify, together with all of their ancestors. It is a known property of AVL trees that $|A|$ is logarithmic in the number of vertices of the tree. In a persistent AVL tree, instead of modifying nodes in A, we perform the modifications on their copies – we create a new node $C(\alpha)$ for every $\alpha \in A$ and set $key(C(\alpha)) = key(\alpha)$.

This way, each addition to the tree introduces a logarithmic number of new nodes. After each addition, we get a new root node, that represents the new tree, the old tree is represented by the previous root and all but a logarithmic number of the nodes are shared by both trees. The graph of old and new nodes together with edges from nodes to their children is an acyclic digraph.

Now, in our algorithm, each tree keeps some set of Boolean variables r_f sorted by the value of ϱ_f. More specifically, with every vertex α we associate a Boolean variable $var(\alpha)$. To add a variable r_f into the tree we add a new node α with $key(\alpha) = \varrho_f$ and $var(\alpha) = r_f$. Additionally, with each node α of the tree we associate a second Boolean variable $var'(\alpha)$ and for each child node β we add a clause $(var'(\alpha) \Rightarrow var'(\beta))$. Finally, for each node α we add a clause $(var'(\alpha) \Rightarrow var(\alpha))$.

To simplify the presentation, assume that we have $n = 2^k$ faces. Let numberings p and q take values $0, \ldots, n-1$. We construct the 2-CNF formula in the following way. First, for each interval of integers $[j \cdot 2^i, (j+1) \cdot 2^i), 0 \leqslant i \leqslant k, j < \frac{n}{2^i}$ we have one persistent balanced binary search tree. The tree for the interval $[a, b)$ is going to keep variables r_f for faces f such that $q_f \in [a, b)$.

We process faces, one by one, in order of increasing values of p. After processing each face f, we add the variable r_f to all trees $[a, b)$ such that $q_f \in [a, b)$. There are $k+1$ such trees and an addition to each tree takes $O(\log n)$ time and introduces $O(\log n)$ new Boolean variables.

This way, when we process face g, then any earlier processed face f satisfies $p_f < p_g$ and no other face satisfies this condition. Now, for any value q_g we can select a logarithmic size subset S of the trees such that the union of the intervals of the trees is exactly $[0, q_g)$. The variables r_f stored in these trees are exactly the variables for faces that satisfy both $p_f < p_g$ and $q_f < q_g$. This is exactly the set of faces that are to the left of the face g.

Each tree in S stores variables r_f sorted by ϱ_f. We can execute a binary search for the left-most node with the key no smaller than λ_g. During the search, when we descend from an inner node α to the left child or when α is the final node of the search, we add clauses $(\neg l_g \Rightarrow var'(right(\alpha))$ and $(\neg l_g \Rightarrow var(\alpha))$. The first implication is forwarded over the tree to all nodes in the right subtree. This way, after completing the search, we have that $\neg l_g$ implies r_f for all faces f such that $p_f < p_g$, $q_f < q_g$ and $\varrho_f \geqslant \lambda_g$ – exactly as intended.

The total running time of this procedure is $O\big(n \log^2 n\big)$ and it produces at most that many variables and clauses. This leads to the following.

Theorem 2.3. *The Rectangular Bar Visibility Representation Extension Problem for a planar st-graph with n vertices can be solved in $O\big(n \log^2 n\big)$ time.*

2.5 Hardness results

In this section we show two hardness results. In the first subsection we show that the bar visibility extension problem for planar graphs is **NP**-complete. Then, we show that the bar visibility extension problem for directed graphs is **NP**-complete when restricted to grid representations.

2.5.1 Representations of undirected graphs

Theorem 2.1. *The Bar Visibility Representation Extension Problem is* **NP***-complete.*

Proof. It is clear that the bar visibility representation extension problem is in **NP**. To prove completeness, we present a reduction from PLANARMONOTONE3SAT. Given a formula ϕ we construct a graph G, a subset V' of vertices of G, and a representation ψ' of V' that is extendable to a representation of the whole G if and only if ϕ is satisfiable. The vertex v is *fixed* when $v \in V'$, otherwise v is *unrepresented*. The reduction constructs a planar Boolean circuit that simulates the formula ϕ. The bars assigned to fixed vertices of G create wires and gates of the circuit. Unrepresented vertices of G correspond to Boolean values transmitted over the wires. Our construction uses several Boolean gates: a NOT gate, a XOR gate, a special gate which we call a CXOR gate, and an OR gate.

In the figures illustrating this proof, the red bars denote the fixed vertices of G and the black bars denote the unrepresented vertices. A bar may have its left (right) endpoint marked or not depending on whether the bar extends to the left (right) of the figure or not. The figures also contain some vertical ranges. These ranges are only required for the description of the properties of the gadgets.

For readability, the figures contain only schemes of previously defined gadgets. Whenever a scheme appears in a figure, its area is colored gray.

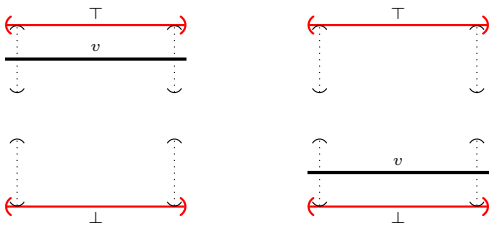


Figure 2.7: Wire transmitting true value (on the left) and false value (on the right)

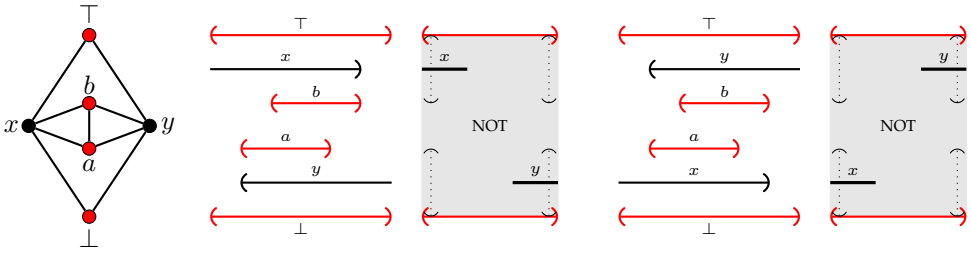


Figure 2.8: The NOT gadget depicted by its two possible representations and their schemes

Each wire, see Figure 2.7, in the circuit is an empty space between two fixed vertices, $\top$ and $\bot$, whose bars are placed one above the other. One unrepresented vertex v, that is adjacent to both $\top$ and $\bot$, corresponds to the value transmitted over the wire. The construction of the gates ensures that in each wire there are exactly two disjoint rectangular regions and v needs to be placed in one of them. In the figures, the small horizontal lines with a dotted line between them are used to mark those regions and are not part of the construction. Placement of v anywhere in the top (bottom) region corresponds to a transmission of a true (false) value. We show a collection of gadgets for the gates that use such wires as inputs, and outputs. Similarly to wires, each gate is bounded from the top and bottom by two bars. This way, it is easy to control the visibility between bars from different gates and wires.

NOT Gadget. Figure 2.8 presents a NOT gate and its scheme. An unrepresented vertex x (y) can transmit the value in the wire that can be placed to the left (right) from the gate. Both bars a and b are adjacent to each other, adjacent to vertices x and y, and not adjacent to the bounding bars. As a and b don't have any other neighbors, the only way to obstruct the visibility gap between a, b and bounding bars is to use x and y. Thus, one of x or y is placed below a and b, and the other is placed above a and b. This way we obtain a desired functionality of a NOT gate. As the visibility between every two bars does not change across the two representations, the corresponding edges of G are well defined.

XOR Gadget. Figure 2.9 presents a XOR gate. It checks that the inputs x_1 and x_2 have different Boolean values. It also produces outputs y_1 and y_2. The partial representation is extendable if and only if $x_1 = \neg x_2 = \neg y_1 = y_2$.

To see that, observe that the visibility gap between b and $\top$ needs to be obstructed and b has only two neighbors x_1 and y_1. Assume that y_1 blocks the visibility between b and $\top$. Now x_1 needs to block the visibility between b and the other bars. Thus, x_1

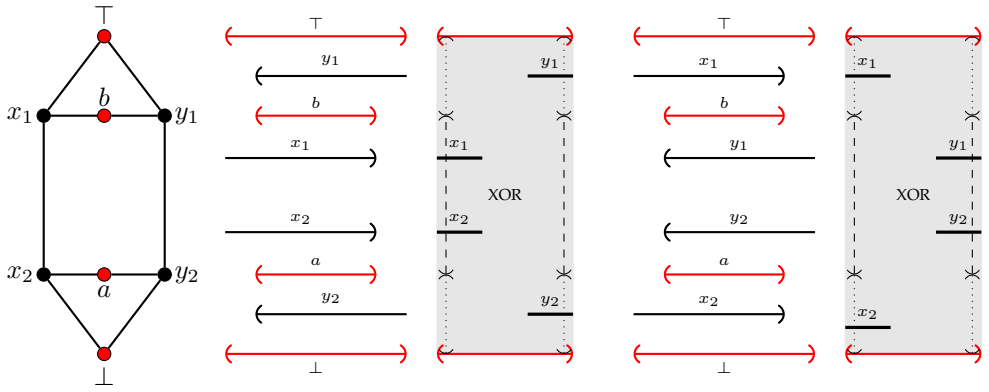


Figure 2.9: The XOR gadget depicted by its two possible representations and their schemes.

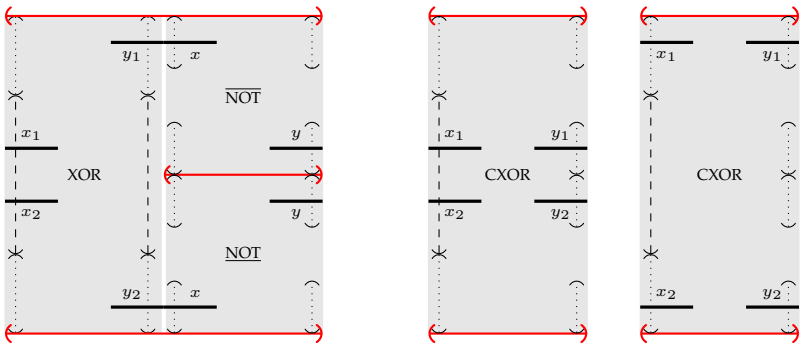


Figure 2.10: The CXOR gadget depicted by the schemes for its two possible representations.

is placed below b. The only other neighbor of x_1 is x_2 and thus, it needs to go above a. The last unrepresented vertex is y_2 and it needs to obstruct the visibility gap between a and $\bot$.

The other possibility is that x_1 blocks visibility between b and $\top$. The analysis of this case is symmetric to the previous one and gives the second possible valuation of the variables.

Finally, note that the visibility between every two bars does not change across the two representations and the corresponding edges of G are well defined.

CXOR Gadget. Figure 2.10 presents a CXOR circuit. It checks that the inputs x_1 and x_2 have different Boolean values and produces copies y_1 and y_2 of the inputs. The CXOR construction combines the XOR gate and two NOT gates in order to obtain a circuit that checks that $x_1 = \neg x_2$, $y_1 = x_1$ and $y_2 = x_2$.

Variable Gadget. Using NOT gates and CXOR circuits it is easy to construct a variable gadget. Figure 2.11 presents a gadget that gives two wires with value x to the right side, and two wires with value $\neg x$ to the left side. If we need k copies of a variable, we simply stack $2k - 1$ NOT gates one on another, and add CXOR gates to check the consistency of the outputs produced by every second NOT gate.

All we need to finish the construction of our building blocks is a clause gadget that checks that at least one of three wires connected to it transmits a true value.

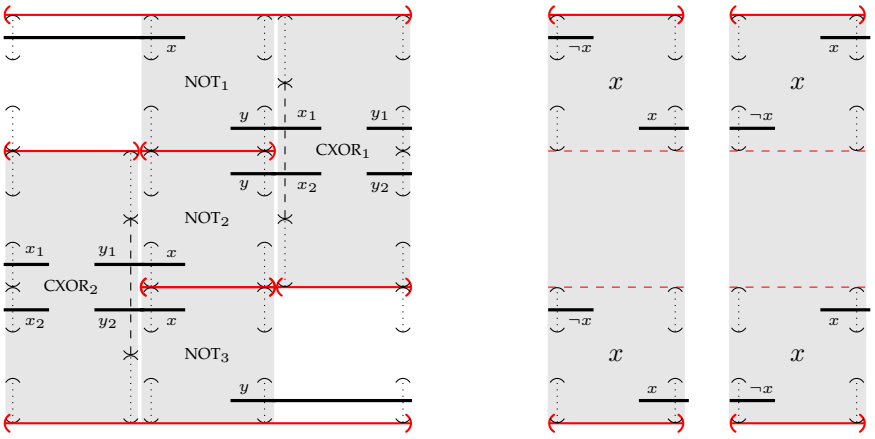


Figure 2.11: On the left: the gadget for the variable x with two output slots and one of its possible representation for the false value of x. On the right: schemes of the gadget for the false and the true value of x, respectively.

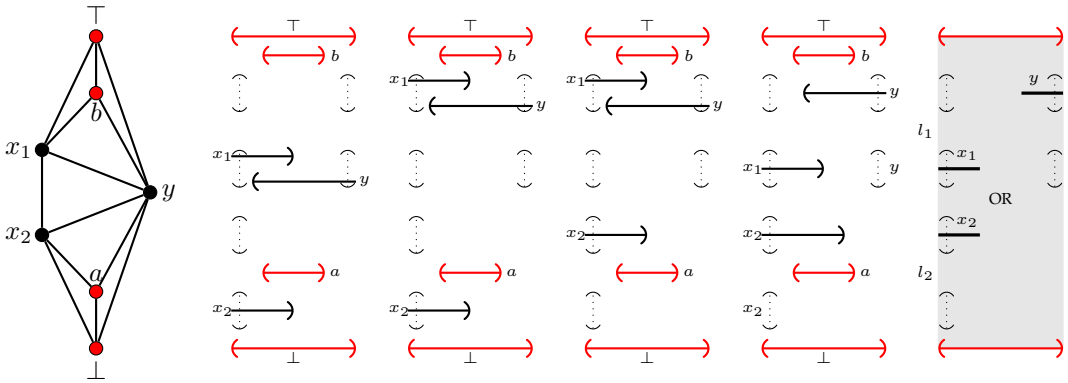


Figure 2.12: The OR gadget.

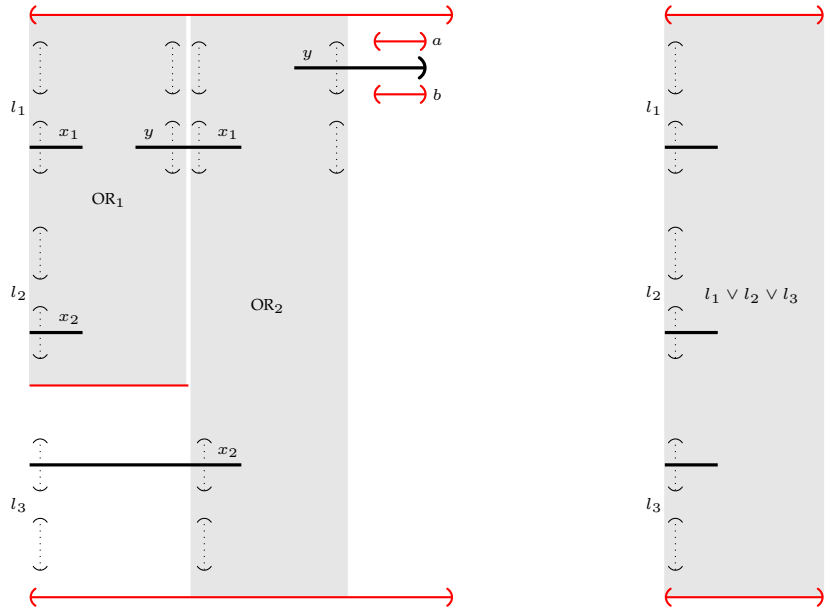


Figure 2.13: The construction for a clause $l_1 \vee l_2 \vee l_3$ and its representation for the false value of l_1 and l_2 and the true value of l_3.

OR Gadget and Clause Gadget. Figure 2.12 presents an OR gate that has two inputs x_1 and x_2 and one output y.

The output value can be true only if at least one of the inputs is true. In each of these three scenarios y can be represented in the higher of its regions. See Figure 2.12 for these three representations. Now, consider a representation of the gadget where both x_1 and x_2 are false. Observe that x_1 and b are neighbors and we have that $r(x_1) > l(b) = l(a)$. Vertices x_1 and a are not adjacent and x_2 is represented below a. The only other bar that can block the visibility gap between a and x_1 is y. Therefore, y is placed in the lower of its regions, as it needs to be below x_1.

Combining two OR gates and two bars that ensure that the output of the second gate is true, we get a clause gadget presented in Figure 2.13.

Given an instance ϕ of PLANARMONOTONE3SAT (together with a rectilinear planar representation of ϕ), we show how we construct the graph G with a partial representation ψ'. We rotate the rectilinear representation by 90 degrees. Now, we replace vertical segments representing variables of ϕ with variable gadgets and vertical segments representing clauses of ϕ with clause gadgets. Finally, for each occurrence of variable x in clause C, we replace the horizontal connection between the segment of x and the segment of C by a wire connecting the appropriate gadgets. The properties of our gadgets assert that ϕ is satisfiable if and only if ψ' is extendable to a bar visibility representation of G. □

2.5.2 Grid representations

In this section we consider the following problem:

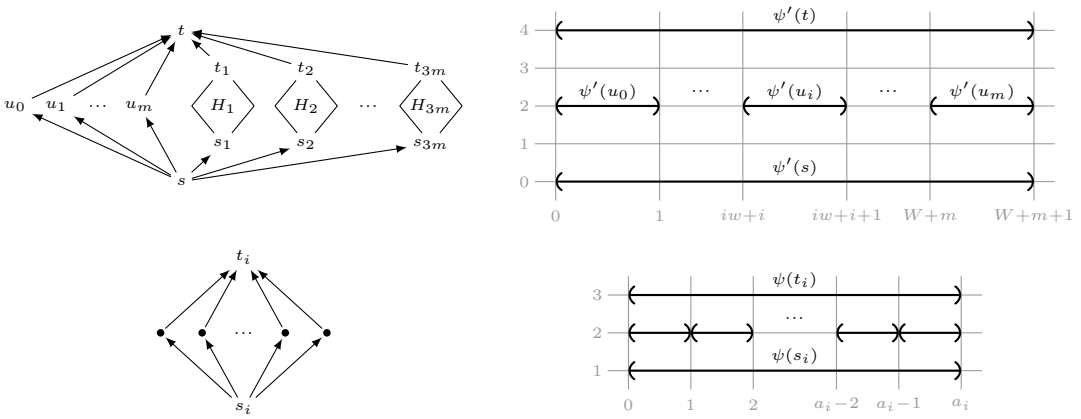


Figure 2.14: The graph G, partial representation ψ', graph H_i, and the representation ψ of H_i with minimum width.

Grid Bar Visibility Representation Extension for directed graphs:

Input: (G, ψ'), where G is a directed graph and ψ' is a mapping assigning bars to some subset V' of $V(G)$.
Question: Does G admit a grid bar visibility representation ψ with $\psi|V' = \psi'$?

In what follows we show that the above problem is **NP**-complete. The proof is generic and can be easily modified to work for other grid representations including undirected, rectangular directed/undirected, and even other models of visibility.

Theorem 2.16. *Grid Bar Visibility Representation Extension Problem is* **NP***-complete.*

Proof. We use the 3PARTITION problem to show **NP**-hardness. Consider an instance w, a_1, $\ldots$, a_{3m} of the 3PARTITION problem. Let $W = \sum_{j=1}^{3m} a_j$, i.e., $W = mw$. From this instance of 3PARTITION, we create a graph G and a partial representation ψ' that assigns bars to a subset V' of $V(G)$. These are constructed as follows and depicted in Figure 2.14.

The graph G is constructed as follows. We start with G' which is a $K_{2,m+1}$ with source s and sink t as the two vertices of degree $m+1$ and the other vertices are labeled $u_0, \ldots, u_m$. Now, for each $i = 1, \ldots, 3m$, create a planar st-graph H_i which is a K_{2,a_i} with source s_i and sink t_i as its two vertices of degree a_i. We remark that the width of any visibility representation of H_i in the integer grid is at least a_i. Finally, the graph G is obtained by attaching each H_i to G' by adding the edges (s, s_i) and (t_i, t).

For the fixed bars, we let $V' = \{s, t, u_1, \ldots, u_{m+1}\}$ and we define a bar $\psi'(v)$ for each element $v \in V'$.

$$y_{\psi'}(v) = \begin{cases} 0 & v = s \\ 2 & v = u_i \\ 4 & v = t \end{cases} \qquad l_{\psi'}(v) = \begin{cases} 0 & v = s, t \\ iw + i & v = u_i \end{cases}$$

$$r_{\psi'}(v) = \begin{cases} W + m + 1 & v = s, t \\ iw + i + 1 & v = u_i \end{cases}$$

Observe that the distance between the right-end of $\psi'(u_i)$ and the left-end of $\psi'(u_{i+1})$ is exactly w.

We now claim that there is a solution for input (G, ψ') if and only if the 3PARTITION instance $\{w, a_1, \ldots, a_{3m}\}$ has a solution. First, we consider a collection of triples $T_1, \ldots, T_m$ which form a solution of the 3PARTITION problem. We extend ψ' to a visibility representation of G where, for each triple $T_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, we place visibility representations of H_{j_1}, H_{j_2}, H_{j_3} in sequence left-to-right and between the fixed bars $\psi'(u_{j-1})$ and $\psi'(u_j)$. Notice that this is possible since $a_{j_1} + a_{j_2} + a_{j_3} = w$ and the distance from the right-end of $\psi'(u_{j-1})$ to the left-end of $\psi'(u_j)$ is w.

To show that any visibility representation ψ of G which extends ψ' provides a solution to the 3PARTITION problem we make the following observations. First, due to the the placement of the bars $\psi'(s), \psi'(t), \psi'(u_0)$ and $\psi'(u_m)$, the outer face of the resulting embedding is s, u_0, t, u_m. In particular, the bars of every H_i occur strictly within the rectangle enclosed by $\psi'(s)$ and $\psi'(t)$. Thus, each H_i must also be drawn between some pair of bars $\psi'(u_{j-1})$, $\psi'(u_j)$ (for some $j \in \{1, \ldots, m-1\}$). Second, due to each a_i being between $\frac{w}{4}$ and $\frac{w}{2}$ and the width of a visibility representation of H_i being at least a_i, at most three H_i's 'fit' between the fixed bars $\psi'(u_{j-1})$ and $\psi'(u_j)$. Thus, since there are $3m$ H_i's, every $\psi'(u_{j-1})$, $\psi'(u_j)$ has exactly three H_i's between them. Moreover, if H_{i_1}, H_{i_2} and H_{i_3} are placed between $\psi'(u_{j-1})$ and $\psi'(u_j)$, then $a_{i_1} + a_{i_2} + a_{i_3} \leqslant w$. Thus, in ψ, the gaps between each pair $\psi'(u_{j-1})$, $\psi'(u_j)$ must contain precisely three H_i's whose sum of corresponding a_i's is w, i.e., the gaps correspond to the triples of a solution of the 3PARTITION problem. □

2.6 Open problems

The main problem left open is to decide whether there exists a polynomial-time algorithm that checks whether a partial representation of a directed planar graph is extendable to a bar visibility representation of the whole graph. Although we showed an efficient algorithm for the case of planar st-graphs, it seems that some additional ideas are needed to resolve this problem in general.

Some further open problems concern extension problems for the weak and strong visibility models.

2.6.1 Weak visibility

Tamassia and Tollis [73] showed that every planar graph admits a weak visibility representation. Nevertheless, the problem of extending a partial representation of a planar graph to a weak visibility representation is **NP**-complete [17].

Di Battista and Tamassia [27] showed that a directed planar graph G admits a weak visibility representation if and only if G admits an upward planar drawing. The latter problem is **NP**-complete [42], so the problem of extending partial weak visibility representations for planar digraphs is also **NP**-complete. Nevertheless, we do not know whether there is an efficient algorithm if we assume that an upward planar drawing of a planar digraph is given on the input.

2.6.2 Strong visibility

Due to Andreae [4], the recognition of planar graphs that admit a strong visibility representation is **NP**-complete. It follows that the problem of testing whether a partial

representation of a planar graph is extendable to a strong visibility representation is also **NP**-complete. Nevertheless, we do not know if there exists an efficient algorithm that tests whether a partial representation of a planar digraph is extendable to a strong representation of the whole graph. It seems that some of our results on the bar visibility model can be adjusted to the strong visibility model.

3

Complexity of minimizing the number of intersecting edges in perfect bipartite matchings

In 2014, Yamanaka et al. [76] introduced TOKEN SWAPPING – a problem, in which we are given a graph, whose every vertex contains a token. Each token has a different destination vertex. In one turn, it is allowed to swap tokens on two adjacent vertices. The goal is to bring all tokens to their destinations in the smallest number of turns.

Two generalizations of the problem were also considered. In the first one, presented in [76] and called COLORED TOKEN SWAPPING, tokens and vertices have colors and the goal is to finally move each token to a vertex of the same color. Note that TOKEN SWAPPING corresponds to COLORED TOKEN SWAPPING, where for each used color there is exactly one vertex and exactly one token of this color. In the second generalization, introduced by Bonnet, Miltzow and Rzążewski [12] and called SUBSET TOKEN SWAPPING, every token is assigned a set of destination vertices and is required to reach any of them. Note here that COLORED TOKEN SWAPPING is a special version of SUBSET TOKEN SWAPPING with sets assigned to different tokens either equal or disjoint.

The decision version of the TOKEN SWAPPING problem was shown to be **NP**-complete by Miltzow, Narins, Okamoto, Rote, Thomas and Uno [62]. This means that both its generalizations are **NP**-complete as well. A precise analysis of the complexity of all three token swapping problems on special graphs has been done in [12]. In particular, SUBSET TOKEN SWAPPING was shown to be **NP**-complete on trees, cliques and stars. However, in the first draft of [12][1] its status has been left completely open on paths. The final version of [12] refers to our proof of **NP**-completeness of SUBSET TOKEN SWAPPING on paths, first published in [45][2].

In case of paths, SUBSET TOKEN SWAPPING has been equivalently restated by Miltzow [61] to be read:

CROSSING-MINIMIZING MATCHING:

Input:
- a bipartite graph $G = (V_1, V_2, E(G))$ that admits a perfect matching,
- a linear order $<_1$ of V_1,
- a linear order $<_2$ of V_2, and
- a nonnegative integer k.

Question: Does G admit a perfect matching $M \subseteq E(G)$ with at most k *crossings*, i.e. pairs of edges $\{u_1, u_2\}, \{v_1, v_2\} \in M$, such that $u_1 <_1 v_1$ and $v_2 <_2 u_2$?

[1] See [11].

[2] Next, this **NP**-completeness proof, together with parametrized algorithms in this and other settings, was included in [1].

For any perfect matching M in G (with fixed orders $<_1$ and $<_2$), we denote the number of crossings in M by $\#\mathsf{cr}(M)$.

The proof of **NP**-completeness for CROSSING-MINIMIZING MATCHING (and, in consequence, of SUBSET TOKEN SWAPPING restricted to paths) is presented both in our paper [45] as well as in this chapter.

We interpret the orders $<_1$ and $<_2$ by placing the vertices of $V(G) = V_1 \cup V_2$ on two different horizontal lines in the plane: vertices in V_1 and V_2 are represented by different points respectively in the upper and lower line, and are ordered by $<_1$ and $<_2$ from the left to the right. Each edge is represented by a corresponding line segment. An example of this geometric interpretation is given in Figure 3.1.

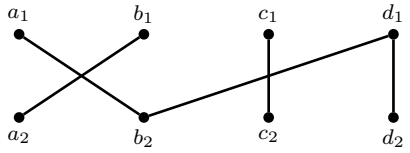


Figure 3.1: A bipartite graph G with $V_1 = \{a_1, b_1, c_1, d_1\}$, $V_2 = \{a_2, b_2, c_2, d_2\}$, edges $\{a_1, b_2\}$, $\{b_1, a_2\}$, $\{c_1, c_2\}$, $\{d_1, b_2\}$, $\{d_1, d_2\}$, and orders $a_1 <_1 b_1 <_1 c_1 <_1 d_1$, $a_2 <_2 b_2 <_2 c_2 <_2 d_2$.

We prove the **NP**-completeness of SUBSET TOKEN SWAPPING on paths by showing the following:

Theorem 3.1. CROSSING-MINIMIZING MATCHING *is* **NP***-complete, even for graphs of maximum degree 2.*

Proof. It is obvious that our problem is in **NP**, so that we are left with its **NP**-hardness. To start with, we restate an instance (H, l) of VERTEX COVER as an appropriate instance $(G, <_1, <_2, k)$ of CROSSING-MINIMIZING MATCHING in logarithmic space. This in turn requires two kinds of gadgets. A gadget of the first kind is created for every vertex of H, while a gadget of the second kind is created for every edge of H. Vertex gadgets are aligned in such a way that there are no crossings between them, but each of them defines regions between the vertices, called slots, that are used to anchor edge gadgets. The details of the construction ensure that H admits a vertex cover of size at most l if and only if G admits a perfect matching with at most k crossings. The heart of the argument is a careful analysis of the number of crossings between different gadgets.

For any $v \in V(H)$, the *vertex gadget* for v is a cycle on $16 \cdot \deg(v)$ vertices together with an isolated edge, ordered as shown in Figure 3.2. There are two possible perfect matchings in each vertex gadget: either containing all the blue edges and the isolated edge in the middle, or all the yellow edges and the isolated one.

The vertex gadget defines $4 \cdot \deg(v)$ *slots*, i.e. spaces between the vertices, whose exact location is marked in Figure 3.2 with gray rectangles. The $2 \cdot \deg(v)$ slots to the left of the isolated edge are called *left slots* and the other ones are called *right slots*. We arbitrarily assign to every edge $e \in E(H)$ incident to v two consecutive right slots IR_v^e, OR_v^e such that IR_v^e is to the left of OR_v^e (here IR and OR stand for *inner right* and *outer right*, respectively). We also assign to e two consecutive left slots OL_v^e, IL_v^e (*outer left*

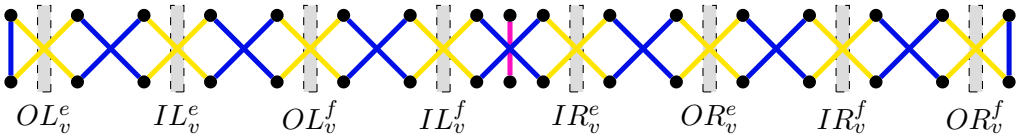


Figure 3.2: A vertex gadget for a vertex v of degree 2 in H, incident to edges e and f, with a possible assignment of slots to these edges.

and *inner left*), but this time IL_v^e is to the right of OL_v^e. Thus, the inner slots of the edge e are closer to the isolated edge than the outer slots. We also make sure that different edges of H get disjoint sets of 4 slots. The fact that there are $2 \cdot \deg(v)$ left and $2 \cdot \deg(v)$ right slots makes such an assignment possible.

We fix an arbitrary linear order $v_1, \ldots, v_n$ on the vertices of H. We arrange the vertex gadgets on the two horizontal lines in such a way that each gadget occupies a separate rectangle between these lines and for every $i < j$, the gadget for v_i is to the left of the gadget for v_j (see Figure 3.3).

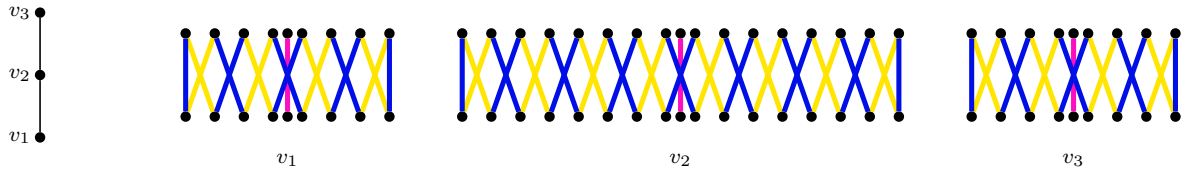


Figure 3.3: A graph H with vertices linearly ordered and an alignment of vertex gadgets for the vertices of H on the two horizontal lines.

For any edge $e = \{v_i, v_j\} \in E(H)$ with $i < j$, the *edge gadget* for e is a cycle on 6 vertices that are labeled and ordered as shown in Figure 3.4. These vertices are carefully placed in the aforementioned slots as follows:

- vertices a and b in the slot $IR_{v_i}^e$,
- vertex c in the slot $OR_{v_i}^e$,
- vertex d in the slot $OL_{v_j}^e$,
- vertices e and f in the slot $IL_{v_j}^e$.

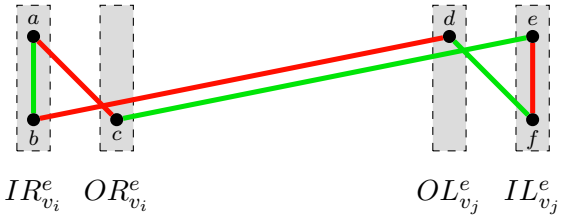


Figure 3.4: The edge gadget for edge $e = \{v_i, v_j\} \in E(H)$ and the placement of its vertices in slots of vertex gadgets for v_i and v_j.

Just like the vertex gadget, the edge gadget admits exactly two perfect matchings: the green one and the red one.

This concludes the construction of the bipartite graph G together with orders $<_1$ and $<_2$. We leave k to be determined later in the proof. An example of this reduction for a small graph is shown in Figure 3.5.

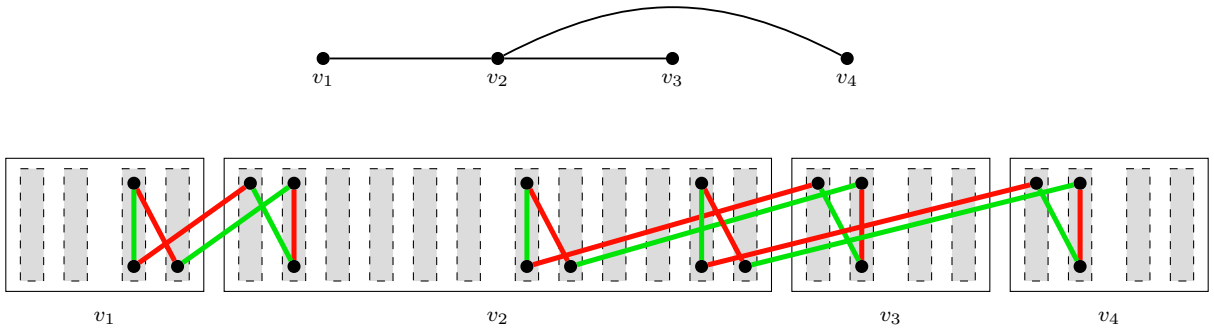


Figure 3.5: A graph H and a possible graph G with orders $<_1$ and $<_2$ obtained by passing H to the reduction algorithm, vertex gadgets presented schematically.

Note that the gadgets are pairwise disjoint graphs and any perfect matching M in G can be viewed as a sum of matchings chosen in each of the gadgets independently.

We interpret the choice of perfect matchings in all vertex gadgets as a selection of a subset of $V(H)$. For every vertex $v \in V(H)$, the perfect matching in the gadget for v containing blue edges corresponds to selecting v into the cover, whereas the other perfect matching is interpreted as not selecting v. For a perfect matching M in G, we denote the set of selected vertices by $S(M)$.

Observe that a vertex cover S of H leads to an orientation of H in which each edge is oriented towards an element of S. Of course, there can be many such orientations for a single vertex cover S. For an $S \subseteq V(H)$ and an orientation $\vec{H}$ of H with each edge oriented towards an element of S, we say that $\vec{H}$ is a *covering orientation* for S.

In our reduction, the choice of perfect matchings in all edge gadgets corresponds to an orientation of H *that may, or may not, be covering* for $S(M)$. For every edge $e = \{v_i, v_j\} \in E(H)$ with $i < j$, the green perfect matching corresponds to orienting e towards v_i, and the red perfect matching orients e towards v_j. For a perfect matching M in G, we denote this orientation by $\vec{H}(M)$.

Note that for any $S \subseteq V(H)$ and for any orientation $\vec{H}$ of H, by choosing appropriate perfect matchings in vertex and edge gadgets, we can obtain a (unique) perfect matching M in G such that $S(M) = S$ and $\vec{H}(M) = \vec{H}$.

We established a one-to-one correspondence between perfect matchings in G and pairs of the form $(S, \vec{H})$, where S is an arbitrary subset of $V(H)$ and $\vec{H}$ is an arbitrary orientation of H. Now, we aim to prove that for a perfect matching M with $\#\mathsf{cr}(M)$ being minimal, the details of the construction ensure that $\vec{H}(M)$ is a covering orientation for $S(M)$, and thus $S(M)$ is a vertex cover of H. Moreover, we will set k so that for $\#\mathsf{cr}(M) \leqslant k$ we will have $|S(M)| \leqslant l$. Conversely, we will show that for any vertex cover S of H with $|S| \leqslant l$ and a covering orientation $\vec{H}$, the perfect matching M

in G with $S(M) = S$ and $\vec{H}(M) = \vec{H}$ admits at most k crossings. Proving these facts requires a detailed analysis of the number of crossings in a perfect matching in G.

We start our analysis by fixing a perfect matching M in G. We split $\#\mathsf{cr}(M)$ into 3 pieces as follows:

$$\#\mathsf{cr}(M) = \#\mathsf{cr}_{\mathsf{VV}}(M) + \#\mathsf{cr}_{\mathsf{EE}}(M) + \#\mathsf{cr}_{\mathsf{VE}}(M),$$

where:

- $\#\mathsf{cr}_{\mathsf{VV}}(M)$ is the number of crossings in M with both edges in vertex gadgets,
- $\#\mathsf{cr}_{\mathsf{EE}}(M)$ is the number of crossings in M with both edges in edge gadgets,
- $\#\mathsf{cr}_{\mathsf{VE}}(M)$ is the number of crossings in M with one edge in a vertex gadget and the other one in an edge gadget.

We begin by calculating $\#\mathsf{cr}_{\mathsf{VV}}(M)$. Observe that there are no crossings in M between edges of different vertex gadgets, as these gadgets do not overlap. Observe also that in the vertex gadget for v, there are $1 + 4 \cdot \deg(v)$ crossings in M if $v \in S(M)$, and only $4 \cdot \deg(v)$ otherwise. Thus, we have

$$\#\mathsf{cr}_{\mathsf{VV}}(M) = |S(M)| + \sum_{v \in V(H)} 4 \cdot \deg(v) = |S(M)| + 2|E(H)|.$$

Now, we aim to show that $\#\mathsf{cr}_{\mathsf{EE}}(M)$ may depend on the way the vertices of H were ordered and the way the slots were assigned to the edges of H, but it does not depend on the particular choice of the matching M. Note first that there is exactly one crossing in each edge gadget (independently of choosing red or green edges to M), totaling $|E(H)|$ crossings. Next observe that two different edge gadgets, $A = \{a, b, c, d, e, f\}$ and $A' = \{a', b', c', d', e', f'\}$ with $a <_1 a'$, can be ordered in three different ways, as shown in Figure 3.6. Note that in each of the cases (a), (b) and (c) the number of crossings does not depend on M. It follows from the analysis in Figure 3.6 that

$$\#\mathsf{cr}_{\mathsf{EE}}(M) = |E(H)| + 0C_a + 3C_b + 2C_c,$$

where C_a, C_b, C_c are the number of pairs of edges that are ordered as in case (a), (b) or (c), respectively.

Finally we are left with $\#\mathsf{cr}_{\mathsf{VE}}(M)$, so that we need to count crossings between the edges of the vertex gadget of $v_q \in V(H)$ and the edges in the edge gadget of $e = \{v_i, v_j\} \in E(H)$ with $i < j$. There are 10 ways these two gadgets may interfere:

(1) $v_q \notin \{v_i, v_j\}$ and either $q < i < j$ or $i < j < q$, i.e. the gadgets do not overlap.

This case does not contribute to $\#\mathsf{cr}_{\mathsf{VE}}(M)$.

(2) $v_q \notin \{v_i, v_j\}$ and $i < q < j$.

In this case, exactly one edge of M in the edge gadget for $v_i v_j$, a green one or a red one, crosses all $1 + 8 \cdot \deg(v_q)$ edges of M in the vertex gadget for v_q, and there are no other crossings. The total number $C_\Diamond$ of crossings coming from all such overlaps is obviously independent of the matching M.

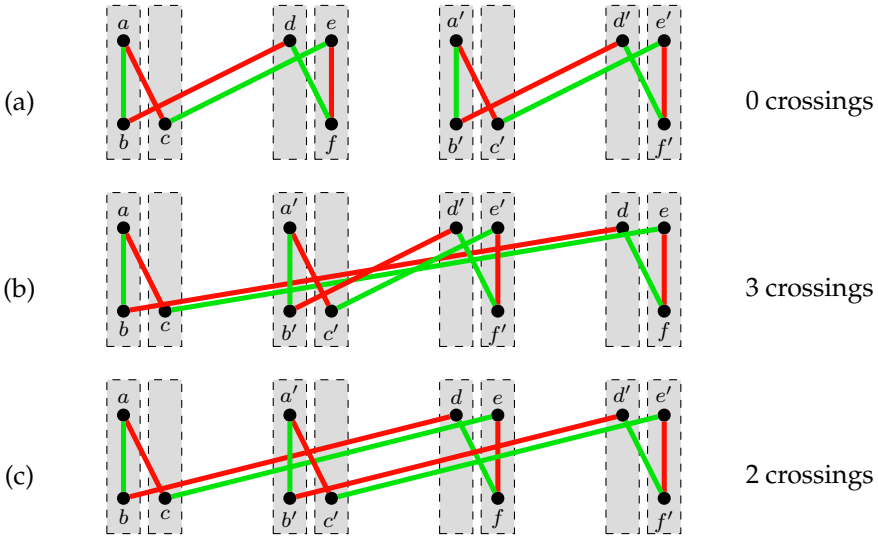


Figure 3.6: The 3 possibilities of how the vertices of two different edge gadgets are ordered. In case (a), there are 0 crossings. In case (b), either the edge $\{b, d\}$ or the edge $\{c, e\}$ crosses all 3 matching edges in the other gadget and there are no other crossings, which gives 3 crossings. In case (c), exactly one of the edges $\{b, d\}$ and $\{c, e\}$ crosses the matching edge of a' and exactly one of the edges $\{b', d'\}$ and $\{c', e'\}$ crosses the matching edge of f, and there are no other crossings, which gives 2 crossings.

Recall that we refer to $v_q \in S(M)$ as v_q being selected. Whenever e is oriented towards v_q, we say that v_q is the *head* of the oriented edge, otherwise we say that v_q is its *tail*. The remaining 8 cases distinguish whether v_q overlaps with the left or the right part of the gadget of e, whether it is selected or not, and whether it is the head or the tail of e.

(3) [left, selected, head] $q = i$, $v_q \in S(M)$, and v_q is the head of e in $\vec{H}(M)$.

(4) [left, selected, tail] $q = i$, $v_q \in S(M)$, and v_q is the tail of e in $\vec{H}(M)$.

(5) [left, not selected, head] $q = i$, $v_q \notin S(M)$, and v_q is the head of e in $\vec{H}(M)$.

(6) [left, not selected, tail] $q = i$, $v_q \notin S(M)$, and v_q is the tail of e in $\vec{H}(M)$.

(7) [right, selected, head] $q = j$, $v_q \in S(M)$, and v_q is the head of e in $\vec{H}(M)$.

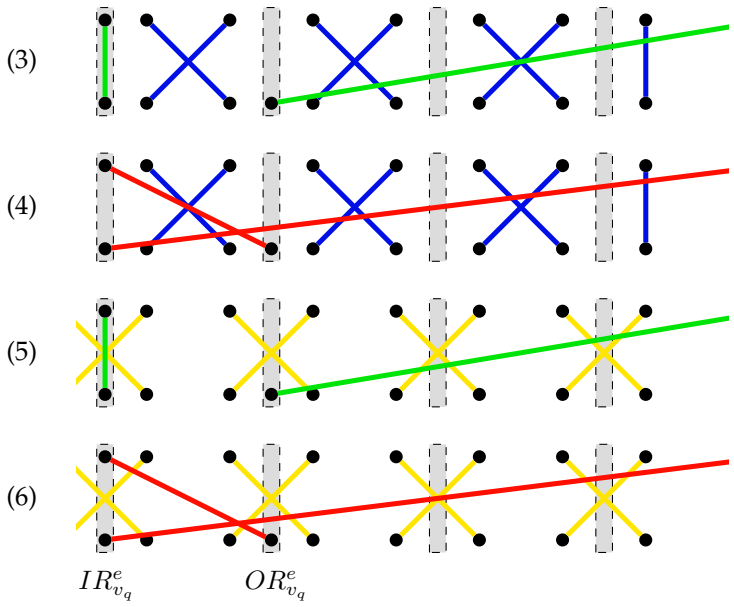
(8) [right, selected, tail] $q = j$, $v_q \in S(M)$, and v_q is the tail of e in $\vec{H}(M)$.

(9) [right, not selected, head] $q = j$, $v_q \notin S(M)$, and v_q is the head of e in $\vec{H}(M)$.

(10) [right, not selected, tail] $q = j$, $v_q \notin S(M)$, and v_q is the tail of e in $\vec{H}(M)$.

Let the values $\#\mathsf{lsh}(M)$, $\#\mathsf{lst}(M)$, $\#\mathsf{lnsh}(M)$, $\#\mathsf{lnst}(M)$, $\#\mathsf{rsh}(M)$, $\#\mathsf{rst}(M)$, $\#\mathsf{rnsh}(M)$, and $\#\mathsf{rnst}(M)$ be the number of occurrences of each of the cases (3) – (10), respectively.

First, we analyze the number of crossings from the cases (3) – (6). Let $s_{q,j}$ be the number of slots in the vertex gadget of v_q that are to the right of the slot $OR^e_{v_q}$. Obviously, the value $s_{q,j}$ does not depend on M. The number of crossings between the edges of the gadget for v_q and the edges of the gadget for e in these 4 cases is calculated in Figure 3.7.



left, selected, head;
$2s_{q,j}+1$ crossings;
$\#\mathsf{lsh}(M)$ such situations

left, selected, tail;
$2s_{q,j}+5$ crossings;
$\#\mathsf{lst}(M)$ such situations

left, not selected, head;
$2s_{q,j}+3$ crossings;
$\#\mathsf{lnsh}(M)$ such situations

left, not selected, tail;
$2s_{q,j}+5$ crossings;
$\#\mathsf{lnst}(M)$ such situations

Figure 3.7: The 4 possible configurations of crossings between the right part of the vertex gadget for v_q and the left part of the edge gadget for e, here $s_{q,j}=2$.

Let C_l be the sum of the values $2s_{q,j}+1$, where $q<j$ range over all edges $\{v_q,v_j\}\in E(H)$. The total number of crossings in the cases (3) – (6) is equal to:

$$\#\mathsf{cr}_{\mathsf{VE}}^{3-6}(M)=C_l+2\cdot\#\mathsf{lnsh}(M)+4\cdot\#\mathsf{lst}(M)+4\cdot\#\mathsf{lnst}(M).$$

Now we analyze the total number of crossings in the cases (7) – (10). By symmetry, this number amounts to:

$$\#\mathsf{cr}_{\mathsf{VE}}^{7-10}(M)=C_r+2\cdot\#\mathsf{rnsh}(M)+4\cdot\#\mathsf{rst}(M)+4\cdot\#\mathsf{rnst}(M).$$

Summing up all 10 cases, we obtain:

$$\begin{aligned}\#\mathsf{cr}_{\mathsf{VE}}(M)&=C_\Diamond+\#\mathsf{cr}_{\mathsf{VE}}^{3-6}(M)+\#\mathsf{cr}_{\mathsf{VE}}^{7-10}(M)\\&=C_\Diamond+C_l+C_r+2(\#\mathsf{lnsh}(M)+\#\mathsf{rnsh}(M))\\&\quad+4(\#\mathsf{lst}(M)+\#\mathsf{lnst}(M)+\#\mathsf{rst}(M)+\#\mathsf{rnst}(M)).\end{aligned}\tag{$\star$}$$

Let $\#\mathsf{nsh}(M)=\#\mathsf{lnsh}(M)+\#\mathsf{rnsh}(M)$. Now, observe that the sum $\#\mathsf{lst}(M)+\#\mathsf{lnst}(M)+\#\mathsf{rst}(M)+\#\mathsf{rnst}(M)$ is simply $|E(H)|$, and ($\star$) simplifies to:

$$\#\mathsf{cr}_{\mathsf{VE}}(M)=C_\Diamond+C_l+C_r+2\cdot\#\mathsf{nsh}(M)+4|E(H)|.$$

We are now ready to conclude our analysis with

$$\begin{aligned}\#\mathsf{cr}(M)&=\#\mathsf{cr}_{\mathsf{VV}}(M)+\#\mathsf{cr}_{\mathsf{EE}}(M)+\#\mathsf{cr}_{\mathsf{VE}}(M)\\&=|S(M)|+3C_b+2C_c+C_\Diamond+C_l+C_r+2\cdot\#\mathsf{nsh}(M)+7|E(H)|\\&=|S(M)|+2\cdot\#\mathsf{nsh}(M)+C,\end{aligned}$$

where $C=3C_b+2C_c+C_\Diamond+C_l+C_r+7|E(H)|$ is a constant completely independent of the matching M.

To complete the description of our reduction from VERTEX COVER to CROSSING--MINIMIZING MATCHING, we set $k = l + C$. It is straightforward to implement the reduction algorithm in logarithmic space. It remains to prove that G admits a perfect matching with at most k crossings if and only if H admits a vertex cover of size l.

First suppose that H admits a vertex cover S of size at most l and $\vec{H}$ is a covering orientation for S. Let M be the perfect matching in G with $S(M) = S$ and $\vec{H}(M) = \vec{H}$. We have $\#\mathsf{nsh}(M) = 0$ because $\vec{H}$ is covering for S, and the number of crossings in M is equal to $|S(M)| + C \leqslant l + C = k$.

For the other direction, assume that G admits a perfect matching with at most k crossings. Let M be any such matching with minimum number of crossings. Observe that $\#\mathsf{nsh}(M) = 0$. Indeed, if there exists an edge $(v_i, v_j) \in E(\vec{H}(M))$ such that $v_j \notin S(M)$, one can replace the yellow perfect matching in the gadget for v_j with the blue one and obtain a perfect matching M' in G such that $v_j \in S(M')$. As $|S(M')| = |S(M)| + 1$ and $\#\mathsf{nsh}(M') = \#\mathsf{nsh}(M) - 1$, we have $\#\mathsf{cr}(M') = \#\mathsf{cr}(M) - 1$, which contradicts the minimality of $\#\mathsf{cr}(M)$. Now observe that $\vec{H}(M)$ is a covering orientation for $S(M)$ thanks to $\#\mathsf{nsh}(M) = 0$. Finally, as the number of crossings in M is equal to $|S(M)| + C$ and is no larger than k, the size of the vertex cover $S(M)$ is at most $k - C = l$.

□

Observe that in the proof above, the size of the CROSSING-MINIMIZING MATCHING instance is linear in the size of the VERTEX COVER instance. Indeed, for every vertex $v \in V(H)$ we produce $16 \cdot \deg(v) + 2$ vertices of G, and for every edge of H, six vertices are produced. Hence, the number of vertices in the graph G, outputted by the reduction, is bounded by $O(|V(H)| + |E(H)|)$. As the vertices in G are of degree at most 2, we have $|E(G)| = O(|V(H)| + |E(H)|)$. We note that VERTEX COVER does not admit an algorithm running in time $2^{o(|V(H)|+|E(H)|)}$, assuming the Exponential Time Hypothesis (Theorem 14.6 in [23]). This yields the following:

Theorem 3.2. *There is no $2^{o(|V(G)|+|E(G)|)}$ algorithm for* CROSSING-MINIMIZING MATCHING, *unless ETH fails.*

4

Smaller universal targets for homomorphisms of edge-colored graphs

Studying universal structures, which reflect and encode all possible structures of a certain kind, has been an important part of computer science. In this chapter, for each q, d, and k we construct a finite k-edge-colored graph $\mathbb{H}_{q,d,k}$ that captures the local behavior of all k-edge-colorings of all graphs with acyclic chromatic number at most q and density at most d. This is a continuation and strengthening of our research published in [47] and presented in 2015 as a master's thesis.

4.1 Introduction

A *k-edge-colored graph* $\mathbb{G}$ is a pair (G, c), where G is a graph, called *an underlying graph* of $\mathbb{G}$, and c is a mapping from $E(G)$ to $[k]$, called a *k-edge-coloring* of $\mathbb{G}$. For $\{u, v\} \in E(G)$, instead of $c(\{u, v\})$ we simply write $c(u, v)$. A *k-edge-colored graph over G* is a k-edge-colored graph with the underlying graph G.

Let $\mathbb{G}_1 = (G_1, c_1)$ and $\mathbb{G}_2 = (G_2, c_2)$ be two k-edge-colored graphs. A mapping $h : V(G_1) \to V(G_2)$ is a *homomorphism* of $\mathbb{G}_1$ to $\mathbb{G}_2$ if, for every two vertices u and v that are adjacent in G_1, $h(u)$ and $h(v)$ are adjacent in G_2 and $c_1(u, v) = c_2(h(u), h(v))$. In other words, a homomorphism of $\mathbb{G}_1$ to $\mathbb{G}_2$ maps every colored edge in $\mathbb{G}_1$ into an edge of the same color in $\mathbb{G}_2$.

A k-edge-colored graph $\mathbb{H}$ is *k-universal*[1] for a class $\mathcal{F}$ of graphs if every k-edge-colored graph over any graph in $\mathcal{F}$ admits a homomorphism to $\mathbb{H}$. We denote by $\lambda_{\mathcal{F}}(k)$ the minimum possible number of vertices in a k-universal graph for $\mathcal{F}$. We set $\lambda_{\mathcal{F}}(k) = \infty$ if such a graph does not exist.

Observe that $\lambda_{\mathcal{F}}(1)$ is the maximum chromatic number of all graphs in $\mathcal{F}$. Although this parameter is of great importance in graph theory, this chapter is focused on the behavior of $\lambda_{\mathcal{F}}(k)$ when k tends to infinity. In particular, the case $k = 1$ differs significantly from the case $k \geqslant 2$. Only the latter one is the subject of this chapter.

The concept of finding a small k-universal graph for a certain class of graphs first arose in 1998, when Alon and Marshall [3] used it to obtain a result on Coxeter groups. They showed for the class of planar graphs $\mathcal{P}$ that $\lambda_{\mathcal{P}}(k)$ is between $k^3 + 3$ and $5k^4$. They also generalized their ideas for graphs with bounded acyclic chromatic number. An *acyclic coloring* of a graph G is an assignment of colors to the vertices of G such that adjacent vertices have different colors and every subgraph of G with vertices in

[1] In literature, universal graphs are also called homomorphism bounds.

at most 2 colors is acyclic. The *acyclic chromatic number* of a graph G, denoted $\chi_a(G)$, is the minimum number of colors in an acyclic coloring of G. For a class $\mathcal{F}$ of graphs, if the acyclic chromatic number of the graphs in $\mathcal{F}$ is bounded by a constant, we write $\chi_a(\mathcal{F}) = \max_{G\in\mathcal{F}} \chi_a(G)$ (and set $\chi_a(\mathcal{F}) = \infty$ otherwise). Alon and Marshall [3] showed that for every graph class $\mathcal{F}$ with $\chi_a(\mathcal{F}) = r < \infty$, we have $\lambda_{\mathcal{F}}(k) \leqslant rk^{r-1}$. Plugging in the famous result of Borodin [13] that $\chi_a(\mathcal{P}) \leqslant 5$ gives $\lambda_{\mathcal{P}}(k) \leqslant 5k^4$.

A concept similar to homomorphisms of edge colorings was considered by Raspaud and Sopena [69]. They show that for every oriented planar graph $\vec{G}$ there exists an oriented graph $\vec{H}$ on at most 80 vertices, such that $\vec{G}$ maps homomorphically to $\vec{H}$, where a homomorphism of an oriented graph $\vec{G}$ to an oriented graph $\vec{H}$ is a mapping $h : V(\vec{G}) \to V(\vec{H})$ such that for every directed edge $(u, v) \in E(\vec{G})$, there is an edge from $h(u)$ to $h(v)$ in $\vec{H}$. This concept is also known under the name *oriented coloring*. A simple consequence of this result is that there exists a single graph $\vec{H}$ on at most 80 vertices, to which every oriented planar graph maps homomorphically. Later, Nešetřil and Raspaud [66] proved a theorem about *mixed graphs* (i.e. graphs with both oriented and unoriented colored edges) that implies both the results of Alon and Marshall and those of Raspaud and Sopena (see also [24]).

Universal graphs were recently analyzed further in our paper with Gutowski [47]. We now shortly summarize these results. First, it is shown that for every $k \geqslant 2$, a class $\mathcal{F}$ of graphs admits a k-universal graph if and only if the acyclic chromatic number of graphs in $\mathcal{F}$ is bounded by a constant. In particular, this means that $\mathcal{F}$ either admits a k-universal graph for all $k \geqslant 2$ or for no $k \geqslant 2$. Next, an analysis of the asymptotic behavior of $\lambda_{\mathcal{F}}(k)$ is performed. It happens that $\lambda_{\mathcal{F}}(k)$ can be much smaller than $O\big(k^{\chi_a(\mathcal{F})-1}\big)$ and is more closely related to the density of the graphs in $\mathcal{F}$. For a graph G, the *density* of G, denoted $D(G)$, is the maximum ratio of the number of edges to the number of vertices over all subgraphs of G. For a class $\mathcal{F}$ of graphs, its density $D(\mathcal{F})$ is the supremum of the densities of the graphs in $\mathcal{F}$. A simple argument gives $D(\mathcal{F}) \leqslant \chi_a(\mathcal{F})-1$ and there are examples of classes of graphs with bounded density and unbounded acyclic chromatic number. The main result of [47] is the following theorem:

Theorem 4.1 ([47]). *Let $\mathcal{F}$ be a class of graphs with $\chi_a(\mathcal{F}) = r < \infty$ and $\lceil D(\mathcal{F})\rceil = d$. Then, for the constant $c = 8dr^4\binom{8dr^4}{d}$, we have*

$$k^{D(\mathcal{F})} \leqslant \lambda_{\mathcal{F}}(k) \leqslant ck^{\lceil D(\mathcal{F})\rceil},$$

for all $k \geqslant 2$.

Since $D(\mathcal{F}) \leqslant \chi_a(\mathcal{F}) - 1$, the above upper bound is asymptotically no worse than the one by Alon and Marshall (although the latter is usually much better for small k's). For any class of graphs with density being an integer, the bounds of Theorem 4.1 are asymptotically tight. For example, we get that $\lambda_{\mathcal{P}}(k) = \Theta(k^3)$.

In this chapter, we continue the study of the asymptotics of $\lambda_{\mathcal{F}}(k)$. The paper [47] concludes with the question whether $\lambda_{\mathcal{F}}(k)$ is always $\Theta\big(k^{D(\mathcal{F})}\big)$. Our next theorem confirms this hypothesis for $D(\mathcal{F})$ being a rational number and provides evidence that it should hold in general.

Theorem 4.2. *Let $\mathcal{F}$ be a class of graphs with $\chi_a(\mathcal{F}) = r < \infty$ and $D(\mathcal{F})$ bounded from above by a quotient s/t of natural numbers. Then, for the constant $c = 2^s(8r^4s)^t\binom{8r^4st}{s}$, we have*

$$k^{D(\mathcal{F})} \leqslant \lambda_{\mathcal{F}}(k) \leqslant ck^{s/t}, \tag{1}$$

for all $k \geqslant 2$.

The immediate consequences of Theorem 4.2 for a class $\mathcal{F}$ with $\chi_a(\mathcal{F}) < \infty$ are:

(1) $\lambda_{\mathcal{F}}(k) = \Theta\big(k^{D(\mathcal{F})}\big)$, if $D(\mathcal{F})$ is a rational number,

(2) $\lambda_{\mathcal{F}}(k) = o\big(k^{D(\mathcal{F})+\varepsilon}\big)$ for every $\varepsilon > 0$.

Our proof of Theorem 4.2 uses the techniques from [47] in a new way. In particular, the proof requires additional ideas that allow us to work with nonintegral graph densities. It is known that a graph G admits a $\lceil D(G)\rceil$-orientation. We employ a more fine-grained concept of fractional orientations to have a better control over the construction size. This technique seems to be quite interesting on its own and we hope that it can be used in other problems related to edge densities or other nonintegral graph parameters.

4.2 Universal graph construction

In this section, we collect several tools from [47] and apply them in a new way to obtain the main theorem.

The paper [47] makes use of a simple observation, attributed to Hakimi [48], that for an integer d, a graph G admits a d-orientation if and only if $D(G) \leqslant d$. Not surprisingly, d may be relaxed to be a rational number s/t. First, we replace the original graph with a multigraph that contains t copies of each edge. Now, for a vertex v, instead of orienting at most d edges towards v in G, we orient at most s edges towards v in this multigraph. We formalize this idea in the following lemma:

Lemma 4.3. *For every graph G and a quotient s/t of natural numbers, we have $D(G) \leqslant s/t$ if and only if there exist orientations $\vec{G}_1, ..., \vec{G}_t$ of G such that for every $v \in V(G)$*

$$\sum_{i=1}^{t} \deg^{\text{in}}_{\vec{G}_i}(v) \leqslant s. \tag{2}$$

Proof. Assume that the orientations $\vec{G}_1, ..., \vec{G}_t$ of G have the property (2). Let H be a subgraph of G. For every $i = 1, ..., t$, we have

$$|E(H)| \leqslant \sum_{v\in V(H)} \deg^{\text{in}}_{\vec{G}_i}(v).$$

Summing up over the i's, we get

$$t \cdot |E(H)| \leqslant \sum_{v\in V(H)} \sum_{i=1}^{t} \deg^{\text{in}}_{\vec{G}_i}(v) \leqslant |V(H)| \cdot s.$$

Therefore, $|E(H)|/|V(H)| \leqslant s/t$.

For the other direction, we assume that $D(G) \leqslant s/t$ and apply Hall's Theorem to a bipartite graph B constructed as follows. The vertex set of B contains s copies of each vertex of G and t copies of each edge of G. For each $v \in V(G)$ and $e \in E(G)$ such that v is incident with e, we add an edge in B between every copy of v and every copy of e.

We show that B admits a matching of all edge copies by checking the Hall's condition for every set X of edge copies. Every such X induces a subgraph of G with at least $|X|/t$ edges and, according to the density bound, at least $|X|/s$ vertices. As every vertex of G has s copies in B, the set X is incident with at least $|X|$ vertices of B, and by Hall's Theorem the required matching indeed exists.

Now, given a matching in B, we construct the orientations $\vec{G_1}, ..., \vec{G_t}$ as follows: in the i-th orientation, every edge e is oriented towards the vertex whose copy is paired in the matching with the i-th copy of e. The bound on the sum of indegrees of $v \in V(G)$ in all orientations follows from the fact that there are s copies of v. □

Apart from the acyclic coloring, we use two other kinds of colorings. The first one, the *star coloring* of a graph, is an assignment of colors to the vertices of the graph such that:

(i) every two adjacent vertices get different colors,

(ii) every subsequent four vertices on any path in the graph get at least 3 different colors.

In other words, a star coloring is a proper coloring such that, for any two colors, every connected component in the graph induced by vertices of these two colors has at most one vertex of degree higher than one. Observe that any star coloring of G is an acyclic coloring of G. Conversely, Albertson, Chappell, Kierstead, Kündgen and Ramamurthi [2] showed that any acyclic coloring with r colors can be used to construct a star coloring with at most $2r^2 - r$ colors.

The other coloring we need is called out-coloring and is a technical tool from [47]. This concept appeared in almost the same form under the name in-coloring in [2], and earlier without name in Nešetřil and Ossona de Mendez [65]. Let $\vec{G}$ be an orientation of a graph G. We use the following notions: if (u, v) is an edge of $\vec{G}$, then u is a *parent* of v; if (u, v) and (v, w) are edges of $\vec{G}$, then u is a *grandparent* of w. An *out-coloring* of an oriented graph is an assignment of colors to the vertices of the graph such that:

(C1) every two adjacent vertices get different colors,

(C2) every two distinct parents of a single vertex get different colors,

(C3) a vertex gets different colors from any of its grandparents.

Out-colorings are closely related to star colorings (and acyclic colorings, in consequence). Indeed, every out-coloring of $\vec{G}$ is a star coloring of G. Moreover, [47] contains an easy construction of an out-coloring of $\vec{G}$ from a star coloring of G.

For the upper bound for $\lambda_{\mathcal{F}}(k)$ given in [47], an explicit construction of a k-universal graph has been provided, and the heart of this construction is contained in the following lemma:

Lemma 4.4 ([47], Lemma 11)**.** *Let $\mathcal{F}$ be a class of graphs for which there are absolute constants q and d such that every graph in $\mathcal{F}$ admits a d-orientation that has an out-coloring with q colors. For any $k \geqslant 2$, the following holds:*

$$\lambda_{\mathcal{F}}(k) \leqslant q\binom{q}{d}k^d.$$

In the current setting, we need the following slightly stronger restatement of Lemma 4.4, in which the universal graph is carefully stratified.

Lemma 4.5. *For integers $k \geqslant 2$, d and q there exist a k-edge-colored graph $\mathbb{H}$ and sets $V_0 \subseteq ... \subseteq V_d = V(\mathbb{H})$ such that:*

(P1) for every k-edge-colored graph $\mathbb{G} = (G, c_G)$ and for every d-orientation $\vec{G}$ of G that admits an out-coloring with q colors there exists a homomorphism h of $\mathbb{G}$ to $\mathbb{H}$ such that for every vertex $v \in V(\mathbb{G})$ with indegree i in $\vec{G}$ we have $h(v) \in V_i$,

(P2) the size of each set V_i satisfies $|V_i| \leqslant q\binom{q}{i}k^i$.

Proof. Our construction of $\mathbb{H} = (H, c_H)$ is modeled after the one used in [47] to prove Lemma 4.4.

The vertex set of $\mathbb{H}$ is the set of all $(q+1)$-tuples of the form

$$(\alpha, x_1, x_2, ..., x_q),$$

where $\alpha \in [q]$, $x_j \in [k]$, and where among $x_1, x_2, ..., x_q$ there are at most d values different from k. We make $\mathbb{H}$ to be a complete graph. The edges of $\mathbb{H}$ are colored by

$$c_H((\alpha, x_1, x_2, ..., x_q), (\beta, y_1, y_2, ..., y_q)) = \min(y_\alpha, x_\beta).$$

Now, the vertices $(\alpha, x_1, ..., x_q)$ of $\mathbb{H}$ are stratified into the sets V_i according to how many x_j's differ from k, i.e. we put

$$V_i = \{(\alpha, x_1, ..., x_q) \in V(H) \mid \#\{j \mid x_j \neq k\} \leqslant i\}.$$

Obviously, $|V_i| \leqslant q\binom{q}{i}k^i$, as required in (P2).

Let $\mathbb{G} = (G, c_G)$ be a k-edge-colored graph such that G admits a d-orientation $\vec{G}$ that has an out-coloring f with q colors. Note that $\vec{G}$ being a d-orientation of G restricts the number of parents of any vertex of G to at most d. Moreover, condition (C2) for out-colorings gives that for each color $j \in [q]$ used by f, a vertex u of G has at most one parent p of color j. This allows us to correctly define a mapping $h : V(G) \to V(H)$ by putting

$$h(u) = (f(u), x_1, x_2, ..., x_q),$$

where

$$x_j = \begin{cases} c_G(u,p), & \text{if } u \text{ has a parent } p \text{ in } \vec{G} \text{ with } f(p) = j, \\ k, & \text{otherwise.} \end{cases}$$

Directly from this definition, it should be obvious that $h(u) \in V_i$ whenever u has at most i parents.

Finally, to see that h is a homomorphism, start with an edge $\{u,v\} \in E(\mathbb{G})$, and suppose (without loss of generality) that v is a parent of u in $\vec{G}$. Representing $h(u)$ and $h(v)$ by

$$\begin{aligned} h(u) &= (f(u), x_1, x_2, ..., x_r), \\ h(v) &= (f(v), y_1, y_2, ..., y_r), \end{aligned}$$

we note that (C1) yields $f(u) \neq f(v)$ and consequently $h(u) \neq h(v)$.

It remains to argue that $c_H(h(u), h(v)) = c_G(u,v)$. However, in the view of $c_H(h(u), h(v)) = \min(x_{f(v)}, y_{f(u)})$ and $x_{f(v)} = c_G(u,v)$, it suffices to show that $y_{f(u)} = k$. Suppose otherwise, so that v has a parent p in $\vec{G}$ that is colored by $f(u)$. But then, p is a grandparent of u and $f(p) = f(u)$, which contradicts condition (C3). □

After these preparations, we are ready to construct a k-universal graph on $O\big(k^{s/t}\big)$ vertices.

Lemma 4.6. *Let $\mathcal{F}$ be a class of graphs with density bounded from above by a quotient s/t of natural numbers and let q be an integer such that every s-orientation of every graph in $\mathcal{F}$ admits an out-coloring with q colors. For any $k \geqslant 2$, the following holds:*

$$\lambda_{\mathcal{F}}(k) \leqslant 2^s q^t \binom{qt}{s} k^{s/t}.$$

Proof. Let $k_0 = \lceil k^{1/t} \rceil$ and let g be a one-to-one mapping from $[k]$ to $[k_0]^t$. After putting $k = k_0$ and $d = s$, Lemma 4.5 supplies us with a k_0-edge-colored graph $\mathbb{H}_0 = (H_0, c_{H_0})$ and a sequence $V_0 \subseteq ... \subseteq V_s$ of subsets of $V(H_0)$ with properties (P1) and (P2). We construct a k-universal graph $\mathbb{H} = (H, c_H)$ for $\mathcal{F}$ as follows. The vertex set of H is given by:

$$V(H) = \bigcup_{\substack{i_1,...,i_t \in \{0,...,s\} \\ i_1+...+i_t = s}} V_{i_1} \times ... \times V_{i_t}.$$

We note that $V(H)$ is a subset of $V(H_0)^t$ and the size of $V(H)$ is bounded by:

$$\sum_{\substack{i_1,...,i_t \in \{0,...,s\} \\ i_1+...+i_t = s}} \prod_{j=1}^{t} q \binom{q}{i_j} \lceil k^{1/t} \rceil^{i_j} \leqslant q^t \binom{qt}{s} \left(k^{1/t} + 1\right)^s \leqslant q^t \binom{qt}{s} 2^s k^{s/t}.$$

An edge between $(u_1, ..., u_t), (v_1, ..., v_t) \in V(H)$ exists if and only if $\{u_i, v_i\}$ is an edge in H_0 for every $i \in [t]$ and $(c_{H_0}(u_1, v_1), ..., c_{H_0}(u_t, v_t))$ is contained in the range of g. In such a case we put:

$$c_H((u_1, ..., u_t), (v_1, ..., v_t)) = g^{-1}((c_{H_0}(u_1, v_1), ..., c_{H_0}(u_t, v_t))).$$

Now, let $\mathbb{G} = (G, c_G)$ be a k-edge-colored graph with $G \in \mathcal{F}$. To construct a homomorphism of $\mathbb{G}$ to $\mathbb{H}$, we start with defining $c_1, ..., c_t$ to be k_0-edge-colorings of G such that for every $\{u,v\} \in E(G)$

$$g(c_G(u,v)) = (c_1(u,v), ..., c_t(u,v)).$$

Next, let $\vec{G}_1, ..., \vec{G}_t$ be the s-orientations of G provided by Lemma 4.3. Applying Lemma 4.5 to the k_0-edge-colored graph (G, c_j) and the s-orientation $\vec{G}_j$, we get a homomorphism $h_j : (G, c_j) \to \mathbb{H}_0$. We want the mapping defined by

$$h(v) = (h_1(v), ..., h_t(v))$$

to be a homomorphism of $\mathbb{G}$ to $\mathbb{H}$.

To see that $h(v) \in V(H)$, note that the bound (2) of Lemma 4.3 together with property (P1) yields $h(v) \in V_{i_1} \times ... \times V_{i_t}$ for some $i_1, ..., i_t$ with $i_1 + ... + i_t = s$. To see that h preserves an edge $\{u, v\} \in E(G)$, first note that each h_i preserves it, so that $\{h_i(u), h_i(v)\} \in E(H_0)$ and $c_i(u, v) = c_{H_0}(h_i(u), h_i(v))$. The tuple

$$(c_{H_0}(h_1(u), h_1(v)), ..., c_{H_0}(h_t(u), h_t(v))) = (c_1(u, v), ..., c_t(u, v)) = g(c_G(u, v))$$

is contained in the range of g, therefore the vertices

$$h(u) = (h_1(u), ..., h_t(u)), \qquad h(v) = (h_1(v), ..., h_t(v))$$

are connected by an edge in $\mathbb{H}$. Its color is given by

$$\begin{aligned} c_H((h_1(u), ..., h_t(u)), (h_1(v), ..., h_t(v))) &= g^{-1}((c_{H_0}(h_1(u), h_1(v)), ..., c_{H_0}(h_t(u), h_t(v)))) \\ &= g^{-1}((c_1(u, v), ..., c_t(u, v))) \\ &= c_G(u, v), \end{aligned}$$

which concludes the proof. □

Now we are ready to prove our main theorem.

Proof of Theorem 4.2. Note that the lower bound in (1) appears already in Theorem 4.1. For the upper bound, first recall that one of the results in [2] says that every graph in $\mathcal{F}$ admits a star coloring with at most $2r^2 - r$ colors. By applying Lemma 10 from [47], we get that every s-orientation of every graph from $\mathcal{F}$ admits an out-coloring with at most $8r^4 s$ colors. We apply Lemma 4.6 with $q = 8r^4 s$ and obtain a k-universal graph for $\mathcal{F}$ of the desired size. □

Bibliography

[1] Akanksha Agrawal, Grzegorz Guśpiel, Jayakrishnan Madathil, Saket Saurabh, and Meirav Zehavi. Connecting the dots (with minimum crossings). In *35th International Symposium on Computational Geometry (SoCG)*, 2019, accepted.

[2] Michael O. Albertson, Glenn G. Chappell, Henry A. Kierstead, André Kündgen, and Radhika Ramamurthi. Coloring with no 2-colored P4's. *Electronic Journal of Combinatorics*, 11(1):R26, 2004.

[3] Noga Alon and Timothy H. Marshall. Homomorphisms of edge-colored graphs and Coxeter groups. *Journal of Algebraic Combinatorics*, 8(1):5–13, 1998.

[4] Thomas Andreae. Some results on visibility graphs. *Discrete Applied Mathematics*, 40(1):5–17, 1992.

[5] Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Vít Jelínek, Jan Kratochvíl, Maurizio Patrignani, and Ignaz Rutter. Testing planarity of partially embedded graphs. *ACM Transactions on Algorithms*, 11(4):32:1–42, 2015.

[6] Kirby A. Baker, Peter C. Fishburn, and Fred S. Roberts. Partial orders of dimension 2. *Networks*, 2(1):11–28, 1972.

[7] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, 3rd ed. edition, 2008.

[8] Mark de Berg and Amirali Khosravi. Optimal binary space partitions for segments in the plane. *International Journal of Computational Geometry & Applications*, 22(3):187–205, 2012.

[9] Therese C. Biedl. Small drawings of outerplanar graphs, series-parallel graphs, and other planar graphs. *Discrete & Computational Geometry*, 45(1):141–160, 2011.

[10] Thomas Bläsius and Ignaz Rutter. Simultaneous PQ-ordering with applications to constrained embedding problems. *ACM Transactions on Algorithms*, 12(2):16:1–46, 2015.

[11] Édouard Bonnet, Tillmann Miltzow, and Paweł Rzążewski. Complexity of token swapping and its variants. 2016. arXiv:1607.07676v1.

[12] Édouard Bonnet, Tillmann Miltzow, and Paweł Rzążewski. Complexity of token swapping and its variants. *Algorithmica*, 80(9):2656–2682, September 2018.

[13] Oleg V. Borodin. On acyclic colorings of planar graphs. *Discrete Mathematics*, 25(3):211–236, 1979.

[14] Richard P. Brent. An improved Monte Carlo factorization algorithm. *BIT Numerical Mathematics*, 20(2):176–184, Jun 1980.

[15] Jean Cardinal and Udo Hoffmann. Recognition and complexity of point visibility graphs. *Discrete & Computational Geometry*, 57(1):164–178, 2017.

[16] Yi-Wu Chang, Joan P. Hutchinson, Michael S. Jacobson, Jenö Lehel, and Douglas B. West. The bar visibility number of a graph. *SIAM Journal on Discrete Mathematics*, 18(3):462–471, 2004.

[17] Steven Chaplick, Paul Dorbec, Jan Kratochvíl, Mickael Montassier, and Juraj Stacho. Contact representations of planar graphs: Extending a partial representation is hard. In *WG 2014: 40th International Workshop on Graph-Theoretic Concepts in Computer Science, Nouan-le-Fuzelier, France, June 2014. Revised Selected Papers*, pages 139–151, 2014.

[18] Steven Chaplick, Radoslav Fulek, and Pavel Klavík. Extending partial representations of circle graphs. In *GD 2013: 21st International Symposium on Graph Drawing, Bordeaux, France, September 2013. Revised Selected Papers*, pages 131–142, 2013.

[19] Steven Chaplick, Grzegorz Guśpiel, Grzegorz Gutowski, Tomasz Krawczyk, and Giuseppe Liotta. The Partial Visibility Representation Extension Problem. In *GD 2016: 24th International Symposium on Graph Drawing and Network Visualization, Athens, Greece, September 2016. Revised Selected Papers*, pages 266–279, 2016.

[20] Steven Chaplick, Grzegorz Guśpiel, Grzegorz Gutowski, Tomasz Krawczyk, and Giuseppe Liotta. The Partial Visibility Representation Extension Problem. *Algorithmica*, 80(8):2286–2323, Aug 2018.

[21] Marek Chrobak and Thomas H. Payne. A linear-time algorithm for drawing a planar graph on a grid. *Information Processing Letters*, 54(4):241–246, 1995.

[22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[23] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

[24] Sandip Das, Soumen Nandi, and Sagnik Sen. On chromatic number of colored mixed graphs. In *Algorithms and Discrete Applied Mathematics - Third International Conference, CALDAM 2017, Sancoale, Goa, India, February 16-18, 2017, Proceedings*, pages 130–140, 2017.

[25] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

[26] Giuseppe Di Battista and Fabrizio Frati. A survey on small-area planar graph drawing. 2014. arXiv:1410.1006.

[27] Giuseppe Di Battista and Roberto Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science*, 61(2-3):175–198, 1988.

[28] Giuseppe Di Battista and Roberto Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25(5):956–997, 1996.

[29] Giuseppe Di Battista, Roberto Tamassia, and Ioannis G. Tollis. Area requirement and symmetry display of planar upward drawings. *Discrete & Computational Geometry*, 7(1):381–401, 1992.

[30] Reinhard Diestel. *Graph Theory, 5th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2017.

[31] James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989.

[32] Piere Duchet, Yahya Ould Hamidoune, Michel Las Vergnas, and Henry Meyniel. Representing a planar graph by vertical lines joining different levels. *Discrete Mathematics*, 46(3):319–321, 1983.

[33] Hicham El-Zein, J. Ian Munro, and Matthew Robertson. Raising permutations to powers in place. In *27th International Symposium on Algorithms and Computation (ISAAC 2016)*, volume 64 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:12, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[34] István Fáry. On straight line representation of planar graphs. *Acta Universitatis Szegediensis Sectio Scientiarum Mathematicarum*, 11:229–233, 1948.

[35] Faith E. Fich, J. Ian Munro, and Patricio V. Poblete. Permuting in place. *SIAM J. Comput.*, 24(2):266–278, April 1995.

[36] Hubert de Fraysseix, Patrice Ossona de Mendez, and János Pach. Representation of planar graphs by segments. In *Intuitive geometry (Szeged, 1991)*, volume 63 of *Colloquia Mathematica Societatis János Bolyai*, pages 109–117. 1994.

[37] Hubert de Fraysseix, Patrice Ossona de Mendez, and Pierre Rosenstiehl. On triangle contact graphs. *Combinatorics, Probability & Computing*, 3:233–246, 1994.

[38] Hubert de Fraysseix, János Pach, and Richard Pollack. Small sets supporting Fáry embeddings of planar graphs. In *STOC 1988: 20th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, May 1988. Proceedings*, pages 426–433, 1988.

[39] Hubert de Fraysseix, János Pach, and Richard Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

[40] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[41] Ashim Garg and Roberto Tamassia. Upward planarity testing. *Order*, 12(2):109–133, 1995.

[42] Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM Journal on Computing*, 31(2):601–625, 2001.

[43] Subir K. Ghosh and Partha P. Goswami. Unsolved problems in visibility graphs of points, segments, and polygons. *ACM Computing Surveys*, 46(2):22:1–29, 2013.

[44] Carsten Gutwenger and Petra Mutzel. A linear time implementation of SPQR-trees. In *GD 2000: 8th International Symposium on Graph Drawing, Colonial Williamsburg, VA, USA, September 2000. Proceedings*, pages 77–90, 2001.

[45] Grzegorz Guśpiel. Complexity of finding perfect bipartite matchings minimizing the number of intersecting edges. 2017. arXiv:1709.06805.

[46] Grzegorz Guśpiel. An in-place, subquadratic algorithm for permutation inversion. 2019. arXiv:1901.01926.

[47] Grzegorz Guśpiel and Grzegorz Gutowski. Universal targets for homomorphisms of edge-colored graphs. *Journal of Combinatorial Theory, Series B*, 127:53–64, 2017.

[48] Seifollah L. Hakimi. On the degrees of the vertices of a directed graph. *Journal of the Franklin Institute*, 279(4):290–308, 1965.

[49] Irith Ben-Arroyo Hartman, Ilan Newman, and Ran Ziv. On grid intersection graphs. *Discrete Mathematics*, 87(1):41–52, 1991.

[50] Xin He, Jiun-Jie Wang, and Huaming Zhang. Compact visibility representation of 4-connected plane graphs. *Theoretical Computer Science*, 447:62–73, 2012.

[51] Goos Kant and Xin He. Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. *Theoretical Computer Science*, 172(1-2):175–193, 1997.

[52] Goos Kant, Giuseppe Liotta, Roberto Tamassia, and Ioannis G. Tollis. Area requirement of visibility representations of trees. *Information Processing Letters*, 62(2):81–88, 1997.

[53] Pavel Klavík, Jan Kratochvíl, Tomasz Krawczyk, and Bartosz Walczak. Extending partial representations of function graphs and permutation graphs. In *ESA 2012: 20th Annual European Symposium on Algorithms, Ljubljana, Slovenia, September 2012. Proceedings*, pages 671–682, 2012.

[54] Pavel Klavík, Jan Kratochvíl, Yota Otachi, Ignaz Rutter, Toshiki Saitoh, Maria Saumell, and Tomáš Vyskočil. Extending partial representations of proper and unit interval graphs. *Algorithmica*, 77:1071–1104, 2017.

[55] Pavel Klavík, Jan Kratochvíl, Yota Otachi, and Toshiki Saitoh. Extending partial representations of subclasses of chordal graphs. *Theoretical Computer Science*, 576:85–101, 2015.

[56] Pavel Klavík, Jan Kratochvíl, Yota Otachi, Toshiki Saitoh, and Tomáš Vyskočil. Extending partial representations of interval graphs. *Algorithmica*, pages 1–23, 2016.

[57] Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., 1997.

[58] Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison Wesley Longman Publishing Co., Inc., 1997.

[59] Paul Koebe. *Kontaktprobleme der konformen Abbildung*. Hirzel, 1936.

[60] Fabrizio Luccio, Silvia Mazzone, and Chak-Kuen Wong. A note on visibility graphs. *Discrete Mathematics*, 64(2-3):209–219, 1987.

[61] Tillmann Miltzow. Subset token swapping on a path and bipartite minimum crossing matchings. Order & Geometry Workshop, Gułtowy Palace, September 13 – 17, 2016, Problem booklet. `http://orderandgeometry2016.tcs.uj.edu.pl/docs/OG2016-Problem-Booklet.pdf`.

[62] Tillmann Miltzow, Lothar Narins, Yoshio Okamoto, Günter Rote, Antonis Thomas, and Takeaki Uno. Approximation and hardness of token swapping. In *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 66:1–66:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[63] Bojan Mohar. A polynomial time circle packing algorithm. *Discrete Mathematics*, 117(1-3):257–263, 1993.

[64] Eugene W. Myers. Efficient applicative data types. In *POPL 84: 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, Salt Lake City, UT, USA, January 1984. Proceedings*, pages 66–75, 1984.

[65] Jaroslav Nešetřil and Patrice Ossona de Mendez. Colorings and homomorphisms of minor closed classes. In *Discrete and Computational Geometry: The Goodman-Pollack Festschrift*, volume 25 of *Algorithms and Combinatorics*, pages 651–664. 2003.

[66] Jaroslav Nešetřil and André Raspaud. Colored homomorphisms of colored mixed graphs. *Journal of Combinatorial Theory, Series B*, 80(1):147–155, 2000.

[67] Ralph H. J. M. Otten and J. G. van Wijk. Graph representations in interactive layout design. In *IEEE International Symposium on Circuits and Systems, New York, NY, USA, May 1978. Proceedings*, pages 914–918, 1978.

[68] Maurizio Patrignani. On extending a partial straight-line drawing. *International Journal of Foundations of Computer Science*, 17(5):1061–1070, 2006.

[69] André Raspaud and Eric Sopena. Good and semi-strong colorings of oriented planar graphs. *Inf. Process. Lett.*, 51(4):171–174, August 1994.

[70] Matthew Robertson. *Inverting permutations in place*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 2015.

[71] Martine Schlag, Fabrizio Luccio, Piero Maestrini, Der-Tsai Lee, and Chak-Kuen Wong. A visibility problem in VLSI layout compaction. *Advances in Computing Research*, 2:259–282, 1985.

[72] James A. Storer. On minimal-node-cost planar embeddings. *Networks*, 14(2):181–212, 1984.

[73] Roberto Tamassia and Ioannis G. Tollis. A unified approach to visibility representations of planar graphs. *Discrete & Computational Geometry*, 1(4):321–341, 1986.

[74] Jiun-Jie Wang and Xin He. Visibility representation of plane graphs with simultaneous bound for both width and height. *Journal of Graph Algorithms and Applications*, 16(2):317–334, 2012.

[75] Stephen K. Wismath. Characterizing bar line-of-sight graphs. In *SCG 1985: 1st Annual Symposium on Computational Geometry, Baltimore, MD, USA, June 1985. Proceedings*, pages 147–152, 1985.

[76] Katsuhisa Yamanaka, Erik D. Demaine, Takehiro Ito, Jun Kawahara, Masashi Kiyomi, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Kei Uchizawa, and Takeaki Uno. Swapping labeled tokens on graphs. *Theoretical Computer Science*, 586:81 – 94, 2015. Fun with Algorithms.

[77] Blog discussion. `https://codeforces.com/blog/entry/50500`. Online; accessed May 1, 2019.