Minimum Linear Arrangement of Series-Parallel Graphs

Martina Eikel, Christian Scheideler, and Alexander Setzer(✉)

University of Paderborn, Paderborn, Germany
{martinah,scheideler,asetzer}@mail.upb.de

Abstract. We present a factor $14D^2$ approximation algorithm for the minimum linear arrangement problem on series-parallel graphs, where D is the maximum degree in the graph. Given a suitable decomposition of the graph, our algorithm runs in time $O(|E|)$ and is very easy to implement. Its divide-and-conquer approach allows for an effective parallelization. Note that a suitable decomposition can also be computed in time $O(|E| \log |E|)$ (or even $O(\log |E| \log^* |E|)$ on an EREW PRAM using $O(|E|)$ processors).

For the proof of the approximation ratio, we use a sophisticated charging method that uses techniques similar to amortized analysis in advanced data structures.

On general graphs, the minimum linear arrangement problem is known to be NP-hard. To the best of our knowledge, the minimum linear arrangement problem on series-parallel graphs has not been studied before.

1 Introduction

The minimum linear arrangement problem is a well-known graph embedding problem, in which an arbitrary graph is mapped onto the line topology, such that the sum of the distances of nodes that share an edge is minimized. We consider the class of series-parallel graphs, which arises naturally in the context of parallel programs: modelling the execution of a parallel program yields a series-parallel graph, where sources of parallel compositions represent fork points, and sinks of parallel compositions represent join points (for the definition of a parallel composition, see Subsect. 1.1). Note that in this context, series-parallel graphs typically have a very low node degree: Since spawning child processes is costly, one would usually not spawn too many of them at a time.

1.1 Problem Statement and Definitions

Throughout this work, we consider undirected graphs only. The following definition of the minimum linear arrangement problem is based on [22]:

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901) and by the EU within FET project MULTIPLEX under contract no. 317532.

Definition 1 (Linear Arrangement). *Given a graph $G = (V, E)$, let $n = |V|$. A* linear arrangement *π of G is a one-to-one function*

$$\pi : V \to \{1, \dots, n\}.$$

For a node $v \in V$, $\pi(v)$ is also called the position of v *in π.*

Definition 2 (Cost of a Linear Arrangement). *Given a graph $G = (V, E)$ and a linear arrangement π of G, we denote the* cost *of π by*

$$COST_\pi(G) := \sum_{\{u,v\} \in E} |\pi(u) - \pi(v)|.$$

Definition 3 (Minimum Linear Arrangement Problem). *Given a graph $G = (V, E)$ (the* input graph*), the* minimum linear arrangement problem *(*minLA*) is to find a linear arrangement π that minimizes $COST_\pi(G)$.*

Next we define the class of series-parallel graphs, (the following is based on [11]):

Two-terminal Graph (TTG). A *two-terminal graph* $G = (V, E)$ is a graph with node set V, edge set E, and two distinct nodes $s_G, t_G \in V$ that are called source and sink, respectively. s_G and t_G are also called the *terminals* of G.

Series Composition. The *series composition* SC of $k \geq 2$ TTGs $X_1, \dots, X_k$ is a TTG created from the disjoint union of $X_1, \dots, X_k$ with the following characteristics: The sink t_{X_i} of X_i is merged with the source $s_{X_{i+1}}$ of X_{i+1} for $1 \leq i < k$. The source s_{X_1} of X_1 becomes the source s_{SC} of SC and the sink t_{X_k} of X_k becomes the sink t_{SC} of SC.

Parallel Composition. The *parallel composition* PC of $k \geq 2$ two-terminal graphs $X_1, \dots, X_k$ is a TTG created from the disjoint union of $X_1, \dots, X_k$ with the following two characteristics: The sources $s_{X_1}, \dots, s_{X_k}$ are merged to create s_{PC} and the sinks $t_{X_1}, \dots, t_{X_k}$ are merged to create t_{PC}.

Two-terminal Series-Parallel Graph (TTSPG). A *two-terminal series-parallel graph* G with source s_G and sink t_G is a graph that may be constructed by a sequence of series and parallel compositions starting from a set of copies of a single-edge two-terminal graph $G' = (\{s, t\}, \{\{s, t\}\})$.

Series-Parallel Graphs. A graph G is a *series-parallel graph* if, for some two distinct nodes s_G and t_G in G, G can be regarded as a TTSPG with source s_G and sink t_G.

Note that the series and parallel compositions are commonly defined over two input graphs only. However, it is not hard to see that our definition of a series-parallel graph is equivalent.

An example of a series-parallel graph is shown in Fig. 1.

1.2 Related Work

The minLA was first stated by Harper [18]. Garey, Johnson, and Stockmeyer were the first to prove its NP-hardness on general graphs [16]. Ambühl, Mastrolilli, and Svensso showed that the minLA on general graphs does not have a polynomial-time approximation scheme unless NP-complete problems can be solved in randomized subexponential time [3]. To the best of our knowledge, the two best polynomial-time approximation algorithms for the minLA on general graphs are due to Charikar, Hajiaghayi, Karloff, and Rao [6], and Feige and Lee [13]. Both algorithms yield an $O(\sqrt{\log n}\log\log n)$-approximation of the minLA. The latter algorithm is a combination of techniques of earlier works by Rao and Richa [24], and Arora, Rao, and Vazirani [4]. For planar graphs (which include the series-parallel graphs), Rao and Richa [24] also present a $O(\log\log n)$-approximation algorithm. Note that even though, for high degree graphs, these algorithms achieve a better approximation factor than the one we present in this work, there are some key differences between these algorithms and ours: First of all, the algorithm we present is a very simple divide-and-conquer algorithm and its functioning can be understood easily. The aforementioned algorithms, however, are much more complex and involve solving a linear or semidefinite program. Furthermore, our algorithm achieves a runtime of only $O(|E|)$ (if the series-parallel graph is given in a suitable format - otherwise, a more complex preprocessing is required that takes time $O(|E|\log|E|)$, but this can be parallelized down to $O(\log|E|\log^*|E|))$ making it suitable in situations where a low runtime is more important than the approximation guarantee. Still, for low graph degrees (which are reasonable to assume in certain applications), our algorithm even improves the approximation factor of Rao and Richa.

For special classes of graphs, the NP-hardness has been shown for bipartite graphs [12], interval graphs, and permutation graphs [8]. On the other hand, polynomial-time optimal algorithms have been found for hypercubes [18], trees [7], d-dimensional c-ary cliques [21], meshes [14], and chord graphs [25]. Note that many people claim that the minLA is optimally solvable on outerplanar graphs, referring to [15]. However, the problem solved in [15] is different from the minLA as we show in [26]. Note that the question whether the minLA is NP-hard on series-parallel graphs is unsettled.

Applications of the minLA include the design of error-correcting codes [18], machine job scheduling (e.g., [2]), VLSI layout (e.g., [1,9]), and graph drawing (e.g., [27]). For an overview of heuristics for the minLA see the survey paper by Petit [23].

The class of series-parallel graphs, first used by MacMahon [20], has been studied extensively. It turns out that many problems that are NP-complete on general graphs can be solved in linear time on series-parallel graphs. Among these are the decision version of the dominating set problem [19], the minimum vertex cover problem, the maximum outerplanar subgraph problem, and the maximum matching problem [28]. Furthermore, since the class of series-parallel graphs is a subclass of the class of planar graphs, any problem that is already in P for that class of graphs can be solved optimally in polynomial time for series-parallel graphs as well (such as the max-cut problem [17]).

Another problem regarding series-parallel graphs is to decide, given an input graph G, whether it is series-parallel and, if so, to output the operations that recursively constructed the series-parallel graph. The first step is referred to as *series-parallel graph recognition* while the second step is referred to as *constructing a decomposition tree.* A parallel linear-time algorithm for this problem on directed graphs was first presented by Valdes, Tarjan, and Lawler [29]. Later, Eppstein [11] developed a parallel algorithm for undirected graphs using a so-called *nested ear decomposition.* The concept of an S-decomposition used in our analysis is technically similar to that concept, though we use a different notation more suitable for our purposes. The algorithm we propose for approximating the minLA on series-parallel graphs also relies on a decomposition tree. For instances in which it is not given, the algorithm by Bodlaender and De Fluiter [5] can be used, since it runs on undirected graphs and outputs so-called *SP-tree* , which can be easily transformed into a format suitable for our algorithm.

1.3 Our Contribution

We describe a simple approximation algorithm for the minimum linear arrangement problem on series-parallel graphs with an approximation ratio of $14D^2$, where D is the degree of the graph, and a running time of $O(|E|)$ if the series-parallel graph is given in a suitable format. If the series-parallel graph is not given in the required format, this format can be computed in time $O(|E| \log |E|)$ (which can even be further parallelized down to $O(\log |E| \log^* |E|)$ on an EREW PRAM using $O(|E|)$ processors). However, for certain applications it is reasonable to assume that the graph is given in the right format, e.g., when the series-parallel graph is used to model the execution of a parallel program, the desired representation can be constructed along with the model. The simplicity and the structure of the algorithm allow for an efficient distributed implementation.

Moreover, our proof of the approximation ratio introduces a sophisticated charging method following an approach that is known from the amortized analysis of advanced data structures. This technique may be applied in other analyses as well.

2 Preliminaries

The algorithm we present is defined recursively and is based on a decomposition of the series-parallel graph into components. Therefore, prior to describing the algorithm, we introduce several definitions needed to formalize this decomposition.

The following definition is similar to the one in [5].

Definition 4 (SP-tree, Minimal SP-tree). *An SP-tree T of a series-parallel graph G is a rooted tree with the following properties:*

1. *Each node in T corresponds to a two-terminal subgraph of G.*
2. *Each leaf is a so-called L-node labelled as $L(k)$ and corresponds to a path with k edges.*
3. *Each inner node is a so-called S-node or P-node, and the two-terminal subgraph G' associated with an S-node (P-node) is the graph obtained by a series (parallel) composition of the graphs associated with the children of G', where*

the order of the children defines the order in which the series composition is applied (the order does not matter for a parallel composition).

4. The root node corresponds to G.

An SP-tree T of a series-parallel graph G is called minimal *if the following two conditions hold:*

1. All children of an S-node are either P-nodes or L-nodes, but at least one is a P-node.
2. All children of a P-node are either S-nodes or L-nodes.

It is easy to see that for any fixed series-parallel graph G, there exists a minimal SP-tree for G.

We are now ready to introduce the following three important notions:

Definition 5 (Simple Node Sequence, Parallel Component, Series Component). *Let G be a series-parallel graph and T be a minimal SP-tree of G. The sub-graph of G associated with a leaf $L(k)$ of T for $k \in \mathbb{N}$ is called a* simple node sequence. *The sub-graph of G associated with a P-node is called a* parallel component *of G. The sub-graph of G associated with a S-node is called a* series component *of G. Furthermore, any simple node sequence is called a* series component*, too.*

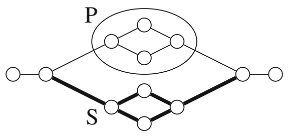

Fig. 1. Example of a simple series-parallel graph. P is a parallel component consisting of two series components (more precisely, two simple node sequences with two edges each). The thick edges belong to the subgraph induced by the series component S. It consists of two single-edge simple node sequences (on the left and right end) and a parallel component.

An illustration of the different types of components is given by Fig. 1. The definition of a minimal SP-tree implies the following: Each parallel component P is the result of a parallel composition of two or more series components. Furthermore, each series component S is the result of a series composition of two or more parallel components or simple node sequences, but not exclusively simple node sequences. This leads to the following definition:

Definition 6 (Child Component). *Let G be a series-parallel graph, let T be a minimal SP-tree, and let X and Y be two nodes in T such that Y is a child of X. Further, let C_i be the (series or parallel) component that is associated with Y and let C be the (parallel or series) component C that is associated with X. Then, C_i is called a* child component *of C, and we say: $C_i \in C$.*

For example, the two simple node sequences that induce the parallel component P in Fig. 1 are child components of P. One implication of this definition is that the terminals of a parallel component and its child components overlap.

For the rest of this work, we assume that for any fixed series-parallel graph G, the simple node sequences, series components and parallel components of G are uniquely defined by a fixed minimal SP-tree T. In the full version [10], we describe an efficient method to compute a minimal SP-tree according to our definition. It is basically an extension of an algorithm by Bodlaender and de Fluiter [5].

3 The Series-Parallel Graph Arrangement Algorithm

The Series-Parallel Graph Arrangement Algorithm (SPGAA) is defined recursively. In order to arrange the nodes of a series or parallel component C, the SPGAA first determines the order of its child components recursively, and then places the child components side by side in an order that depends on their size. For any component C, when the algorithm has just arranged the nodes of C, it holds that its source receives the leftmost position among all nodes of C and that its sink receives the position directly to the right of the source. However, later computations (in a higher recursion level) may re-arrange the terminals and pull them apart. More specific details are given in the corresponding subsections for the different types of components. Illustrations of all arrangements and all different cases can be found in the full version [10].

3.1 Arrangement of a Simple Node Sequence

For any simple node sequence L, we label the nodes of L from left to right by 1 to k. That is, the source receives label 1 and the sink receives label k. The arrangement of this sequence then is: $1, k, 2, k-1, 3, k-2, \dots$. One can see that this arrangement fulfills the property that the source is on the leftmost position and that the sink is its right neighbor.

3.2 Arrangement of a Parallel Component

For any parallel component P with source u, sink v, and $m \geq 2$ child components $S_1, S_2, \dots, S_m$ (note that any parallel component has at least two child components), the SPGAA recursively determines the arrangement of the child components. We denote the computed arrangement of S_i excluding the two terminal nodes (which would have been placed at the first two positions of the arrangement, see Subsect. 3.3) by S_i^-. W.l.o.g. let S_m be a biggest child component (w.r.t. the number of nodes in it). Then, the algorithm places u at the first position, v at the second position, the nodes of S_1^- to the right of that (in their order), and the nodes of S_i^- to the right of S_{i-1}^- for $i \in \{2, \dots, m\}$.

3.3 Arrangement of a Series Component

For any series component S with source u, sink v, and $m \geq 2$ child components $P_1, P_2, \ldots, P_m$ (note that any series component has at least two child components, otherwise it would be a simple node sequence), the SPGAA first recursively determines the arrangement of the child components. Second, it puts u and v at the first two positions, in this order. The third step differs from the case of a parallel component: To keep the cost of the arrangement low while ensuring that a biggest child component P_a receives the rightmost position, the general order of the child components is: $P_1, P_2, \ldots, P_{a-1}, P_m, P_{m-1}, \ldots, P_{a+2}, P_{a+1}, P_a$. Here, the components from P_m to P_{a+1} are flipped (the order of their nodes is reversed). For $m = a$, the order is $P_1, P_2, \ldots, P_m$ and for $a = 1$, the components are ordered in reverse (i.e., $P_m, P_{m-1}, \ldots, P_1$) (where all components except for P_1 are flipped).

However, since each two neighboring child components P_i and P_{i+1} share a (terminal) node, it must be decided which of the two components may "keep" its node. The strategy here is as follows: Each component P_i (except for the first component, whose source has received the leftmost position already) keeps its source and lends its sink to P_{i+1} (of which it is a source), except for P_m (whose sink has been placed at the second position already). This may stretch existing edges, which we will keep track of in the analysis.

An illustration of the arrangement for the case $1 < a < m$ can be found in Fig. 2.

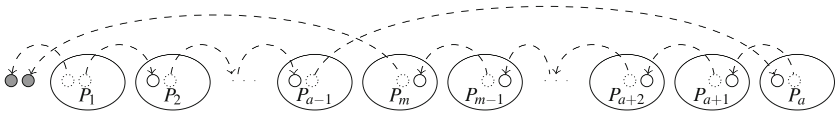

Fig. 2. Order in which the SPGAA arranges a series component consisting of m child components for $1 < a < m$ (where P_a is a biggest component). Dotted nodes indicate the position at which a node would be placed according to the previous recursion level. Dashed arrows indicate the change in position at the current recursion level.

4 Analysis

In this section, we prove the approximation ratio of $14 \cdot D^2$ for the Series-Parallel Graph Arrangement Algorithm described in Sect. 3. As a first step, we provide lower bounds on the *amortized cost* in an optimal arrangement for each kind of component. The amortized cost of a component is the sum of two values: First, the exclusive cost of this component (cost of the current component minus the individual cost of all child components). Second, some cost that has been accounted for in a lower recursion level. This cost is chosen such that the sum of all amortized costs does not contain this cost more than three times. We use these bounds to establish a lower bound on the total cost of an optimal solution. The details are described in Subsect. 4.2. As a second step, we state upper bounds on the exclusive costs generated at each recursion step of the SPGAA in order to determine an upper bound on the total cost in Subsect. 4.3. Last, we use both the lower bound as well as the upper bound to relate the cost of an optimal arrangement to that of an arrangement computed by the SPGAA. This is done in Subsect. 4.4. In addition to providing the approximation ratio of the SPGAA, we establish a polynomial runtime bound of our algorithm in Subsect. 4.5. Note that all the proofs in this section can be found in the full version [10].

4.1 Prerequisites

For the analysis, we need several notions, which we now introduce.

Definition 7 (Length of an Edge). *Given a graph $G = (V, E)$ and a linear arrangement π of G, let $u, v \in E$. The* length of (u,v) in π, *denoted by $length_\pi(u, v)$ is defined as:*

$$length_\pi(u, v) = |\pi(u) - \pi(v)|.$$

Definition 8. *Given a linear arrangement π of a series-parallel graph $G = (V, E)$ and a (series or parallel) component C in G, we define:*

Restricted Arrangement. *The* arrangement π cted to C, *denoted by $\pi(C)$ is obtained by removing all nodes from π that do not belong to C, as well as their incident edges, i.e., $\pi(C)$ maps the nodes from C to $\{1, \ldots, |C|\}$.*

Restricted Length of an Edge. *For any edge (u, v) that belongs to C, the* length of (u,v) restricted to C, *denoted by $length_{\pi(C)}$*(u,v), *is the distance between u and v in $\pi(C)$.*

Restricted Cost of an Arrangement. *Let E_C be the set of all edges from G whose both endpoints are in C. The* cost of C restricted to C, *denoted by R-$COST_\pi(C)$, is defined as:*

$$R\text{-}COST_\pi(C) := \sum_{(u,v)\in E_C} length_{\pi(C)}(u, v).$$

Definition 9 (Exclusive Cost of a Series/Parallel Component). *Given a linear arrangement π of a series-parallel graph G and a (series or parallel) component C in G containing $m \geq 0$ child components $C_1, \ldots, C_m$, the* exclusive cost of C in π, *denoted by E-$COST_\pi(C)$, is defined as*

$$E\text{-}COST_\pi(C) := R\text{-}COST_\pi(C) - \sum_{i=1}^{m} R\text{-}COST_\pi(C_i).$$

Note that the exclusive cost of a simple node sequence S is equal to the restricted cost of S.

We can make the following observation regarding the relationship between the exclusive costs of the components and the total cost:

Observation 1. *Let G be a series-parallel graph and let π be a linear arrangement of G. Further, let $\mathcal{C}$ be the set of all (series or parallel) components in G. It holds:*

$$\sum_{C \in \mathcal{C}} E\text{-}COST_\pi(C) = COST_\pi(G).$$

In the analysis of the SPGAA, we need to find at least one path from s_P to t_P through P for each parallel component P such that any two such paths are edge-disjoint for two different parallel components. Therefore, we introduce the following notion of an *S-decomposition*, which yields these paths and is recursively defined as follows:

Definition 10 (A-path, S-path, S-decomposition). *Let P be an "innermost" parallel component in a series-parallel graph G (i.e., one whose child components are simple node sequences only) with source s, sink t, and k child components. Select an arbitrary simple path from s to t through P (i.e., select one of the simple node sequences). This path is called the* auxiliary path *or simply* A-path *of P. The remaining paths from s to t through P are called the* selected paths *or simply* S-paths *of P.*

Recursively, for an arbitrary parallel component P, with source s, sink t, and $m \geq 2$ child components $S_1, \ldots, S_m$, for each child component S_i, $1 \leq i \leq m$, select a simple path Q_i from s to t through S_i in the following way: If S_i is a simple node sequence, Q_i is the whole sequence. Otherwise, S_i is a series component, which consists of $k \geq 0$ simple node sequences and $l \geq 1$ parallel components (note that $k + l \geq 2$). Denote these child components by $P_1, \ldots P_{k+l}$ in the order in which they appear in S_i. Construct the path Q_i step by step: Start with P_1 and add P_1 completely to Q_i if P_1 is a simple node sequence. If, however, P_1 is a parallel component, select the A-path of P_1 and extend Q_i by it. Continue in the same manner up to P_{k+l}. After this, the whole path Q_i is constructed. Q_1 is called the A-path *of P and the remaining paths $Q_2, \ldots, Q_m$ are called the* S-paths *of P.*

The selection of S-paths (and A-paths accordingly) for all parallel components of G is called an S-decomposition *of G.*

An example of an S-decomposition can be found in the full version [10].

Intuitively, an auxiliary path of a parallel component P_j is a path through the whole component which is *reserved* to be used in higher recursion levels (to eventually become part of an S-path there). Any edges of an S-path are not used for any S-path or A-path in any higher recursion level.

The main contribution of the S-decomposition is that it gives a mapping from parallel components to paths through the respective components (the S-paths) such that all these paths are edge-disjoint. More formally:

Lemma 1. *For each series-parallel graph G, there exists an S-decomposition SD. Besides, in any S-decomposition, each edge belongs to at most one S-path in SD.*

Provided with the definition of an S-decomposition, we are ready to define the amortized cost as follows:

Definition 11 (Amortized Cost). *Let π_{OPT} be an (optimal) linear arrangement of a series-parallel graph G, let SD be an S-decomposition of G, and let S be a series component in G. Further, let E_S be the set that contains all edges of simple node sequences that are child components of S and all edges of S-paths of the child components of S that are parallel components. The* amortized cost *of S, denoted by $A\text{-}COST_{\pi_{OPT}}(S)$, is defined as:*

$$A\text{-}COST_{\pi_{OPT}}(S) := E\text{-}COST_{\pi_{OPT}}(S) + \sum_{\{x,y\}\in E_S} length_{\pi_{OPT}(S)}(x,y).$$

For any parallel component P in a series-parallel graph G and any optimal linear arrangement π_{OPT},

$$A\text{-}COST_{\pi_{OPT}}(P) := E\text{-}COST_{\pi_{OPT}}(P).$$

Note that the addend in the amortized cost for simple node sequences is zero (as the set E_S is empty in this case).

This definition will be helpful for the analysis of the minimum cost of an optimal arrangement. The amortized cost adds a certain value to the exclusive cost of a (series or parallel) component C, with the following property:

Lemma 2. *Let $G = (V, E)$ be a series-parallel graph, and π_{OPT} be an (optimal) linear arrangement for G. Further, let $\mathcal{C}$ be the set of all (series or parallel) components of G. It holds:*

$$\sum_{C\in\mathcal{C}} A\text{-}COST_{\pi_{OPT}}(C) \leq 3 \cdot \sum_{C\in\mathcal{C}} E\text{-}COST_{\pi_{OPT}}(C).$$

For the analysis of an optimal arrangement, we also need the following notation:

Definition 12 (Δ_C). *Given an (optimal) linear arrangement π_{OPT} of a series-parallel graph G, and a (series or parallel) component C in G, consider π_{OPT} restricted to C. We denote the smallest number of nodes to the left or to the right (depending on which number is smaller) of a terminal node of C in $\pi_{OPT}(C)$ by Δ_C.*

It is convenient to define:

Definition 13 (Cardinality of a Component). *For any series-parallel graph G and any (series or parallel) component C in G: $|C|$ is the number of nodes in C, $|C^{\ominus}|$ is the number of all nodes in C without the sink of C, and $|C^{-}|$ is the number of nodes in C without the two terminal nodes of C.*

4.2 A Lower Bound on the Total Cost of Optimal Solutions

In this subsection, we give lower bounds on the amortized costs of an optimal arrangement for simple node sequences, parallel components, and series components. In the end, we consolidate the results and state a general lower bound on the total cost of an optimal arrangement. For the proofs, we refer to the full version [10].

First of all, the following is a simple result about simple node sequences:

Lemma 3. *For any simple node sequence L in a series-parallel graph G in an optimal arrangement π_{OPT}, it holds:*

$$A\text{-}COST_{\pi_{OPT}}(L) \geq |L| - 1 + \Delta_L.$$

We now provide a lower bound on the amortized cost of series components:

Lemma 4. *For any series component S in a series-parallel graph G with $m \geq 2$ child components $P_1, \ldots, P_m$ in an optimal arrangement π_{OPT}, it holds:*

$$A\text{-}COST_{\pi_{OPT}}(S) \geq \frac{1}{2}\left(\sum_{i=1}^{m} |P_i^{\ominus}| - \max_i |P_i^{\ominus}|\right) + 1 + \sum_{i=1}^{m} \Delta_{P_i} - \Delta_S.$$

For the amortized cost of parallel components, we have the following result:

Lemma 5. *For any parallel component P in a series-parallel graph G with $m \geq 2$ child components $S_1, \ldots, S_m$, in an optimal arrangement π_{OPT}, it holds:*

$$A\text{-}COST_{\pi_{OPT}}(P) \geq \frac{1}{2}\left(\sum_{i=1}^{m} |S_i^{-}| - \max_i |S_i^{-}|\right) + \sum_{i=1}^{m} \Delta_{S_i} - \Delta_P.$$

These three lower bounds for the different types of components in any series-parallel graph can be combined into a single lower bound:

Corollary 1. *Let $G = (V, E)$ be an arbitrary series-parallel graph and π_{OPT} an optimal arrangement of G. Further, denote the total cost of π_{OPT} by $COST_{\pi_{OPT}}(G)$, the set of simple node sequences in G by L_G, the set of parallel components by P_G, the set of series components by S_G. Then, it holds:*

$$7 \cdot COST_{\pi_{OPT}}(G) \geq \sum_{L \in L_G} 2 \cdot (|L| - 1) + \sum_{P \in P_G} \left(\sum_{S_i \in P} |S_i^{-}| - \max_{S_i \in P} |S_i^{-}|\right) + \sum_{S \in S_G} \left(\sum_{P_i \in S} |P_i^{\ominus}| - \max_{P_i \in S} |P_i^{\ominus}|\right).$$

4.3 An Upper Bound on the Total Cost of SPGAA Arrangements

For the approximation ratio of the SPGAA, we also need to find an upper bound on the cost of arrangements computed by the SPGAA. One can show the following result:

Lemma 6. *Let $G = (V, E)$ be an arbitrary series-parallel graph and let π_{ALG} be an arrangement of G computed by the SPGAA. Furthermore, denote the total cost of π_{ALG} by $COST_{\pi_{ALG}}(G)$, the set of simple node sequences in G by L_G, the set of parallel components by P_G, the set of series components by S_G. Then, it holds:*

$$COST_{\pi_{ALG}}(G) \leq \sum_{L \in L_G} 2 \cdot (|L| - 1) + \sum_{P \in P_G} 2D^2 \cdot \left(\sum_{S_i \in P} |S_i^{-}| - \max_{S_i \in P} |S_i^{-}|\right) + \sum_{S \in S_G} 2D \cdot \left(\sum_{P_i \in S} |P_i^{\ominus}| - \max_{P_i \in S} |P_i^{\ominus}|\right).$$

4.4 The Approximation Ratio of $14 \cdot D^2$

Finally, based on the groundwork of the previous subsections, proving the main theorem of this chapter is straightforward.

Theorem 2. *For a series-parallel graph G, let π_{ALG} be the linear arrangement of G computed by the SPGAA, and let π_{OPT} be an optimal linear arrangement of G. It holds:*

$$COST_{\pi_{ALG}}(G) \leq 14 \cdot D^2 \cdot COST_{\pi_{OPT}}(G).$$

4.5 Runtime

Regarding the runtime of the SPGAA, one can show the following result:

Theorem 3. *On a series-parallel graph $G = (V, E)$, the SPGAA has a runtime of $O(|E|)$ if a minimal SP-tree of G is given as an input, and a runtime of $O(|E| \log |E|)$ otherwise.*

References

1. Adolphson, D., Hu, T.C.: Optimal linear ordering. SIAM J. Appl. Math. **25**(3), 403–423 (1973)
2. Adolphson, D.L.: Single machine job sequencing with precedence constraints. SIAM J. Comput. **6**(1), 40–54 (1977)
3. Ambühl, C., Mastrolilli, M., Svensson, O.: Inapproximability results for sparsest cut, optimal linear arrangement, and precedence constrained scheduling. In: Proceedings of FOCS, pp. 329–337. IEEE Computer Society (2007)
4. Arora, S., Rao, S., Vazirani, U.: Expander flows, geometric embeddings and graph partitioning. J. ACM **56**(2), 5 (2009)
5. Bodlaender, H.L., de Fluiter, B.: Parallel algorithms for series parallel graphs. In: Diaz, J., Serna, M. (eds.) ESA 1996. LNCS, vol. 1136, pp. 277–289. Springer, Heidelberg (1996)
6. Charikar, M., Hajiaghayi, M.T., Karloff, H., Rao, S.: L22 spreading metrics for vertex ordering problems. In: Proceedings of SODA, pp. 1018–1027. Society for Industrial and Applied Mathematics, Philadelphia (2006)
7. Chung, F.R.K.: Labelings of graphs. In: Beineke, L., Wilson, R. (eds.) Selected Topics in Graph Theory, vol. 3, pp. 151–168. Academic Press, New York (1988)
8. Cohen, J., Fomin, F.V., Heggernes, P., Kratsch, D., Kucherov, G.: Optimal linear arrangement of interval graphs. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 267–279. Springer, Heidelberg (2006)
9. Díaz, J., Petit, J., Serna, M.: A survey of graph layout problems. ACM Comput. Surv. **34**(3), 313–356 (2002)
10. Eikel, M., Scheideler, C., Setzer, A.: Minimum linear arrangement of series-parallel graphs (full paper). ArXiv e-prints, October 2014
11. Eppstein, D.: Parallel recognition of series-parallel graphs. Inf. Comput. **98**(1), 41–55 (1992)
12. Even, S., Shiloach, Y.: NP-completeness of several arrangement problems. Department of Computer Science, Technion, Haifa, Israel, Technical Report 43 (1975)

13. Feige, U., Lee, J.R.: An improved approximation ratio for the minimum linear arrangement problem. Inf. Process. Lett. **101**(1), 26–29 (2007)
14. Fishburn, P., Tetali, P., Winkler, P.: Optimal linear arrangement of a rectangular grid. Discret. Math. **213**(1–3), 123–139 (2000)
15. Frederickson, G.N., Hambrusch, S.E.: Planar linear arrangements of outerplanar graphs. IEEE Trans. Circ. Syst. **35**(3), 323–333 (1988)
16. Garey, M., Johnson, D., Stockmeyer, L.: Some simplified NP-complete graph problems. Theoret. Comput. Sci. **1**(3), 237–267 (1976)
17. Hadlock, F.: Finding a maximum cut of a planar graph in polynomial time. SIAM J. Comput. **4**(3), 221–225 (1975)
18. Harper, L.H.: Optimal assignments of numbers to vertices. J. SIAM **12**(1), 131–135 (1964)
19. Kikuno, T., Yoshida, N., Kakuda, Y.: A linear algorithm for the domination number of a series-parallel graph. Discret. Appl. Math. **5**(3), 299–311 (1983)
20. Macmahon, P.A.: The combination of resistances. Electrician **28**, 601–602 (1892)
21. Nakano, K.: Linear layouts of generalized hypercubes. In: van Leeuwen, J. (ed.) WG 1993. LNCS, vol. 790, pp. 364–375. Springer, Heidelberg (1994)
22. Petit, J.: Experiments on the minimum linear arrangement problem. J. Exp. Algorithmics **8**, 2–3 (2003)
23. Petit, J.: Addenda to the survey of layout problems. Bull. EATCS **3**(105), 177–201 (2013)
24. Rao, S., Richa, A.W.: New approximation techniques for some ordering problems. In: Proceedings of SODA, pp. 211–218. Society for Industrial and Applied Mathematics, Philadelphia (1998)
25. Rostami, H., Habibi, J.: Minimum linear arrangement of chord graphs. Appl. Math. Comput. **203**(1), 358–367 (2008)
26. Setzer, A.: The planar minimum linear arrangement problem is different from the minimum linear arrangement problem. ArXiv e-prints, September 2014
27. Shahrokhi, F., Sýkora, O., Székely, L., Vrto, I.: On bipartite drawings and the linear arrangement problem. SIAM J. Comput. **30**(6), 1773–1789 (2001)
28. Takamizawa, K., Nishizeki, T., Saito, N.: Linear-time computability of combinatorial problems on series-parallel graphs. J. ACM **29**(3), 623–641 (1982)
29. Valdes, J., Tarjan, R.E., Lawler, E.L.: The recognition of series parallel digraphs. In: Proceedings of STOC, pp. 1–12. ACM, New York (1979)