Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 17, no. 3, pp. 265–299 (2013)
DOI: 10.7155/jgaa.00294

On the Complexity of Partitioning Graphs for Arc-Flags

Reinhard Bauer Moritz Baum Ignaz Rutter Dorothea Wagner

Karlsruhe Institute of Technology (KIT)

Abstract

Precomputation of auxiliary data in an additional off-line step is a common approach towards improving the performance of shortest-path queries in large-scale networks. One such technique is the *arc-flags algorithm*, where the preprocessing involves computing a partition of the input graph. The quality of this partition significantly affects the speed-up observed in the query phase. It is evaluated by considering the search-space size of subsequent shortest-path queries, in particular its maximum or its average over all queries. In this paper, we substantially strengthen existing hardness results of Bauer et al. and show that optimally filling this degree of freedom is $\mathcal{NP}$-hard for trees with unit-length edges, even if we bound the height or the degree. On the other hand, we show that optimal partitions for paths can be computed efficiently and give approximation algorithms for cycles and trees.

Submitted:	Reviewed:	Revised:	Accepted:	Final:
January 2013	April 2013	May 2013	June 2013	June 2013

Published: June 2013

Article type: Regular Paper — Communicated by: G. Liotta

Partially supported by DFG grant WA 654/16-2, by the Federal Ministry of Economics and Technology under grant no. 01ME12013, and by the EU FP7/2007-2013 (DG CONNECT (Communications Networks, Content and Technology Directorate General), Unit H5 - Smart Cities & Sustainability), under grant agreement no. 288094 (project eCOMPASS) The sole responsibility for the contents of the publication lies with the authors.

E-mail addresses: reinhard.bauer@kit.edu (Reinhard Bauer) moritz.baum@kit.edu (Moritz Baum) ignaz.rutter@kit.edu (Ignaz Rutter) dorothea.wagner@kit.edu (Dorothea Wagner)

1 Introduction

In recent years, route planning has become a widely known application of algorithm engineering. Although Dijkstra's algorithm [8] is of polynomial-time complexity on arbitrary graphs, its performance on large realistic graphs is not acceptable for practical applications. Speed-up techniques that yield improved query times split the work into two parts. In the off-line phase a precomputation step is executed on the input graph to gain additional information about the underlying network. The retrieved data is then used during the on-line phase to improve the performance of shortest-path queries. For a survey of recent approaches exploiting this pattern we refer to Delling et al. [7]. There is a comparatively small number of works that consider theoretical aspects of these techniques [1, 2, 3]. Here, we focus on one particular technique. The idea of *arc-flags* was first introduced by Lauther [11]. The basic approach was exhaustively evaluated in experimental studies, see for example Köhler et al. [10] and Möhring et al. [13]. Moreover, it was combined with other techniques in order to gain additional speed-up [4, 5].

We use the following definition of arc-flags. Given a directed graph $G = (V, E)$ and a partition $\mathcal{C} = \{C_1, \ldots, C_k\}$ of V into *cells*, the arc-flags for a directed edge $e \in E$ consist of k binary flags, where the i-th flag is set if and only if e is part of *some* shortest path to a target node belonging to the cell C_i. In a query to a node t lying in cell C_j, all edges whose j-th flag is not set may safely be ignored, as no shortest path to any node in cell C_j contains e. The preprocessing of the arc-flags algorithm computes a partition $\mathcal{C}$ of the input graph into k cells and determines the corresponding arc-flags. Observe that the flags are uniquely specified by the partition. In particular, the i-th flag of an edge only depends on the nodes contained in cell C_i. Thus, the only degree of freedom in the preprocessing is the choice of $\mathcal{C}$.

Although the outstanding performance of the arc-flags algorithm has been substantiated in many experimental studies, little is known about its theoretical backgrounds. Yet, theoretical analysis is a vital aspect of algorithm engineering. The choice of the partition $\mathcal{C}$ has a large impact on query times in the on-line phase. Bauer et al. prove that it is is $\mathcal{NP}$-hard to compute a partition that minimizes the average search-space size (sss) of on-line queries [2]. However, the graph used in their reduction has a number of properties unlikely to be shared by realistic instances.

1. The graph includes a huge cycle that is an inherent part of the reduction. Since the graph is not acyclic, it does not apply to time-expanded graphs typically used in time-table queries [14].

2. The graph contains substantially differing edge weights.

3. The graph is not strongly connected, and for undirected graphs the complexity is still open.

4. The graph is unusually dense; it contains a quadratic number of edges.

Table 1: Complexity of the two examined problems on different graph classes.

Graph Class	Worst Case directed	Worst Case undirected	Average Case directed	Average Case undirected
Stars	$\mathcal{O}(\lvert V\rvert)$	$\mathcal{O}(\lvert V\rvert)$	$\mathcal{O}(\lvert V\rvert)$	$\mathcal{O}(\lvert V\rvert)$
Trees ($h \leq 2$)	$\mathcal{NPC}$	$\mathcal{NPC}$	$\mathcal{NPC}$	$\mathcal{NPC}$
Paths	$\mathcal{O}(\lvert V\rvert)$	$\mathcal{O}(\lvert V\rvert)$	$\mathcal{O}(\lvert V\rvert)$	$\mathcal{O}(\lvert V\rvert)$
Trees ($\Delta \leq 3$)	$\mathcal{NPC}$	$\mathcal{NPC}$	?	?
Cycles	$\mathcal{O}(\lvert V\rvert)$	$\mathrm{OPT}+1$	$\mathcal{O}(\lvert V\rvert)$	$\mathcal{P}$ [1]

Contributions and Outline. We substantially strengthen known results about the complexity of preprocessing arc-flags. We examine several restricted classes of graphs and establish a border of tractability for this problem. Besides the previously used average sss as a quality measure we also consider the worst-case sss for assessing the quality of partitions. Moreover, we consider directed as well as undirected graphs.

We present preliminaries in Section 2. In Section 3, we show that computing a partition that minimizes the worst-case sss is $\mathcal{NP}$-hard, both for directed and for undirected unit-weight trees. These results hold for binary trees as well as trees with limited height of at most 2. On the other hand, we present a constant-factor approximation algorithm for general trees with arbitrary edge weights. For cycles the number of cells k necessary to bound the sss by a given value W can be approximated within an additive constant of 1. For the average sss, we show that it is $\mathcal{NP}$-hard to compute an optimal partition both for directed and undirected trees in Section 4. These results hold for the case of unit-weight edges and restricted height. For paths an optimal partition can be computed efficiently, and the same holds for cycles if we require cells to be connected. Table 1 shows an overview of our results. We conclude our work and discuss open questions in Section 5.

2 Preliminaries

We assume familiarity with basic concepts from graph theory and shortest-path search; see the book by Cormen et al. [6] for foundations in this area. We consider *directed weighted graphs*, denoted by a triple $G = (V, E, \omega)$, where $\omega \colon E \to \mathbb{R}^+$ is a weight function. If the weight function ω of a graph is not the matter of concern, we omit it from the notation. Our treatment of *undirected graphs* is somewhat non-standard, as depending on the direction of traversal, an undirected edge may have different arc-flags set. Thus, we model undirected edges as a pair of two separate, oppositely oriented edges of the same weight between the endpoints. The *size* $|P|$ of a path $P = \langle v_1, \ldots, v_k \rangle$ is the number k of nodes it contains. The *length* of P is $\omega(P) = \sum_{i=1}^{k-1} \omega(v_i, v_{i+1})$ and the distance between

[1] We present a polynomial-time algorithm that computes optimal *connected* cells.

two nodes s and t is denoted by $d(s,t)$. We say that a cell $C \subseteq V$ is (strongly) connected if the subgraph induced by C is (strongly) connected. A directed tree with root node r is a tree in which all edges point away from r towards the leaves.

Dijkstra's Algorithm, Arc-Flags, and Search Spaces. Dijkstra's algorithm [8] solves the single-source shortest path problem on directed graphs with non-negative edge weights. It manages a priority queue, which initially contains only the source node. In each step, it extracts the node u from the queue with smallest distance label. We say that the node u is *settled* at this time. We assume that each node has a unique index in $\{1, \ldots, |V|\}$ that determines the extracted node if there are two or more nodes with minimum key. Next, any edge (u, v) outgoing from u is *relaxed*, that is, the distance label of v is updated if this edge yields a shorter path from the source node to v via u. In an s-t-query, the algorithm may stop once the target node t is settled (at this point the correct distance as well as a shortest path is known). The query of the arc-flags algorithm modifies this procedure slightly; it relaxes only edges whose flag for the target cell is set, while all other edges are ignored.

Given a graph G and a partition $\mathcal{C}$, the *search space* of an s-t-query is the set of all nodes settled by the query algorithm and its cardinality is denoted by $\mathrm{S}(G, \mathcal{C}, s, t)$. As long as the considered graph is sparse (which holds for realistic instances of street networks), the query time is proportional to $\mathrm{S}(G, \mathcal{C}, s, t)$. Therefore, the sss provides a machine-independent efficiency measure which is also commonly used in experimental studies (see, e.g., Delling et al. [7]). To assess the quality of $\mathcal{C}$ we use either the worst-case efficiency, i.e., $\mathrm{S}_{\max}(G, \mathcal{C}) := \max_{s,t \in V} \mathrm{S}(G, \mathcal{C}, s, t)$ or the average sss over all queries $\mathrm{S}_{\mathrm{avg}}(G, \mathcal{C}) := \sum_{s,t \in V} \mathrm{S}(G, \mathcal{C}, s, t)$. To obtain the actual average sss we would need to divide $\mathrm{S}_{\mathrm{avg}}(G, \mathcal{C})$ by $|V|^2$. Since the corresponding measure only differs by the fixed factor $|V|^2$, we omit this. If G and $\mathcal{C}$ are clear from the context, we may omit both from the notation.

Algorithmic Problems. All reductions in this work are made from the strongly $\mathcal{NP}$-hard problem 3-Partition [9]. An instance of 3-Partition is a tuple (S, B), where B is a positive integer and $S = \{s_1, \ldots, s_{3m}\}$ is a set of $3m$ elements, such that each element s_i is associated with an integer weight $B/4 < \omega_i < B/2$ and $\sum_{i=1}^{3m} \omega_i = mB$. The instance (S, B) is a Yes-instance if and only if there exists a partition of S into m subsets S_j, $j \in \{1, \ldots, m\}$, such that for all j it is $|S_j| = 3$ and the weight of each subset equals B, i.e., $\sum_{s_i \in S_j} \omega_i = B$. Since the problem is strongly $\mathcal{NP}$-hard, we may use unary encodings of the element weights in our reductions. The task considered in this work is to find a partition of a graph that yields low sss. More precisely, given a graph G and a positive integer k, the problems MinWorstCasePartition and MinAvgCasePartition are to find a partition $\mathcal{C}$ with at most k cells that minimizes $\mathrm{S}_{\max}$ or $\mathrm{S}_{\mathrm{avg}}$, respectively.

3 Minimizing the Worst-Case Search-Space Size

In the following, we examine the problem MinWorstCasePartition on certain restricted classes of graphs. We present efficient (approximation) algorithms for paths, stars, and cycles and show $\mathcal{NP}$-hardness for trees with bounded height or out-degrees, respectively. Moreover, we distinguish directed and undirected graphs for all graph classes.

3.1 Trees with Bounded Height

In this section, we examine both directed and undirected trees of limited height. To begin with, consider rooted, directed trees $T = (V, E, \omega)$ with height at most 1, i.e., the class of directed stars. The sss of a query starting at an arbitrary leaf is always 1, because a leaf has no outgoing edges. Hence, the task reduces to minimization of the worst-case sss of all queries from the root node r. Clearly, a query from the root node settles only leaves that are assigned to the target cell. Since these leaves are visited in a deterministic order, each cell C_i of a partition $\mathcal{C} = \{C_1, \dots, C_k\}$ contains a distinct target node t_i such that all nodes of C_i are settled in an r-t_i-query. Additionally, r itself is always settled in a query starting at r, which yields a worst-case sss of $\mathrm{S}_{\max} = 1 + \max_{C_i \in \mathcal{C}} |C_i \setminus \{r\}|$. Obviously, we minimize this number if and only if the cell sizes are balanced.

In what follows, we prove that MinWorstCasePartition becomes $\mathcal{NP}$-hard already if we allow trees of height two. Theorem 1 given below shows $\mathcal{NP}$-hardness even under severe restrictions to the graph structure. Moreover, we obtain a tight border of tractability for the problem MinWorstCasePartition on directed trees.

Theorem 1 MinWorstCasePartition *is $\mathcal{NP}$-hard for rooted directed trees of height* 2*, even in the case of uniform edge weights.*

Proof: We reduce from 3-Partition. Given an instance (S, B) of 3-Partition, we construct (in polynomial time) an instance (T, m) of MinWorstCasePartition as follows. For each element $s_p \in S$, we create a *limb* ℓ_p consisting of one *element node* s_p, $\omega_p - 1$ *weight nodes*, and directed edges from s_p to all its weight nodes. We add a root node r along with directed edges connecting r to all element nodes s_p; see Figure 1 for an example. We claim that (T, m) admits a partition with worst-case sss at most $B + 1$ if and only if (S, B) is a Yes-instance.

Assume (S, B) is a Yes-instance and $S_1, \dots, S_m$ a corresponding solution. Let $\mathcal{C} = \{C_1, \dots, C_m\}$ be the partition where C_i consists of all nodes of limbs corresponding to elements of S_i, and additionally $r \in C_1$. We have $|C_1| = B + 1$ and $|C_i| = B$ for $i \geq 2$. The sss $\mathrm{S}(s, t)$ of an arbitrary s-t-query with $s \neq r$ is bounded by $\lceil B/2 - 1 \rceil$, the maximum size of a limb. Consider queries starting at r. Clearly, a query to an arbitrary target node t never settles nodes outside the cell of t except for r itself. Hence, for queries into any cell $C_i, i \geq 2$, the sss cannot exceed $B + 1$, and the same holds for C_1, as it already contains r.

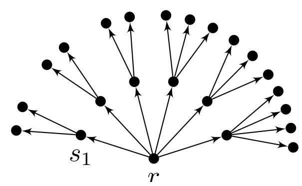


Figure 1: The reduction of an instance with $m = 2, B = 11$ and weights $3, 3, 3, 4, 4, 5$.

Conversely, assume that $\mathcal{C} = \{C_1, \ldots, C_m\}$ is a partition of T inducing a worst-case sss of at most $B+1$. Without loss of generality, assume that $r \in C_1$. We call $\mathcal{C}$ *balanced* if $|C_1| = B + 1$ and $|C_i| = B$ for $i \geq 2$. A limb ℓ_j is *monochromatic* if all its nodes belong to the same cell. A balanced partition containing only monochromatic limbs is called *perfect*. Clearly, a perfect partition corresponds to a solution of 3-Partition and it suffices to show that $\mathcal{C}$ is perfect.

We know that each cell C_i contains a distinct target node t_i such that all nodes of C_i are settled in an r-t_i-query. Together with the fact that r is settled in every such query, this implies that $|C_1| \leq B+1$ and $|C_i| \leq B$ for $i \geq 2$. Since the total number of nodes is $mB + 1$, these conditions must be satisfied with equality, and thus $\mathcal{C}$ is balanced. Now, assume for a contradiction that there is a limb ℓ_p that is not monochromatic, and let s_p be the element node of ℓ_p. Then there exists a weight node of ℓ_p that is assigned to a cell C_i different from the cell of s_p. Now, the query from r to $t_i \in C_i$ settles r, all nodes in C_i and additionally s_p, resulting in a sss of at least $B+2$; a contradiction. Hence, all limbs are monochromatic and the claim follows. The theorem holds since the reduction can clearly be performed in polynomial time. □

Next, we consider undirected trees. In an undirected star, starting from a leaf, the second node that is settled is always the root node. Hence, it again suffices to minimize the worst-case sss of queries from the root node, which was shown to be achieved if the cell sizes are balanced. Using a very similar approach compared to the proof of Theorem 1, we obtain the following hardness result for undirected trees.

Theorem 2 MinWorstCasePartition *is $\mathcal{NP}$-hard for undirected trees with height* 2*, even in case of uniform edge weights.*

Proof: To simplify notation, we denote the worst-case sss of all queries starting at a fixed node $v \in V$ by $\mathrm{S}_{\max}(v) = \max_{t \in V} \mathrm{S}(v, t)$. We use exactly the same reduction as in Theorem 1, except for edges now being undirected. Given an instance (S, B) of 3-Partition, we create a *limb* ℓ_p for each element $s_p \in S$ consisting of an *element node* s_p together with $\omega_p - 1$ *weight nodes* and *undirected* edges from s_p to all its weight nodes. Finally, we add a root node r along with *undirected* edges connecting r to all element nodes s_p. We claim that (T, m) admits a partition with worst-case sss at most $B + 3$ if and only if (S, B) is a Yes-instance of 3-Partition.

Assume that (S, B) is a Yes-instance and let $S_1, \dots, S_m$ be a corresponding solution. Let $\mathcal{C} = \{C_1, \dots, C_m\}$ be the partition of T where C_i consists of all nodes of limbs corresponding to elements of S_i and additionally $r \in C_1$, yielding $|C_1| = B + 1$ and $|C_i| = B$ for $i \geq 2$. Consider an arbitrary query from a source node $s \in V$ to a target node $t \in V$. The only nodes outside the target cell of t that are possibly settled in this query are s_p if $s \in \ell_p$, r and s itself. This yields a worst-case sss of at most $\max_{C_i \in \mathcal{C}} |C_i \cup \{s, s_p, r\}| = B + 3$.

For the other direction, assume that $\mathcal{C} = \{C_1, \dots, C_m\}$ is a partition of T with a worst-case sss of at most $B + 3$. Without loss of generality, assume that $r \in C_1$. Along the lines of Theorem 1, we call $\mathcal{C}$ *balanced* if $|C_1| = B + 1$ and $|C_i| = B$ for $i \geq 2$. A limb ℓ_p is *monochromatic* if all its nodes belong to the same set C_i, and $\mathcal{C}$ is *perfect* if it is balanced and all limbs are monochromatic. Since a perfect partition corresponds to a solution of 3-Partition, it again suffices to show that C is perfect.

Consider an arbitrary s-t-query in T given the partition $\mathcal{C}$. If s and t belong to the same limb ℓ_p, the sss of an s-t query cannot exceed $B/2$, because at most all nodes in ℓ_p and r are settled. Thus, we focus on queries where the s-t-path contains r. Observe that in this case, the query must settle at least all nodes that are settled in an r-t-query. Moreover, the situation for queries that start from the root node r has not changed compared to the directed case treated in Theorem 1, because only edges pointing away from r may lead to unsettled nodes. Hence, we know that in the worst case, a query from r to a certain target node t_i settles all nodes of the target cell C_i, and $\mathrm{S}_{\max}(r)$ equals $B + 1$ if and only if $\mathcal{C}$ is perfect. For now, assume that there exists at least one limb ℓ_p such that none of its nodes are in C_i. Then the sss of a query from an arbitrary leaf $w_{p,q}$ of this limb to t_i is $2 + \mathrm{S}_{\max}(r)$. Thus, we have a worst-case sss of at most $B + 3$ if and only if $\mathcal{C}$ is perfect. To complete the proof, we show that if such a limb ℓ_p does not exist, the worst-case sss must be at least $B + 4$.

Assume there is a cell C_i that contains nodes from each limb ℓ_p in T. Observe that in this case, every edge (r, s_p) has the flag for C_i set. We claim that this implies that $\mathrm{S}_{\max}(r) \geq B + 3$ and $\mathrm{S}_{\max} \geq B + 4$. There is a query starting at r that settles r, all element nodes in T and all leaves in C_i. This yields $\mathrm{S}_{\max}(r) \geq 1 + 3m + |\{w_{p,q} \in C_i\}|$. For the worst-case sss of r to be less than $B + 3$, the number of leaves in C_i is at most $B - 3m + 2$. The total number of leaves in T is exactly $m(B - 3)$, so there are at least $m(B - 1) - 2$ leaves distributed among the $m - 1$ remaining cells. Hence, by the pigeon-hole principle, for $m \geq 4$ there is a cell C_j that contains at least $B - 1$ leaves. Furthermore, we know by construction of T, that a limb contains less than $(B - 1)/2$ leaves. Thus, C_j holds nodes of at least three different limbs. Then there is a query from r to a target $t \in C_j$ that settles r itself, at least three element nodes, and $B - 1$ leaves, which yields $\mathrm{S}_{\max}(r) \geq B + 3$. Furthermore, if $\mathrm{S}_{\max}(r) = B + 3$, there must be a leaf $w_{p,q}$ not included in the search space of the corresponding r-t-query, such that the $w_{p,q}$-t-path contains r, and hence $\mathrm{S}_{\max}(w_{p,q}) = B + 4$. Observe that such a node $w_{p,q}$ outside of C_j must exist, because otherwise C_j would contain all leaves of at least $3m - 1$ limbs, which immediately implies a worst-case sss greater than $B + 4$. □

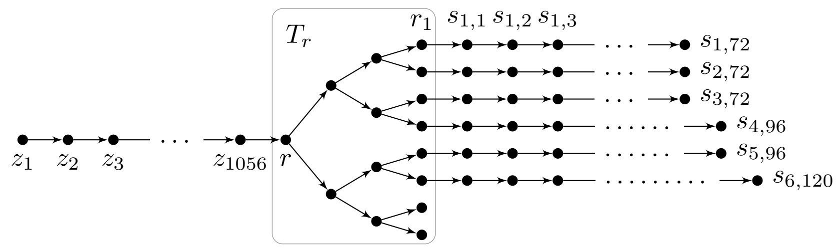


Figure 2: The reduction of an instance with $m = 2, B = 11$ and weights $3, 3, 3, 4, 4, 5$.

3.2 Trees with Bounded Degree

Instead of the height of a tree, we may also bound its maximum degree. If we bound the maximum degree by 2, we obtain the simple class of graphs consisting of a single path. Observe that on a path, the worst-case sss always occurs in a query between its endpoints, regardless of the underlying partition. Hence, the worst-case sss on a path is $|V|$ in both the directed and undirected case. Next, we show that MinWorstCasePartition becomes $\mathcal{NP}$-hard if we consider binary trees. This result again provides a tight border of tractability with respect to maximum node degree.

Theorem 3 MinWorstCasePartition *is $\mathcal{NP}$-hard for rooted directed trees with a maximum degree of* 3*, even in case of uniform edge weights.*

Proof: To simplify notation, we denote the worst-case sss of all queries starting at a fixed node $v \in V$ by $\mathrm{S}_{\max}(v) = \max_{t \in V} \mathrm{S}(v, t)$. Given an instance (S, B) of 3-Partition, the instance (T, m) with a binary tree $T = (V, E)$ is constructed as follows. We replace the limbs occuring in the previous reduction of Theorem 1 by more complex binary structures. The former root node r is now represented by a full binary tree T_r with m' leaves $r_1, \ldots, r_{m'}$, where $m' = 2^{\lceil \log_2 3m \rceil}$. From now on, let r denote the root of T_r, and all edges point away from r. A limb ℓ_p corresponding to an element $s_p \in S$ now consists of a chain of $12m\omega_p$ *element nodes* $s_{p,1}, \ldots, s_{p,12m\omega_p}$ with edges $(s_{p,q}, s_{p,q+1})$ for $q = 1, \ldots, 12m\omega_p - 1$. We connect each chain to T_r by adding the edge $(r_p, s_{p,1})$ to the tree. Moreover, we add a chain Z of $24m^2B$ nodes $z_1, \ldots, z_{24m^2B}$ with edges (z_j, z_{j+1}) for $j = 1, \ldots, 24m^2B - 1$ and connect it to r by adding an edge (z_{24m^2B}, r). An example of a resulting tree is shown in Figure 2. We claim that (T, m) admits a partition with worst-case sss at most $24m^2B + 12mB + 12m$ if and only if (S, B) is a Yes-instance of 3-Partition.

First, assume that (S, B) is a Yes-instance and let $S_1, \ldots, S_m$ be a corresponding solution. Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be the partition where C_i consists of all nodes of limbs corresponding to elements of S_i. All remaining nodes z_j on the chain Z and the binary tree T_r are assigned to arbitrary cells. Clearly, the sss of an arbitrary query originating at r or any node reachable from r is bounded by the size of the corresponding subtree rooted at that node. The size of T_r is bounded by $2m' < 12m$, and the total number of element nodes $s_{p,q}$ is $mB \cdot 12m$. Thus, $\mathrm{S}_{\max}(s)$ is bounded by $12m^2B + 12m$ for any such node. For a node $z_j \in Z$, the invariant $\mathrm{S}_{\max}(z_j) = \mathrm{S}_{\max}(z_{j+1}) + 1$ holds. In particular, we have $\mathrm{S}_{\max}(z_1) > 24m^2B > 12m^2B + 12m$, and therefore $\mathrm{S}_{\max}(z_1) > \mathrm{S}_{\max}(v)$ holds for all $v \in V \setminus \{z_1\}$. Thus, the overall worst-case sss equals $\mathrm{S}_{\max}(z_1)$. Observe that due to the described cell assignment, any cell contains exactly $12mB$ element nodes, and no other element nodes are settled in a query into this cell. Hence, the number of nodes corresponding to elements of (S, B) settled in a query from z_1 is bounded by $12mB$. In addition to that, there are at most $24m^2B$ settled nodes in Z and at most $12m$ settled nodes in T_r. We obtain $\mathrm{S}_{\max}(z_1) \leq 24m^2B + 12mB + 12m$.

Conversely, assume that $\mathcal{C} = \{C_1, \ldots, C_m\}$ is a partition of T with a worst-case sss of at most $24m^2B + 12mB + 12m$. Again, we know that the overall worst-case sss is always equal to $\mathrm{S}_{\max}(z_1)$, regardless of the underlying partition. To show that $\mathcal{C}$ corresponds to a Yes-instance of 3-Partition, we examine queries between z_1 and the leaves of T, as one such query must induce the worst-case sss (note that we do not explicity prove that limbs are monochromatic). Consider the leaves $s_{p,12m\omega_p}, p = 1, \ldots, 3m$ of the $3m$ limbs that correspond to elements of S. For every cell C_i in $\mathcal{C}$, there is one query starting at z_1 that settles all nodes $s_{p,12m\omega_p} \in C_i$. The sss of such a query is at least $24m^2B + 1 + 12m \sum_{s_{p,12m\omega_p} \in C_i} \omega_i$, because all nodes in the chain Z, at least the root node r in T_r, and all chains for which $s_{p,12m\omega_p}$ is in C_i are settled. For this term to fall below the bound $12m^2B + 12m(B + 1)$, the condition $\sum_{s_{p,12m\omega_p} \in C_i} \omega_p \leq B$ must be fulfilled for every cell C_i. But then we can derive a Yes-instance of 3-Partition with $S_i = \{s_p \mid s_{p,12m\omega_p} \in C_i\}$ and the claim follows. Since the reduction can be performed in polynomial time, the proof is complete. □

Again, this proof carries over to the case of undirected trees with a degree that is restricted to 3. Restricting both the degree and the height of the tree restricts its size, and thus renders the problem MinWorstCasePartition efficiently solvable.

Theorem 4 MinWorstCasePartition *is $\mathcal{NP}$-hard for undirected trees with a maximum degree of* 3*, even in case of uniform edge weights.*

Proof: Let (S, B) be an instance of 3-Partition as described in Section 2. We construct an instance (T, m) of the problem MinWorstCasePartition for a reduction. First, we create an undirected binary tree T_r with m' leaves $s_{1,1}, \ldots, s_{m',1}$, where $m' = 2^{\lceil \log_2 3m \rceil}$. Let r be the root of this tree. For each element $s_p \in S$, we add a *limb* that is constructed as follows. We create a chain of x nodes $s_{p,2}, \ldots, s_{p,x-1}$ connected by undirected edges, where the value of x is specified later. We connect the first node $s_{p,2}$ of each chain to a respective leaf $s_{p,1}$ of T_r. To the last node $s_{p,x-1}$ of every chain, we attach another binary tree with $B' = 2^{\lceil \log_2 \lfloor B/2 \rfloor \rceil}$ leaves and root node $s_{p,x}$. Finally, we add ω_p chains of $12m$ nodes and connect each chain to a distinct leaf of the p-th tree (recall that $\omega_p < B/2$ for all $p \in \{1, \ldots, 3m\}$). We call the subgraph containing all descendants of $s_{p,x}$ including $s_{p,x}$ the *element tree* T_p for a $p \in \{1, \ldots, 3m\}$. An example is shown in Figure 3. Since we can safely assume that $B' < B$ and $m' < 6m$, this construction is polynomial in the input size as long as x is polynomial in m and B. We claim that (T, m) admits a partition with worst-case sss at most $c := 2m' + 4x + 6B' + \log_2 B' + 12m(B + 1)$ if and only if (S, B) is a YES-instance of 3-PARTITION.

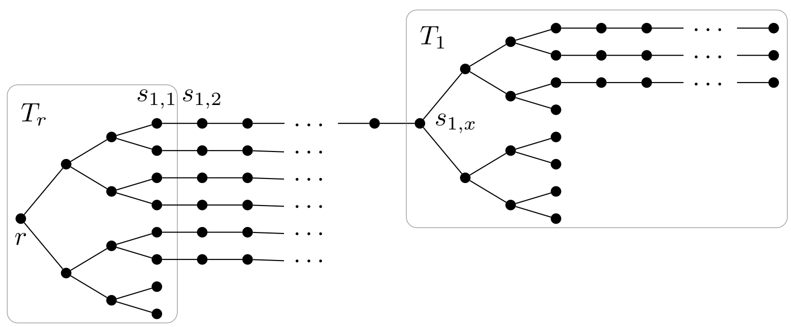


Figure 3: The reduction of an instance with $m = 2, B = 11$ and weights $3, 3, 3, 4, 4, 5$.

First, assume that (S, B) is a YES-instance and let $S_1, \ldots, S_m$ be a corresponding solution. Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be the partition where C_i consists of all nodes of limbs corresponding to the elements in S_i (a limb is a subtree rooted at a node $s_{p,1}$). The nodes of T_r are assigned to arbitrary cells. Thus, the size of each cell is bounded by the size of T_r, which has $2m' - 1$ nodes, plus the size of three entire limbs with B attached chains in total. This yields $|C_i| < 2m' + 3(x + 2B') + 12mB$ for $i = 1, \ldots, m$. First, note that a query where s and t are contained in the same cell settles no nodes outside the target cell except for nodes in T_r. Thus, the sss of an intra-cell query is bounded by $|C_i \cup T_r| < 2m' + 3x + 6B' + 12mB \leq c$. Similarly, an inter-cell query (i.e., where s and t belong to different cells) that starts inside T_r settles at most the nodes in T_r and all nodes in the target cell, yielding a sss bounded by c. An inter-cell query that starts at a node s in an arbitrary limb of the tree settles the path from s to T_r, and afterwards at most all nodes inside T_r and the complete target cell. Since the size of the longest path from a node inside a limb to T_r is $x + \log_2 B' + 12m$, the sss of an inter-cell query is bounded by $2m' + 4x + 6B' + \log_2 B' + 12m(B + 1) = c$.

Conversely, assume that $\mathcal{C} = \{C_1, \ldots, C_m\}$ is a partition of T with a worst-case sss of at most c. We show that each cell C_i of $\mathcal{C}$ contains nodes in at most three different element trees T_p. Assume for a contradiction that C_i includes nodes of z element trees $T_{p_1}, \ldots, T_{p_z}$, $z \geq 4$. As long as $z < 3m$, there is a query that starts at a node $s_{q,x}, q \notin \{p_1, \ldots, p_z\}$ and settles all nodes in C_i, that is, at least $z + 1 \geq 5$ complete chains of size x. Setting $x > 2m' + 6B' + \log_2 B' + 12m(B + 1)$, this value exceeds the bound c introduced above. A similar argument holds for the case where $z = 3m$, and thus we may safely assume that any cell contains nodes of at most three different element trees. Since there are $3m$ element trees and m cells in total, this immediately implies that all element trees are monochromatic. But then we know that there exists a query from a source node s, where s is a leaf of an element tree, that passes all nodes on the path from s to the root tree T_r, and then settles at least three entire limbs with their element trees assigned to a target cell $C_i \not\ni s$. The path from s to the first node of the root tree has size $x + \log_2 B' + 12m$. The three entire limbs contain $3(x + 2B' - 1) + 12mB_i$ nodes in total, where B_i is the sum of the weights ω_p of the elements s_p corresponding to the three element trees T_p. Moreover, at least three internal nodes from T_r must be settled to connect four limbs. In total, we get a worst-case sss of at least $4x+6B'+\log_2 B'+12m(B_i+1)$. For this value to fall below c, we have to ensure that $B_i \leq B$ for each cell C_i. But then each cell corresponds to a set S_i of total weight B, and hence $\mathcal{C}$ corresponds to a solution of 3-Partition. □

3.3 Cycles

Since the search space on a directed cycle always consists of exactly the unique s-t-path, the worst-case sss is $|V|$ and does not depend on the underlying partition. Therefore, we may focus on undirected cycles. We consider the following problem that is strongly related to MinWorstCasePartition. We are given as input an undirected cycle $G = (V, E, \omega)$ and a desired worst-case sss W, and the task is to compute a partition of minimum cardinality such that the induced worst-case sss is at most W. Observe that solving this problem efficiently would immediately imply the existence a polynomial-time algorithm for MinWorstCasePartition, as we can use binary search to obtain the minimum bound W that allows a partition with at most k cells. In what follows, let $k_{\mathrm{opt}}(G, W)$ denote the minimum number of cells that is necessary to achieve a worst-case sss of at most W on G. Clearly, the shortest path of maximum size yields a lower bound L on the worst-case sss. For $W \geq L$, we show how to approximate $k_{\mathrm{opt}}(G, W)$.

Theorem 5 *Given an undirected cycle G and a positive integer $W \geq L$, a partition $\mathcal{C}$ with $k_{\mathrm{opt}}(G, W) + 1$ cells and $\mathrm{S}_{\max}(G, \mathcal{C}) \leq W$ can be computed in polynomial time.*

Proof: For the sake of simplicity, assume that all shortest paths in $G = (V, E, \omega)$ are unique. Consider the shortest-path tree T_s rooted at an arbitrary node s. Since G is a cycle, there is exactly one undirected edge e_s that is not in T_s, called the *cut edge* of s. We assign to each node t the sss of a Dijkstra search from s to t. Note that each target node t gets a distinct number in $\{1, \ldots, |V|\}$, its *Dijkstra rank* with respect to s. Obviously, nodes on the two branches of T_s originating at s have ascending ranks. Consider a pair s and t of nodes such that the Dijkstra rank of t with respect to s is in $\{W + 1, \ldots, |V|\}$ and let C_t be the cell containing t. Recall that the nodes assigned to C_t completely determine the sss of all arc-flags queries to t. To make sure that the sss of an s-t-query is at most W, we have to ensure that the arc-flags query prunes the search at the branch of T_s that does not contain t. This is achieved by assigning nodes that cause a large sss to cells distinct from C_t. More precisely, we determine the set X_t of nodes such that $\max_{s \in V} \mathrm{S}(s,t) \leq W$ if and only if $C_t \cap X_t = \emptyset$.

Assume we traverse the cycle starting at t in both directions. Let e_u and e_v be the first edges in the respective direction that are cut edges for some nodes $u, v \in V$. Consider the backward shortest-path tree of t, i.e., the shortest-path tree of t obtained if edges are traversed in reverse direction. Edges in this tree have the flag for C_t set. If we omit edge directions, this tree coincides with T_t. Let e_t be its cut edge. Removing e_u, e_v, and e_t from G yields three connected components $G_{u,v}$, $G_{u,t}$ and $G_{v,t}$ with t in $V(G_{u,v})$, see Figure 4.

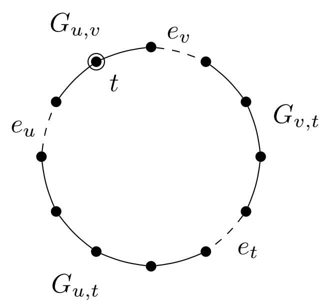


Figure 4: The three subgraphs $G_{u,v}$, $G_{u,t}$, and $G_{v,t}$ with respect to a certain node t.

Claim. The set X_t is determined as follows.

(1) $V(G_{u,t}) \subseteq X_t$ if $\mathrm{S}(s,t) > W$ for a node $s \in V(G_{v,t})$, and $V(G_{u,t}) \cap X_t = \emptyset$ otherwise.

(2) $V(G_{v,t}) \subseteq X_t$ if $\mathrm{S}(s,t) > W$ for a node $s \in V(G_{u,t})$, and $V(G_{v,t}) \cap X_t = \emptyset$ otherwise.

(3) $V(G_{u,v}) \cap X_t = \emptyset$.

To see this, consider the fixed target node t, and assume that there exists a node $s \in V$ such that the sss of Dijkstra's algorithm in an s-t-query exceeds W. First, consider the case where s is in $V(G_{u,t})$. We show that the arc-flags enhanced query from s to t settles at most W nodes if and only if no node in $G_{v,t}$ is assigned to the cell C_t that contains t.

Assume that there is a node w in $V(G_{v,t}) \cap C_t$. We show that then the flag for the cell of t must be set on all edges of T_s that are relaxed by Dijkstra's algorithm until t is reached, resulting in a worst-case sss greater than W in the s-t-query (because then the arc-flags algorithm settles the same nodes as Dijkstra's algorithm). To see this, let e be an arbitrary edge of T_s. If e is part of the shortest path from s to t or from s to w, its flag is clearly set. If e is in the subtree of T_s rooted at t, it is not relaxed by Dijkstra's algorithm in an s-t-query and therefore its flag setting is irrelevant. Otherwise, e is in the subtree of T_s rooted at w, and this subtree contains only nodes in $G_{v,t}$ because $w \in V(G_{v,t})$ and for the cut edge e_s it is either $e_s = e_v$ or $e_s \in E(G_{v,t})$. Otherwise, $e_s \in E(G_{u,t}) \cup \{e_u\}$ would hold (since $G_{u,v}$ contains no cut edges and $e_s = e_t$ implies that the s-t query settles at most all nodes on the shortest path from the endnode of e_t that lies in $G_{u,t}$ to t, yielding a sss of at most $L \leq W$, which contradicts our assumption). Therefore, the shortest path from t to s, and thus also from s to t, would include e_s, contradicting the definition of e_s. But then e is contained in the backward shortest-path tree T_t of t and its flag must be set as well.

Conversely, assume that $V(G_{v,t}) \cap C_t = \emptyset$. First of all, we show that all nodes in $V(G_{u,v})$ share the same cut edge e_t. Assume to the contrary that there is a node t' in $V(G_{u,v})$ with a cut edge $e_{t'} \neq e_t$. Removing e_t and $e_{t'}$ yields two non-empty connected subgraphs of G. Let z be a node that is in the unique component that does not contain the nodes $V(G_{u,v})$. Then the shortest z-t-path and the shortest z-t'-path lie on different branches of T_z (due to the position of z between the respective cut edges and symmetry of shortest paths), and hence e_z must lie in $E(G_{u,v})$. Since by construction $G_{u,v}$ contains no cut edges, this is a contradiction. Thus, all nodes in $V(G_{u,v})$ have the cut edge e_t. Let x_1 and x_2 be the endpoints of e_t with $x_1 \in V(G_{u,t})$. The flag from x_1 to x_2 for C_t is not set, as we have $C_t \subseteq V(G_{u,t}) \cup V(G_{u,v})$, and a set flag from x_1 to x_2 would imply that there is a shortest path from a node in $G_{u,t}$ to a node in $G_{u,v}$ via e_t, contradicting the definition of e_t. Thus, a query from s to t settles at most all nodes in $G_{u,t} \cup G_{u,v}$. But those are exactly the nodes on the shortest path from x_1 to the endpoint of e_v that lies in $G_{u,v}$, and hence $\mathrm{S}(s,t) \leq L \leq W$.

Analogously, for the case of s not in $V(G_{v,t})$, a query from s to t settles at most W nodes if and only if no node in $G_{u,t}$ is assigned to the cell C_t that contains t. Finally, if $s \in V(G_{u,v})$, an s-t-query settles at most all nodes on the shortest s-t-path plus all nodes on the remaining branch of T_s. Since the cut edge of s is e_t, this branch ends either at x_1 or at x_2. Hence, the query settles only nodes on the shortest path from t to x_1 or x_2, respectively. This implies that $\mathrm{S}(s,t) \leq L$, regardless of the underlying partition $\mathcal{C}$. Summarily, it is $X_t = \emptyset$ if no source node s induces a sss greater than W in a query with target t, and otherwise X_t consists of either $V(G_{u,t})$, $V(G_{v,t})$ or the union of both sets.

Next, consider the sets $U_t = \{w \in V(G_{u,v}) \mid X_w \supseteq V(G_{v,t})\}$ and $U'_t = \{w \in V(G_{u,v}) \mid X_w \supseteq V(G_{u,t})\}$ of nodes in $G_{u,v}$ whose sets X_w share a subgraph of G.

Claim. If $U_t \neq \emptyset$, it contains an endpoint of e_v. If $U'_t \neq \emptyset$, it contains an endpoint of e_u. Both U_t and U'_t induce connected subgraphs of G.

Assume that there is a $w \in U_t$ such that the set X_w contains $V(G_{v,t})$. This implies that there is a node $s \in V(G_{u,t})$ such that the sss of Dijkstra's algorithm in an s-w-query exceeds W. Since e_s is in $E(G_{v,t}) \cup \{e_v\}$, all nodes in $G_{u,v}$ are in the same branch of T_s. In particular, starting from the endpoint of e_u that lies in $G_{u,v}$, these nodes have ascending Dijkstra ranks. Hence, if w induces a sss of $\mathrm{S}(s,w) \geq W+1$, all nodes $x \neq w$ in the subtree of T_s rooted at w (and especially the endpoint of e_v in $E(G_{u,v})$) induce an even greater sss in a query from s. Thus, all corresponding sets X_x contain the set $G_{v,t}$ as well. An analogous argument follows for nodes $w' \in U'$ $V(G_{u,t}) \subseteq X_{w'}$.

Because all nodes in U_t lie between two consecutive cut edges, it follows from Claim our first claim that it is either $U_t \subseteq X_w$ or $U_t \cap X_w = \emptyset$ for all nodes w of the graph. Thus, restricting to partitions where all nodes in the set U_t are assigned to the same cell neither causes the sss to exceed W nor does it increase the number of necessary cells. The same holds for the set U'_t.

Summarizing the sets of nodes x, y where $U_x = U_y$ or $U_x = U'_y$, we obtain a number of distinct connected subsets $U_i \subseteq V$ (connectivity holds by our second Claim). Each set U_i corresponds to a set $X_i \neq \emptyset$, such that nodes in X_i must not be assigned to the cell that contains U_i. It is easy to see that at most two sets U_i, U_j with $X_i, X_j \neq \emptyset$ can be put into the same cell (roughly speaking, this is due to the fact that each set X_i blocks one of two branches of a corresponding shortest-path tree). We can find a minimum number of cells for the sets U_i if we find a maximum matching of them, where two sets U_i and U_j can be matched if and only if $U_i \cap X_j = U_j \cap X_i = \emptyset$. This can be done in polynomial time [12] and yields a lower bound $k \leq k_{\text{opt}}(G, W)$ on the necessary number of cells. Finally, we have to assign all remaining nodes u with $X_u = \emptyset$. A sophisticated matching may possibly allow for an exhaustive assignment of these nodes to cells that are already used. However, this appears to be difficult to guarantee in general. Instead, we use an extra cell and assign all nodes u with $X_u = \emptyset$ to this cell, and therefore we use at most one more cell than necessary. In summary, given a bound W on the worst-case sss we can compute a partition that needs at most $k + 1 \leq k_{\text{opt}}(G, W) + 1$ cells. □

3.4 Approximation Algorithms

We present an algorithm that approximates the optimal worst-case sss with a given number of cells within a factor of 5/2 and 3 for undirected and directed trees, respectively. The essential task concerning the instances constructed in the proof of Theorem 1 is to find balanced cells that are *almost* connected. We exploit this observation to derive an approximation algorithm. We say that a cell C of a partition $\mathcal{C}$ in a graph $T = (V, E, \omega)$ is 1-*disconnected* if there is a node $v \in V$ such that $C \cup \{v\}$ induces a connected subgraph of T.

We describe the algorithm TreeApprox that, given an undirected tree T (if T is directed, we simply ignore edge directions) and a parameter k, computes at most k 1-disconnected cells of size at most $2\lceil |V|/k \rceil$. Starting from the leaves of the tree, we traverse it in a bottom-up fashion and keep track of the size of the subtree induced by each node. Once a node v is reached whose subtree contains at least $s_v \geq \lceil |V|/k \rceil$ nodes, we assign all nodes in this subtree including v to $c = \max\{a \in \mathbb{N} \mid a \cdot \lceil |V|/k \rceil \leq s_v\}$ newly introduced cells. For each descendant w of v, we add the subtree rooted at w to one of the c new cells such that the cell size does not exceed $2\lceil |V|/k \rceil$. The subtree rooted at v is removed and the algorithm continues recursively until T contains less than $\lceil |V|/k \rceil$ nodes. All remaining nodes are put into a final new cell, which is added to $\mathcal{C}$ as well. The partition $\mathcal{C}$ generated by the algorithm fulfills the following desired conditions.

Lemma 1 *Given input parameters $T = (V, E, \omega)$ and k, the algorithm* TreeApprox *terminates and computes a partition $\mathcal{C} = \{C_1, \dots, C_{k'}\}$ satisfying the following properties.*
(a) All cells $C_i \in \mathcal{C}$ are 1-disconnected.
(b) For all $C_i \in \mathcal{C}$ it is $|C_i| \leq 2\lceil |V|/k \rceil$.
(c) The number of cells k' in the computed partition $\mathcal{C}$ is at most k.

Proof: Since the algorithm traverses the tree in a bottom-up fashion, it terminates if and only if there always exists a cell with enough room left for the next subtree during cell assignment. We prove that the properties (a), (b), and (c) are fulfilled. Termination of the algorithm follows immediately.

(a) The assignment of nodes to cells is done in the main loop of the algorithm. By construction, each newly created cell C_i contains only connected subtrees rooted at the descendant of a given node v. Clearly, $C_i \cup \{v\}$ induces a connected subtree of T. By removing V_v (the nodes of the subtree rooted at v) in the main loop, we ensure that nodes in the subtree rooted at v are never reassigned.

(b) New cells are created whenever the number of unassigned nodes in the subtree of a node v exceeds the size $\lceil |V|/k \rceil$. Thus, we may safely assume that for all descendants w of v, the set V_w contains less than $\lceil |V|/k \rceil$ nodes. There are $c \cdot \lceil |V|/k \rceil \leq s_v < (c+1) \cdot \lceil |V|/k \rceil$ nodes to be assigned to $c \geq 1$ cells $C_1, \dots, C_c$. The sets V_w (and analogously, the set $\{v\}$) are consecutively assigned to an arbitrary available cell C_i with $|C_i| + |V_w| \leq 2\lceil |V|/k \rceil$. Assume for a contradiction that at some point we are forced to exceed the size limit of $2\lceil |V|/k \rceil$ when trying to add a set V_x. Let $|V_x| = \lceil |V|/k \rceil - \varepsilon$ for an $\varepsilon \geq 1$ and hence $|C_i| > \lceil |V|/k \rceil + \varepsilon$ for all $i \in \{1, \dots, c\}$. Then the total number s_v of assigned nodes is at least $\sum_{i=1}^{c} |C_i| + |V_x| \geq c \cdot (\lceil |V|/k \rceil + \varepsilon) + \lceil |V|/k \rceil - \varepsilon \geq (c+1) \cdot \lceil |V|/k \rceil$, a contradiction.

(c) New cells are introduced whenever a node v is reached with $s_v \geq c \cdot \lceil |V|/k \rceil$ for some $c \geq 1$. At this point, at least $c \cdot \lceil |V|/k \rceil$ nodes have to be assigned to c cells. If v is the last node visited before the algorithm terminates and $s_v < \lceil |V|/k \rceil$, the s_v remaining nodes are assigned to a final new cell. Let k' be the number of cells computed by TreeApprox. By construction of the algorithm, we know that all cells, except for the last one, contain at least $\lceil |V|/k \rceil$ nodes. We distinguish two cases. If the last created cell contains at least $\lceil |V|/k \rceil$ nodes as well, the total number of nodes assigned to cells is $|V| \geq k' \lceil |V|/k \rceil \geq k'|V|/k$, which implies $k' \leq k$. Otherwise, let $x < \lceil |V|/k \rceil$ be the size of the last cell created by the algorithm. The number of assigned nodes is $|V| \geq (k'-1)\lceil |V|/k \rceil + x \geq ((k'-1)|V|/k) + 1$, and hence we have $k' - 1 < k$. In both cases the number of cells is bounded by k. □

We prove approximation guarantees for the algorithm TreeApprox. Theorem 6 provides a first bound, which can be improved for undirected trees.

Theorem 6 *Algorithm* TreeApprox *is a 3-approximation for the problem* MinWorstCasePartition *on directed and undirected trees.*

Proof: Let $\mathcal{C} = \{C_1, \dots, C_{k'}\}$ be the output of algorithm TreeApprox given the input parameters $T = (V, E, \omega)$ and k. Let ALG denote the worst-case sss induced by $\mathcal{C}$ and OPT the optimal worst-case sss for T and k. Since all cells in $\mathcal{C}$ are 1-disconnected, after entering the target cell, a query settles at most one more node outside this cell. Moreover, only edges pointing towards the target cell have the corresponding flag set. Hence, a worst-case query into a given cell C_i settles the largest possible path outside C_i leading into this cell plus at most all nodes in C_i plus an additional node,. Let $P_{s,t}$ denote the unique s-t-path for any $s, t \in V$ and let $\Delta = \max_{s,t \in V} |P_{s,t}|$ be the diameter of T. Clearly, the worst-case sss is bounded by $\text{ALG} \leq \max_{1 \leq i \leq k'} \{\Delta + |C_i|\} \leq \Delta + 2\lceil |V|/k \rceil \leq 3 \cdot \max\{\Delta, \lceil |V|/k \rceil\}$ (note that the longest path of size Δ is at least as large as the longest path outside C_i plus the additional node possibly settled). On the other hand, an optimal partition contains at least one cell of size at least $\lceil |V|/k \rceil$ and there is a query that settles all nodes of this cell. Since the diameter is a lower bound on the worst-case sss, the optimal solution for T must be $\text{OPT} \geq \max\{\Delta, \lceil |V|/k \rceil\}$ (this holds for directed trees as well, since there must exist a root node from which all nodes are reachable). It follows that $\text{ALG} \leq 3 \cdot \text{OPT}$. □

A more sophisticated analysis leads to an improvement of the lower bound on the optimal solution for undirected trees and yields the following guarantee.

Theorem 7 *Algorithm* TREEAPPROX *is a* $5/2$*-approximation for the problem* MINWORSTCASEPARTITION *on undirected trees.*

Proof: Given an undirected tree $T = (V, E, \omega)$ with diameter Δ and a parameter k, let OPT be the minimum worst-case sss for the corresponding instance of MINWORSTCASEPARTITION. Let $\mathcal{C}_{\text{opt}} = \{C_1, \ldots, C_k\}$ be an optimal partition. Without loss of generality, let $|C_1| \geq |C_2| \geq \cdots \geq |C_k|$, and in particular $|C_1| \geq \lceil |V|/k \rceil$. We show that $\text{OPT} \geq \lceil |V|/k \rceil + \Delta/4$. Let $P_{\max}$ be a path of maximal size in T, i.e., $|P_{\max}| = \Delta$. We consider queries from the endpoints of $P_{\max}$ and distinguish two cases depending on the number of nodes on $P_{\max}$ assigned to C_1.

First, assume that $|C_1 \cap P_{\max}| \leq \Delta/2$. From each endpoint of $P_{\max}$, there is a query that settles all nodes in C_1. Moreover, every node on $P_{\max}$ is settled by at least one of these two queries. To see this, consider an arbitrary target t in C_1. Each of the two unique paths from the endpoints of $P_{\max}$ to t contains exactly the subpath of $P_{\max}$ from the corresponding endpoint to the unique node of $P_{\max}$ that roots the subtree containing t. Hence, the two (almost) complementary subpaths together must cover $P_{\max}$. Since t is settled in both of the two queries (with target nodes t_1, t_2 possibly distinct from t) that settle all nodes in C_1, the observation follows. As the number of nodes on $P_{\max}$ not in C_1 is at least $\Delta/2$, one of these two queries must settle at least $\Delta/4$ nodes not in C_1. In total, we obtain a worst-case sss of at least $|C_1| + \Delta/4$.

For the second case, assume that the number of nodes on $P_{\max}$ assigned to C_1 is $\Delta/2 + x$ for some $x \geq 1$. There are queries from both endpoints of $P_{\max}$ that settle all nodes in C_2. Obviously, at least one of these two queries must settle at least $\Delta/4 + \lceil x/2 \rceil + |C_2|$ nodes. If $|C_2| \geq \lceil |V|/k \rceil$, the claim follows. Conversely, let $|C_2| = \lceil |V|/k \rceil - y$ for some $y \geq 1$. The worst-case sss is at least $\Delta/4 + \lceil x/2 \rceil + \lceil |V|/k \rceil - y$. In addition to that, we know that $|C_1| \geq \lceil |V|/k \rceil + y$

must hold (recall that $|C_2| \geq |C_i|$ for all $i \geq 2$). Moreover, there is a query that settles at least $\Delta/4 - \lceil x/2 \rceil$ nodes of $P_{\max}$ not in C_1 plus all nodes in C_1, hence the worst-case sss is at least $\Delta/4 - \lceil x/2 \rceil + \lceil |V|/k \rceil + y$ nodes. Thus, there is always a query settling $\Delta/4 + \lceil |V|/k \rceil$ nodes, independent of whether $x/2 \geq y$ or $x/2 < y$.

Next, we infer the resulting approximation ratio. Let ALG be the worst-case sss induced by a partition computed by TreeApprox. We know that ALG $\leq \Delta + 2\lceil |V|/k \rceil$ and OPT $\geq \max\{\Delta, \Delta/4 + \lceil |V|/k \rceil\}$. To prove the theorem, we distinguish two cases. First, let $\Delta \geq 4\lceil |V|/k \rceil/3$. In this case we have ALG/OPT $\leq (\Delta + 2\lceil |V|/k \rceil)/\Delta \leq 5/2$. If $\Delta < 4\lceil |V|/k \rceil/3$, it is ALG/OPT $\leq (\Delta + 2\lceil |V|/k \rceil)/(\Delta/4 + \lceil |V|/k \rceil) = (\Delta + 2\lceil |V|/k \rceil)/(4/5 \cdot (5\Delta/16 + \lceil |V|/k \rceil/4 + \lceil |V|/k \rceil)) < (\Delta + 2\lceil |V|/k \rceil)/(4/5 \cdot (\Delta/2 + \lceil |V|/k \rceil)) = 5/2$. □

4 Minimizing the Average Search-Space Size

Since MinAvgCasePartition is known to be $\mathcal{NP}$-hard in general [2], we investigate restricted input instances. Along the lines of Section 3, we examine paths, cycles, stars, and trees.

To begin with, we establish preliminary tools in Lemma 2 and Corollary 1, used in the subsequent proofs of this section. Recall that a function $f\colon \mathbb{R}^{\geq 0} \to \mathbb{R}$ is convex on $\mathbb{R}^{\geq 0}$ if and only if the difference quotient $(f(x_0+h) - f(x_0))/h$ of f is non-decreasing in x_0 for any fixed h. The following lemma provides a crucial statement about the sum of several functional values of a convex function. Later on, such functions will come up as sss induced by a set of cells.

Lemma 2 *Let $f\colon \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ be a cost function that is convex and increasing on $\mathbb{R}^{\geq 0}$. Let x and n be two fixed positive integers. Furthermore, let $x_1, \ldots, x_n$ be positive integers subject to $\sum_{i=1}^{n} x_i = x$. Then the total cost $\Gamma = \sum_{i=1}^{n} f(x_i)$ is non-decreasing if the values x_i are modified subject to one of the following rules while maintaining the constraints $\sum_{i=1}^{n} x_i = x$ and $x_i \geq 0$ for all $x_i, i \in \{1, \ldots, n\}$.*

1. *Two arbitrary values x_i and x_j are swapped.*
2. *Given two integers x_i, x_j with $x_i \geq x_j$ and a number $d \in \mathbb{N}^+$, the value x_i is increased by d while x_j is decreased by d.*

Proof: Clearly, swapping two elements has no influence on the cost Γ. Thus, we can concentrate on the latter case.

For the second case, assume we are given two values x_i, x_j such that $x_i \geq x_j$ holds. Obviously, the resulting cost after increasing x_i and decreasing x_j by the same value $d \in \mathbb{N}^+$ is equal to

$$\Gamma' = \Gamma + f(x_i + d) - f(x_i) + f(x_j - d) - f(x_j).$$

Since f is increasing in $\mathbb{R}^{\geq 0}$, we know that we have $f(x_i + d) - f(x_i) \geq 0$ and similarly $f(x_j - d) - f(x_j) \leq 0$. Consequently, all we need to show is that

$$|f(x_i + d) - f(x_i)| \geq |f(x_j) - f(x_j - d)|$$

holds for any $x_i \geq x_j$. This, however, is clear because we demanded that f is convex and thus the difference quotient

$$g(x) = \frac{f(x+h) - f(x)}{h}$$

is non-decreasing in x for fixed h. We set $h = d$ and $x = x_i$ or $x = x_j - d$, respectively. Since $x_i \geq x_j$ implies $g(x_i) \geq g(x_j - d)$ and due to the constraints of the lemma $g(x_j - d) \geq 0$ holds, we obtain the following desired result.

$$|f(x_i + d) - f(x_i)| = d \cdot g(x_i) \geq d \cdot g(x_j - d) = |f(x_j) - f(x_j - d)|$$

This completes the proof. $\square$

Assume we are given positive integers $\{x_1, \ldots, x_n\}$ with $x_i \in \{\lfloor x/n \rfloor, \lceil x/n \rceil\}$ for all x_i such that their sum equals x and a cost function Γ as in Lemma 2. Using steps 1 and 2 from the lemma, we can create any set of values x_i that fulfills the constraint $\sum_{i=1}^{n} x_i = x$. In each of these steps, the overall cost is non-decreasing. Hence, we minimize a given convex, increasing cost function if all values x_i are as close to $\lfloor x/n \rfloor$ as possible. Corollary 1 follows directly from this observation.

Corollary 1 *Let $f \colon \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ be an increasing, convex function and x and n two positive integers. For arbitrary positive integers $x_1, \ldots, x_n$ subject to the constraint $\sum_{i=1}^{n} x_i = x$, the cost $\sum_{i=1}^{n} f(x_i)$ is minimized if $x_i = \lceil x/n \rceil$ for $i \leq x \bmod n$ and $x_i = \lfloor x/n \rfloor$ for $i > x \bmod n$.*

4.1 Paths

In the proofs of Theorems 8 and 9, we use an alternative notion of sss. Given a directed or undirected path $P = (V, E, \omega)$, let $P_{s,t}$ denote the unique simple s-t-path between arbitrary nodes $s, t \in V$. We say that the *penalty* of a corresponding query is $\mathrm{pen}(s,t) = \mathrm{S}(s,t) - |(P_{s,t})|$ if $d(s,t) < \infty$ and $\mathrm{pen}(s,t) = \mathrm{S}(s,t) - 1$ otherwise. We know that a query must at least settle all nodes on the path from s to t, and it must at least settle s in the case that t is unreachable. Consequently, the values $|P_{s,t}|$ and 1 yield respective tight lower bound on $\mathrm{S}(s,t)$. Thus, $\mathrm{pen}(s,t) \geq 0$ always holds and since the sum $\sum_{s,t \in P} |P_{s,t}|$ of all path sizes of a graph is constant, minimizing $\mathrm{S}_{\mathrm{avg}}$ is equal to minimizing $\sum_{s,t \in P} \mathrm{pen}(s,t)$. The essential step in both of the two following proofs is to show that cell-induced penalties can be interpreted as convex functions, and by Corollary 1 we thus minimize the average sss if we balance the cell sizes.

Given a graph consisting of a single directed path P and a parameter k, let the partition $\mathcal{C}_{\mathrm{opt}}$ consist of k connected cells $C_1, \ldots, C_k$ of balanced size, i.e., $|C_i| \in \{\lfloor |V|/k \rfloor, \lceil |V|/k \rceil\}$ for all $1 \leq i \leq k$.

Theorem 8 *Let $P = (V, E, \omega)$ be a directed path and k a positive integer. The partition $\mathcal{C}_{\mathrm{opt}}$ described above yields an optimal partition if k bounds the number of cells.*

Proof: Each node of the graph has at most one outgoing edge. Hence, the next node to be settled in a query is always unique and we can ignore edge weights. The proof consists of two elementary steps. First, we show that given an arbitrary partition $\mathcal{C}$, we can always construct a partition $\mathcal{C}'$ that contains only (weakly) connected cells without increasing the sss. In the second step, we show how to minimize the sss for strongly connected cells with a uniform weight function, which proves the theorem.

Given an arbitrary partition $\mathcal{C} = \{C_1, \ldots, C_k\}$ that contains at least one cell that is not strongly connected, we construct the partition $\mathcal{C}' = \{C'_1, \ldots, C'_k\}$ as follows. Starting at the leftmost node v_1, we assign subsequent nodes of the path to ascending cell indices while retaining the cell sizes of $\mathcal{C}$. More formally, given $V = \{v_1, \ldots, v_{|V|}\}$ and $E = \{(v_i, v_{i+1}) \mid 1 \leq i \leq |V| - 1\}$ we set $C'_1 = \{v_1, \ldots, v_{|C_1|}\}$, $C'_2 = \{v_{|C_1|+1}, \ldots, v_{|C_1|+|C_2|}\}$ and so forth. We now show that $\sum_{s,t \in P} \mathrm{pen}_{\mathcal{C}'}(s,t) \leq \sum_{s,t \in P} \mathrm{pen}_{\mathcal{C}}(s,t)$ holds. To this end, we distinguish the penalties of intra-cell queries (i.e., queries where s and t belong to the same cell) and inter-cell queries (i.e., queries where s and t are assigned to different cells) on P. Since all cells in $\mathcal{C}'$ are strongly connected subgraphs, only edges that actually lead from s towards the target cell have the corresponding flag set. Hence, inter-cell queries cause a total penalty of 0, because either the exact s-t-path is settled or no outgoing edge has the target flag set. Intra-cell query penalties cannot increase in comparison to the original partition, for either exactly the s-t-path gets settled or the query settles all nodes up to the rightmost node of the cell (i.e., the unique node v that has no reachable node of the same cell). The total penalty accounting for the latter case clearly is minimized if cells are connected. Finally, we know that the size of each cell in $\mathcal{C}$ is preserved in the corresponding cell in $\mathcal{C}'$, i.e., $|C_i| = |C'_i|$ for all $i \in \{1, \ldots, k\}$. Hence, the total number of inter-cell queries and intra-cell queries is identical for both partitions and the overall penalty does not increase for $\mathcal{C}'$.

To prove the second claim, we have to minimize the overall penalty given that all cells are connected. The only positive penalties that occur are those of intra-cell queries where the target node is unreachable. Imagine the nodes of a certain cell C_i to be ordered by increasing number of unreachable nodes in C_i, i.e., the order in which the nodes in C_i are traversed when starting at its front node. For a node v at position j in this order, there are $j - 1$ distinct intra-cell nodes that are unreachable from v. A query from v to any of these nodes then causes all reachable nodes in C_i to be settled, which induces a penalty of $|C_i| - j$. With j taking any value in $\{1, \ldots, |C_i|\}$, this yields a total penalty as shown below.

$$\begin{aligned}\sum_{s,t \in V} \mathrm{pen}_{\mathcal{C}}(s,t) &= \sum_{i=1}^{k} \sum_{j=1}^{|C_i|-1} (j-1) \cdot (|C_i| - j) \\ &= \sum_{i=1}^{k} \left(\frac{1}{6} |C_i|^3 - \frac{1}{2} |C_i|^2 + \frac{1}{3} |C_i| \right) \qquad (1)\end{aligned}$$

This implies that we can assign a penalty $p(x) = x^3/6 - x^2/2 + x/3$ to a cell of cardinality x. If we interpret the polynomial p as a continuous function with the cell size as a parameter, we obtain a cost function that is non-negative, increasing and convex on $\mathbb{R}^{\geq 0}$. From Corollary 1 we know that the total penalty of P then is minimized if we have $n \bmod k$ cells of size $\lceil n/k \rceil$ and $n - (n \bmod k)$ cells of size $\lfloor n/k \rfloor$. Together with the demand for strongly connected cells, the partition $\mathcal{C}$ stated in the theorem fulfills this requirement and hence yields a minimum penalty for P. □

The following Theorem 9 shows that the partition $\mathcal{C}_{\text{opt}}$ optimizes the average sss on undirected paths as well. The proof uses similar arguments as in the directed case.

Theorem 9 *Let P be an undirected path and k a positive integer. The partition $\mathcal{C}_{\text{opt}}$ described above yields an optimal partition if k bounds the number of cells.*

Proof: The proof consists of three elementary steps. Along the lines of Theorem 8, we show that, given an arbitrary partition $\mathcal{C}$, we can construct a partition $\mathcal{C}'$ containing only strongly connected cells without increasing the sss. In the second step, we show that we may ignore the weight function of the graph as long as cells are strongly connected. Finally, we show how to minimize the sss for strongly connected cells with a uniform weight function.

We use the same procedure as in the first step of the proof of Theorem 8 to convert an arbitrary partition into one that only has connected cells without increasing the total sss. Let s and t be nodes of different cells. Since all cells in $\mathcal{C}'$ are strongly connected subgraphs, only edges that actually lead from s towards the target cell have the corresponding flag set. Therefore, the query algorithm starts at s and settles only nodes of $P_{s,t}$ until the target cell is reached. The query is then aborted as soon as the target node t is reached. Hence, all nodes that are settled during the query belong to the unique shortest s-t-path, yielding a penalty of 0 for inter-cell queries.

As for intra-cell queries, we consider an arbitrary isolated pair of corresponding cells C_i and C'_i of both partitions, and show that the sum of all intra-cell penalties cannot increase for the cell C'_i. Assume we are given a certain cell $C_i = \{r_1, \ldots, r_c\}$ and its transformation $C'_i = \{r_1, \ldots, r_c\}$ for an $i \in \{1, \ldots, k\}$. We compare the sum of all penalties of intra-cell queries starting at two nodes r_j and s_j at the same relative position in the respective cell (i.e., r_j and s_j are the j-th node encountered when traversing P from a certain direction). Since each cell in $\mathcal{C}'$ is strongly connected, no nodes outside C'_i are settled in an intra-cell query. For any source node s_j, the order in which the nodes in C'_i get extracted from the queue is independent of t, and we can assign a rank p ranging from 1 to $|C'_i|$ to each node of the cell that represents its position in this order. There are $|C'_i|$ possible target nodes for an intra-cell query starting at s_j and each of them has a distinct rank in $\{1, \ldots, |C'_i|\}$. To obtain the corresponding penalties, we must consider the sizes of all paths $P_{r_j,t}$ from r_j to any $t \in C'_i$. Since the cell is strongly connected, we know that the cell-induced subgraph contains exactly $j-1$ nodes with an index lower than s_j and $|C'_i| - j$ nodes with an index greater than s_j. The penalty thus only depends on the cell size as well as the relative position of r_j.

Conversely, when analyzing all queries that start at the corresponding source s_j of the original cell C_i, we have to take into account that nodes of other cells may get settled as well. To obtain the penalty induced by intra-cell queries from s_j, let us assume that we temporarily remove all nodes of other cells from the graph. To preserve correct intra-cell distances, edges between corresponding pairs of nodes with their original distance as weight are inserted. Clearly, the penalty caused by s_j now is identical to the case of r_j. Now, assume that we reinsert the nodes from other cells into the graph. For each s_j-t-query with $t \in C_i$, if an inserted node u lies on the s_j-t path, both the corresponding sss and the path size are incremented, so the penalty remains unaffected. Otherwise, the path size remains unchanged and the penalty cannot decrease. In total, the penalty for s_j cannot be lower than the one for r_j.

We have shown that for both the total inter-cell penalty and the total intra-cell penalty the partition $\mathcal{C}'$ yields a solution that is at least as good as $\mathcal{C}$. In what follows, we can therefore safely assume that all cells are strongly connected.

Now, consider the weight function ω of the graph P. Again, we distinguish intra-cell queries and inter-cell queries. Since all cells must be strongly connected, we know that the inter-cell search spaces cover exactly the shortest paths from the source node to the target node. Intra-cell search spaces, however, were shown above to be equal to the search spaces caused by Dijkstra's algorithm on the corresponding cell, which only depends on the cell size. So neither the inter-cell sss nor the intra-cell sss depend on the edge weights.

Our objective is to find a partition that minimizes the total penalty. Provided that all cells are strongly connected, the overall inter-cell penalty is 0. Thus, we only have to minimize the intra-cell penalty of all cells given uniform edge weights. To determine the correct penalty, we enumerate the sizes of all distinct paths for a given cell C_i. Without loss of generality, let $C_i = \{v_1, \ldots, v_c\}$ with $c = |C_i|$. There are exactly two paths of size c, namely the paths $\langle v_1, \ldots, v_c \rangle$ and $\langle v_c, \ldots, v_1 \rangle$. Analogously, we have exactly the four paths $\langle v_1, \ldots, v_{c-1} \rangle$, $\langle v_2, \ldots, v_c \rangle$, $\langle v_{c-1}, \ldots, v_1 \rangle$, $\langle v_c, \ldots, v_2 \rangle$ of size $(c-1)$ and so forth. Finally, we have to account for $2(c-1)$ paths of size 2 and c paths of size 1 (paths where the source and target node are identical). Note that the latter case forms an exception, as we do not have to distinguish two directions. The sum of all these path sizes is summarized below.

$$\begin{aligned} \sum_{u,v \in C_i} |P_{u,v}| &= c + 2 \sum_{j=1}^{c-1} (c-j)(j+1) \\ &= \frac{1}{3}c^3 + c^2 - \frac{1}{3}c \end{aligned}$$

From the observations made above, we know that the intra-cell sss of all distinct queries within a given cell C_i is $\sum_{s,t\in C_i} \mathrm{S}(s,t) = |C_i| \sum_{j=1}^{|C_i|} j = |C_i|^2(|C_i|+1)/2$. Hence, we obtain the following total penalty.

$$\sum_{s,t\in V} \mathrm{pen}_{\mathcal{C}}(s,t) = \sum_{i=1}^{k} \left(\sum_{u,v\in C_i} \mathrm{S}(u,v) - \sum_{u,v\in C_i} |P_{u,v}| \right)$$
$$= \sum_{i=1}^{k} \left(\frac{1}{6}\,|C_i|^3 - \frac{1}{2}\,|C_i|^2 + \frac{1}{3}\,|C_i| \right)$$

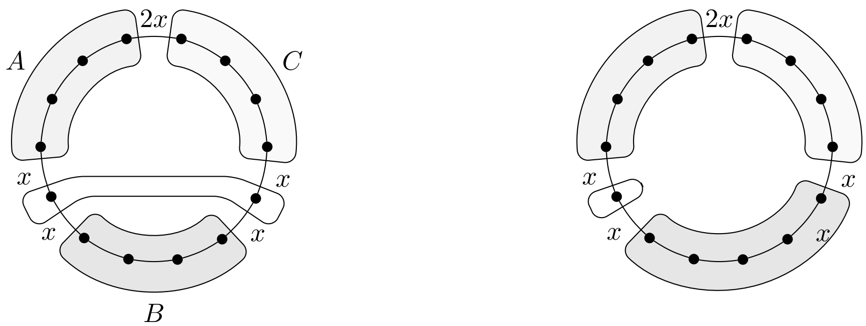
This is the same result as we obtained in Equation 1. Hence, we get similar optimal partitions for directed paths. □

4.2 Cycles

Observe that the sss of queries in a directed cycle is independent of the underlying partition, rendering the problem trivial for these graphs. On the other hand, we have seen in Section 3.3 that finding optimal cells on undirected cycles is nontrivial for worst-case optimization. Since the average-case minimization seems more difficult in general, we make the following simplification. We present an algorithm that computes optimal *connected* cells for cycles. Note that in general, an optimal partition may require disconnected cells, as shown in Figure 5. Here, x is a large number while all other edge weights are 1. The values of $\mathrm{S}_{\mathrm{avg}}$ induced by both partitions were obtained using an ILP solver and the algorithm presented below, respectively. An optimal partition with at most four cells inherently contains a disconnected cell. The construction of this counterexample is based on the following observation. The set flags of a given cell on a cycle depend on the overlap of the backward shortest-path trees that correspond to the cell. As a result, it is easy to see that an inter-cell query into this cell will either settle the same number of nodes as Dijkstra's algorithm or exactly the nodes on the shortest path from source to target. In other words, an inter-cell query achieves either a perfect speed-up or no speed-up at all. Since nodes in the sets A,B, and C share similar respective backward shortest-path trees, assigning these nodes to the same cell results in almost minimal sss of inter-cell queries into these cells. Since the number of cells is bounded by four, this leaves the two remaining (disconnected) nodes for the last cell.

The algorithm for computing an optimal partition with connected cells is based on the following observation. After choosing an orientation of the cycle $G = (V, E, \omega)$, a connected cell $C_{u,v}$ is uniquely described by two border nodes u and v, such that $C_{u,v}$ contains all nodes encountered when traversing the cycle from u to v along the chosen orientation, including u and v. Recall from the introduction that the flags for the cell $C_{u,v}$ only depend on $C_{u,v}$. Thus, given $C_{u,v}$, the sss $S_C(u,v) = \sum_{s\in V, t\in C_{u,v}} \mathrm{S}(s,t)$ of all s-t-queries with an arbitrary source $s \in V$ and a target $t \in C_{u,v}$ can be computed efficiently.

Using this observation, we describe a dynamic programming approach to compute optimal connected cells on undirected cycles. Let $V = \{v_1, \ldots, v_{|V|}\}$ be



(a) An optimal partition of the cycle, inducing $\mathrm{S_{avg}} = 994$.

(b) Optimal partition for connected cells, inducing $\mathrm{S_{avg}} = 996$.

Figure 5: A cycle with an optimal partition containing a disconnected cell.

indexed along the orientation of G and, without loss of generality, we assume that v_1 is the left boundary of a cell in an optimal partition (to preserve correctness, we simply consider each node v_i as the starting point once). We define a two-dimensional $|V| \times k$-table T, where $T[i,\ell]$ is the optimal sss of all s-t-queries with $s \in V$ and $t \in \{v_1, \ldots, v_i\}$ provided that $v_1, \ldots, v_i$ are partitioned into ℓ distinct cells. We initialize the first row by setting $T[i,1] = S_C(v_1, v_i)$. Moreover, T satisfies the following recurrence relation.

$$T[i,\ell] = \min_{1 \leq j \leq i-\ell+1} T[i-j, \ell-1] + S_C(v_{i-j+1}, v_i), \text{ for } i \geq \ell \geq 2.$$

This follows directly from the fact that the sss of queries into the ℓ-th cell is independent of the choice of the first $\ell - 1$ cells. Using this recurrence, the table entries can be filled in polynomial time. By definition, $T[n,k]$ is the sss of an optimal partition that contains the boundary v_1. By keeping track of the boundary nodes yielding the table entries, a partition with this sss can be computed in the same running time. We have the following theorem.

Theorem 10 *The problem* MinAvgCasePartition *on cycles can be solved in polynomial time if partitions are restricted to strongly connected cells.*

Clearly, replacing $S_C(u,v)$ by the corresponding worst-case sss and taking the maximum instead of the sum in the recurrence yields an algorithm that computes connected cells with minimum worst-case sss.

4.3 Hardness Results for Trees

We show that provided $\mathcal{P} \neq \mathcal{NP}$, there is no efficient algorithm that guarantees to find optimal cell assignments on undirected trees. The reductions given below are similar to those in Section 3.1, but proofs are significantly more involved due to the consideration of the average-case sss rather than the worst case.

Theorem 11 MinAvgCasePartition *is $\mathcal{NP}$-hard on undirected trees with uniform edge weights and a maximum height of* 2.

Proof: We use the reduction given in the proof of Theorem 2 to construct a tree $T = (V, E, \omega)$ from an instance (S, B) of 3-Partition. Let the root r have the smallest index in the ordering that is used for tie breaks in the query, that is, in any s-t-query, r is settled before all other nodes v with distance $d(s, v) = d(s, r)$. We establish a bound Γ such that (T, m) admits a partition $\mathcal{C}$ with $\mathrm{S}_{\mathrm{avg}} \leq \Gamma$ if and only if (S, B) is a Yes-instance.

Assume (S, B) is a Yes-instance and $S_1, \ldots, S_m$ a corresponding solution. Consider the partition $\mathcal{C} = \{C_1, \ldots, C_m\}$ where C_i contains all nodes of limbs corresponding to elements in S_i, and $r \in C_1$ (just as in the reduction we used to prove Theorem 2). We have $|C_1| = B + 1$ and $|C_i| = B$ for $i \geq 2$. We distinguish queries starting from three different types of nodes, namely the root node, element nodes and weight nodes.

For a query starting at r, we know that besides r, no nodes outside the target cell are settled. Also, for every cell C_i and every index $1 \leq j \leq |C_i|$, there is a unique node $t_{i,j}$ such that the query from r to $t_{i,j}$ settles exactly j nodes of C_i (this follows from the fact that nodes are settled in a deterministic order). Therefore, the total sss of queries from r to nodes in C_1 is $\sum_{t \in C_1} \mathrm{S}(r, t) = \sum_{j=1}^{B+1} j = (B + 1)(B + 2)/2$. For C_i with $i \geq 2$, we obtain $\sum_{t \in C_i} \mathrm{S}(r, t) = B + B(B + 1)/2$, because r is additionally settled in each of the B queries. This yields

$$\gamma_1 := \sum_{t \in V} \mathrm{S}(r, t) = |V| + m \cdot \frac{B(B + 1)}{2}, \text{ where } |V| = mB + 1.$$

Next, consider queries starting at an element node s_p. The node s_p is settled in every query. Since r has the least index regarding tie breaks and all flags on all incoming edges of r are set, the second node settled, if any, is always r. Let $\mathcal{S}(u, v)$ denote the set of settled nodes in a u-v-query. Clearly, we have $\sum_{t \in V} |\mathcal{S}(s_p, t) \cap \{s_p, r\}| = 2\,|V| - 1$ and besides s_p and r, no node outside the target cell is settled in an s_p-t-query. For a cell $C_i \in \mathcal{C}$, the total number of nodes in $C_i \setminus \{s_p, r\}$ settled in the $|C_i|$ distinct queries from s_p equals $|C_i \setminus \{s_p, r\}|(|C_i \setminus \{s_p, r\}| + 1)/2$. Observe that we have $|C_i \setminus \{s_p, r\}| = B$ if $s_p \notin C_i$ and $|C_i \setminus \{s_p, r\}| = B - 1$ otherwise. For the sss of all queries originating at s_p, this yields

$$\gamma_2 := \sum_{t \in V} \mathrm{S}(s_p, t) = 2|V| - 1 + (m - 1)\frac{B(B + 1)}{2} + \frac{B(B - 1)}{2}.$$

Finally, we account for queries from a leaf $w_{p,q}$ of the tree. We know that $w_{p,q}$ is settled in all $|V|$ distinct queries starting at $w_{p,q}$. The corresponding element node s_p is the only reachable node from $w_{p,q}$ and is always settled unless we have $s = t = w_{p,q}$. As we observed before, the first note settled after s_p (if any) is always r, which leaves us with $\sum_{t \in V} |\mathcal{S}(w_{p,q}, t) \cap \{w_{p,q}, s_p, r\}| = 3\,|V| - 3$. Along the lines of the argumentation for the element-node case, we infer a sss for the remaining parts of queries from $w_{p,q}$ that equals $|C_i \setminus \{w_{p,q}s_p, r\}|(|C_i \setminus \{w_{p,q}, s_p, r\}| + 1)/2$ for each cell $C_i \in \mathcal{C}$. We obtain the following sss for queries from an arbitrary leaf $w_{p,q}$.

$$\gamma_3 := \sum_{t \in V} \mathrm{S}(w_{p,q}, t) = 3|V| - 3 + (m-1)\frac{B(B+1)}{2} + \frac{(B-1)(B-2)}{2}.$$

The tree T consists of one root node, $3m$ element nodes and $mB - 3m$ weight nodes. Thus, setting $\Gamma = \gamma_1 + 3m\gamma_2 + m(B-3)\gamma_3$, we can assure that the inequality $\sum_{s,t \in V} \mathrm{S}(s,t) \leq \Gamma$ stated above is fulfilled by the partition $\mathcal{C}$.

For the other direction, assume we are given a partition $\mathcal{C} = \{C_1, \ldots, C_m\}$ of T such that the resulting sss is at most Γ. We show that T corresponds to a Yes-instance of 3-Partition. Again, we divide the sss into three components by distinguishing different types of source nodes. Without loss of generality, assume that $r \in C_1$. Then it suffices to show that $\mathcal{C}$ is perfect (cf. Theorem 1). To this end, we show that Γ in fact yields a tight lower bound on the total sss of T that is only reached if $\mathcal{C}$ is perfect. For every source node $s \in T$ we determine a subset $U \subseteq V$ such that $\sum_{t \in V} |\mathcal{S}(s,t) \cap U|$ is independent of the underlying partition $\mathcal{C}$. Observe that we actually did this before in order to obtain the values of γ_1, γ_2, and γ_3. To account for the remaining parts of the search spaces, consider the subgraph induced by the nodes in $V \setminus U$. For each target cell $C_i \in \mathcal{C}$, there are $c_i := |C_i \cap (V \setminus U)|$ distinct s-t-queries with $t \in C_i \cap (V \setminus U)$ and these c_i nodes are settled in a deterministic order. Thus, the overall sss of queries from s into the cell C_i within the considered subgraph must be at least $\sum_{t \in C_i \setminus U} |\mathcal{S}(s,t) \setminus U| \geq c_i(c_i+1)/2$. In order to reach this lower bound, one has to ensure that in no such query, any nodes outside $C_i \cup U$ are settled. Following this approach, we can show the claim given below, which immediately implies the theorem.

Claim 1 *The terms γ_1, γ_2, and γ_3 are tight lower bounds on the average sss of queries from the root node, an element node, and weight node, respectively. To reach the lower bound γ_1, the underlying partition must be perfect.*

To prove the claim, we first examine *queries starting at the root node r*. If we set $U = \{r\}$, the total number of nodes in U settled during all distinct queries from r equals $|V|$, regardless of the underlying partition. The argumentation given above then yields a lower bound on the sss of all queries from r that equals γ_1. Clearly, all limbs need to be monochromatic in order to reach this bound (otherwise, there exists at least one query in which nodes other than r outside the target cell are settled). Furthermore, we claim that the bound is reached only if $\mathcal{C}$ is a balanced partition. To see this, we derive a lower bound on the sss of queries from r in an arbitrary non-balanced partition by adapting balanced cell sizes using the two steps given in Lemma 2. Starting from the balanced case with $c_i = |C_i \setminus U| = B$ for all $1 \leq i \leq m$, we can without loss of generality perform our first step by simultaneously increasing one cell size by 1 and decreasing another cell size by 1. This yields an increase of the sss of

$$\frac{(B+1)(B+2)}{2} + \frac{B(B-1)}{2} - B(B+1) = 1.$$

We know from Lemma 2 that the sss is non-decreasing in all remaining steps. Hence, only a perfect partition enables us to reach the lower bound of γ_1.

For *queries from element nodes* $s_p \in V$, we set $U = \{s_p, r\}$. The source node s_p is settled in every query, so this node accounts for an amount of $|V|$ in the sss. Furthermore, we know that r is the second node to be settled if the corresponding flag of the edge (s_p, r) is set. Assume that this is the case for all flags pointing at r. Then r is settled on exactly $|V| - 1$ queries and we immediately obtain a lower bound that equals γ_2 for all queries starting at s_p. Conversely, assume that there exists a cell C_i such that its flag on the edge (s_p, r) has the value 0 for C_i. We show that the bound γ_2 must be exceeded in this case. Note that the edge (s_p, r) is only relevant for the sss of queries starting inside the limb ℓ_p. Moreover, queries from a node in ℓ_p to a node outside ℓ_p cannot benefit from the zero flag, because r must be passed in such a query and thus the target node is not in C_i (otherwise, the flag would be set). For a query where both the source and the target belong to ℓ_p, r is the only node outside ℓ_p that is settled by Dijkstra's algorithm, and hence we can save at most one settled node per query compared to any other partition. Since there are less than $B/2$ distinct targets inside ℓ_p, the gain is bounded by $B/2$ for a fixed source node. However, for the corresponding target flag to be 0, no node outside ℓ_p is allowed to be in C_i. Consider the sss of queries from s_p that we examined before. The sss of these queries is minimized if we find cells with balanced partial sizes $c_j = |C_j \setminus \{s_p, r\}|$. However, for the corresponding target flag to be 0, the cell C_i must contain less than $B/2$ nodes. Given a balanced partition, we can adapt the cell sizes c_j for $j \in \{1, \ldots, 3m\}$ using the operations of Lemma 2 to create a partition where $|C_i| < B/2$. Without loss of generality, in each step let only one cell size be decremented by 1, while another one is incremented by 1. Then we can identify more than $B/2$ steps in which some size c_j is increased by 1 and simultaneously, the size $c_i \leq c_j$ is decreased by 1. Since

$$\frac{(c_j+1)(c_j+2)}{2} + \frac{c_i(c_i-1)}{2} = \frac{c_j(c_j+1)}{2} + \frac{c_i(c_i+1)}{2} + c_j - c_i + 1$$

for $c_j \geq c_i > 0$, the total sss of queries from s_p increases by at least 1 in each of these steps. In total, the overall sss must exceed γ_2, and we can safely assume that in an optimal partition, all flags are set on an edge (s_p, r).

Finally, consider *queries from a leaf* $w_{p,q}$ *of* T. Setting $U = \{w_{p,q}, s_p, r\}$, we obtain a partial sss of $3n - 3$ along the lines of our previous observations. To find a lower bound on the remaining parts of queries from a leaf, we have to even the cell sizes $|C_i \setminus U|$. A partition with $|C_i \setminus U| = |C_j \setminus U| = B - 1$ for two cells C_i, C_j and $|C_k \setminus U| = B$ for all $k \neq i, j$ yields a lower bound on the sss of queries from $w_{p,q}$ that equals $\gamma_3 - 1$. Any other combinations leads to a sss of at least γ_3. Observe that the lower bound of $\gamma_3 - 1$ for $w_{p,q}$ can only be achieved if all limbs except for ℓ_p are monochromatic. Furthermore, it cannot be fulfilled by a perfect partition, for which we obtained a sss of γ_3. Assume we are given a partition such that $\sum_{t \in V} \mathrm{S}(w_{p,q}, t) = \gamma_3 - 1$ for at least one leaf $w_{p,q}$ of the limb ℓ_p. We distinguish two cases and prove that each time, the global sss must be strictly greater than Γ. In the investigations below, we assume that $m \geq 2$ and $B \geq 9$, which is fulfilled by any nontrivial instance of 3-Partition.

First, assume that the limb ℓ_p itself is monochromatic as well. Let C_i be the cell that all nodes in ℓ_p are assigned to. We have $\{w_{p,q}, s_p\} \subseteq C_i$ and thus $|C_i \setminus \{r\}| \geq B + 1$ (because $|C_i \setminus U| \geq B - 1$ holds). If we ignore r, the partition $\mathcal{C}$ therefore consists of one cell $C_i \setminus \{r\}$ of size $B + 1$, a cell $C_j \setminus \{r\}$ of size $B - 1$ and $m - 2$ cells $C_k \setminus \{r\}, k \neq i, j$ of size B. Consider the sss of queries starting at weight a node $w_{x,y}$ in a different limb ℓ_x assigned to a cell C_l, $l \neq i, j$. It is $|C_i \setminus \{w_{x,y}, s_x, r\}| = B+1$, $|C_j \setminus \{w_{x,y}, s_x, r\}| = B-1$, $|C_l \setminus \{w_{x,y}, s_x, r\}| = B-2$, and $|C_k \setminus \{w_{x,y}, s_x, r\}| = B$ for all $k \neq i, j, l$. This yields a lower bound on the sss of queries from $w_{x,y}$ of $\sum_{t \in V} \mathrm{S}(w_{x,y}, t) \geq \gamma_3 + 1$. Analogously, we obtain $\sum_{t \in V} \mathrm{S}(w_{x,y}, t) \geq \gamma_3 + 3$ for leaves $w_{x,y}$ of the cell C_j. Summing up, we achieve a sss of $\gamma_3 - 1$ for less than $B/2$ leaves, while all other leaves obtain a sss of at least $\gamma_3 + 1$. Since $m \geq 2$, there must be at least $3B/2 - 3 > B/2$ such leaves, and therefore the sss of all leaves must exceed $m(B - 3)\gamma_3$.

For the second case, assume that ℓ_p is not monochromatic. Let a denote the number of nodes assigned to a cell different from the cell that contains s_p, and let C_i denote this cell. The sss of an arbitrary weight node $w_{x,y}$ outside ℓ_p then is bounded by $\sum_{t \in V} \mathrm{S}(w_{x,y}, t) \geq \gamma_3 - 1 + a$, because in each query to one of the a mentioned leaves, the element node s_p gets settled despite not being in the target cell. Hence, if $a \geq 2$, we have a sss of at least $\gamma_3 + 1$ for at least $m(B - 1) - B/2 > B/2$ leaves of the tree, which outclasses the gain of the less than $B/2$ leaves of ℓ_p. Thus, we may safely assume that $a = 1$. Then we have a unique leaf in ℓ_p that is assigned to a certain cell C_j, while all remaining nodes in ℓ_p are assigned to C_i. Due to the fact that $|C_j \setminus \{w_{p,q}, s_p, r\}| \in \{B - 1, B\}$ and $s_p \notin C_j$, we know that $|C_j \setminus \{r\}|$ is in $\{B, B + 1\}$, depending on the cell assignment of $w_{p,q}$. If $|C_j \setminus \{r\}| = B$, it is $|C_j \setminus \{w_{x,y}, s_x r\}| \leq B - 2$ for all weight nodes $w_{x,y}$ of other limbs $\ell_x, x \neq p$ assigned to C_j (recall that ℓ_x is monochromatic). This yields a lower bound of at least γ_3 for all these leaves. Observe that there are at least $B - 4 > B/2$ such leaves $w_{x,y}$ outside ℓ_p assigned to C_j. If $|C_j \setminus \{r\}| = B + 1$, we have $|C_j \setminus \{w_{x,y}, s_x, r\}| = B + 1$ for all leaves outside $\ell_p \cup C_j$. Moreover, there must be more than $B/2$ nodes in $V \setminus (\ell_p \cup C_j)$. Hence, in both cases we obtain a lower bound of γ_3 for at least $B/2$ leaves. In addition to that, the lower bound of every leaf outside ℓ_p must have its lower bound increased by 1, because s_p is now additionally settled during the query to the unique leaf in ℓ_p that is assigned to C_j. In total, we obtain a bound of $\gamma_3 - 1$ for less than $B/2$ nodes, a bound of $\gamma_3 + 1$ for more than $B/2$ nodes and γ_3 for all other nodes. Therefore, the global sss of all leaves must exceed the bound of $m(B - 3)\gamma_3$ of a balanced partition with monochromatic limbs. It follows that $m(B - 3)\gamma_3$ is indeed a tight lower bound on the sss of all leaves of the tree, and only a perfect partition reaches this bound. $\square$

The next theorem shows that the problem MinAvgCasePartition is $\mathcal{NP}$-hard for directed trees, a subclass of directed acyclic graphs. Since directed acyclic graphs occur in the form of time-expanded graphs in time-dependent scenarios [14], this result is of vast importance for practical applications.

The outline of the proof of Theorem 12 is similar to the proof of Theorem 11. Replacing undirected edges by directed ones in the reduction, we first examine the sss of a perfect partition. Then we can show that this bound yields a tight lower bound on the sss that is reached if and only if the partition of the graph is perfect.

Theorem 12 MinAvgCasePartition *is $\mathcal{NP}$-hard on directed trees with uniform edge weights and a maximum height of* 2.

Proof: Given an instance (S', B') with $S = \{s'_1, \ldots, s'_{3m}\}$ of 3-Partition, we first make the following changes. We set $B := B' + 3\alpha$, and $\omega_i = \omega'_i + \alpha$ for $1 \leq i \leq 3m$, with α being specified below. Note that this does not affect solvability of the resulting instance. The reduction from (S, B) to an instance (T, m) then works very similar to the undirected case. Starting with a root node $r \in V$, for each element $s_p \in S$, we create a *limb* ℓ_p consisting of one *element node* s_p, $\omega_p - 1$ *weight nodes*, and directed edges connecting both r to s_p and s_p to all its weight nodes. We proceed along the lines of Theorem 11. We claim that there exists a bound Γ such that (T, m) admits a partition $\mathcal{C}$ with $\sum_{s,t \in T} \mathrm{S}(s,t) \leq \Gamma$ if and only if (S, B) is a Yes-instance of 3-Partition.

Assume that (S, B) is a Yes-instance and let $S_1, \ldots, S_m$ be a corresponding solution. Consider the partition $\mathcal{C} = \{C_1, \ldots, C_m\}$ where C_i contains all nodes of the limbs corresponding to elements in S_i, and additionally $r \in C_1$. Again, we have $|C_1| = B + 1$ and $|C_i| = B$ for $i \geq 2$. To analyze the sss induced by $\mathcal{C}$, we distinguish three types of queries. To begin with, a query starting at r uses only edges pointing away from r, we face the exact same situation as in the undirected case. The total sss of queries from r is

$$\gamma_1 := \sum_{t \in V} \mathrm{S}(r,t) = |V| + m \cdot \frac{B(B+1)}{2}.$$

Next, we examine the sss of queries starting at a fixed element node s_p that is assigned to a cell C_i with a target node $t \neq r$. First, observe that an s_p-t-query settles only the source node s_p if $t \notin C_i$. This yields a sss of 1 for $B(m-1)$ distinct queries from s_p. Additionally, we have to consider intra-cell queries where $t \in C_i$. There are ω_p nodes (including s_p itself) in cell C_i that are in ℓ_p and thus reachable from s_p. If the target node is part of the same limb, the number of settled nodes depends on the target node index used for tie-breaks and ranges from 1 to ω_p. As a main difference to the undirected case, however, all queries from s_p to target nodes $t \in C_i$ outside ℓ_p cause all ω_p reachable nodes to be settled. Since $|C_i \setminus \{r\}| = B$, we obtain the following total sss of queries starting at s_p to all targets except r.

$$\gamma_2 := \sum_{t \in V \setminus \{r\}} \mathrm{S}(s_p, t) = \frac{\omega_p(\omega_p + 1)}{2} + (B - \omega_p)\omega_p + B(m-1)$$

We also have to account for the query from s_p to r. The number of settled nodes in this query is ω_p if $p = 1$ (i.e., the s_p-r-query is an intra-cell query), and 1 otherwise. Summing up for all element nodes $s_p \in V$, this yields $B + 3(m-1)$ in total (recall that the weights ω_p sum up to B for each cell and in particular for C_1). Together with γ_2, we get the following sss for queries from all element nodes to all distinct targets of the tree.

$$\begin{aligned}\sum_{i=1}^{3m} \sum_{t \in V} \mathrm{S}(s_p, t) &= \sum_{p=1}^{3m} \left(\frac{\omega_p^2 + \omega_p}{2} + B\omega_p - \omega_p^2 \right) + 3m(mB - B + 1) + B - 3 \\ &= \underbrace{m\left(B^2 + \frac{B}{2}\right)}_{\lambda_1^*} - \underbrace{\sum_{p=1}^{3m} \frac{\omega_p^2}{2}}_{\lambda_2^*} + \underbrace{3m(mB - B + 1) + B - 3}_{\lambda_3^*} \qquad (2)\end{aligned}$$

Finally, there are no reachable nodes from an arbitrary weight node $w_{p,q}$ of the tree, so the sss of a query from $w_{p,q}$ is always 1 and the total sss of all queries where the source node is a leaf of the directed tree is constant.

$$\gamma_3 := \sum_{t \in V} \mathrm{S}(w_{p,q}, t) = mB + 1$$

All in all, the value Γ stated below makes sure that the partition $\mathcal{C}$ satisfies the inequality $\sum_{s,t \in V} \mathrm{S}(s,t) \leq \Gamma$.

$$\Gamma = \gamma_1 + m\left(B^2 + \frac{B}{2}\right) - \sum_{i=1}^{3m} \frac{\omega_i^2}{2} + 3m(mB - B + 1) + B - 3 + m(B-3)\gamma_3$$

For the other direction, assume we are given a partition $\mathcal{C} = \{C_1, \ldots, C_m\}$ of T such that the resulting sss is at most Γ. We show that T corresponds to a Yes-instance of 3-Partition. Without loss of generality, assume that $r \in C_1$. We show that $\sum_{s,t \in V} \mathrm{S}(s,t) \leq \Gamma$ if and only if $\mathcal{C}$ is perfect, i.e., $\mathcal{C}$ is balanced and contains only monochromatic limbs.

Recall that the situation for queries from r is similar to the undirected case. Following the arguments in the proof of Theorem 11, we thus know that $\sum_{t \in V} \mathrm{S}(r,t) \leq \gamma_1$ if and only if $\mathcal{C}$ is a perfect partition. Moreover, the sss of an arbitrary query that starts at a leaf is 1, independent of the underlying partition. Therefore, the total sss of queries starting at leaves of the tree is always $m(B-3)\gamma_3$.

What is left to take into consideration is the sss of queries from element nodes. We examine the sss of queries starting at a fixed element node $s_p \in C_i$.

First, consider intra-cell queries where s and t belong to the same cell C_i. Let $\rho_{p,i}$ denote the number of nodes in cell C_i that are reachable from s_p. If the target node is part of the same limb, the number of settled nodes depends on the target node index and is at most $\rho_{p,i}$. Otherwise, all $\rho_{p,i}$ reachable nodes of the target cell are settled. We obtain the following total sss of intra-cell queries starting at s_p into its own cell C_i.

$$\begin{aligned}\sum_{t \in C_i} \mathrm{S}(s_p, t) &= (|C_i| - \rho_{p,i})\, \rho_{p,i} + \sum_{z=1}^{\rho_{p,i}} z \\ &= |C_i|\, \rho_{p,i} + \frac{1}{2}\, \rho_{p,i} - \frac{1}{2}\, {\rho_{p,i}}^2 \qquad (3)\end{aligned}$$

As for an inter-cell query, the only difference is that we have to account for the fact that the source node s_p gets settled in every query, although it is not a member of the target cell. This yields the following sss of all queries from s_p into a cell $C_j \neq C_i$.

$$\begin{aligned}\sum_{t \in C_j} \mathrm{S}(s_p, t) &= (|C_j| - \rho_{p,j})\, (\rho_{p,j} + 1) + \sum_{z=2}^{\rho_{p,i}+1} z \\ &= |C_j|\, \rho_{p,j} + \frac{1}{2}\, \rho_{p,j} - \frac{1}{2}\, {\rho_{p,j}}^2 + |C_j| \qquad (4)\end{aligned}$$

Let $\overline{s}_i = |\{s_p \mid s_p \notin C_i\}|$ denote the number of element nodes in V that are *not* assigned to C_i. Using Equations 3 and 4, we obtain the following sss summed up for all queries from element nodes. Note that for all element nodes assigned to the cell C_i, the values $\rho_{p,i}$ sum up to $|C_i|$ if $r \notin C_i$ and $|C_i| - 1$ otherwise.

$$\begin{aligned}\lambda &:= \sum_{p=1}^{3m} \sum_{i=1}^{m} \sum_{t \in C_i} \mathrm{S}(s_p, t) \\ &= \sum_{i=1}^{m} \left(\overline{s}_i\, |C_i| + \sum_{p=1}^{3m} \left(|C_i|\, \rho_{p,i} + \frac{1}{2}\, \rho_{p,i} - \frac{1}{2}\, {\rho_{p,i}}^2 \right) \right) \\ &= \underbrace{\sum_{i=1}^{m} \left(|C_i|^2 + \frac{|C_i|}{2} \right) - |C_1| - \frac{1}{2}}_{\lambda_1} - \underbrace{\sum_{i=1}^{m} \sum_{p=1}^{3m} \frac{{\rho_{p,i}}^2}{2}}_{\lambda_2} + \underbrace{\sum_{i=1}^{m} \overline{s}_i\, |C_i|}_{\lambda_3} \qquad (5)\end{aligned}$$

We examine the partial terms $\lambda_1, \lambda_2, \lambda_3$ separately and compare each term to a respective term $\lambda_1^*, \lambda_2^*, \lambda_3^*$ of the sss of a perfect partition given in Equation 2. First, we show that λ_1^* is in fact a lower bound on the term λ_1. The minimization of the sum $\sum_{i=1}^{m}(|C_i|^2 + |C_i|/2)$ by a balanced partition follows directly from Corollary 1. Choosing C_1 to be the largest cell clearly is beneficial as the value $|C_1|$ is subtracted in λ_1. Moreover, we can use Lemma 2 to show that increasing the size of the cell C_1 to a value greater than $B + 1$ implies that the bound λ_1 is exceeded. Starting from a balanced partition, we construct any partition with

$|C_1| > B + 1$ in small steps, during which the size $|C_1|$ is increased by 1, while another cell size $|C_i| \leq |C_1|$ is decreased by 1. For the sum of the quadratic terms $|C_i|^2$, this yields a difference of at least 2, because

$$|C_1 + 1|^2 + |C_i - 1|^2 = |C_1 + 1|^2 + |C_i - 1|^2 + 2|C_1| - 2|C_i| + 2.$$

The sss is increasing in any possible remaining steps. Hence, the sss increases whenever C_1 is increased, and we reach the lower bound λ_1^* only if the partition $\mathcal{C}$ is balanced and additionally $|C_1| = B + 1$.

In order to minimize the sss, the terms λ_2 in Equation 5 should be maximized. Therefore, the values of $\rho_{p,i}$ need to be as great as possible. Obviously, this is the case if and only if all limbs are monochromatic, as was the case for λ_2^*. Up to now, we have shown that γ_1, γ_3, λ_1^*, and λ_2^* are indeed tight lower bounds on the sss, and the partition $\mathcal{C}$ has to be perfect to reach all these bounds. Finally, we have to consider the terms $\lambda_3 = \sum_{i=1}^{m} \overline{s}_i \, |C_i|$. For the first time, we may actually come below the value λ_3^* of a perfect partition. For example, assigning all nodes of the graph to the same cell yields $\sum_{i=1}^{m} \overline{s}_i \, |C_i| = 0$ (recall that $\overline{s}_i$ denotes the number of element nodes *not* in C_i). Globally, this clearly is not beneficial. In what follows, we show that any partition other than a perfect one that yields a value $\lambda_3 < \lambda_3^*$ inherently leads to a global sss greater than Γ.

Recall that in a perfect partition it is $\lambda_3 = \lambda_3^*$. An arbitrary partition with $\lambda_3 < \lambda_3^*$ can be constructed by modifying a given perfect partition. In order to decrease λ_3 starting from a perfect partition, one has to change the sizes of some cells. In particular, one needs to assign many element nodes to large cells. We can construct any value for λ_3 that corresponds to a valid partition $\mathcal{C}$ by reassigning $c \geq 0$ nodes of a given perfect partition in total, while reassigning $e \geq 0$ element nodes. Clearly, in order to reach a small sss, larger cells should have smaller corresponding values $\overline{s}_i$ and vice versa. Hence, we may restrict ourselves to modifications that create such cells. Then we can construct a combination of values $\overline{s}_i$ and $|C_i|$ that induce an arbitrary sss $\lambda_3 \leq \lambda_3^*$ as follows. Given a perfect partition with $|C_i| \in \{B, B+1\}$ and $\overline{s}_i = 3(m-1)$ for all i, we increase some values $\overline{s}_i$ by $e_i \geq 0$ while decreasing $|C_i|$ by $c_i \leq 0$, and decrease some other values $\overline{s}_i$ by $e_i \leq 0$ while increasing $|C_i|$ by $c_i \geq 0$. This yields the following partial sss.

$$\lambda_3 = \lambda_3^* - \left(\sum_{i=1}^{m} \left((\overline{s}_i + e_i)(|C_i| + c_i) - \overline{s}_i |C_i| \right) \right) = \lambda_3^* - \sum_{i=1}^{m} e_i c_i - e_1 \quad (6)$$

In what follows, we derive bounds on the increase of the sss induced by its remaining terms, to show that we cannot achieve a global improvement. For the term λ_1, we now have to pay the following penalty.

$$\lambda_1 \geq \lambda_1^* + \sum_{i=1}^{m} \left((B + c_i)^2 - B^2 \right) - c_1 = \sum_{i=1}^{m} c_i^2 - c_1 \quad (7)$$

Next, we consider the sss induced by the root node in comparison to γ_1. Compared to a perfect partition, we reassign $\sum |e_i|/2$ element nodes in total. Consider a cell C_i for which we then have $\overline{s}_i = 3(m-1) - e_i$ for an $e_i < 0$, i.e., C_i contains $3 + |e_i| > 3$ element nodes. Let W_i be the set of leaves reachable from these $3 + |e_i|$ element nodes. Observe that by construction of the graph T, the set W_i contains at least $(3 + |e_i|)\alpha + B'$ nodes. Let $\overline{w}_i$ be the number of leaves in W_i *not* assigned to C_i, i.e., $\overline{w}_i = |W_i \setminus C_i|$. We distinguish two different cases.

First, assume that $\overline{w}_i \geq |e_i|\alpha/2$. There must be at least $|e_i|\alpha/2$ distinct r-t-queries in which an element node that lies outside the target cell additionally gets settled. Therefore, the lower bound γ_1 is exceeded by at least $|e_i|\alpha/2$. Conversely, let $\overline{w}_i < e_i\alpha/2$. Analogously to Equation 7, the increased cell size of C_i makes the sss induced by the root node γ_1 grow by $\sum_{j=1}^{m} |c_j^2|/2$. Since we have $\overline{w}_i < |e_i|\alpha/2$, the cell C_i has a size of at least $3\alpha + B' + |e_i|\alpha/2$ and thus $c_i \geq |e_i|\alpha/2$. This yields $c_i^2/2 \geq (|e_i|\alpha)^2/2 \geq |e_i|\alpha/2$. In total, we obtain the following bound that sums up the penalties for all cells.

$$\gamma_1 = \gamma_1^* + \sum_{i=1}^{m} |e_i|\alpha/4 \tag{8}$$

To prove the claim that a partial sss of $\lambda_3 < \lambda_3^*$ is not globally optimal, we consider a fixed cell C_i. The gain that we can assign to this cell is at most $e_i c_i + e_i$ as in Equation 6. If $c_i \geq 6m$, the gain for this cell is thus bounded by $c_i^2/2 + c_i/2$, because e_i is at most $3m \leq c_i/2$. On the other hand, we know from Equation 7 that the sss increases by at least $c_i^2 - c_i$, which exceeds the gain for $c_i \geq 6m$. If $c_i < 6m$, the gain is below $6me_i + e_i = (6m+1)e_i$ compared to a penalty of $e_i\alpha/4$ given by Equation 8. Setting $\alpha > 24m + 4$, the penalty outweighs the gain. Observe that we can apply these arguments to all cells independently, which proves the claim.

It follows that we minimize the overall sss if and only if we balance the limbs to cells of equal size each, corresponding to triples of total weight B. Furthermore, the reduction can be performed in polynomial time. □

Finally, we mention that MinAvgCasePartition on stars can be solved efficiently. Using arguments similar to the worst-case analysis at the end of Section 3.1, it is easy to see that balanced cell sizes yield optimal partitions. Thus, we have established a border between hard instances and those solvable in polynomial time for the average case as well.

5 Conclusion

We investigated the complexity of two computational problems concerning graph partitioning for arc-flags on several classes of graphs. It turned out that in both cases, solving even very restricted classes of trees is $\mathcal{NP}$-hard. This yields a substantial improvement of the known general hardness result. Together with the efficiently computable partitions on paths and stars, our results also provide a tight border of tractability for both problems. In addition to that, it seems that the introduction of cycles, and thus ambiguity of shortest paths, vastly increases the difficulty of the problems. In fact, the complexity of both problems remains unknown on cycles.

As an insight from the analysis of trees, a major difficulty seems to be the computation of connected cells of balanced size. Both the reductions used and the approximation algorithms presented support this hypothesis. One may take this as a theoretical approval of practical heuristics, which essentially aim at finding such cells. Obtained hardness results were similar for both problems on all examined graph classes. Since the worst-case sss seems to allow for a much simpler examination, the investigation of the problem MinWorstCasePartition provides a reasonable alternative to gain further insights into the complexity of preprocessing arc-flags or speed-up techniques in general.

Besides the complexity of cycles, the primary open question would be whether there exist better approximation algorithms or inapproximability results for trees as well as more general classes of graphs. For example, is it possible to generalize the approximation algorithms for trees to graphs of bounded treewidth? Moreover, the complexity of MinWorstCasePartition or MinAvgCasePartition is unclear if the input parameter k is replaced by a fixed constant.

References

[1] I. Abraham, A. Fiat, A. V. Goldberg, and R. F. Werneck. Highway Dimension, Shortest Paths, and Provably Efficient Algorithms. In M. Charikar, editor, *Proceedings of the 21st Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'10)*, pages 782–793. SIAM, 2010.

[2] R. Bauer, T. Columbus, B. Katz, M. Krug, and D. Wagner. Preprocessing Speed-Up Techniques is Hard. In *Proceedings of the 7th Conference on Algorithms and Complexity (CIAC'10)*, volume 6078 of *Lecture Notes in Computer Science*, pages 359–370. Springer, 2010. `doi:10.1007/978-3-642-13073-1_32`.

[3] R. Bauer, T. Columbus, I. Rutter, and D. Wagner. Search-Space Size in Contraction Hierarchies. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP'13)*, Lecture Notes in Computer Science. Springer, 2013. To appear.

[4] R. Bauer and D. Delling. SHARC: Fast and Robust Unidirectional Routing. *ACM Journal of Experimental Algorithmics*, 14(2.4):1–29, August 2009. Special Section on Selected Papers from ALENEX 2008. `doi:10.1145/1498698.1537599`.

[5] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner. Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm. *ACM Journal of Experimental Algorithmics*, 15(2.3):1–31, January 2010. Special Section devoted to WEA'08.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[7] D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering Route Planning Algorithms. In J. Lerner, D. Wagner, and K. A. Zweig, editors, *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009. `doi:10.1007/978-3-642-02094-0_7`.

[8] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of $\mathcal{NP}$-Completeness.* W. H. Freeman and Company, 1979.

[10] E. Köhler, R. H. Möhring, and H. Schilling. Acceleration of Shortest Path and Constrained Shortest Path Computation. In *Proceedings of the 4th Workshop on Experimental Algorithms (WEA'05)*, volume 3503 of *Lecture Notes in Computer Science*, pages 126–138. Springer, 2005. `doi:10.1007/11427186_13`.

[11] U. Lauther. An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background. In *Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung*, volume 22, pages 219–230. IfGI prints, 2004.

[12] S. Micali and V. V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matchings in general graphs. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science (FOCS'80)*, pages 17–27, 1980. doi:10.1109/SFCS.1980.12.

[13] R. H. Möhring, H. Schilling, B. Schütz, D. Wagner, and T. Willhalm. Partitioning Graphs to Speedup Dijkstra's Algorithm. *ACM Journal of Experimental Algorithmics*, 11(2.8):1–29, 2006. doi:10.1145/1187436.1216585.

[14] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12(2.4):1–39, 2008. doi:10.1145/1227161.1227166.