Project Number 288094

eCOMPASS

eCO-friendly urban Multi-modal route PlAnning Services for mobile uSers

STREP

Funded by EC, INFSO-G4(ICT for Transport) under FP7

eCOMPASS – TR – 007

On the Complexity of Partitioning Graphs for Arc-Flags

Reinhard Bauer, Moritz Baum, Ignaz Rutter, and Dorothea Wagner

September 2012

On the Complexity of Partitioning Graphs for Arc-Flags*

Reinhard Bauer, Moritz Baum, Ignaz Rutter, and Dorothea Wagner

Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
`firstname.lastname@kit.edu`

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.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases shortest paths, arc-flags, search space, preprocessing, complexity

Digital Object Identifier 10.4230/OASIcs.ATMOS.2012.71

1 Introduction

In recent years, route planning has become a widely known application of algorithm engineering. Although Dijkstra's algorithm [6] 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. [5]. Here, we focus on one particular technique. The idea of *arc-flags* was first introduced by Lauther [9]. The basic approach was exhaustively evaluated in experimental studies, see for example Köhler et al. [8] and Möhring et al. [11]. Moreover, it was combined with other techniques in order to gain additional speed-up [2, 3].

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.

* Partially supported by DFG grant WA 654/16-2, by BMWi grant iZeus, and by the EU FP7/2007-2013 (DG INFSO.G4-ICT for Transport), under grant agreement no. 288094 (project eCOMPASS).

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

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

The preprocessing of the arc-flags algorithm computes a partition $\mathcal{C}$ of the input graph into k cells and detects 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 [1]. 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 [12].
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.

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 an 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 force cells to be connected. Table 1 shows an overview of our results. We conclude our work and discuss open questions in Section 5.

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

2 Preliminaries

We assume familiarity with basic concepts from graph theory and shortest-path search; see the book by Cormen et al. [4] for foundations in this area. We consider *directed weighted graphs*, denoted by a triple $G = (V, E, \omega)$, where ω is a weight function. 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* of a path $P = \langle v_1, \dots, v_k \rangle$ 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 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 [6] 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 $\{1, \dots, |V|\}$ 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. [5]). 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 $\mathrm{S}_{\text{avg}}(G, \mathcal{C}) := \sum_{s,t \in V} \mathrm{S}(G, \mathcal{C}, s, t)$. To obtain the actual average sss we would need to divide $\mathrm{S}_{\text{avg}}(G, \mathcal{C})$ 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 [7]. An instance of 3-Partition is a tuple (S, B), where B is a positive integer and $S = \{s_1, \dots, s_{3m}\}$ is a set of $3m$ elements, such that each element s_i is associated with a 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, $j \in \{1, \dots, m\}$, 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 S_{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 and cycles and show $\mathcal{NP}$-hardness for directed and undirected trees.

3.1 Paths and Cycles

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 is always $|V|$. A similar argument holds for directed cycles.

To examine 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 immediately yields 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 $k_{\text{opt}}(G, W)$ 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 approximate $k_{\text{opt}}(G, W)$.

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

Proof. For 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 1.

▶ Claim 1. 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) $V(G_{u,v}) \cap X_t = \emptyset$.

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

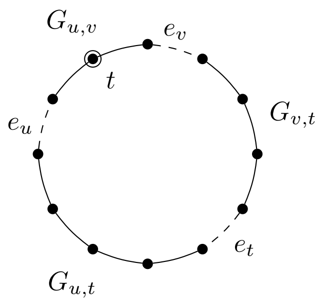

Figure 1 The three subgraphs $G_{u,v}$, $G_{u,t}$, and $G_{v,t}$ with respect to a certain node t.

▶ Claim 2. 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.

We omit the proofs of both claims. Because all nodes in U_t lie between two consecutive cut edges, it follows from Claim 1 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 t, t' where $U_t = U_{t'}$ or $U_t = U'_t$, we obtain a number of distinct connected subsets $U_i \subseteq V$ (connectivity holds by Claim 2). 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 [10] 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. ◀

3.2 Hardness Results for Trees

We prove hardness on trees with uniform edge weights and height 2 in Theorem 2 given below. Hence, the problem MinWorstCasePartition remains $\mathcal{NP}$-hard even with severe restrictions to the graph structure.

▶ **Theorem 2.** *The problem* MinWorstCasePartition *is $\mathcal{NP}$-hard for rooted directed trees of height at most* 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 2 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.

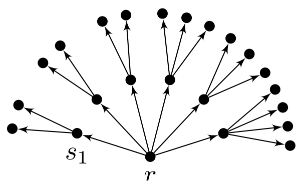

Figure 2 The reduction of an instance with $m = 2, B = 11$ and weights $3, 3, 3, 4, 4, 5$.

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.

Conversely, assume that $\mathcal{C} = \{C_1, \dots, C_m\}$ 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.

Observe 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 (because the order in which nodes are settled from a fixed source node is deterministic). 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. ◀

Modifying the reduction used in Theorem 2, we can also prove hardness if we limit the maximum outdegree of a tree to a constant greater or equal 2.

▶ **Theorem 3.** MINWORSTCASEPARTITION *is $\mathcal{NP}$-hard for rooted directed trees with a maximum outdegree of at most* 2*, even in case of uniform edge weights.*

Moreover, we consider undirected trees. Using a very similar reduction compared to the proof of Theorem 2, we obtain the following result.

▶ **Theorem 4.** MINWORSTCASEPARTITION *is $\mathcal{NP}$-hard for undirected trees with height at most* 2*, even in case of uniform edge weights.*

Again, this proof carries over to the case where the degree is restricted to 3. Note that a maximum outdegree of 2 leads to the trivial graph class of paths.

▶ **Theorem 5.** MINWORSTCASEPARTITION *is $\mathcal{NP}$-hard for undirected trees with a maximum degree of at most* 3*, even in case of uniform edge weights.*

Restricting both the degree and the height of the tree restricts its size, and thus renders the problem MINWORSTCASEPARTITION efficiently solvable. Essentially, the remaining class of trees that we have not covered so far is the class of stars (i.e., trees with height at most 1). Considering a directed star, the sss of a query starting at an arbitrary leaf is 1. On an undirected star, starting from a leaf, the second node that is settled is always the root node. Hence, in both cases it suffices to minimize the worst-case sss of queries from the root node. Clearly, this is achieved if the cell sizes are balanced. In total, we obtain a tight border of tractability for the problem MinWorstCasePartition.

3.3 An Approximation Algorithm for Trees

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 2 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 $\mathcal{C}$ given 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 6.** *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.*

We prove approximation guarantees for the algorithm TreeApprox. Theorem 7 provides a first bound, which can be improved for undirected trees.

▶ **Theorem 7.** *Algorithm* TreeApprox *is a 3-approximation for* 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 at most all nodes in C_i plus an additional node, and the largest possible path outside C_i leading into this cell. 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 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 $\mathrm{OPT} \geq \max\{\Delta, \lceil |V|/k \rceil\}$ (this holds for directed trees as well, since there must exist a root node from which all nodes are reachable). It follows immediately that $\mathrm{ALG} \leq 3 \cdot \mathrm{OPT}$. ◀

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 8.** *Algorithm* TreeApprox *is a* $5/2$*-approximation for* MinWorstCasePartition *on undirected trees.*

4 Minimizing the Average Search-Space Size

Since MinAvgCasePartition is $\mathcal{NP}$-hard in general [1], we investigate restricted input instances. Along the lines of Section 3, we examine paths, cycles, stars, and trees.

4.1 Paths and Cycles

First, we consider paths. Given a graph consisting of a single undirected 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 9.** *Let P be an undirected 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.*

The following Theorem 10 shows that the partition $\mathcal{C}_{\mathrm{opt}}$ optimizes the average sss on directed paths as well. The proof is very similar to the undirected case.

▶ **Theorem 10.** *Let P 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.*

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 3.1 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 3. Here, x is a large number while all other edge weights are 1. It can be shown that an optimal partition with at most four cells inherently contains the disconnected white cell. The rough idea is that making A,B, and C cells of the partition results in a very small sss of all queries into these comparatively large cells. Since the number of cells is bounded by four, this leaves the two remaining (disconnected) nodes for the last cell.

The algorithm 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 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

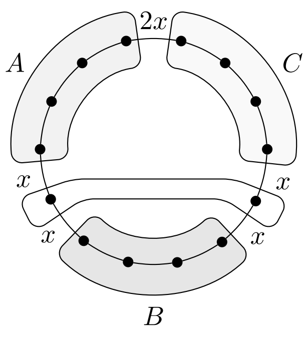

Figure 3 An example of a cycle with an optimal partition containing a disconnected cell.

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 $t \in \{v_1, \dots, v_i\}$ provided that $v_1, \dots, v_i$ 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.

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

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 11.** *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.

4.2 Hardness Results for Trees

We show that provided $\mathcal{P} \neq \mathcal{NP}$, there is no efficient algorithm that can guarantee to find optimal cell assignments on undirected trees.

▶ **Theorem 12.** 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 4 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 $\mathrm{S}_{\text{avg}} \le \Gamma$ 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. Consider the partition $\mathcal{C} = \{C_1, \dots, C_m\}$ where C_i contains all nodes of limbs corresponding to elements in S_i, and $r \in C_1$. We have $|C_1| = B + 1$ and $|C_i| = B$ for $i \ge 2$. We distinguish queries starting from three different types of nodes.

For a query starting at r, we know that besides r, no nodes outside the target cell are settled. For every cell C_i and every index $1 \le j \le |C_i|$, there is a distinct node $t_{i,j}$ such that the query from r to $t_{i,j}$ settles exactly j nodes of C_i. 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 an 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 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, leaving 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 and distinguish queries with respect to their source nodes. Without loss of generality, assume that $r \in C_1$. Then it suffices to show that $\mathcal{C}$ is perfect (cf. Theorem 2). 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, nodes from another cell are additionally settled. Following this approach, we can show the following claim.

▶ Claim 3. 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, or a leave of the tree, respectively. To reach the lower bound γ_1, the underlying partition must be perfect.

We omit the rather technical proof here. Since only a YES-instance admits a perfect partition, this completes the proof. ◀

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 [12], this result is of vast importance for practical applications.

▶ **Theorem 13.** MINAVGCASEPARTITION *is $\mathcal{NP}$-hard on directed trees with uniform edge weights and a maximum height of* 2.

The outline of the proof of Theorem 13 is similar to the proof of Theorem 12. 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.

Finally, we mention that MINAVGCASEPARTITION on stars can be solved efficiently. Using arguments similar to the worst-case analysis at the end of Section 3.2, 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 the computational problems MINWORSTCASEPARTITION and MINAVGCASEPARTITION 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 algorithm presented support this hypothesis. One may take this as a theoretical approval of practical heuristics, which essentially aim at finding cells that have such structure. The 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.

References

1 Reinhard Bauer, Tobias Columbus, Bastian Katz, Marcus Krug, and Dorothea 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.

2 Reinhard Bauer and Daniel 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.

3 Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea 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.

4 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 2nd edition, 2001.

5 Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering Route Planning Algorithms. In Jürgen Lerner, Dorothea Wagner, and Katharina A. Zweig, editors, *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009.

6 Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

7 Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of $\mathcal{NP}$-Completeness*. W. H. Freeman and Company, San Francisco, CA, USA, 1979.

8 Ekkehard Köhler, Rolf H. Möhring, and Heiko 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.

9 Ulrich 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.

10 Silvio Micali and Vijay 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.

11 Rolf H. Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, and Thomas Willhalm. Partitioning Graphs to Speedup Dijkstra's Algorithm. *ACM Journal of Experimental Algorithmics*, 11(2.8):1–29, 2006.

12 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12(2.4):1–39, 2007.