On the Complexity of Local Graph Transformations

Christian Scheideler
Paderborn University, Germany
https://cs.uni-paderborn.de/en/ti/
scheideler@upb.de

Alexander Setzer
Paderborn University, Germany
https://cs.uni-paderborn.de/en/ti/
asetzer@mail.upb.de

Abstract
We consider the problem of transforming a given graph $G_s$ into a desired graph $G_t$ by applying a minimum number primitives from a particular set of local graph transformation primitives. These primitives are local in the sense that each node can apply them based on local knowledge and by affecting only its 1-neighborhood. Although the specific set of primitives we consider makes it possible to transform any (weakly) connected graph into any other (weakly) connected graph consisting of the same nodes, they cannot disconnect the graph or introduce new nodes into the graph, making them ideal in the context of supervised overlay network transformations. We prove that computing a minimum sequence of primitive applications (even centralized) for arbitrary $G_s$ and $G_t$ is NP-hard, which we conjecture to hold for any set of local graph transformation primitives satisfying the aforementioned properties. On the other hand, we show that this problem admits a polynomial time algorithm with a constant approximation ratio.

This publication is the full version of a paper appearing at ICALP’19. The main part of this paper is identical to that paper and any additional material is provided in the appendix.

1 Introduction
Overlay networks are used in many contexts, including peer-to-peer systems, multipoint VPNs, and wireless ad-hoc networks. In fact, any distributed system on top of a shared communication infrastructure usually has to form an overlay network (i.e., its participating sites have to know each other or at least some server) in order to allow its members to exchange information.

A fundamental task in the context of overlay networks is to maintain or adapt its topology to a desired topology, where the desired topology might either be pre-defined or depend on a certain objective function. The problem of reaching a pre-defined topology has been extensively studied in the context of self-stabilizing overlay networks (e.g., [29] [21] [12] [5] [22] [7]), and the problem of adapting the topology based on a certain objective function has been studied in the context of self-adapting and -optimizing overlay networks (e.g., [32] [14] [2] [19] [11] [3] [10] [8]). Many of these approaches are decentralized, and because of that, the work (in terms of number of edge changes) they need to adapt to a desired topology might be far away from the minimum possible work to reach that topology. In fact, no non-trivial results on the competitiveness of decentralized overlay network adaptations are known.
so far other than handling single join or leave operations, and it is questionable whether any good competitive result can be achieved with a decentralized approach. An alternative approach would be that a server is available for controlling the network adaptations, and this has already been considered in the context of so-called supervised overlay networks.

In a supervised overlay network there is a dedicated, trusted node called supervisor that controls all network adaptations but otherwise is not involved in the functionality of the overlay network (such as serving search requests), which is handled in a peer-to-peer manner. This has the advantage that even if the supervisor is down, the overlay network is still functional. Solutions for supervised overlay networks have been proposed in [24, 15], for example, and the results in [24] imply that, for specific overlay networks, any set of node arrivals and departures can be handled in a constant competitive fashion (concerning the work needed for adding and removing edges) to get back to a desired topology. But no general result is known so far for supervised overlay networks concerning the competitiveness of converting an initial topology into a desired topology. Also, no result is known so far on how to handle the problem that a supervisor could be faulty or even act maliciously.

A malicious supervisor would pose a significant problem for an overlay network since it could easily launch Sybil attacks (i.e., flooding the overlay network with fake or adversarial nodes) or Eclipse attacks (i.e., isolating nodes from other nodes in the overlay network). We thus ask: Can we limit the power of a supervisor such that it cannot launch an eclipse or sybil attack while still being able to convert the overlay network from any connected topology to any other connected topology?

We answer the question to the affirmative by determining a set of graph transformation commands, also called primitives, that only the supervisor may issue to the nodes. These primitives are powerful enough to transform any (weakly) connected topology into any other (weakly) connected topology but still allow the nodes to locally check that applying them does not disconnect the network or introduce a new node into the network. We additionally aim at minimizing the reconfiguration overhead, i.e., the number of commands to be issued (and, related to this, the number of changes to be made to node neighborhoods) to reach a desired topology. Unfortunately, as we will show, this cannot be done efficiently for the set of primitives we consider unless $P \neq NP$, and we conjecture that this holds for any set of commands that has the aforementioned property of giving the participants the ability to locally check that they cannot be used for eclipse or sybil attacks. However, we are able to give an $O(1)$-approximation algorithm for this problem.

### 1.1 Model and Problem Statement

We model the overlay network as a graph, i.e., nodes represent participants of the network and if there is a directed edge $(u, v)$ in the graph, this means that there is a connection from $u$ to $v$. Undirected edges $\{u, v\}$ model the two connections from $u$ to $v$ and from $v$ to $u$. Since there may be multiple connections between the same pair of participants, the graphs we consider in this work are multigraphs, i.e., edges may appear several times in the (multi-)set of edges. For convenience throughout this work we will use the term “graph” instead of multigraph and refer to “edge sets” even though their elements need not be unique.

We consider the following set $P_d$ of four primitives for the manipulation of directed graphs, first introduced by Koutsopoulos et al. [25] in the context of overlay networks:

- **Introduction** If a node $u$ has a reference of two nodes $v$ and $w$ with $v \neq w$, $u$ introduces $w$ to $v$ if $u$ sends a message to $v$ containing a reference of $w$ while keeping the reference.

- **Delegation** If a node $u$ has a reference of two nodes $v$ and $w$ s.t. $u, v, w$ are all different, then $u$ delegates $w$’s reference of $v$ if $u$ sends a message to $v$ containing a reference of $w$
and deletes the reference of \( w \).

**Fusion** If a node \( u \) has two references \( v \) and \( w \) with \( v = w \), then \( u \) fuses the two references if it only keeps one of these references.

**Reversal** If a node \( u \) has a reference of some other node \( v \), then \( u \) reverses the connection if it sends a reference of itself to \( v \) and deletes its reference of \( v \).

The four primitives are visualized in Figure 1. Note that for the Introduction primitive, it is possible that \( w = u \), i.e., \( u \) introduces itself to \( v \). To simplify the description, we sometimes say that a node \( u \) introduces or delegates the edge \((u, v)\) if \( u \) introduces \( v \) to some other node or delegates \( v \)'s reference to some other node, respectively.

![Figure 1](image)

The primitives in \( P_d \) are known to be universal (c.f. [25]), i.e., it is possible to transform any weakly connected graph into any other weakly connected graph by using only the primitives in \( P_d \). Note that for every edge \((u, v)\) used in any of the primitives, either \((u, v)\) still exists after the corresponding primitive is applied, or there is still an (undirected) path from \( u \) to \( v \) in the resulting graph. This directly implies that no application of the primitives can disconnect the graph. We assume that all connections are *authorized*, meaning that both endpoints are aware of the other endpoint of this connection. Thus, if for an edge \((u, v)\) that is supposed to be transformed into \((v, u)\) by an application of the Reversal Primitive, \( v \) checks that \( u \) actually was the previous endpoint of the former edge then the primitives cannot be used to introduce new nodes into the graph.

For undirected graphs, consider the set \( P_u \) containing only the primitives Introduction, Delegation and Fusion (defined correspondingly). These three primitives, accordingly, are universal on undirected graphs, i.e., any connected undirected graph can be transformed into any other connected undirected graph by applying the primitives in \( P_u \) (c.f. [25]).

We make the following observation:

\[
\text{Observation 1.} \quad \text{The Introduction primitive is the only primitive that can increase the number of edges in a graph. The Fusion primitive is the only primitive that can decrease the number of edges in a graph. The Delegation primitive is the only primitive that can remove the last edge between two nodes (i.e., an edge of multiplicity one).}
\]

A computation \( C \) is a finite sequence \( G_1 \Rightarrow G_2 \Rightarrow \cdots \Rightarrow G_l \) of either directed or undirected graphs, in which each graph \( G_{i+1} \) is obtained from \( G_i \) by the application of a single primitive from \( P_d \) or \( P_u \), respectively. The graphs \( G_1 \) and \( G_l \) are called the initial and the final graphs of \( C \), respectively. The variable \( l \) is called the length of the computation.

We define the Undirected Local Graph Transformation Problem (ULGT) as follows: given two connected undirected graphs \( G_s, G_t \), find a computation of minimum length whose initial
On the Complexity of Local Graph Transformations

Graph is $G_s$ and whose final graph is $G_t$. The corresponding decision problem $k$-ULGT is defined as follows: given a positive integer $k$ and two connected undirected graphs $G_s$ and $G_t$, decide whether there is a computation with initial graph $G_s$ and final graph $G_t$ of length at most $k$. Accordingly we define the Directed Local Graph Transformation Problem (DLGT) and $k$-DLGT, which differ from the according problems in that the graphs are directed.

1.2 Related Work

Graph transformations have been studied in many different contexts and applications, including but not limited to pattern recognition, compiler construction, computer-aided software engineering, description of biological developments in organisms, and functional programming languages implementation (for a more detailed introduction and literature overview, we refer the reader to [4], [20], or [31, 13]). Simply put, a graph transformation (or graph-rewriting) system consists of a set of rules $L \rightarrow R$ that may be applied to subgraphs isomorphic to $L$ of a given graph $G$ thus replacing $L$ with $R$ in $G$. Since changing the labels assigned to a graph (graph relabeling) is also a kind of graph transformation, basically every distributed algorithm can be understood as a graph transformation system (c.f. [13]). The type of graph transformations probably closest related to our work is the area of Topology Control (TC). In simple terms, the goal of TC is to select a subgraph of a given input graph that fulfills certain properties (such as connectivity) and optimizes some value (such as the maximum degree). This problem has been studied in a variety of settings (for surveys on this topic see, e.g., [27], or [6]) and although the usual approach is decentralized, there are also some centralized algorithms in this area (see, e.g., [50]). However, these works only consider the complexity of computing an optimal topology (instead of the complexity of transforming the graph by a minimum number of rule applications). There is one work by Lin [28] proving the NP-hardness of the Graph Transformation Problem, in which the goal is to find the minimum integer $k$ such that an initial graph $G_s$ can be transformed into a final graph $G_t$ by adding and removing at most $k$ edges in $G_s$. Our work differs from that work in that we do not allow arbitrary edge relocations but restrict them to a set of rules that can be applied locally (and we also provide constant-factor approximation algorithms).

Our approximation algorithms use an approximation algorithm for the Undirected Steiner Forest Problem as a black-box (also known as the Steiner Subgraph Problem with edge sharing, or, in generalizations, the Survivable Network Design Problem or the Generalized Steiner Problem). 2-approximations of this problem were first given by Agrawal, Klein, and Ravi [1], and by Goemans and Williamson [16], and later also by Jain [23]. Gupta and Kumar [18] showed a simple greedy algorithm to have a constant approximation ratio and recently, Groß et al. [17] presented a local-search constant approximation for Steiner Forest.

1.3 Our Contribution

The main contributions of this paper are as follows: We prove the Undirected and the Directed Local Graph Transformation Problem to be NP-hard in Section 2. Furthermore, in Section 3 we show that they belong to APX, i.e., there exist constant approximation algorithms for these two problems.

2 NP-hardness results

In this section, we show the NP-hardness of the Undirected Local Graph Transformation Problem by proving the NP-hardness of $k$-ULGT (see Section 2.1). Since $k$-DLGT’s NP-
hardness is very similar for k-ULGT, we only briefly sketch the differences in Appendix B. Throughout this section, for any positive integer i we use the notation [i] to refer to the set \{1, 2, \ldots, i\}.

2.1 k-ULGT is NP-hard

We prove k-ULGT’s hardness via a reduction from the Boolean satisfiability problem (SAT) which was proven to be NP-hard by Cook [9] and, independently, by Levin [26]. We briefly recap SAT as follows:

Definition 1 (SAT). Given a set X of n Boolean variables x₁, . . . , xₙ and a Boolean formula Φ over the variables in X in conjunctive normal form (CNF), decide whether there is a truth assignment t : X → \{0, 1\} that satisfies Φ.

To reduce SAT to k-ULGT, we use the following reduction function:

Definition 2 (Reduction function). Let S = (X, Φ) be a SAT instance, in which X = \{x₁, . . . , xₙ\} is the set of Boolean variables and Φ = C₁ ∧ · · · ∧ Cₘ for clauses C₁, . . . , Cₘ. Then f(S) = (Gₛ, Gᵣ, k) in which k = 2n + m and Gₛ and Gᵣ are undirected graphs defined as follows. Without loss of generality, assume that each literal yᵢ ∈ \{xᵢ, ¬xᵢ\} occurs only once in each clause. We say yᵢ ∈ Cⱼ if literal yᵢ occurs in Cⱼ.

We define the following sets of nodes: Vₙ = \{C₁, . . . , Cₘ\}, and Vₓᵢ = \{xᵢ, ¬xᵢ, sᵢ, tᵢ\}. Then, the set of nodes of Gₛ and Gᵣ is V = \bigcup_{1 \leq i \leq n} Vₓᵢ ∪ Vₙ ∪ \{r\}. For the set of edges, define Eₓᵢ = \{(sᵢ, xᵢ), (¬xᵢ, sᵢ), (xᵢ, tᵢ), (tᵢ, ¬xᵢ)\}, Eₖ = \{(yᵢ, Cⱼ) | yᵢ ∈ \{xᵢ, ¬xᵢ\} ∧ yᵢ occurs in Cⱼ\}, Eₙr = \{(tᵢ, r) | 1 ≤ i ≤ n\}, Eᵣ = \{(Cⱼ, r) | 1 ≤ j ≤ m\}. Both Gₛ and Gᵣ have the edges in \bigcup_{1 \leq i \leq n} Eₓᵢ ∪ \bigcup_{1 \leq j \leq m} Eₖ. Additionally, Gₛ has the edges in Eₙr and Gᵣ has the edges in Eᵣ ∪ Eᵣᵣ.

Intuitively, each variable xᵢ is mapped to a gadget Xᵢ consisting of the four nodes xᵢ, ¬xᵢ, sᵢ, and tᵢ. Also each clause Cⱼ is connected with each literal occurring within it. Lastly, in Gₛ, each of the sᵢ is connected with the node r, whereas in Gᵣ, each of the tᵢ and each of the Cⱼ are connected with r. Figure 2 shows an example of the output of the reduction function for a given formula in CNF.

We now show that every SAT instance S is satisfiable if and only if f(S) is a “yes” instance of k-ULGT. We start with the “only if” part for this is the simpler direction:

Lemma 3. If a SAT instance S as in Definition 2 is satisfiable then f(S) = (Gₛ, Gᵣ, k) with k = 2n + m is a k-ULGT instance and there is a computation with initial graph Gₛ and final graph equal to Gᵣ of length at most 2n + m.

Proof. Assume there is a satisfying truth assignment t : X → \{0, 1\} of S. For every 1 ≤ i ≤ n let yᵢ := xᵢ if t(xᵢ) = 1 or yᵢ := ¬xᵢ if t(xᵢ) = 0. We construct the following computation with initial graph Gₛ and final graph Gᵣ:

1. For every 1 ≤ i ≤ n, sᵢ delegates the edge \{sᵢ, r\} to yᵢ.
2. For every Cⱼ ∈ \{C₁, . . . , Cₘ\} choose one neighbor zⱼ ∈ \{y₁, . . . , yₙ\} (we show below that this exists), and let zⱼ introduce r to Cⱼ.
3. For every 1 ≤ i ≤ n, yᵢ delegates the edge \{yᵢ, r\} to tᵢ.

Obviously, the length of this computation is 2n + m. To prove the missing part, recall that every Cⱼ is satisfied under t, i.e., there is at least one literal zⱼ in Cⱼ that evaluates to true, i.e., there is an i ∈ [n] such that zⱼ = xᵢ if t(xᵢ) = 1, or zⱼ = ¬xᵢ if t(xᵢ) = 0. By definition of yᵢ, zⱼ = yᵢ. Thus because zⱼ occurs in Cⱼ, yᵢ was a neighbor of Cⱼ during Step 2. ▸
On the Complexity of Local Graph Transformations

Figure 2 Graph $G_s$ returned by the reduction function for the (example) Boolean formula $(x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2) \land (x_2 \lor \neg x_3)$. $G_t$ differs from $G_s$ in that the dashed edges do not exist and all grey nodes share an edge with node $r$.

The “if” part is more complex. We begin with the following insight that will prove helpful in the course of this part.

Lemma 4. Suppose the nodes in the initial graph of a computation $C$ can be decomposed into disjoint sets $V_1, \ldots, V_k, P$ such that there is no edge $\{u, v\}$ for some $u \in V_i, v \in V_j$, $i, j \in [k], i \neq j$ and throughout $C$ none of the nodes in $P$ applies a primitive. Then there is no edge $\{u, v\}$ for some $u \in V_i, v \in V_j, i, j \in [k], i \neq j$ in any graph of the computation.

Proof. Assume there is a computation $C$ and sets $V_1, \ldots, V_k, P$ as defined above and assume for contradiction that the claim is not true. We consider the first edge $\{u, v\}$ such that $u \in V_i, v \in V_j, i, j \in [k], i \neq j$. Clearly, it cannot have been created by the application of a Fusion primitive. Thus it must have been created by an Introduction or Delegation primitive applied by a node $w$ that knew both $u$ and $v$ before the application of this primitive. By definition of $P$, $w \notin P$, i.e., $w \in V_l$ for some $l \in [k]$. However, by the definition of $\{u, v\}$, $u$ and $v$ must have been from $V_l$ as well, yielding a contradiction.

The next lemma we show represents a main building block of the proof of the “if” part.

Lemma 5. Let $S$ be a SAT instance and let $(G_s, G_t, k) = f(S)$. For every computation $C$ with initial graph $G_s$ and final graph equal to $G_t$ of length at most $2n + m$ it holds: There are $y_1, \ldots, y_n, y_i \in \{x_i, \overline{x_i}\}$ for every $i \in [n]$, such that in $C$ there are no edges other than $E(G_s) \cup E(G_t) \cup \{\{y_i, r\}|i \in [n]\}$ and no edge occurs twice (where $E(G_s)$ and $E(G_t)$ denote the edge set of $G_s$ and $G_t$, respectively).

To keep the main part of this full version similar to the conference version, we only sketch the proof here, whereas the full proof can be found in Appendix A. The general idea of the proof of Lemma 5 is the following: To obtain the target graph, for each $j \in [m]$ the edge $\{C_j, r\}$ has to be created and for each $i \in [n]$ the edge $\{t_i, r\}$ has to be created. Each of these creations involves a distinct application of a primitive. Therefore, only $n$ applications of primitives are left in a feasible computation. We show that the nodes in each gadget $i$ have to apply at least one primitive $p_i$ that does not create one of the above edges. This
Lemma 6. Let $S$ be a SAT instance as in Definition 3. If $f(S) = (G_s, G_t, k)$ with $k = 2n + m$ is a $k$-ULGT instance and there is a computation with initial graph $G_s$ and final graph equal to $G_t$ of length at most $2n + m$ then $S$ is satisfiable.

Proof. In the following, we refer to the variables defined in Definition 3. Furthermore, we say a computation is feasible if and only if its initial graph is $G_s$, its target graph is $G_t$ and its length is at most $2n + m$. Moreover, we say that the edge that is established during the application of an Introduction or Delegation primitive (the edge $(v, w)$ in Figures 1a and 1b) is the result of the Introduction or Delegation, respectively.

Assume that $f(S) = (G_s, G_t, 2n + m)$ is a $k$-ULGT instance and there is a feasible computation $C$ for $f(S)$. According to Lemma 5 there are $y_1, \ldots, y_n, y_1 \in \{x, \overline{x}\}$ for every $i \in [n]$ such that in $C$ there are no edges other than $E(G_s) \cup E(G_t) \cup \{(y_i, y_i)[n]\}$. Note that in $G_t$, for every $j \in [m]$ there is an edge $\{C_j, r\}$ and each such edge must have been the result of an Introduction or Delegation primitive applied by an $y_i, i \in [n]$ (as throughout $C$, the $C_j$s do not have any other neighbors with an edge to $r$ that could possible create this edge). Let $g : \{C_1, C_2, \ldots, C_m\} \rightarrow \{y_1, y_2, \ldots, y_n\}$ be the mapping of each $C_j$ to the $y_i$ who applied a primitive that resulted in the edge $\{C_j, r\}$. Consider the truth assignment $t : X \rightarrow \{0, 1\}$ such that $t(x_i) = 1$ if $y_i = x_i$ and $t(x_i) = 0$ if $y_i = \overline{x}$. Observe that $t(y_i) = 1$ for every $i \in [n]$. Assume for contradiction that there is a clause $C_j$ in $S$ that does not evaluate to 1 under $t$. Note that $g(C_j)$ must occur in $C_j$ by construction. However, since $g(C_j) = y_i$ for some $i \in [n]$ and $t(y_i) = 1$, we obtain the desired contradiction. ▶

3 Approximation Algorithms

In this section, we first describe an approximation algorithm for ULGT (see Section 3.1) and prove it to have a constant approximation ratio (see Section 3.2). As an ingredient our algorithm uses a 2-approximation algorithm (see Section 1.2) for the Undirected Steiner Forest Problem (USF) defined as follows: Given a graph $G$ and a set $S$ of pairs of nodes from $G$, find a forest $F$ in $G$ with a minimum number of edges such that the two nodes of each pair in $S$ are connected by a path in $F$. Note that a constant approximation factor algorithm for DLGT can be obtained by a slight adaptation of our Algorithm from Section 3.1. We describe how to do this in Appendix C.

3.1 Algorithm Description

For an initial graph $G_s = (V, E_s$) and a final graph $G_t = (V, E_t)$, we define the set of additional edges $E_\oplus := E_t \setminus E_s$ and the set of excess edges $E_\ominus := E_s \setminus E_t$. We now describe the algorithm in detail and then summarize its pseudo-code in Algorithm 1. Our algorithm consists of two parts, the first of which dealing with establishing all additional edges and the second of which dealing with removing all excess edges. In the first part, using an arbitrary 2-approximation algorithm for the USF as a black box the algorithm computes a 2-approximate solution to the following USF instance: The given graph is $G_s$, and the set of pairs of nodes is $E_\oplus$. Note that the result is a forest such that for every edge $(u, v) \in E_\oplus$,
Algorithm 1 Approximation algorithm for ULGT

- **Input:** Initial graph $G_s$ and final graph $G_t$.

- **First part:**
  1. Compute a 2-approximate solution $F_{\text{ALG,}⊕}$ for the USF with input $G_s$, and the set $E_{⊕}$ as the set of pairs of nodes.
  2. For each tree $T$ in $F_{\text{ALG,}⊕}$, select a root node $r_T$ and connect all nodes in $T$ that are incident to an edge in $E_{⊕}$ with $r_T$ (details below).
  3. For each $\{u, v\} \in E_{⊕}$, let $s(u)$ be an arbitrary of the two endpoints of $u$. For each tree $T$ in $F_{\text{ALG,}⊕}$, select a root node $r_T$ and connect $s(u)$ with $r_T$ (similar to Step 2, details below).
  4. For each tree $T$ in $F_{\text{ALG,}⊕}$, delegate all superfluous edges (i.e., not belonging to $G_s$ or $E_{⊕}$) created during Step 2 bottom up in $T$ rooted at $r_T$, starting with the lowest level. At each intermediate node fuse all of these edges before delegating them to the next predecessor.

- **Second part:**
  5. Compute a 2-approximate solution $F_{\text{ALG,}⊖}$ for the USF with input $G_t$, and the set $E_{⊖}$ as the set of pairs of nodes.
  6. For each $e \in E_{⊖}$, let $s(e)$ be an arbitrary of the two endpoints of $e$. For each tree $T$ in $F_{\text{ALG,}⊖}$, select a root node $r_T$ and for each $e \in E_{⊖}$ whose endpoints belong to $T$, connect $s(e)$ with $r_T$ (similar to Step 2, details below).
  7. For each $e \in E_{⊖}$, $s(e)$ delegates the other endpoint to $r_T$.
  8. For each tree $T$ in $F_{\text{ALG,}⊖}$, delegate all superfluous edges bottom-up and fuse multiple edges as in Step 4.

In this section we show that Algorithm 1 is a constant-approximation algorithm for ULGT, which proves the following theorem:

**Theorem 7.** ULGT $\in \text{APX}$. 

For convenience we will analyze the two parts of the algorithm individually. Therefore, for a given initial graph $G_s$ and final graph $G_t$, let $\text{ALG}_1(G_s, G_t)$ be the length of the graph transformation $G_s \rightarrow G_t$.

3.2 Analysis

In this section we show that Algorithm 1 is a constant-approximation algorithm for ULGT, which proves the following theorem:

**Theorem 7.** ULGT $\in \text{APX}$. 

For convenience we will analyze the two parts of the algorithm individually. Therefore, for a given initial graph $G_s$ and final graph $G_t$, let $\text{ALG}_1(G_s, G_t)$ be the length of the graph transformation $G_s \rightarrow G_t$. 

$u$ and $v$ belong to the same tree. For each tree $T$ in this forest the algorithm then selects an arbitrary root $r_T$ and connects all nodes in $T$ that are incident to an edge in $E_{⊕}$ to $r_T$. The exact details of this will be described when we analyze the length of the resulting computation. In the next step, for every $T$, for every $\{u, v\} \in E_{⊕}$ such that $u$ and $v$ belong to $T$, $r_T$ introduces $u$ to $v$ to each other, thereby creating the edge $\{u, v\}$. After that, the superfluous edges are deleted in a bottom-up fashion: every node that does not have a descendant with a superfluous edge (in the tree $T$ this node belongs to when viewing this tree as rooted by $r_T$), fuses all superfluous edges and delegates the last such to its predecessor in the tree. Note that all superfluous edges in the same tree $T$ have $r_T$ as one of their endpoints. The second part of the algorithm is similar to the first, with the following differences: In the fifth step, the USF is approximated for the graph $G_t$ and $E_{⊖}$ as the set of pairs. Note that the solution is a subgraph of the graph obtained after the first part of the algorithm. In the sixth step, only one of the two endpoints of an edge from $E_{⊖}$ is selected to become connected with the root of the tree the endpoints belong to. In the seventh step (where in the first part the additional edges are created by the $r_T$ nodes), for each edge $e \in E_{⊖}$, the endpoint selected in the sixth step delegates this edge to $r_T$ (resulting in the edge $\{r_T, v\}$).
computation of the first part of the algorithm for this instance, $ALG_2(G_s, G_t)$ be the length of the computation of the second part, and $ALG(G_s, G_t) := ALG_1(G_s, G_t) + ALG_2(G_s, G_t)$. Furthermore, let $OPT(G_s, G_T)$ be the length of an optimal solution to ULGT for initial graph $G_s$ and final graph $G_t$. We also define the intermediate graph $G' = (V, E_s \cup E_o)$. In the course of the analysis we will establish a relationship between $ALG_1(G_s, G_t)$ and $OPT(G_s, G')$ and between $ALG_2(G_s, G_t)$ and $OPT(G', G_t)$. This will aid us in determining the approximation factor of Algorithm due to the following lemma:

Lemma 8. $OPT(G_s, G') + OPT(G', G_t) \leq 2OPT(G_s, G_t) + |E_o|$. 

Proof. Let $P$ denote the problem equal to k-ULGT with initial graph $G_s$ and final graph $G_t$, with the additional requirement that the computation must contain $G'$ and let $OPT'(G_s, G_t)$ be the length of an optimal solution to it. Clearly, $OPT(G_s, G') + OPT(G', G_t) \leq OPT'(G_s, G_t)$ (otherwise, split the computation at $G'$ and improve either $OPT(G_s, G')$ by the first part obtained or $OPT(G', G_t)$ by the second part obtained). We now show that $OPT'(G_s, G_t) \leq 2OPT(G_s, G_t) + |E_o|$. 

Consider a computation $C$ whose initial graph is $G_s$, whose final graph is $G_t$, and whose length is $OPT(G_s, G_t)$ (note that such a computation is an optimal solution to ULGT). We now transform $C$ into a computation that represents a solution to $P$. This transformation increases its length by only $OPT(G_s, G_t) + |E_o|$ and thus proves the above claim (recall that any solution to $P$ has at least the size of an optimal solution to it). First, because the final graph does not contain any edge $(u, v) \in E_o$, for every such edge there is one last Delegation in $C$ that removes this edge (recall Observation). We replace each of these last delegations by an introduction and obtain a new computation $C'$ of equal length. Note that changing these delegations to introductions does not make the computation infinite as this only causes the graph to have additional edges. The final graph of $C'$ is $(V, E_s \cup E_o) = G'$ (recall that $E_t = (E_s \cup E_o) \setminus E_o$). Next we append $C'$ by $C$ and obtain the computation $C''$ of length $2OPT(G_s, G_t)$. Note that since $C'$ transformed $G_s$, $G_t$, this second half of $C''$, which starts from $G' = (V, E_s \cup E_o)$, has the final graph $G'' = (V, E_t \cup E_o)$, i.e., each edge from $E_o$ appears twice in $G''$. Thus we extend $C''$ by fusing each edge from $E_o$ with its double, resulting in a computation $C'''$ of length $2OPT(G_s, G_t) + |E_o|$. Since $C'''$ represents a solution to $P$ for initial graph $G_s$ and final graph $G_t$, this completes the proof.

In the rest of the analysis we show that $ALG_1(G_s, G_t) \leq 11OPT(G_s, G')$ (Lemma 9) and that $ALG_2(G_s, G_t) \leq 7OPT(G', G_t)$ (Lemma 10). By Lemma 8, this implies that $ALG(G_s, G_t) \leq 11(2OPT(G_s, G_t) + |E_o|) \leq 33OPT(G_s, G_t)$ (since, clearly, $OPT(G_s, G_t) \geq |E_o|$), which yields the claim of Theorem. We start with the former claim:

Lemma 9. $ALG_1(G_s, G_t) \leq 11OPT(G_s, G')$.

Proof. Let $F_{OPT,o}$ be an optimal solution for the USF with input $G_s$ and $E_o$ as the set of nodes and recall that $F_{ALG,o}$ is the USF approximation computed in Step 1 of Algorithm. Throughout the analysis, $|F_{OPT,o}|$ and $|F_{ALG,o}|$ will denote the number of edges in these solutions. In the first part of this proof, we show that $ALG_1(G_s, G_t) \leq 4|F_{OPT,o}| + 3|E_o|$. The second part then consists in proving $OPT(G_s, G') \geq |F_{OPT,o}| - |E_o|$, which together with the observation that $OPT(G_s, G') \geq |E_o|$ yields the claim.

To upper bound $ALG_1(G_s, G_t)$, we analyze the number of primitives applied in each of the steps of the first part of the approximation algorithm. In Step 1, no primitive is applied. To keep the number of edges as low as possible (which saves Fusion primitives in Step 4), the algorithm for every $T$ in $F_{ALG,o}$ connects the desired nodes to $r_T$ in Step 2 in the following
On the Complexity of Local Graph Transformations

(a) Step 2 connects all endpoints of edges in $E_\oplus$ belonging to $T$ with $r_T$.

(b) In Step 3, $r_T$ creates the edges in $E_\oplus$ that belong to $T$ by an Introduction.

(c) Step 4 removes all superfluous edges by delegating and fusing them up in the tree.

Figure 3 Example of a tree $T$ with root $r_T$ for Step 2-4 of Algorithm 1 assuming $\{u, v\} \in E_\oplus$. $ST(x)$ consists of $x$, $w$, and $u$. $x$ is relevant, whereas $y$ is not. Dashed edges exist temporarily during the displayed step.

way: To simplify the description, we view $T$ as rooted at $r_T$ and for a node $u \in T$ denote by $ST(u)$ the set consisting of $u$ and all of its descendants in the tree $T$ rooted at $r_T$. We say a node $u$ is relevant if $ST(u)$ contains a node with an endpoint in $E_\oplus$. See Figure 3 for an illustration of these notions. First of all, $r_T$ introduces itself to all relevant children. Then, starting from the second level, we proceed level-wise in the tree: For each level $i$, every node $u$ at level $i$ checks whether $u$ is an endpoint of an edge in $E_\oplus$ or $\{u, r_T\}$. If so, it introduces $r_T$ to all relevant children. Otherwise, it introduces $r_T$ to all but one of its relevant children (chosen arbitrarily) and delegates $r_T$ to the relevant child it did not introduce $r_T$ to. One can check that the result of this procedure is that each node incident to an edge in $E_\oplus$ has an edge to $r_T$ for the tree $T$ it belongs to, see Figure 3. Note that according to the definition of $F_{ALG, \oplus}$, for each pair $\{u, v\} \in E_\oplus$ $u$ and $v$ belong to the same tree $T$. The above procedure increases the number of edges by at most $2|E_\oplus|$ and requires at most $|F_{ALG, \oplus}|$ applications of primitives (since each tree $T$ with $k$ edges contains at most $k + 1$ nodes and for each node $u$ in $T$, at most one primitive is applied to create $\{u, r_T\}$ and this is done for neither $r_T$ nor the nodes at level 1). It is easy to see that Step 3 (c.f. Figure 3a) involves exactly $E_\oplus$ applications of primitives. For the length of Step 4 (c.f. Figure 3b), note that for every tree $T$ at most $|T|$ delegations have to be applied because every node in each tree has to apply at most one Delegation (causing $|F_{ALG, \oplus}|$ delegations in total) and at most $2|E_\oplus|$ fusions have to be applied for this is the number of superfluous edges created during Step 2. All in all, Step 2, Step 3, and Step 4 involve $|F_{ALG, \oplus}|$, $E_\oplus$, and $|F_{ALG, \oplus}| + 2|E_\oplus|$ applications of primitives, respectively. This makes a total of $2|F_{ALG, \oplus}| + 3|E_\oplus|$. Since $F_{ALG, \oplus}$ is a 2-approximation of $OPT(G_s, G')$, we obtain $ALG_1(G_s, G_i) \leq 4|OPT(G_s)| + 3|E_\oplus|$.

For the lower bound on $OPT(G_s, G')$, assume for contradiction that there is a computation $C$ with initial graph $G_s$ and final graph $G'$ of length $L < |FOPT(G_s)| = |E_\oplus|$. Let $G_s = G_1 \Rightarrow G_2 \Rightarrow \cdots \Rightarrow G_L$ be the sequence of graphs of this computation. For every $\{u, v\} \in E_\oplus$ we iteratively create a path from $u$ to $v$ in the following way: Begin with $P_{u,v}^L := (u, v)$. Note that $P_{u,v}^L$ exists in $G_L$. We iterate through $C$ in reverse order and for every graph $G_i$, if $P_{u,v}^{i+1}$ exists in $G_i$, $P_{u,v}^i := P_{u,v}^{i+1}$. Otherwise, since $G_{i+1}$ is the result of a single application of a primitive to $G_i$, there is exactly one edge $\{x, y\}$ in $P_{u,v}^{i+1}$ that exists in $G_{i+1}$ but not in $G_i$ and this edge was created by the application of an Introduction or Delegation primitive of some node $w$ such that $\{w, x\}$ and $\{w, y\}$ exist in $G_i$. Thus, let $P_{u,w}^i$ be $P_{u,v}^{i+1}$ with $(x, y)$ replaced by $(x, w, y)$ and note that $P_{u,w}^i$ exists in $G_i$. Eventually, we obtain a path $P_{u,w}^i$ that exists in $G_s$. For $i \in \{1, \ldots, L\}$, let $F^i := \cup_{\{x, y\} \in E_\oplus} E(P_{u,x}^i)$ (where $E(P)$ is the set of all edges on the path $P$) and note that $F^1$ represents a solution to
Lemma 10. \(\text{we obtain} (e)\) Initial graph \(G_1\). (b) \(G_2\): \(x\) has introduced \(u\) to \(y\). (c) \(G_3\): \(y\) has delegated \(u\) to \(v\). (d) \(G_4\): \(y\) has introduced \(v\) to \(w\).

Figure 4 Example of an optimal computation \(C\) with initial graph \(G_1\) and \(E_{\oplus} = \{u, v\}, \{v, w\}\), and the notions used in the proof of Lemma 9. The upper row shows \(C\) in order, the lower row illustrates the path sets \(P^{u,v}_{u,v}\) and \(P^{w,v}_{u,v}\), which are defined by iterating through \(C\) in reverse order. In the lower row, the edges drawn black in \(G_1\) are the edges belonging to to \(F^t\). Observe that \(F_t\) represents a solution to the USF for graph \(G_1\) and node pairs \(E_{\oplus}\).

the USF with input \(G_s\) and \(E_{\oplus}\) as the set of node pairs. An example is given in Figure 4.

For an arbitrary \(i \in \{1, \ldots, L - 1\}\), note that \(|F^i| \leq |F^{i+1}| + 1:\) if \(G_{i+1}\) was obtained from \(G_i\) by the application of a Fusion primitive, this inequality trivially holds as none of the above paths changes in this case. Otherwise, \(G_{i+1}\) was obtained from \(G_i\) by an application of an Introduction or Delegation primitive by some node \(w\) causing at most one edge \(\{x, y\}\) to exist in \(G_{i+1}\) that does not exist in \(G_i\). In this case, we further know that \(\{w, x\}\) and \(\{w, y\}\) exist in \(G_i\) and by the definition of the above paths, for every pair \(\{u, v\}\) such that \(P^{i+1}_{u,v}\) contains the edge \(\{x, y\}\) the path \(P^i_{u,v}\) contains \((x, w, y)\) as a sub-path instead and for all other pairs \(\{u', v'\}\), \(P^{i+1}_{u',v'} = P^i_{u',v'}\). By the definition of \(F^i\) and \(F^{i+1}\), this implies \(|F^i| \leq |F^{i+1}| + 1\) also in this case. All in all we obtain that \(|F^i| \leq |F^L| + L = |E_{\oplus}| + L\) because \(F^L = E_{\oplus}\) (note the definition of \(F^L\)). By the assumption that \(L < |F_{OPT,\oplus}| - |E_{\oplus}|\), we obtain \(|F^1| < |F_{OPT,\oplus}|\), which represents a contradiction.

Lemma 10. \(\text{ALG}_2(G_s, G_t) \leq 7\text{OPT}(G', G_t)\).

Proof. The general structure of this proof follows the line of the proof of Lemma 9, but differs in the details. Similar to the notation used in that proof, let \(F_{OPT,\oplus}\) be an optimal solution for the USF with input \(G_1\) and \(E_{\oplus}\) as the set of nodes and recall that \(F_{ALG,\oplus}\) is the USF approximation computed in Step 5 of Algorithm 1. Analogously, \(|F_{OPT,\oplus}|\) and \(|F_{ALG,\oplus}|\) denote the number of edges in these solutions. In the first part of this proof, we show that \(\text{ALG}_2(G_s, G_t) \leq 4|F_{OPT,\oplus}| + 3|E_{\oplus}|\). The second part then consists in proving \(\text{OPT}(G', G_t) \geq |F_{OPT,\oplus}|\), which together with the observation that \(\text{OPT}(G', G_t) \geq |E_{\oplus}|\) yields the claim.

To upper bound \(\text{ALG}_2(G_s, G_t)\), we analyze the number of primitives applied in each step of the second part of the approximation algorithm. Of course, no primitive is applied in Step 5. The connections required in Step 6 can be created in a similar fashion as in Step 2 (see the proof of Lemma 9). For each tree \(T\), we proceed top-down in the \(T\) rooted at \(T_T\) again. Here, each intermediate node \(u\) checks whether \(u = s(e)\) for some \(e \in E_{\oplus}\). If so, it introduces \(r_T\) to all relevant children (here a node \(v\) is relevant if \(ST(v)\) contains a node \(w\) such that \(w = s(e')\) for some \(e' \in E_{\oplus}\)). Otherwise, it introduces \(r_T\) to all but one relevant children and delegates it to the remaining one. In the end, for every
edge \( e \in E_{\mathcal{G}} \), \( s(e) \) has an edge to \( r_T \), the number of edges in the graph has increased by at most \( |E_{\mathcal{G}}| \), and the process involved at most \( |F_{\text{ALG},\mathcal{G}}| \) applications of primitives. In Step 7, clearly exactly \( |E_{\mathcal{G}}| \) edges have to be delegated. Step 8 is similar to Step 4 and for analogous reasons requires at most \( |F_{\text{ALG},\mathcal{G}}| \) delegations and at most \( 2|E_{\mathcal{G}}| \) fusions (recall that up to \( |E_{\mathcal{G}}| \) edges were added in Step 6 and the edges delegated in Step 7 have to be removed as well). All in all, Step 6, Step 7 and Step 8 of the algorithm involve at most \( |F_{\text{ALG},\mathcal{G}}| \cdot |E_{\mathcal{G}}| \) and \( |F_{\text{ALG},\mathcal{G}}| + 2|E_{\mathcal{G}}| \) applications of primitives, respectively, which yields:

\[
\text{ALG}_2(G_s, G_t) \leq 2|F_{\text{ALG},\mathcal{G}}| + 3|E_{\mathcal{G}}| \leq 4|F_{\text{OPT},\mathcal{G}}| + 3|E_{\mathcal{G}}| \quad \text{(since } F_{\text{ALG},\mathcal{G}} \text{ is a 2-approximation of } F_{\text{OPT},\mathcal{G}}. \text{)}
\]

To lower bound the value of \( \text{OPT}(G', G_t) \), assume for contradiction that there is a computation \( C \) with initial graph \( G' \) and final graph \( G_s \) of length \( L < |F_{\text{OPT},\mathcal{G}}| - |E_{\mathcal{G}}| \). Let \( G_s = G_1 \Rightarrow G_2 \Rightarrow \cdots \Rightarrow G_L \) be the sequence of graphs of this computation. Similar to the proof of Lemma 12, for every \( \{u, v\} \in E_{\mathcal{G}} \), we create a path from \( u \) to \( v \), but this time we start with \( P_{u,v}^1 := (u, v) \) and consider the graphs in increasing order: For \( i \in \{2, \ldots, L\} \), if \( P_{u,v}^{i-1} \) exists in \( G_{i-1} \), \( P_{u,v}^i := P_{u,v}^{i-1} \). Otherwise since \( G_i \) is the result of a single application of a primitive to \( G_{i-1} \), there is exactly one edge \( \{x, y\} \) in \( P_{u,v}^{i-1} \) that exists in \( G_{i-1} \) but not in \( G_i \) and this edge must have been delegated by either \( x \) or \( y \) to some node \( w \). In the following denote the node that applied the Delegation by \( z \) and denote by \( \tau \) the other node from \( \{x, y\} \). In \( G_{i-1} \), \( z \) must share an edge with \( w \) and this edge still exists in \( G_i \) (for only one primitive is applied in the transition from \( G_{i-1} \) to \( G_i \)). Since \( \{z, \tau\} \) was delegated to \( w \), in \( G_i \) the edge \( \{w, \tau\} \) exists in \( G_i \). Thus, let \( P_{u,v}^i \) be \( P_{u,v}^{i-1} \) with \( \{x, y\} \) replaced by \( \{x, w, y\} \) and observe that \( P_{u,v}^i \) exists in \( G_i \). Eventually, we obtain a path \( P_{u,v}^L \) that exists in \( G_L \). Define \( F^i := \bigcup_{\{u,v\} \in E_{\mathcal{G}}} E(P_{u,v}^i) \) (where \( E(P) \) is the set of all edges on the path \( P \)) for all \( i \in \{1, \ldots, L\} \), and note that \( F^L \) represents a solution to the USF with input \( G_1 \) and \( E_{\mathcal{G}} \) as the set of nodes. Furthermore, for an arbitrary \( i \in \{1, \ldots, L-1\} \), note that \( |F^{i+1}| \leq |F^i| + 1 \) because there is at most one edge \( \{x, y\} \) that exists in \( G_i \) but not in \( G_{i+1} \) and thus causes the replacement of \( \{x, y\} \) by \( \{x, w, y\} \) for some fixed node \( w \) for all paths that contain \( \{x, y\} \) as a sub-path. This yields that \( |F^L| \leq |F^1| + L = |E_{\mathcal{G}}| + L \) because \( F^1 = E_{\mathcal{G}} \) (note the definition of \( F^1 \)). By the assumption that \( L < |F_{\text{OPT},\mathcal{G}}| - |E_{\mathcal{G}}| \), we obtain \( |F^L| < |F_{\text{OPT},\mathcal{G}}| \), which represents a contradiction.

4 Conclusion

We proposed a set of primitives for topology adaptation that a server may use to adapt the network topology into any desired (weakly) connected state but at the same time cannot use to disconnect the network or to introduce new nodes into the system. So far, we only assumed that the server could act maliciously but that the participants of the network are honest and correct, i.e., they refuse any graph transformation commands beyond the four primitives. What, however, if some participants also behave in a malicious manner? Is it still possible to avoid Eclipse or Sybil attacks? It seems that in this case the only measure that would help is to form quorums of nodes that are sufficiently large so that at least one node in each quorum is honest.

Besides these security-related aspects, our results give rise to additional questions: For example, does the NP-hardness apply to any set of local primitives, or is there a set of local primitives that can transform arbitrary initial graphs much faster into arbitrary final graphs than the set considered in this work? Furthermore, is it possible to obtain decentralized versions of the algorithms presented in Section 3 and, if so, what is their competitiveness when compared to the centralized ones?
References

1. Ajit Agrawal, Philip N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995. doi:10.1137/S0097539792236237

2. Susanne Albers, Stefan Eilts, Eyal Even-Dar, Yishay Mansour, and Liam Roditty. On nash equilibria for a network creation game. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2006, Miami, Florida, USA, January 22-26, 2006, pages 89–98, 2006.

3. Noga Alon, Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Tom Leighton. Basic network creation games. *SIAM J. Discrete Math.*, 27(2):656–668, 2013. doi:10.1137/090771478

4. Marc Andries, Gregor Engels, Annegret Habel, Berthold Hoffmann, Hans-Jörg Kreowski, Sabine Kuske, Detlef Plump, Andy Schürr, and Gabriele Taentzer. Graph transformation for specification and programming. *Sci. Comput. Program.*, 34(1):1–54, 1999.

5. James Aspnes and Yinghua Wu. O(log n)-time overlay network construction from graphs with out-degree 1. In *Proceedings of the 11th International Conference on Principles of Distributed Systems, (OPODIS ’07)*, pages 286–300, 2007.

6. Azrina Abd Aziz, Y. Ahmet Sekercioglu, Paul G. Fitzpatrick, and Milosh V. Ivanovich. A survey on distributed topology control techniques for extending the lifetime of battery powered wireless sensor networks. *IEEE Communications Surveys and Tutorials*, 15(1):121–144, 2013. doi:10.1109/SURV.2012.031612.00124

7. Andrew Berns, Sukumar Ghosh, and Sriram V. Pemmaraju. Building self-stabilizing overlay networks with the transitive closure framework. *Theor. Comput. Sci.*, 512:2–14, 2013.

8. Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Locality-based network creation games. *TOPC*, 3(1):6:1–6:26, 2016. doi:10.1145/2938426.

9. Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC ’71, pages 151–158, New York, NY, USA, 1971. ACM. doi:10.1145/800157.805047

10. Andreas Cord-Landwehr, Martina Hüllmann, Peter Kling, and Alexander Setzer. Basic network creation games with communication interests. In *Algorithmic Game Theory - 5th International Symposium, SAGT 2012, Barcelona, Spain, October 22-23, 2012. Proceedings*, pages 72–83, 2012. doi:10.1007/978-3-642-33996-7_9

11. Erik D. Demaine, Mohammad Taghi Hajiaghayi, Hamid Mahini, and Mortezagadgadgadgadgadgadgadgadgadgadgadgadgad. The price of anarchy in network creation games. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, Portland, Oregon, USA, August 12-15, 2007*, pages 292–298, 2007. doi:10.1145/1281100.1281142

12. Shlomi Dolev and Ronen I. Kat. Hypertree for self-stabilizing peer-to-peer systems. *Distributed Computing*, 20(5):375–388, 2008.

13. Hartmut Ehrig, Hans-Jörg Kreowski, Ugo Montanari, and Grzegorz Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 3: Concurrency, Parallelism, and Distribution*. World Scientific, 1999.

14. Alex Fabrikant, Ankur Luthra, Eltiza N. Maneva, Christos H. Papadimitriou, and Scott Shenker. On a network creation game. In *Proceedings of the Twenty-Second ACM Symposium on Principles of Distributed Computing, PODC 2003, Boston, Massachusetts, USA, July 13-16, 2003*, pages 347–351, 2003. doi:10.1145/872035.872088

15. Michael Feldmann, Christina Kolb, Christian Scheideler, and Thim Strothmann. Self-stabilizing supervised publish-subscribe systems. In *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 21-25, 2018*, pages 1050–1059, 2018. doi:10.1109/IPDPS.2018.00114

16. M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995. doi:10.1137/S0097539793242618
On the Complexity of Local Graph Transformations

17 Martin Groß, Anupam Gupta, Amit Kumar, Jannik Matuschke, Daniel R. Schmidt, Melanie Schmidt, and José Verschae. A Local-Search Algorithm for Steiner Forest. In Anna R. Karlin, editor, 9th Innovations in Theoretical Computer Science Conference (ITCS 2018), volume 94 of Leibniz International Proceedings in Informatics (LIPIcs), pages 31:1–31:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITCS.2018.31

18 Anupam Gupta and Amit Kumar. Greedy algorithms for steiner forest. In Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing, STOC ’15, pages 871–878, New York, NY, USA, 2015. ACM. doi:10.1145/2746539.2746590

19 Yair Halevi and Yishay Mansour. A network creation game with nonuniform interests. In Internet and Network Economics, Third International Workshop, WINE 2007, San Diego, CA, USA, December 12-14, 2007, Proceedings, pages 287–292, 2007. doi:10.1007/978-3-540-77105-0_28

20 Reiko Heckel. Graph transformation in a nutshell. Electr. Notes Theor. Comput. Sci., 148(1):187–198, 2006. doi:10.1016/j.entcs.2005.12.018

21 Riko Jacob, Andrée W. Richa, Christian Scheideler, Stefan Schmid, and Hanjo Täubig. Skip +: A self-stabilizing skip graph. J. ACM, 61(6):36:1–36:26, 2014. doi:10.1145/2629695

22 Riko Jacob, Stephan Ritscher, Christian Scheideler, and Stefan Schmid. Towards higher-dimensional topological self-stabilization: A distributed algorithm for delaunay graphs. Theor. Comput. Sci., 457:137–148, 2012.

23 Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. Combinatorica, 21(1):39–60, 2001. doi:10.1007/s004930170004

24 Kishore Kothapalli and Christian Scheideler. Supervised peer-to-peer systems. In 8th International Symposium on Parallel Architectures, Algorithms, and Networks, ISPAN 2005, December 7-9, 2005, Las Vegas, Nevada, USA, pages 188–193, 2005. doi:10.1109/ISPAN.2005.81

25 Andreas Koutsopoulos, Christian Scheideler, and Thim Strothmann. Towards a universal approach for the finite departure problem in overlay networks. Inf. Comput., 255:408–424, 2017. doi:10.1016/j.ic.2016.12.006

26 Leonid Anatolevich Levin. Universal sequential search problems. Problemy Peredachi Informatsii, 9(3):115–116, 1973.

27 Mo Li, Zhenjiang Li, and Athanasios V. Vasilakos. A survey on topology control in wireless sensor networks: Taxonomy, comparative study, and open issues. Proceedings of the IEEE, 101(12):2538–2557, 2013. doi:10.1109/JPROC.2013.2257631

28 Chih-Long Lin. Hardness of approximating graph transformation problem. In Algorithms and Computation, 5th International Symposium, ISAAC ’94, Beijing, P. R. China, August 25-27, 1994, Proceedings, pages 74–82, 1994. URL: https://doi.org/10.1007/3-540-58325-4_168

29 Rizal Mohd Nor, Mikhail Nesterenko, and Christian Scheideler. Corona: A stabilizing deterministic message-passing skip list. Theor. Comput. Sci., 512:119–129, 2013. doi:10.1016/j.tcs.2012.08.029

30 Ram Ramanathan and Regina Hain. Topology control of multihop wireless networks using transmit power adjustment. In Proceedings IEEE INFOCOM 2000, The Conference on Computer Communications, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Reaching the Promised Land of Communications, Tel Aviv, Israel, March 26-30, 2000, pages 404–413, 2000. doi:10.1109/INFCOM.2000.832213

31 Grzegorz Rozenberg, editor. Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations. World Scientific, 1997.

32 Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker. Splaynet: Towards locally self-adjusting networks. IEEE/ACM Trans. Netw., 24(3):1421–1433, 2016. doi:10.1109/TNET.2015.2410313
In this section, we provide the proof of Lemma 5. We split the proof into several claims that we show individually. Throughout this section, we refer to the variables defined in Definition 2. We say a computation is feasible if and only if its initial graph is $G_t$, its target graph is $G_t$ and its length is at most $2n + m$. Furthermore, we say that the edge that is established during the application of an Introduction or Delegation primitive (the edge $(v, w)$ in Figures 1a and 1b) is the result of the Introduction or Delegation, respectively.

The first claim we show is the following:

\begin{itemize}
  \item [\textbf{Claim 11.}] In every feasible computation for every $i \in [n]$, there is a node $z_i \in \{s_i, x_i, \overline{x}_i\}$ that applies either a Fusion primitive or an Introduction or Delegation primitive whose result is not in $E_c := \{(C_j, r)| 1 \leq j \leq m\} \cup \{(C_j, C_i)| i \leq n\} \cup \{(C_j, t)| i \in [n], j \in [m]\} \cup \{(C_j, t)| i \in [n], j \in [m]\} \cup \{(t_i, t_k)| i, k \in [n]\}.$
\end{itemize}

\bf{Proof.} Assume for contradiction that there is a feasible computation and an $i \in [n]$ such that there is no $z_i \in \{s_i, x_i, \overline{x}_i\}$ that applies either a Fusion primitive or an Introduction or Delegation primitive whose result is not in $E_c$. Note $s_i$ cannot delegate away one of its incident edges $\{s_i, x_i\}$ or $\{s_i, \overline{x}_i\}$ as the result would not be in $E_c$ (for it would be incident to $x_i$ or $\overline{x}_i$). Similarly, $x_i$ and $\overline{x}_i$ could not delegate away $\{x_i, s_i\}$ and $\{x_i, \overline{x}_i\}$, respectively. Therefore, the edges $\{x_i, s_i\}$ and $\{s_i, \overline{x}_i\}$ must be kept throughout the computation. Now observe that $s_i$ has an initial degree of three. Since $s_i$ has a degree of two in the final graph, there must be at least one application of a primitive in which $s_i$’s degree decreases. Consider the last such application, i.e., the resulting neighborhood of $s_i$ is $x_i$ and $\overline{x}_i$ (remember that these edges persist throughout the computation). If it was an application of a Fusion primitive, then $x_i$, $s_i$ or $\overline{x}_i$ must have applied this primitive, yielding a contradiction. Otherwise, it must have been a Delegation primitive applied by $s_i$ for no other primitive could reduce $s_i$’s degree then. However, the result of this Delegation primitive must be an edge incident to $x_i$ or $\overline{x}_i$ then, i.e., an edge not in $E_c$ yielding a contradiction in this case as well. All in all, we have proven Claim 11.
\end{itemize}

The second claim we prove is:

\begin{itemize}
  \item [\textbf{Claim 12.}] In every feasible computation there is no graph that contains, for arbitrary $i, k \in [n], j, l \in [m]$: (i) more than one edge $\{t_i, r\}$ or $\{C_j, r\},$ (ii) an edge $\{C_j, C_i\}, \{t_i, C_j\}$, or $\{t_i, t_k\},$ (iii) an edge $\{u, v\}$ such that $u \in X_i$ and $v \in X_k$, and (iv) whenever an edge $\{t_i, r\}$ exists in a graph of the computation, it exists in all subsequent graphs as well, and (v) for each $i \in [n]$ the nodes in $V_{X_i}$ may apply at most one primitive whose result is not an edge $\{t_k, r\}$ or $\{C_j, r\}$ for $k \in [n], j \in [m]$, and (vi) the nodes in $\{r\} \cup \{C_j| j \in [m]\}$ do not apply any primitives at all.
\end{itemize}

\bf{Proof.} For the proof of this claim note that every application of a primitive can create just one of the following types of edges: A) an edge $\{t_i, r\}$ for some $i \in [n]$, B) an edge $\{C_j, r\}$ for some $j \in [m]$, or C) an edge $e \in E_c$. Note that in order to obtain the final graph, the nodes need to establish $n$ edges of type A, $m$ edges of type B, and, according to the Claim 11 for every $i \in [n]$, one of the nodes in $\{x_i, s_i, \overline{x}_i\}$ has to apply a primitive whose result is an edge of type C. Since there may be at most $2n + m$ applications of primitives, no other primitives may be applied. In particular, no edge $\{t_i, r\}, i \in [n]$ or $\{C_j, r\}, j \in [m]$ may exist multiple times, which proves (i), and no edge $\{C_j, C_i\}, \{t_i, C_j\}$, or $\{t_i, t_k\}$ may be created for any $i, k \in [n], j, l \in [m]$, proving (ii), and for each $i \in [n]$ the nodes in $V_{X_i}$ may not apply more than one primitive that creates neither an edge of type A nor type B (only
On the Complexity of Local Graph Transformations

the single primitive creating an edge of type C), proving (v). Since \( r \) cannot create the first instance of an edge of type A or B by itself, \( r \) cannot apply any primitive at all as this would require more than \( 2n + m \) primitive applications then \( (n + m) \) are used to create the type A/B edges and an additional \( m \) primitives are applied by the nodes from \( \{x_i, s_i, \overline{x}_i| i \in [n]\} \).

For the same reason, the nodes in \( \{t_i, C_j| i \in [n], j \in [m]\} \) cannot apply any primitives, which together with the fact that \( r \) cannot apply any primitives yields (iv) and (vi). In addition this implies that the initial graph can be decomposed into \( V_{X_1}, V_{X_2}, \ldots, V_{X_n}, P \) with \( P = \{r, C_j| j \in [m]\} \) such that there is no edge \( \{u, v\} \) with \( u \in V_{X_i}, v \in V_{X_j}, u \neq v \) and the nodes in \( P \) do not apply any primitive at all. Thus, (iii) follows from Lemma 4.

The last claim we show is the following:

\[ \triangleright \text{Claim 13.} \quad \text{In every feasible computation, for every } i \in [n] \text{ there must be a graph containing an edge } \{x_i, r\} \text{ or } \{\overline{x}_i, r\}. \]

This gives that during a feasible computation, the edges \( \{t_i, r\} \) for all \( i \in [n], \{C_j, r\} \) for all \( j \in [m] \) and \( \{y_i, r\}, y_i \in \{x_i, \overline{x}_i\} \) for all \( i \in [n] \) have to be created. Since each primitive can create at most one of these edges and the length of the computation is at most \( 2n + m \), this implies the claim of Lemma 5 (recall that the \( m \) edges \( \{C_j, r\} \) and the \( n \) edges \( \{t_i, r\} \) have to be created, too).

\textbf{Proof.} To prove Claim 13, we show that \( s_i \) has to delegate \( \{s_i, r\} \) to \( x_i \) or \( \overline{x}_i \). We do so by proving that \( s_i \) cannot have a neighbor other than \( x_i, \overline{x}_i \) or \( r \), which is sufficient because \( r \) does not apply any primitive according to (vi) of Claim 12. Assume for contradiction that \( s_i \) has an edge \( \{s_i, v\} \) such that \( v \notin \{x_i, \overline{x}_i, r\} \) and let \( \{s_i, v\} \) be the last such edge that occurs in a graph in the computation. Due to (iii) of Claim 12 \( v \in \{t_i \cup C_j| j \in [m]\} \). Consider the node \( w \) that applied an Introduction or Delegation primitive whose result was the edge \( \{s_i, v\} \). For this to be possible, there must have been an edge \( \{w, s_i\} \) when \( w \) applied the primitive. Since \( r, t \) and the \( C_j \) for \( j \in [m] \) do not apply any primitives and because of (iii) of the Claim 12 \( w \in \{x_i, s_i, \overline{x}_i\} \). According to (v), the computation may not contain another application (than this one) of a Delegation or Introduction primitive by a node in \( V_X \), whose result is not an edge \( \{t_k, r\} \text{ or } \{C_j, r\} \text{ for } k \in [n], j \in [m] \text{ (*)}. Since \( \{s_i, v\} \notin E(G_i) \), this edge must be removed by some application of a Delegation primitive. Note that since \( v \in \{t_i \cup C_j| j \in [m]\} \), \( s_i \) must apply this primitive. Since \( \{s_i, v\} \) is the last edge different from \( \{s_i, r\}, \{s_i, x_i\} \), and \( \{s_i, \overline{x}_i\} \), \( v \) must be delegated to one of the nodes \( r, x_i, \text{ and } \overline{x}_i \). If \( s_i \) delegates \( \{s_i, v\} \) to \( x_i \) or \( \overline{x}_i \), then the result is an edge \( \{x_i, v\} \text{ or } \{\overline{x}_i, v\} \), which contradicts (*). Thus assume \( s_i \) delegates \( \{s_i, v\} \) to \( r \). Note that after this delegation, the edge \( \{s_i, r\} \) exists and \( s_i \) does not have a neighbor \( v' \in \{t_i \cup C_j| j \in [m]\} \) in any of the subsequent graphs (recall that \( v \) was the last edge of its kind). Since \( \{s_i, r\} \notin E(G_i) \), and \( r \) does not apply any primitives according to (vi) of Claim 12 this edge must be delegated to either \( x_i \text{ or } \overline{x}_i \) yielding an edge \( \{x_i, r\} \text{ or } \{\overline{x}_i, r\} \), which contradicts (*) as well. As mentioned above this proves that \( s_i \) has to delegate \( \{s_i, r\} \) to \( x_i \) or \( \overline{x}_i \), and, as argued before as well, also implies the claim of the lemma.

\[ \triangleright \]

\[ \textbf{B \quad k-DLGT is NP-hard} \]

The proof of the \( \text{NP}\)-completeness of \( \text{k-DLGT} \) is very similar to that of \( \text{k-ULGT} \). Whereas the proof that \( \text{k-ULGT} \) is in \( \text{NP} \) directly transfers to \( \text{k-DLGT} \), the proof of \( \text{k-DLGT} \)’s \( \text{NP}\)-hardness is actually a simpler variant of the proof of \( \text{k-ULGT} \)’s \( \text{NP}\)-hardness. We thus do not state the proof in full length here, but only sketch its idea: The reduction function
Theorem 14. We begin with establishing a relationship between solutions to \((\text{in which } \text{ALG}_G)\) versus \((\text{DLGT})\). In this subsection we describe how to adapt Algorithm 1 to obtain a constant approximation proof. ▶

**Proof.** We begin with establishing a relationship between solutions to DLGT and to ULGT. For a graph \(G = (V, E)\), let \(U(G) := (V, U(E))\) with \(U(E) := \{(u, v) | (u, v) \in E\}\). For arbitrary directed graphs \(G_s\) and \(G_t\), let \(C_d(G_s, G_t)\) be an optimal solution to DLGT with (directed) initial graph \(G_s\) and (directed) final graph \(G_t\) (this problem we denote by \(\text{MP}_d(G_s, G_t)\)) and let \(\text{OPT}_d(G_s, G_t)\) be its length. Let \(C_u((U(G_s), U(G_t)))\) be an optimal solution to ULGT with initial graph \(U(G_s)\) and final graph \(U(G_t)\), denote its length by \(\text{OPT}_u((U(G_s), U(G_t)))\), and denote this problem by \(\text{MP}_u((U(G_s), U(G_t)))\). Let the computation \(C'_u\) be obtained from \(C_d\) by ignoring all edge directions and removing all applications of Reversal primitives from \(C_d\). Note that \(C'_u\) is a solution to \(\text{MP}_u((U(G_s), U(G_t)))\) and that its length is thus lower bounded by \(\text{OPT}_u((U(G_s), U(G_t)))\) and upper bounded by \(\text{OPT}_d(G_s, G_t)\) (since we only shortened the optimal solution to \(\text{MP}_d(G_s, G_t)\)). Therefore, \(\text{OPT}_u((U(G_s), U(G_t))) \leq \text{OPT}_d(G_s, G_t)\) for all directed graphs \(G_s\) and \(G_t\). This insight allows us to compare the length of the solution computed by Algorithm 2 for initial graph \(G_s\) and final graph \(G_t\) to \(\text{OPT}_u(U(G_s), U(G_t))\).

Consult Algorithm 2 for the adapted version of the approximation algorithm (with differences to Algorithm 1 highlighted in boldface). Due to the similarities with Algorithm 1, the only part that requires some additional explanation is the creation of edges in Step 2 and Step 6: In general, the procedure is very similar to that used in Algorithm 1 (described in the proofs of Section 3.2), with the only clarification that the edges delegated / introduced through the tree must be directed towards the root node (and that in Step 6, for each edge \((u, v) \in E_\oplus\), \(u\) corresponds to \(s(e)\) in Algorithm 1). For Step 2, however, this is not sufficient because this process creates the additional edges \((u, r_T)\) for every \(u\) incident \(r_T\), whereas \((r_T, u)\) is desired. Thus, for each such edge, we apply the Reversal primitive to turn \((r_T, u)\) into \((u, r_T)\), which requires at most \(2|E_\oplus|\) applications of primitives. Note that in the first part only change that requires additional primitives is the reversal in Step 4, whose number is, again, upper bounded by \(2|E_\oplus|\). Thus the length \(\text{ALG}_2^1(G_s, G_t)\) of the computation computed in the first part by Algorithm 2 is at most \(\text{ALG}_1(U(G_s), U(G_t)) + 4|E_\oplus|\) (in which \(\text{ALG}_1(U(G_s), U(G_t))\) is the length of the computation of the first part of Algorithm 1 for initial graph \(U(G_s)\) and final graph \(U(G_t)\)). In the second part, only Step 8 causes additional primitive applications and their number is at most \(|E_\oplus|\) (c.f. the proof of Lemma 10). Therefore, the length \(\text{ALG}_2^2(G_s, G_t)\) of the computation computed in the second part by Algorithm 2 is at most \(\text{ALG}_2(U(G_s), U(G_t)) + |E_\oplus|\) (in which \(\text{ALG}_2(U(G_s), U(G_t))\) is the length of the computation of the second part of Algorithm 1 for initial graph \(U(G_s)\) and final graph \(U(G_t)\)). For the same arguments used in

### C. An approximation algorithm for DLGT

In this subsection we describe how to adapt Algorithm 1 to obtain a constant approximation algorithm, Algorithm 2 for DLGT, which implies the following theorem:

**Theorem 14.** DLGT ∈ APX.
Algorithm 2 Approximation algorithm for DLGT

**Input:** Initial graph $G_s$ and final graph $G_t$.

**First part:**
1. Compute a 2-approximate solution $F_{ALG,\oplus}$ for the USF with input $U(G_s)$, and the set $U(E_{\oplus})$ as the set of pairs of nodes.
2. For each tree $T$ in $F_{ALG,\oplus}$, select a root node $r_T$ and for all nodes $u$ in $T$ that are incident to an edge in $E_{\oplus}$, establish the edge $(r_T, u)$ (details below).
3. For each $(u, v) \in E_{\oplus}$, the root of the tree $u$ and $v$ belong to applies the Introduction primitive to create the edge $(u, v)$.
4. For each tree $T$ in $F_{ALG,\oplus}$, reverse all superfluous edges (i.e., not belonging to $G_s$ or $E_{\oplus}$) created during Step 2 and delegate them bottom up in $T$ rooted at $r_T$, starting with the lowest level. At each intermediate node fuse all of these edges before delegating them to the next predecessor.

**Second part:**
5. Compute a 2-approximate solution $F_{ALG,\ominus}$ for the USF with input $U(G_t)$, and the set $U(E_{\ominus})$ as the set of pairs of nodes.
6. For each tree $T$ in $F_{ALG,\ominus}$, select a root node $r_T$ and for each $(u, v) \in E_{\ominus}$ whose endpoints belong to $T$, establish the edge $(u, r_T)$ (similar to Step 2, details below).
7. For each $(u, v) \in E_{\ominus}$, $u$ delegates $(u, v)$ to $r_T$.
8. For each tree $T$ in $F_{ALG,\ominus}$, reverse all superfluous edges, delegate them bottom-up and fuse multiple edges as in Step 4.

The proofs of Lemma 9 and Lemma 10 it follows that $ALG_1^2(G_s, G_t) \leq 15OPT_u(U(G_s), G')$ and $ALG_2^2(G_s, G_t) \leq 8OPT_u(G', U(G_t))$, where $G' = U((V, E_s \cup E_{\oplus}))$ for for $E_s$ being the edge set of $G_s$. By Lemma 3 we obtain $ALG^2(G_s, G_t) := ALG_1^2(G_s, G_t) + ALG_2^2(G_s, G_t) \leq 15(2OPT_u(U(G_s), U(G_t)) + |E_{\oplus}|) \leq 45OPT_u(U(G_s), U(G_t)) \leq 45OPT_d(G_s, G_t)$. \hfill \(
