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 of 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.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Approximation algorithms analysis

Keywords and phrases Graphs transformations, NP-hardness, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.150

Category Track C: Foundations of Networks and Multi-Agent Systems: Models, Algorithms and Information Management

Related Version A full version of the paper is available at https://arxiv.org/abs/1904.11395.

Funding This work was supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901) under Grant No.: GZ SFB 901/02.

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) to allow the exchange of 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., [33, 14, 2, 19, 11, 3, 10, 8]). Many of these approaches are decentralized, and because of that,

46th International Colloquium on Automata, Languages, and Programming (ICALP 2019).
Editors: Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi;
Article No. 150; pp. 150:1–150:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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 $\mathsf{P} \neq \mathsf{NP}$, 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:

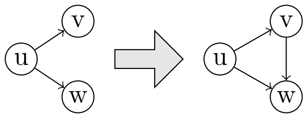
Introduction. If a node u has a reference of two nodes v and w with $v \neq w$, u *introduces* w to v if u sends a message to v containing a reference of w while keeping the reference.

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.

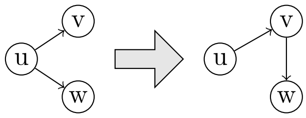
Fusion. If a node u has two references v and w with $v = w$, then u *fuses* the two references if it only keeps one of these references.

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.

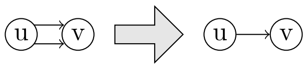
The four primitives are visualized in Figure 1. Note that for the Introduction primitive, it is possible that $w = u$, i.e., u introduces itself to v. To simplify the description, we sometimes say that a node u introduces or delegates the *edge* (u, v) if u introduces v to some other node or delegates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



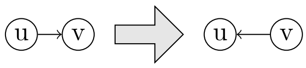
(a) Introduction Primitive.



(b) Delegation Primitive.



(c) Fusion primitive.



(d) Reversal primitive.

Figure 1 The four primitives in P_d in pictures.

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:

▶ **Observation 1.** *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 graph is G_s and whose final graph is G_t. The corresponding decision problem κ-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 κ-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 $L \to R$ 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., [30]). 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 omit its proof and only briefly sketch the differences in the full version of this paper [32].

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 $x_1, \ldots, x_n$ and a Boolean formula Φ over the variables in X in conjunctive normal form (CNF), decide whether there is a truth assignment $t : X \to \{0, 1\}$ that satisfies Φ.*

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

▶ **Definition 2** (Reduction function). *Let $S = (X, \Phi)$ be a SAT instance, in which $X = \{x_1, \ldots, x_n\}$ is the set of Boolean variables and $\Phi = C_1 \wedge \cdots \wedge C_m$ for clauses $C_1, \ldots, C_m$. Then $f(S) = (G_s, G_t, k)$ in which $k = 2n + m$ and G_s and G_t are undirected graphs defined as follows. Without loss of generality, assume that each literal $y_i \in \{x_i, \overline{x_i}\}$ occurs only once in each clause. We say $y_i \in C_j$ if literal y_i occurs in C_j.*

We define the following sets of nodes: $V_C = \{C_1, \ldots, C_m\}$, and $V_{X_i} = \{x_i, \overline{x_i}, s_i, t_i\}$. Then, the set of nodes of G_s and G_t is $V = \bigcup_{1 \le i \le n} V_{X_i} \cup V_C \cup \{r\}$. For the set of edges, define $E_{X_i} = \{\{s_i, x_i\}, \{\overline{x_i}, s_i\}, \{x_i, t_i\}, \{t_i, \overline{x_i}\}\}$, $E_{C_j} = \{\{y_i, C_j\} | y_i \in \{x_i, \overline{x_i}\} \wedge y_i \text{ occurs in } C_j\}$, $E_{sr} = \{\{s_i, r\} | 1 \le i \le n\}$, $E_{tr} = \{\{t_i, r\} | 1 \le i \le n\}$, $E_{Cr} = \{\{C_j, r\} | 1 \le j \le m\}$. Both G_s and G_t have the edges in $\bigcup_{1 \le i \le n} E_{X_i} \cup \bigcup_{1 \le j \le m} E_{C_j}$. Additionally, G_s has the edges in E_{sr} and G_t has the edges in $E_{tr} \cup E_{Cr}$.

Intuitively, each variable x_i is mapped to a *gadget* X_i consisting of the four nodes $x_i, \overline{x_i}, s_i$, and t_i. Also each clause C_j is connected with each literal occurring within it. Lastly, in G_s, each of the s_i is connected with the node r, whereas in G_t, each of the t_i and each of the C_j 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 $f(S) = (G_s, G_t, k)$ 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 $2n + m$.*

Proof. Assume there is a satisfying truth assignment $t : X \to \{0, 1\}$ of S. For every $1 \le i \le n$ let $y_i := x_i$ if $t(x_i) = 1$ or $y_i := \overline{x_i}$ if $t(x_i) = 0$. We construct the following computation with initial graph G_s and final graph G_t:

1. For every $1 \le i \le n$, s_i delegates the edge $\{s_i, r\}$ to y_i.
2. For every $C_j \in \{C_1, \ldots, C_m\}$ choose one neighbor $z_j \in \{y_1, \ldots, y_n\}$ (we show below that this exists), and let z_j introduce r to C_j.
3. For every $1 \le i \le n$, y_i delegates the edge $\{y_i, r\}$ to t_i.

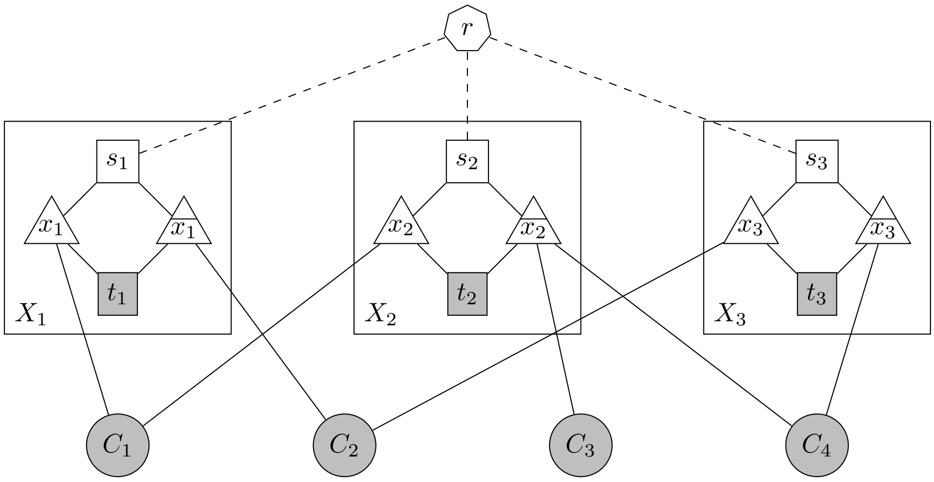


Figure 2 Graph G_s returned by the reduction function for the (example) Boolean formula $(x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (\overline{x_2}) \wedge (\overline{x_2} \vee \overline{x_3})$. G_t differs from G_s in that the dashed edges do not exist and all grey nodes share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

Obviously, the length of this computation is $2n + m$. To prove the missing part, recall that every C_j is satisfied under t, i.e., there is at least one literal z_j in C_j that evaluates to true, i.e., there is an $i \in [n]$ such that $z_j = x_i$ if $t(x_i) = 1$, or $z_j = \overline{x_i}$ if $t(x_i) = 0$. By definition of y_i, $z_j = y_i$. Thus because z_j occurs in C_j, y_i was a neighbor of C_j during Step 2. ◀

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 $V_1, \dots, V_k, P$ 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 $V_1, \dots, V_k, P$ 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 $y_1, \dots, y_n$, $y_i \in \{x_i, \overline{x_i}\}$ 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).*

Due to space constraints, we only sketch the proof here, whereas the full proof can be found in the full version of this paper [32]. 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 implies that each gadget may apply no other primitive than p_i to create an edge that is not in the target graph and that the nodes r and C_j themselves cannot apply any primitives at all which by Lemma 4 means that there are no inter-gadget edges. We use these facts to prove that p_i is used to remove the edge $\{s_i, r\}$ thereby creating either $\{x_i, r\}$ or $\{\overline{x_i}, r\}$.

The rest of the proof of the "if" part, as formalized by the following lemma, is comparably straightforward.

▶ **Lemma 6.** *Let S be a* SAT *instance as in Definition 2. If $f(S) = (G_s, G_t, k)$ with $k = 2n + m$ is a* κ-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 2. 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 κ-ULGT instance and there is a feasible computation C for $f(S)$. According to Lemma 5 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 $E(G_s) \cup E(G_t) \cup \{\{y_i, r\} | [n]\}$. 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 introduce or Delegation primitive applied by an y_i, $i \in [n]$ (as throughout C, the C_js do not have any other neighbors with an edge to r that could possible create this edge) . Let $g : \{C_1, C_2, \ldots, C_m\} \to \{y_1, y_2, \ldots, y_n\}$ 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 $t : X \to \{0, 1\}$ such that $t(x_i) = 1$ if $y_i = x_i$ and $t(x_i) = 0$ if $y_i = \overline{x_i}$. 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). Note that a constant approximation factor algorithm for DLGT can be obtained by a slight adaptation of this algorithm. For a description of this, we refer the reader to the full version [32] due to space constraints.

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.

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.

Algorithm 1 Approximation algorithm for ULGT.

Input: Initial graph G_s and final graph G_t.

First part:

1: Compute a 2-approximate solution $F_{ALG,\oplus}$ for the USF with input G_s, and the set $E_\oplus$ as the set of pairs of nodes.
2: For each tree T in $F_{ALG,\oplus}$, select a root node r_T and connect all nodes in T that are incident to an edge in $E_\oplus$ with r_T (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 $F_{ALG,\oplus}$, delegate all superfluous edges (i.e., not belonging to G_s or $E_\oplus$) 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 $F_{ALG,\ominus}$ for the USF with input G_t, and the set $E_\ominus$ as the set of pairs of nodes.
6: For each $e \in E_\ominus$, let $s(e)$ be an arbitrary of the two endpoints of e. For each tree T in $F_{ALG,\ominus}$, select a root node r_T and for each $e \in E_\ominus$ whose endpoints belong to T, connect $s(e)$ with r_T (similar to Step 2, details below).
7: For each $e \in E_\ominus$, $s(e)$ delegates the other endpoint to r_T.
8: For each tree T in $F_{ALG,\ominus}$, delegate all superfluous edges bottom-up and fuse multiple edges as in Step 4.

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 $\{u, v\} \in E_\oplus$, 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 $E_\oplus$ 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 $\{u, v\} \in E_\oplus$ 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 $E_\ominus$ 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 $E_\ominus$ 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 $e \in E_\ominus$, the endpoint selected in the sixth step delegates this edge to r_T (resulting in the edge $\{r_T, v\}$).

3.2 Analysis

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

▶ **Theorem 7.** ULGT $\in$ 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 $ALG_1(G_s, G_t)$ be the length of the 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 $G' = (V, E_s \cup E_\oplus)$. 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 1 due to the following lemma:

▶ **Lemma 8.** $OPT(G_s, G') + OPT(G', G_t) \leq 2OPT(G_s, G_t) + |E_\oplus|$.

Proof. Let $\mathcal{P}$ denote the problem equal to κ-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 $OPT'(G_s, G_t) \leq 2OPT(G_s, G_t) + |E_\oplus|$.

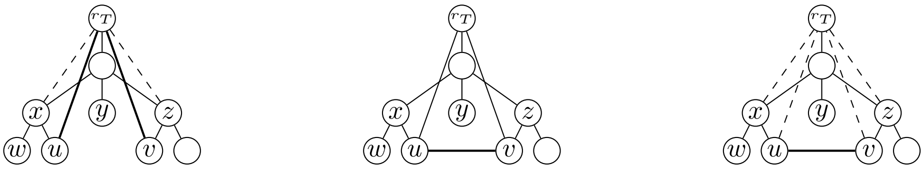
Consider a computation C whose initial graph is G_s, whose final graph is G_t and whose length is $OPT(G_s, G_t)$ (note that such a computation is an optimal solution to ULGT). We now transform C into a computation that represents a solution to $\mathcal{P}$. This transformation increases its length by only $OPT(G_s, G_t) + |E_\oplus|$ and thus proves the above claim (recall that any solution to $\mathcal{P}$ has at least the size of an optimal solution to it). First, because the final graph does not contain any edge $\{u, v\} \in E_\ominus$, for every such edge there is one last Delegation in C that removes this edge (recall Observation 1). 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 infeasible as this only causes the graph to have additional edges. The final graph of C' is $(V, E_t \cup E_\ominus) = (V, E_s \cup E_\oplus) = G'$ (recall that $E_t = (E_s \cup E_\oplus) \setminus E_\ominus$). 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 to G_t, this second half of C'', which starts from $G' = (V, E_s \cup E_\oplus)$, has the final graph $G'' = (V, E_t \cup E_\oplus)$, i.e., each edge from $E_\oplus$ appears twice in G''. Thus we extend C'' by fusing each edge from $E_\oplus$ with its double, resulting in a computation C''' of length $2OPT(G_s, G_t) + |E_\oplus|$. Since C''' represents a solution to $\mathcal{P}$ 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 $ALG(G_s, G_t) \leq 11(2OPT(G_s, G_t) + |E_\oplus|) \leq 33OPT(G_s, G_t)$ (since, clearly, $OPT(G_s, G_t) \geq |E_\oplus|$), which yields the claim of Theorem 7.

We begin with the former claim, which is formalized by the following lemma:

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

Proof. Let $F_{OPT,\oplus}$ be an optimal solution for the USF with input G_s and $E_\oplus$ as the set of nodes and recall that $F_{ALG,\oplus}$ is the USF approximation computed in Step 1 of Algorithm 1. Throughout the analysis, $|F_{OPT,\oplus}|$ and $|F_{ALG,\oplus}|$ will denote the number of edges in these



(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.

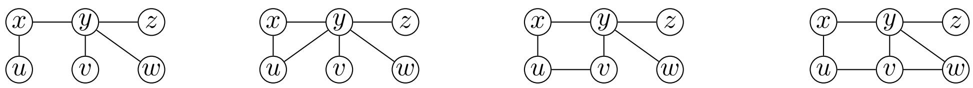
(c) Step 4 removes all superfluous edges by delegating and fusing them up in the tree.

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.

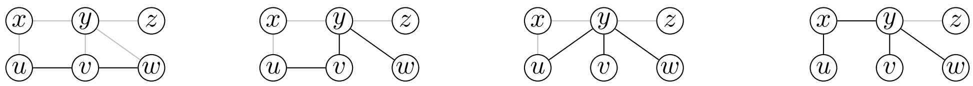
solutions. In the first part of this proof, we show that $ALG_1(G_s, G_t) \le 4|F_{OPT,\oplus}| + 3|E_\oplus|$. The second part then consists in proving $OPT(G_s, G') \ge |F_{OPT,\oplus}| - |E_\oplus|$, which together with the observation that $OPT(G_s, G') \ge |E_\oplus|$ yields the claim.

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 $F_{ALG,\oplus}$ connects the desired nodes to r_T in Step 2 in the following 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 3a. 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 3b) involves exactly $E_\oplus$ applications of primitives. For the length of Step 4 (c.f. Figure 3c), 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 $F_{OPT,\oplus}$, we obtain $ALG_1(G_s, G_t) \le 4|F_{OPT,\oplus}| + 3|E_\oplus|$.

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 $L < |F_{OPT,\oplus}| - |E_\oplus|$. 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 $P^L_{u,v} := (u,v)$. Note that $P^L_{u,v}$ exists in G_L. We iterate through C in reverse order and for every graph G_i, if $P^{i+1}_{u,v}$ exists in G_i, $P^i_{u,v} := P^{i+1}_{u,v}$. 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 $P^{i+1}_{u,v}$ 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 $P^i_{u,v}$ be $P^{i+1}_{u,v}$ with (x,y) replaced



(a) 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.



(e) G_4: $P^4_{u,v} = (u, v)$ and $P^4_{v,w} = (v, w)$. **(f)** G_3: $P^3_{u,v} = (u, v)$ and $P^3_{v,w} = (v, y, w)$. **(g)** G_2: $P^2_{u,v} = (u, y, v)$ and $P^2_{v,w} = (v, y, w)$. **(h)** G_1: $P^1_{u,v} = (u, x, y, v)$ and $P^1_{v,w} = (v, y, w)$.

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 $P^i_{u,v}$ and $P^i_{v,w}$, which are defined by iterating through C in reverse order. In the lower row, the edges drawn black in G_i are the edges belonging to to F^i. Observe that F_1 represents a solution to the USF for graph G_1 and node pairs $E_\oplus$.

by (x, w, y) and note that $P^i_{u,v}$ exists in G_i. Eventually, we obtain a path $P^1_{u,v}$ that exists in G_s. For $i \in \{1, \ldots, L\}$, let $F^i := \bigcup_{\{u,v\}\in E_\oplus} E(P^i_{u,v})$ (where $E(P)$ is the set of all edges on the path P) and note that F^1 represents a solution to 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 $|F^i| \le |F^{i+1}| + 1$: 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 $\{u', v'\}$, $P^i_{u',v'} = P^{i+1}_{u',v'}$. By the definition of F^i and F^{i+1}, this implies $|F^i| \le |F^{i+1}| + 1$ also in this case. All in all we obtain that $|F^1| \le |F^L| + L = |E_\oplus| + L$ 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.** *$ALG_2(G_s, G_t) \le 7OPT(G', G_t)$.*

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 $F_{OPT,\ominus}$ be an optimal solution for the USF with input G_t and $E_\ominus$ as the set of nodes and recall that $F_{ALG,\ominus}$ is the USF approximation computed in Step 5 of Algorithm 1. Analogously, $|F_{OPT,\ominus}|$ and $|F_{ALG,\ominus}|$ denote the number of edges in these solutions. In the first part of this proof, we show that $ALG_2(G_s, G_t) \le 4|F_{OPT,\ominus}| + 3|E_\ominus|$. The second part then consists in proving $OPT(G', G_t) \ge |F_{OPT,\ominus}|$, which together with the observation that $OPT(G', G_t) \ge |E_\ominus|$ yields the claim.

To upper bound $ALG_2(G_s, G_t)$, 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 r_T again. Here, each intermediate node u checks whether $u = s(e)$ for some $e \in E_\ominus$. 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 $e' \in E_\ominus$). Otherwise, it introduces r_T to all but one relevant children and delegates it to the remaining one. In the end, for every edge $e \in E_\ominus$, $s(e)$ has an edge to r_T, the number of edges in the graph has increased by at most $|E_\ominus|$, and the process involved at most $|F_{ALG,\ominus}|$ applications of primitives. In Step 7, clearly exactly $|E_\ominus|$ edges have to be delegated. Step 8 is similar to Step 4 and for analogous reasons requires at most $|F_{ALG,\ominus}|$ delegations and at most $2|E_\ominus|$ fusions (recall that up to $|E_\ominus|$ 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 $|F_{ALG,\ominus}|$, $|E_\ominus|$ and $|F_{ALG,\ominus}| + 2|E_\ominus|$ applications of primitives, respectively, which yields: $ALG_2(G_s, G_t) \leq 2|F_{ALG,\ominus}| + 3|E_\ominus| \leq 4|F_{OPT,\ominus}| + 3|E_\ominus|$ (since $F_{ALG,\ominus}$ is a 2-approximation of $F_{OPT,\ominus}$).

To lower bound the value of $OPT(G', G_t)$, assume for contradiction that there is a computation C with initial graph G' and final graph G_s of length $L < |F_{OPT,\ominus}| - |E_\ominus|$. 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 9, for every $\{u, v\} \in E_\ominus$, we create a path from u to v, but this time we start with $P^1_{u,v} := (u, v)$ and consider the graphs in increasing order: For $i \in \{2, \ldots, L\}$, if $P^{i-1}_{u,v}$ exists in G_i, $P^i_{u,v} := P^{i-1}_{u,v}$. 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 $P^{i-1}_{u,v}$ 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 $\overline{z}$ 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 $\{z, \overline{z}\}$ was delegated to w, in G_i the edge $\{w, \overline{z}\}$ exists in G_i. Thus, let $P^i_{u,v}$ be $P^{i-1}_{u,v}$ with (x, y) replaced by (x, w, y) and observe that $P^i_{u,v}$ exists in G_i. Eventually, we obtain a path $P^L_{u,v}$ that exists in G_t. Define $F^i := \bigcup_{\{u,v\} \in E_\ominus} E(P^i_{u,v})$ (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_t and $E_\ominus$ 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 $|F^L| \leq |F^1| + L = |E_\ominus| + L$ because $F^1 = E_\ominus$ (note the definition of F^1. By the assumption that $L < |F_{OPT,\ominus}| - |E_\ominus|$, we obtain $|F^L| < |F_{OPT,\ominus}|$, 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. doi:10.1016/S0167-6423(98)00023-9.

5 James Aspnes and Yinghua Wu. O(logn)-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_7.

11 Erik D. Demaine, MohammadTaghi Hajiaghayi, Hamid Mahini, and Morteza Zadimoghaddam. 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, Elitza 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.

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, Andréa 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. doi: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 Christian Scheideler and Alexander Setzer. On the Complexity of Local Graph Transformations (full version), 2019. arXiv:1904.11395.

33 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.