log(n)-approximation d'un arbre de Steiner auto-stabilisant et dynamique

Lélia Blin, Maria Gradinariu Potop-Butucaru, Stephane Rovedakis

$\log(n)$-approximation d'un arbre de Steiner auto-stabilisant et dynamique

Lélia Blin [1,2], Maria Gradinariu Potop-Butucaru [2,3], Stephane Rovedakis[1]

[1]*IBISC, Université d'Evry, 91000 Evry, France.* [2]*LIP6-CNRS UMR 7606, France.* [3]*INRIA REGAL, France.*

Ce travail est motivé entre autre, par le maintient distribué d'infrastructures optimisées pour la communication d'un groupe d'utilisateurs dispersé sur un réseau dynamique. Les domaines d'application typiques de telles structures sont les systèmes de publish/subscribe, bases de données distribuées, systèmes multicasts. Dans ce papier nous décrivons un algorithme distribué qui construit et maintient un arbre de Steiner approché connectant un groupe dynamique de membres dispersé sur un réseau dynamique. Le coût de la solution retournée par notre algorithme est au plus $\log|S|$ fois le coût de la solution optimale, S étant le groupe de membres à interconnecter. Notre algorithme améliore les solutions existantes de plusieurs façons. Premièrement, il tolère le dynamisme des membres et du réseau, autrement dit les membres peuvent rejoindre ou quitter le groupe et les noeuds ou liens du réseau peuvent apparaître ou disparaître du réseau. Deuxièmement notre algorithme est auto-stabilisant, en d'autres termes il tolère les fautes transitoires. Enfin, notre algorithme est *super-stabilisant*, ce qui signifie que l'on garantie des propriétés sur la structure construite durant la convergence de l'algorithme et malgré le dynamisme du réseau.

1 Introduction

In this paper we are interested in the study of overlays targeted to efficiently connect a group of members that are not necessarily located in the same geographical area (e.g. sensors that should communicate their sensed data to servers located outside the deployment area, P2P nodes that share the same interest and are located in different countries, robots that should participate to the same task but need to remote coordinate). Steiner trees are good candidates to implement the above mentioned requirements since the problem have been designed for efficiently connect a subset of nodes, referred as Steiner members.

The Steiner tree problem can be informally expressed as follows : given a weighted graph in which a subset S of nodes is identified, find a minimum-weight tree spanning S. The Steiner tree problem is one of the most important combinatorial optimization problems and is NP-hard to solve optimally.

A survey on heuristics with different competitive ratios can be found in [Win87]. In our work, we are interested in dynamic variants of Steiner trees first addressed in [IW91] in a centralized online setting. Several algorithms are proposed, one of them is a $\log|S|$-approximation algorithm for this problem that cope only with Steiner member arrivals. This algorithm can be implemented in a decentralized environment (see [GRG05]).

Our work considers the fully dynamic version of the problem where both Steiner members and ordinary nodes or links can join or leave the system (i.e., group and topological dynamicity). The above mentioned solutions are not fault-tolerant or self-stabilizing. The property of self-stabilization [Dol00] enables a distributed algorithm to recover from a transient fault regardless of its initial state. The superstabilization [Dol00] is an extension of the self-stabilization property for dynamic settings. The idea is to provide some minimal structural guarantees while the system repairs after a topology change.

To our knowledge there are only two self-stabilizing approximations algorithms constructing Steiner trees [KK02a, KK02b]. Both works assume a communication via shared memory model and an unfair centralized scheduler. In [KK02a], the authors propose a self-stabilizing algorithm based on a pruned minimum spanning tree. The computed solution has a competitiveness of $|V|-|S|+1$, where V is the set of nodes in the network. In [KK02b], the authors proposed a four-layered algorithm that builds upon the techniques of

[WWW86] in order to obtain a 2-approximation algorithm. The two above cited algorithms work only for static networks.

Our results We describe a self-stabilizing algorithm for the Steiner tree problem. This algorithm has the following properties.

- First, it is distributed, i.e., completely decentralized. That is, nodes locally self-organize in a Steiner tree. The cost of the constructed Steiner tree is at most $\log|S|$ times the cost of an optimal solution, where S is the Steiner group.
- Second, our algorithm is specially designed to cope with user dynamism. In other words, our solution withstand when nodes (or links) join and leave at will.
- Third, our algorithm includes self-stabilization policies. Starting from an arbitrary state (nodes local memory corruption, counter program corruption, or erroneous messages in the network buffers), our algorithm is guaranteed to converge to a tree spanning the Steiner members.
- Fourth, our algorithm is *superstabilizing*. That is, while a fault occurs, i.e., during the restabilization period, the algorithm offers the guarantee that only the subtree connected through the crashed node or edge is reconstructed. This latter property is the core of our contribution.

2 Model and notations

We consider an undirected weighted connected network $G = (V,E,w)$ where V is the set of nodes, E is the set of edges and $w : E \rightarrow \mathbb{R}^+$ is a positive cost function. Nodes represent processors and edges represent bidirectional communication links. Each node in the network has an unique identifier. $S \subseteq V$ defines the set of members we have to connect. For any pair of nodes $u, v \in V$, the *distance* from u to v is the distance of the shortest path between u and v in G. A Steiner tree, T in G is a connected acyclic sub-graph of G such that $T = (V_T, E_T)$, $S \subseteq V_T \subseteq V$ and $E_T \subset E$. We denote by $W(T)$ the cost of a tree T, i.e., $W(T) = \sum_{e \in T} w(e)$.

We consider an asynchronous communication message passing model with FIFO channels (on each link messages are delivered in the same order as they have been sent).

A *local state* of a node is the value of the local variables of the node and the state of its program counter. We consider a fined-grained communication atomicity model [BK07, Dol00]. That is, each node maintains a local copy of the variables of its neighbors. These variables are refreshed via special messages (denoted in the sequel `InfoMsg`) exchanged periodically by neighboring nodes. A *configuration* of the system is the cross product of the local states of all nodes in the system plus the content of the communication links. The transition from a configuration to the next one is produced by the execution of an atomic step at a node. An *atomic step* at node p is an internal computation based on the current value of p's local variables and a single communication operation (send/receive) at p. A *computation* of the system is an infinite sequence of configurations, $e = (c_0, c_1, \ldots c_i, \ldots)$, where each configuration c_{i+1} follows from c_i by the execution of a single atomic step. In the sequel we consider the system can start in any configuration. That is, the local state of a node can be corrupted. Note that we don't make any assumption on the bound of corrupted nodes. In the worst case all nodes in the system may start in a corrupted configuration. In order to tackle these faults we use *self-stabilization* techniques.

Given $\mathcal{L}_{\mathcal{A}}$ a non-empty *legitimacy predicate*[†] an algorithm $\mathcal{A}$ is *self-stabilizing* iff the following two conditions hold : (i) Every computation of $\mathcal{A}$ starting from a configuration satisfying $\mathcal{L}_{\mathcal{A}}$ preserves $\mathcal{L}_{\mathcal{A}}$ (*closure*). (ii) Every computation of $\mathcal{A}$ starting from an arbitrary configuration contains a configuration that satisfies $\mathcal{L}_{\mathcal{A}}$ (*convergence*).

A *legitimate configuration* for the Steiner Tree is a configuration where an unique tree T spanning S is computed. Additionally, we expect a competitiveness of $\log(z)$, i.e., $\frac{W(T)}{W(T^*)} \leq \log(z)$, with $|S| = z$ and T^* an optimal Steiner tree.

In the following we propose a self-stabilizing Steiner tree algorithm. We discuss the extension of the algorithm to fully dynamic settings (the add or removal of members, nodes or links join or leave). This second extension offers no guarantees during the restabilization period.

[†] A legitimacy predicate is defined over the configurations of a system and is an indicator of its correct behavior.

3 A self-stabilizing approximation of a Steiner tree

This section describes a self-stabilizing algorithm for the Steiner tree problem. It implements the technique proposed by Imase and Waxman [IW91], in a self-stabilizing manner. That is, each Steiner member is connected to the existing Steiner tree via a shortest path. Note that in a self-stabilizing setting the initial configuration may be arbitrary, hence nodes have to perpetually verify the coherency of their state. We consider there is a special member node that acts as the root of the Steiner tree. To this end, we assume a leader oracle that returns to every node : leader or follower. Several implementations fault-tolerant and self-stabilizing can be found in [DGDF07] and extension to dynamic settings in [PB08] for example.

In the following, we give only a high level description of our algorithm due to lack of space, an extended abstract can be found in [BPBR09].

3.1 Description of the algorithm

Every node $v \in V$ sends periodically its local variables to each of its neighbors using `InfoMsg` messages. Upon the reception of this message a neighbor updates the local copy of its neighbor variables.

Our algorithm is a four phase computation : (1) first nodes update their distance to the existing Steiner tree, then (2) nodes request connection (if they are members or they received a connection demand), then (3) they establish the connection, and finally (4) they update the state of the current Steiner tree. These phases have to be performed in the given order. Note that if a node detects a distance modification in its neighborhood, it can change its connection to the current tree. Therefore a node before computing any other action must update its distance to the current tree.

Every node in the network, maintains a parent link stored in variable parent which is one of its neighbors having the shortest distance to the current tree stored in variable dist. To break cycles due to erroneous initial configurations, we use the notion of tree level, defined by the variable level. Each node has three boolean variables, need to ask a connection to the tree, connected to notice if it is connected or not and connect_pt to signal the node is a connection point used to maintain distances in the Steiner tree. A connection point is a connected node which is a member or has more than one connected child. An extra boolean variable member notices to a node if it belongs to the Steiner group (this variable is only read by the algorithm).

Root node : In a coherent state the root has a distance and a level equal to zero, variables need, connected and connect_pt are *true* since the root is always a connected member (it always belongs to the Steiner tree). Whenever the state of the root is incoherent the root corrects its state as described above.

Distance update : To compute its distance, an unconnected node compares the minimum edge weight with a connected neighbor and the minimum distance with an unconnected neighbor, and takes the minimum among the two values. If an unconnected node detects it has a better shortest path then it updates its distance and changes its other variables accordingly (connected, connect_pt to *false* and level to parent level plus one). The same rule is used to reinitiate the state of a node if it has an incoherent parent. To update its distance, a connected node must have coherent variables with its connected parent. Its distance is computed as for an unconnected node but it compares its distance with local distances towards connection points and takes the minimum. Remark : Since every node computes its shortest path to the Steiner tree (at least to the root), the node's parent relation defines a spanning tree of G.

Member connection : Variable need is set to *true* by a node to ask to its parent a connection to the current Steiner tree. Each member always have need to *true*. An unconnected node which detects a connection request sets its variable need to *true*. This connection request is forwarded in the spanning tree until a connected node is reached. An unconnected node sets its variable need to *false* if it is not a member and it has no connection request from a child.

When an unconnected neighbor of a connected node detects a connection request from a child, an acknowledgment is sent backward using variable connected, by setting it to *true*. Therefore every unconnected node on the request path sets connected to *true* until the member that requested the connection is connected. Only a node that has (1) no better path, (2) its variable need $=$ *true* and (3) a connected parent can set connected to *true*. A connected node becomes unconnected (i.e., sets connected to *false*) if its variables are not coherent with its parent.

Update the Steiner tree : Thanks to connection points (i.e., the root of the branch connecting a member) and distance computation, we maintain a shortest path between a member and the Steiner tree in order to respect the construction in [IW91]. A connected node with coherent variables with its parent sets its variable connect_pt to *true* if it is a member or has more than one connected children,*false* otherwise. This variable allows to maintains shortest paths from members to the current Steiner tree, since every connected node with a better path updates its distance.

Topology change Whenever a removal topology change event e (i.e., member, node or edge removal) occurs, a connected node whose parent is affected by e must be disconnected. So the node sets connected to *false* and dist to infinity and waits until all its subtree is disconnected (i.e., it has no connected children), before beginning a new connection.

In case of an add topology change event e (i.e., member, node or edge add) occurs, since the algorithm is based on shortest paths computation eventually the algorithm leads in a finite number of steps to a legitimate configuration. However, during the algorithm stabilization no structural guarantee is provided.

Due to lack of space, we only give below the main properties of the proposed algorithm, the proof of its correctness and its complexity can be found in [BPBR09].

Lemma 1 *Starting from any configuration (legitimate or illegitimate), eventually the algorithm reaches in a finite time a legitimate configuration describing a Steiner tree.*

Lemma 2 *Starting from a legitimate configuration, if a member leaves the member set S or a node or edge is removed from the network then parent relations can be modified for nodes in the subtree connected via the removed member, edge or node, but parent relations are preserved for any other node in the tree.*

Lemma 3 *The self-stabilizing algorithm returns a Steiner tree with a competitiveness of* $\log|S|$ *in* $O(D\cdot|S|)$ *rounds and using* $O(\Delta\log n)$ *bits per node, where S is the Steiner group, and n, D and* Δ *are respectively the number of nodes, the diameter and the maximal degree of the current network.*

Références

[BK07] Janna Burman and Shay Kutten. Time optimal asynchronous self-stabilizing spanning tree. In *DISC*, pages 92–107, 2007.

[BPBR09] Lélia Blin, Maria Gradinariu Potop-Butucaru, and Stephane Rovedakis. A superstabilizing $\log(n)$-approximation algorithm for dynamic steiner trees. Technical Report hal-00363003, HAL, February 2009.

[DGDF07] Carole Delporte-Gallet, Stéphane Devismes, and Hugues Fauconnier. Robust stabilizing leader election. In *SSS*, pages 219–233, 2007.

[Dol00] Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.

[GRG05] Luca Gatani, Giuseppe Lo Re, and Salvatore Gaglio. A dynamic distributed algorithm for multicast path setup. In *Euro-Par*, pages 595–605, 2005.

[IW91] Makoto Imase and Bernard M. Waxman. Dynamic steiner tree problem. *SIAM J. Discrete Math.*, 4(3) :369–384, 1991.

[KK02a] Sayaka Kamei and Hirotsugu Kakugawa. A self-stabilizing algorithm for the steiner tree problem. In *SRDS*, pages 396–, 2002.

[KK02b] Sayaka Kamei and Hirotsugu Kakugawa. A self-stabilizing algorithm for the steiner tree problem. *IEICE TRANSACTIONS on Information and System*, E87-D(2) :299–307, 2002.

[PB08] Sara Tucci Piergiovanni and Roberto Baldoni. Brief announcement : Eventual leader election in the infinite arrival message-passing system model. In *DISC*, pages 518–519, 2008.

[Win87] Pawel Winter. Steiner problem in networks : a survey. *Networks*, 17(2) :129–167, 1987.

[WWW86] Ying-Fung Wu, Peter Widmayer, and Chak-Kuen Wong. A faster approximation algorithm for the steiner problem in graphs. *Acta Inf.*, 23(2) :223–229, 1986.