Brief Announcement: The Dynamic Steiner Tree Problem: Definitions, Complexity, Algorithms

Stefan Balev, Yoann Pigné, Éric Sanlaville, Mathilde Vernet

Brief Announcement: The Dynamic Steiner Tree Problem: Definitions, Complexity, Algorithms

Stefan Balev
Université Le Havre Normandie, Univ Rouen Normandie, INSA Rouen Normandie, Normandie Univ, LITIS UR 4108, F-76600 Le Havre, France

Yoann Pigné
Université Le Havre Normandie, Univ Rouen Normandie, INSA Rouen Normandie, Normandie Univ, LITIS UR 4108, F-76600 Le Havre, France

Éric Sanlaville[1]
Université Le Havre Normandie, Univ Rouen Normandie, INSA Rouen Normandie, Normandie Univ, LITIS UR 4108, F-76600 Le Havre, France

Mathilde Vernet
LIA, Avignon Université, Avignon, France

Abstract

This note introduces an extension of the Steiner tree problem applied to dynamic graphs. It discusses its interest, studies its complexity and proposes an algorithm tested on generated and real data.

2012 ACM Subject Classification Mathematics of computing → Paths and connectivity problems

Keywords and phrases Steiner Tree, Dynamic Graph, Complexity, experimental study

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.24

Funding This work was Supported by the French ANR, project ANR-22-CE48-0001 (TEMPOGRAL)

1 Introduction and contributions of this work

The goal of the classical Steiner tree problem in graphs is to maintain connectivity among selected vertices, called terminals, while minimizing the cost associated with the chosen edges or vertices in the process. The problem has practical applications in many domains, such as communication networks, social networks, and logistics networks. Although it is simple to formulate, it was proven to be NP-hard very early on. It remains NP-hard in most of its simple versions, even with unit costs on the edges, where the objective is to minimize the number of vertices needed to connect the terminal vertices. However, there exist cases for which the Steiner problem is easy, as it can be solved using polynomial algorithms. The first case is when the number of terminals is two, as it then reduces to a shortest path problem. The second case is when every vertex of the graph is a terminal, as it then reduces to a spanning tree problem.

In the application domains cited above, the underlying graph could change over time. For some years now, a growing literature has been considering the well-known classical combinatorial problems in this new setting. Recalling all papers dealing with dynamic graphs (the name may vary) is out of the scope of this short paper, but one may cite, as the most relevant for this work, [8] for paths and their extensions, [2, 6] for some considerations about connectivity issues, ... When time is considered as a discrete variable, such a graph is basically constituted of an ordered sequence of graphs indexed by time: $G = (G_i),\ i \in \mathcal{T}$.

[1] corresponding author

This is the way it will be considered throughout the paper. The term of dynamic graph is preferred because, as we shall see, it is not mandatory to know the graph evolution in advance to find a *Steiner set* (the set of vertices used to connect the terminals).

Connectivity in a temporal setting may have different meanings. A static Steiner tree is the less costly way to ensure connectivity between a subset of terminal nodes. We are looking for a structure that maintains connectivity whereas the graph is changing. Basically, connectivity here may be defined in two ways. In the first way, one may wish to maintain some "instantaneous" or path-based connectivity: at each time step, a path exists between each pair of terminals. Conversely, in the second way one looks for the existence of a journey, also called temporal path, from each terminal to any other terminal. This journey-based connectivity ensures that some information, or some good, will eventually be transferred. This is a one shot property: typically, after a given time, no journey may exist for a couple of terminals and the transfer is no more possible. *We claim that instantaneous connectivity, is better adapted to some situations*, like a set of mobile robots that cooperate for a given common goal, thus needing frequent communications between some distinguished nodes of the network. Note that to the best of our knowledge, no previous work considers direct extensions of Steiner trees. However, many works exist that study the existence of journeys [8]. Many works also study some journey-based extensions of spanning trees, namely spanners, see [1, 3]. The path-based extensions of spanning tree are of less interest, as it may consist either in computing the minimum spanning tree at each snapshot (without building any persistent structure), or in computing the spanning tree of the intersection of all snapshots (using only edges constantly present).

The main contributions of this paper are:

- We discuss how to extend the Steiner tree problem to dynamic graphs. Among the possible extensions, we identify the more relevant one for practical applications.
- We show that unlike its static counterpart, the dynamic Steiner problem is NP-hard even with two terminals.
- We propose an exact algorithm that computes all Steiner sets of a given size, study its complexity and test it experimentally on generated and real-word instances.

2 The Dynamic Steiner Set problem

Let us first recall the static problem definition. Let $G = (V, E)$ be a (static) graph. A non-negative weight w_e is associated to each edge $e \in E$. For a given subset of vertices $S \subset V$, called *terminals*, the objective is to find a tree of minimum weight covering all vertices of S. Note that the simpler version with unit cost is already NP-Hard. It reduces to find a Steiner set of minimal cardinality.

Let us see now how the Steiner Tree Problem can be extended to dynamic graphs. We consider here only the unit cost version, or minimal cardinality version. Let G be a undirected dynamic graph such that $G = (V, E, T) = (G_1, \ldots, G_T)$ with $G_i = (V, E_i)$ and $E = \bigcup_i E_i$. The successive G_is are called *snapshots* of G at each time step. As in the static case, one may introduce a subset S of special vertices, called terminals. The goal is still to ensure connectivity between the terminals. Note that we limit our study to the case where no travel time is associated to the edges. Hence there is no travel time associated to paths either: the (instantaneous) connectivity requirement applies to each time step.

The most straightforward way to extend the Steiner Tree problem to dynamic graphs is of course to compute, at each time step, the Steiner tree associated to S. This extension has one major drawback: at each time step, the Steiner set is different. It is not really convenient,

for instance in a communication network, to change the intermediate nodes at each time step. More importantly, we do not make any use of the knowledge of the dynamic graph as a whole, considering each G_i separately. Note also that this approach is very time consuming, as a complete Steiner tree is recomputed at each time step. Hence in the remaining of the section, we focus on computing a Steiner set that is fixed during all the lifetime of the graph.

▶ **Definition 1** (Fully Connected Steiner Set). *For a given vertex set $S \subset V$ of terminals, find V' with $S \subset V' \subset V$ and $E'_i \subset E_i$ $\forall i \leq T$ such that:*

- *$G'_i = (V', E'_i)$ is a connected graph $\forall i \leq T$*
- *$|V'|$ is minimum*

We look for a subset V' of vertices containing S, such that the subgraph of G_i generated by V' is always connected, and V' has minimal cardinality (which is equivalent to minimize the total number of edges used to connect V'). But in fact the condition verified by each G'_i is stronger than we need to keep the terminals connected: the definition imposes the connectivity of *all* vertices in the Steiner set. The following definition relaxes this condition.

▶ **Definition 2** (Partially Connected Steiner Set). *For a given vertex set $S \subset V$ of terminals, find V' with $S \subset V' \subset V$ and $E'_i \subset E_i$ $\forall i \leq T$ such that:*

- *All vertices of S are part of the same connected component in the static graph $G'_i = (V', E'_i)$*
- *$|V'|$ is minimum*

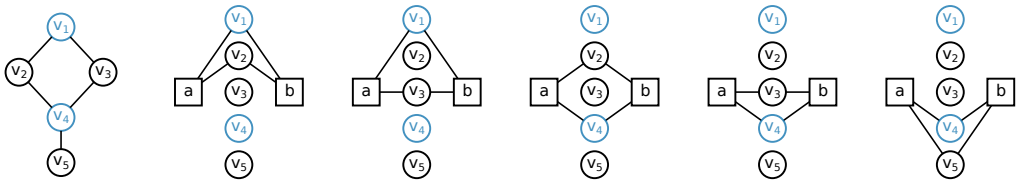
The (partially connected) Steiner set V' is the subset of nodes that keep the terminals connected for the whole lifetime of the graph, at a minimum cost while V' itself is not necessarily connected. Note that when $T = 1$, this problem is the Steiner Tree problem. However, we cannot deduce an optimal solution from the solution of static Steiner Problem at each time step.

To illustrate the difference between the two definitions, consider the simple dynamic graph with two time steps, where vertices $v_1 - v_2 - \cdots - v_n$ form a path on both snapshots, and two terminals a and b. At the first time step a and b are connected to v_1 and at the second step they are both connected to v_n. If we want to keep the Steiner set fully connected, we have to take all the vertices ($k = n + 2$), but to keep the terminals connected, we only need v_1 and v_n ($k = 4$). Consider now the same setting but without the edges (v_i, v_{i+1}). There is no fully connected Steiner set but a, b, v_1 and v_n still form a partially connected Steiner set.

We now retain definition 2 and study the complexity of the Dynamic Minimum Steiner Set (DMSS) problem which consists of finding a (partially connected) Steiner Set of minimum cardinality. Let us recall that the minimum cardinality version is NP-hard in the static case, even for bipartite or for chordal graphs [7]. On the other hand, one might consider the case of a small number of terminals. Even with arbitrary costs, the problem with two terminals is easy in the static case as the Steiner tree reduces to a single path.

▶ **Theorem 3.** *The DMSS problem is NP-hard even with two terminals.*

The idea of the proof is to transform Vertex Cover (VC) a well known NP-complete problem to DMSS with two terminals. Let $G = (V, E)$ be the graph of the VC instance. The dynamic graph G^{DYN} has the same vertices plus two terminals a and b. For each $(u, v) \in E$ there is a time step where the terminals are connected to u and v. A vertex cover in G corresponds to a Steiner set in G^{DYN}. An example is given in Fig. 1.



(a) G **(b)** G_1^{DYN} **(c)** G_2^{DYN} **(d)** G_3^{DYN} **(e)** G_4^{DYN} **(f)** G_5^{DYN}

Figure 1 Transforming a VC instance G with 5 edges to a *DMSS* instance with 5 time steps.

3 Algorithmic issues and experiments

This section focuses on efficiently solving the DMSS problem. We only consider the problem of finding a Steiner set of a given cardinality. Throughout this section, s is the number of terminals, k is the cardinality of the Steiner sets, and $k' = k - s$ is the number of vertices to add to the terminal vertices.

3.1 Basic ideas

For a given time step i, there might be a very large number of vertex sets of cardinality k' that allow the terminals to be connected. These sets will be called *candidates* for time step i. An obvious upper bound of this number is the number of vertex subsets of cardinality k' among $V - S$, that is $\binom{n-s}{k'}$. The dynamic Steiner sets we are looking for belong to this set but, hopefully, are much less numerous.

A Steiner set of cardinality k must by definition connect the terminals for each time step. Therefore, the algorithmic possibilities are straightforward. Let us discard the naive idea that consists in computing independently all candidate subsets for all time steps, and then computing their intersection. It is much more efficient to use an iterative process: suppose we have a set of candidates PSS for time steps $1, \ldots, i$. At iteration $i+1$, only the solutions in PSS which are also candidates for time step $i+1$ are kept. So the core procedure of the algorithm is a search on small subgraphs generated by the candidate sets. Its complexity is $\mathcal{O}(m')$ where m' the number of edges of the subgraph is bounded by k^2. The process starts with all candidates for $i = 1$. It ends when $i = T$ with all Steiner sets of cardinality k or when no more candidates exist. Of course, this algorithm is still a brute force algorithm as some enumerations have to be done.

▶ **Theorem 4.** *There exists an algorithm that enumerates all solutions of DMSS, of complexity* $\Theta(T \times (n-s)^{k'} \times \frac{k^2}{(k')!})$. *It follows that DMSS is in XP class relatively to parameter* k'. *When* k *and* s *are fixed parameters, the complexity is* $\Theta(n^{k-s})$.

The algorithm sketched above might be used on-line: at each time step θ, all vertex sets that are Steiner sets for time interval $[0, \theta]$, are computed.

3.2 Experimental study

We performed an experimental study of our algorithm, both on randomly generated graphs and on real-world networks. The virtual machines used for this experiment have 64 GB of RAM.

We generated dynamic graphs using a method previously presented in [6] to test connected component computation. First we generate the underlying graph, then we add dynamicity to the edges using a Markovian process, see [4].

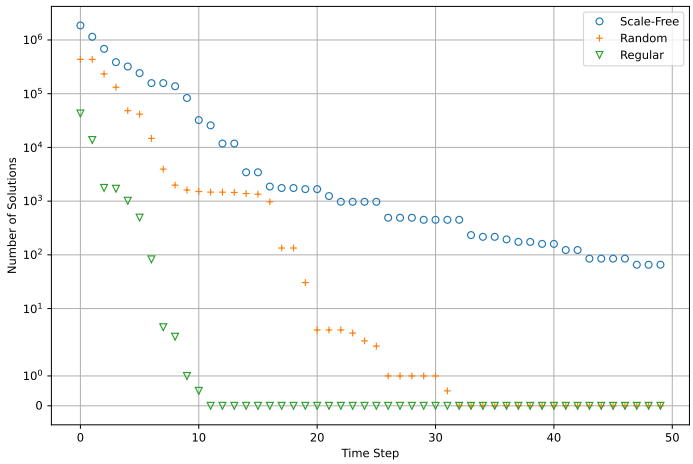


Figure 2 Median number of Steiner sets with regard to the number of time steps, with $s = 3$.

The underlying graph is generated using generators from the GraphStream library[2]. We tested three different types of graphs that present specific features. They differentiate mostly according to their clustering coefficient.

Experiments were run on randomly generated graphs up to 100 vertices, 50 time steps and up to 6 terminals. Results show the algorithm can solve these instances usually in less than an hour. The evolution of the number of solutions, hence of the computation time, heavily depends on the type of underlying graph, as shown in figure 2.

As a real-word instance, we used the CRAWDAD VT/MANIAC dataset [5], accessible through the IEEE Dataport comes from the CRAWDAD collection. This dataset encompasses routing and topology traces gathered during the Mobile Ad hoc Networks Interoperability And Cooperation (MANIAC) Challenges that took place in 2007 and 2009 in conjunction with the IEEE Globecom IEEE and PerCom conferences. These traces provide insights into the communication patterns, node mobility, and network structure characteristic of MANETs (Mobile Adhoc Networks) in real-world scenarios.

The resulting dynamic graph has 14 nodes, 74 edges (for the footprint), and a high number 1244 of timesteps. An edge remains present during less than 10% of the time horizon on average. The algorithm runs very fast on this instance (a few seconds) to provide all possible Steiner sets for all possible values of k.

References

1 Kyriakos Axiotis and Dimitris Fotakis. On the size and the approximability of minimum temporally connected subgraphs. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, 2016. arXiv preprint arXiv:1602.06411.

2 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

3 Arnaud Casteigts, Joseph G Peters, and Jason Schoeters. Temporal cliques admit sparse spanners. *Journal of Computer and System Sciences*, 121:1–17, 2021.

[2] `http://graphstream-project.org`

4 Andrea EF Clementi, Claudio Macci, Angelo Monti, Francesco Pasquale, and Riccardo Silvestri. Flooding time in edge-markovian dynamic graphs. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 213–222, 2008.

5 Amr Hilal, Jawwad N Chattha, Vivek Srivastava, Michael S Thompson, Allen B MacKenzie, Luiz A DaSilva, and Pallavi Saraswati. Crawdad vt/maniac, 2022. doi:10.15783/C7WG6T.

6 Mathilde Vernet, Yoann Pigne, and Eric Sanlaville. A study of connectivity on dynamic graphs: computing persistent connected components. *4OR*, 21(2):205–233, 2023.

7 Kevin White, Martin Farber, and William Pulleyblank. Steiner trees, connected domination and strongly chordal graphs. *Networks*, 15(1):109–124, 1985.

8 B Bui Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003.