The Dynamic Steiner Tree problem: definitions, complexity, algorithms

Stefan Balev, Yoann Pigné, Eric Sanlaville, Mathilde Vernet

The Dynamic Steiner Tree problem: definitions, complexity, algorithms

Stefan Balev

Yoann Pigné

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 paper introduces an extension of the Steiner tree problem applied to dynamic graphs. In various application domains of this problem, the associated graphs undergo temporal changes, such as variations in the edge set or associated costs. The paper presents three extension models, and we opt for the one demonstrating the most desirable features: a fixed Steiner set (a set of selected vertices) maintained throughout the time horizon and of minimum cardinality, ensuring connectivity among the terminal vertices.

We show that the resulting problem of minimizing the size of the Steiner set is NP-hard, even when dealing with only two terminals and a lifetime of 2 time steps. This contrasts with the corresponding static problem, which is among the rare polynomial versions of the Steiner tree problem. However, polynomial algorithms exist when the size of the Steiner set is bounded. An algorithm is designed, analyzed and tested on specially generated and on real data sets. We show that it solves exactly non trivial instances when the Steiner set size in bounded.

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

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

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

1 Introduction and contributions of this work

The Steiner tree problem is one of the most studied combinatorial problems on graphs. Roughly speaking, the goal is to preserve the connectivity among some selected vertices, called terminals, minimizing the cost associated to the chosen edges, or vertices, in the process. This kind of problems occurs in many application domains, as in communication networks, sensor networks, social networks, logistic networks, etc.

Although it is simple to formulate, there is no easy way to solve it as it was very early proven to be NP-hard [15]. Furthermore, the problem remains NP-hard even in most of its simple versions. Indeed, it is the case even with unitary costs on the edges[12], that is, when the objective is to minimize the number of vertices to connect the terminal vertices. In the following, the vertex set used to connect the terminals, including the terminal vertices themselves, will be called *the Steiner set*.

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

[1] corresponding author

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 may change over time. This is undoubtedly the case for social networksas interactions among people have a limited lifespan. This is also true for some communication networks (for instance ad-hoc networks or mobile networks). And also for logistic networks, even if the dynamic is much slower, because of works on the road or congestion phenomena for instance. 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 paper, but one may cite, as the most relevant for this work, [22, 17] for paths and their extensions like hamiltonian cycles, [11, 18, 19] for flows, [6, 20] for some considerations about connectivity issues, ... Observe that there are different ways (and names) to define a dynamic graph (temporal graphs, dynamic or temporal networks, or time evolving graphs). Still, and when time is considered as a discrete variable, such a graph is basically constituted of an ordered sequence of graphs indexed by time: $\mathcal{G} = (G_i)_{i \in \mathcal{T}}$. 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).

Designing the temporal counterpart of a given optimization problem on static graphs may not always be easy. This is specially the case when the problem involves connectivity issues, as connectivity in a temporal setting may have different meanings. A Steiner tree is the cheapest way to ensure connectivity between a subset of nodes called terminals. In a temporal setting, 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 simply looks for the existence of a journey, also called temporal path, from each terminal to any other terminal. This journey-based connectivity ensures some information, or some good, will eventually be transferred. But 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. Conversely, in the second way one may wish to maintain some "instantaneous" connectivity: at each time, a path exists between each pair of terminals. *We claim that instantaneous, or path-based connectivity (see [20]) , may be 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. Many works exist that study the existence of journeys, see for instance the seminal work of [22]. Many works also study some journey-based extensions of connected components, see [4, 3], or of spanning trees, namely spanners, see [2, 7]. 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). Note that to the best of our knowledge, no previous work considers the solving of direct extensions of Steiner trees. Still, the recent work of Klobas et al [16] introduced a journey-based temporal version : when does it exist a journey between all terminal vertices? However, their paper studies the dual problem of finding a minimum temporal labelling allowing the existence of a temporal Steiner tree. Although this problem is NP-Hard, The number of labels is bounded by $2n - 4$, and these labels may be assigned to a tree whose leafs are the terminals, plus one edge. When the temporal labelling is given, we shall see that the number of edges (and of vertices) to ensure instantaneous connectivity among the terminals may be much larger.

This paper first investigates the temporal counterpart of Steiner tree problem when the

goal is to maintain instantaneous connectivity at minimal cost. Several possible extensions of the Steiner Tree problem to the case of dynamic graphs are presented. Note that the study is restricted to the case of unit costs. The obvious extension is to compute the Steiner trees for all time steps. Its drawbacks are that one must keep a different set of intermediate vertices at each time step, which is not of practical use; and it is computationally intensive. The other proposed extensions build a *Steiner set*, that is, a subset of vertices that will guarantee the connectivity of the terminal vertices throughout the lifetime of the graph. The second extension builds at each time step a spanning tree whose vertices are the terminals plus all the vertices of the Steiner set. We show that this problem cannot be reduced to successive solving of classical Steiner tree problems. However, the drawback of this model is that it retains too many edges that are not necessary at each time step to keep the terminal vertices connected. Therefore, the total cost might exceed by far the cost for the other models. *Our last extension minimizes the size of the Steiner set.* Note that the sum of the costs of the edges used at each time step might be used as a second criterion in case of arbitrary costs. This definition is more suitable for the applications, as the obtained solutions will be less resource consuming in terms of vertices and edges used, and is used in the remaining of the paper. However, we simply prove that *the associated decision problem is NP-Complete even when the costs are not considered, and there are only two terminals.* A more intricated proof shows it remains the case even with two time steps.

This calls for the design of exact algorithms that should be efficient for Steiner sets of small size, or of approximation algorithms, that will be designed specifically for the dynamic setting. *This paper proposes an exact algorithm* that computes all Steiner sets of a given size, studies its complexity, and provides some experimental tests. The algorithm might be used on-line: at each time step, all vertex sets that are so far Steiner sets, are computed. It might be worth to discuss here about the distinction between temporal or dynamic graphs and *multi-layered graphs*. Multi-layered graphs have been studied in a number of papers, see for instance [5] where several kinds of subgraph searches (including connected components) are investigated. They are similar to temporal graphs in the sense that they can also be defined as a collection of static graphs with an identical set of vertices. However, these static graphs are not ordered, contrary to the snapshots of temporal graphs. As we shall see in the next section, definitions of Steiner problems based on maintaining instantaneous connectivity do not need to consider some ordering of the snapshots. However, it is interesting to solve these problems through on-line algorithms, so that at each time step, the collection of current Steiner sets is given. That is why we introduced these extensions of the Steiner problems through the setting of dynamic graphs (to our knowledge, there is no paper generalizing the Steiner problem to multi-layered graphs).

Experiments with different randomly generated dynamic graphs show that *our algorithm is able to solve medium size instances.* The number of solutions is highly dependent on the type of graphs. The results confirm that Steiner sets are much more numerous on scale-free graphs. The algorithm is also applied to a dynamic graph obtained from real data about mobile ad hoc networks. The graph is small, but the time horizon is large. *The algorithm is able to find rapidly all Steiner sets, for different parameter values.*

The main contributions of this paper are therefore:

- We discuss how to extend the Steiner tree problem to dynamic graphs. Among the possible extensions, we identify the more relevant one.
- 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 settings. 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. If the case of unit costs, It is of course equivalent to minimize the number of vertices of the tree.

Let us see now how the Steiner Tree Problem can be extended to dynamic graphs. Let $\mathcal{G}$ be an non-directed dynamic graph such that $\mathcal{G} = (G_1, \ldots, G_L)$, with $G_i = (V, E_i)$ $\forall 1 \leq i \leq L$ and $E = \bigcup_i E_i$. The successive G_is are called *snapshots* of $\mathcal{G}$ at each time step. As in the static case, let us consider 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.

2.1 Direct extension: allowing a varying Steiner set

The direct way to extend the Steiner Tree problem to such 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 deeply, 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 time of study.

2.2 Second model: Fully Connected Minimum Steiner Set

2.2.1 Definition

▶ **Definition 1** (Fully Connected Minimum Steiner Set)**.** *Let us consider some dynamic graph $\mathcal{G}$. For a given vertex set $S \subset V$ of terminals, find $E'_i \subset E_i$ $\forall i \leq L$ and V' with $S \subset V' \subset V$ such that:*

- *$G'_i = (V', E'_i)$ is a connected graph $\forall i \leq L$*
- *$|V'|$ is minimum*

We look for a minimum subset V' of V containing S, such that the subgraphs of all snapshots induced by V' are connected (subgraph of G induced by X : subgraph containing X and all edges linking vertices of X, denoted $G[X]$).

The subset V', still called the Steiner set, is a subset of nodes that keep the terminals connected. But in fact the condition verified by each G'_i is stronger: it keeps the connectivity between all vertices of V', not only the terminals.

2.2.2 Remarks

Several elements are important to notice regarding the problem given by Definition 1.

1. When $T = 1$, this problem is the Steiner Tree problem.

2. Enforcing the subset E'_i to be the same at each time step makes this problem solvable as a static problem by intersecting all G_i
3. The vertices from the optimal Steiner tree of S at a given time step do not necessarily belong to the optimal solution.
4. The union of the optimal Steiner trees at each time step is not optimal.
5. The intersection of the Steiner trees obtained by the solution of this problem at each time step is not a connected graph.
6. Choosing $V' = V$ is always a solution, but the worst possible in terms of cardinality.

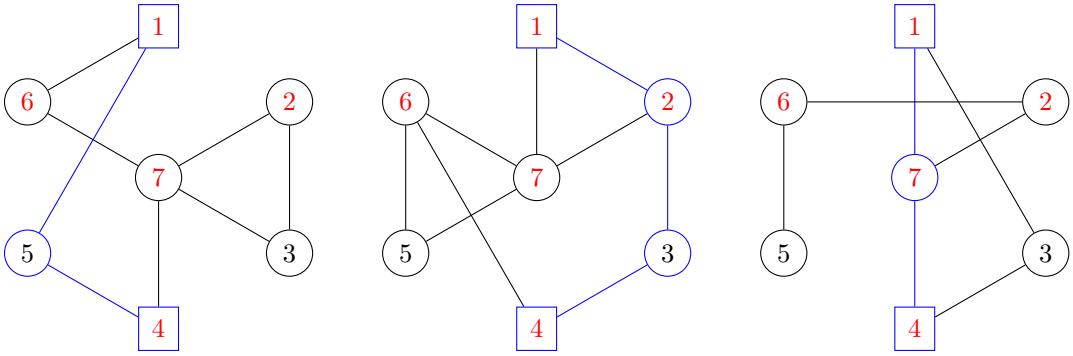


Figure 1 A dynamic graph with 7 vertices and 3 time steps. The graph is represented at each time step.

2.2.3 Example

Figure 1 presents an example of the evolution of a dynamic graph on 3 time steps.

Rectangular vertices (vertices 1 and 4) correspond to the terminal subset S. Vertices with identification in red (vertices 1, 2, 4, 6 and 7) correspond to the subset V' of the optimal solution of the Fully Connected Steiner Set Problem (Definition 1).

Blue circled vertices and blue edges are the optimal Steiner trees for the subset S at each time step. Their size are 3, 4 and 3, respectively. The set $\{1, 2, 3, 4, 5, 7\}$ resulting from their union is not connected for the thierd snapshot.

The Steiner set for this example is $V' = \{1, 2, 4, 6, 7\}$, its cardinality is 5.

This example illustrates the remarks presented in section 2.2.2. The blue vertex set and the red vertex set are not equivalent, which is the point of remark 3. Intersecting the Steiner trees at each time step (in other words, intersecting blue edges and vertices) gives an independent set in this example. Therefore, no solution can be extracted from this, which is the point of remark 5. The union of the optimal Steiner Sets at each time step (remark 4) is not always connected in this specific example, and as stated by remark 6, taking all vertices gives of course a solution of value 7 instead of 5.

2.2.4 Issues

This *Fully Connected Steiner Set* problem cannot be simply reduced to the computation of a Steiner tree on some static graph built from the dynamic graph. Therefore, it deserves a specific study.

Unfortunately, the full connectivity constraint on each G'_i is rather artificial with regard to applications. Remember the initial goal of building Steiner sets is to keep only the terminals

connected. One could wonder why enforce the connectivity of the whole Steiner Set as long as the terminals are connected. An optimal solution of this problem might keep some edges and vertices that will not be used to connect the terminals at each snapshot, *thus significantly increasing the overall cost* (this remains true if costs are added to edges). This is the case in the example presented before. For instance, 4 edges are necessary to ensure connectivity at each snapshot. But during the first and third time steps, only two edges are necessary to connect the two terminals.

Another issue is the difficulty to solve the problem in practice. Note however that the problem is polynomial when the size of the searched Steiner set is bounded. Indeed, for a fixed maximum size of the Steiner set, there is a polynomial number of sets to test (if we consider the terminals to be part of the Steiner Set, the number of terminals does not matter). And this test consists in searching for the L minimum spanning trees associated with the induced subgraphs. First we should enumerate all connected subsets W of V containing S and find the minimum spanning tree of the graph induced by W at each time step. The optimal solution is the subset W for which the sum of the weights at each time step is minimum. This method is obviously not applicable is practice as their might be as much as 2^{n-k} sets W to check on a graph with n vertices and with S having k vertices.

2.3 Third model: The terminal set is connected, not the Steiner set

2.3.1 Definition

From the previous extension of the Steiner Problem, one major remark was that it seemed artificial to keep the whole Steiner set connected because the objective is to connect the terminals. This is how we came to another possible definition to extend the Steiner problem to dynamic graphs: the Partially Connected Steiner Set.

▶ **Definition 2** (Partially Connected Minimum Steiner Set). *Let us consider some dynamic graph $\mathcal{G}$. For a given vertex set $S \subset V$ of terminals, find V' with $S \subset V' \subset V$, and $E'_i \subset E_i \; \forall i \leq L$ such that:*

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

The (partially connected) Steiner set V' is the subset of nodes that keep the terminals connected for the whole time interval of study of the graph, at a minimum cost. V' itself is not necessarily connected. This is an answer to a drawback of the previous definition.

2.3.2 Remarks and complexity issues

Most remarks from the previous definition remain valid. Again, when $T = 1$, this problem is the Steiner Tree problem. We still cannot deduce an optimal solution from the solution of static Steiner Problem at each time step. However, the problem does not directly extend to the case of weights on the edges. Indeed, if one minimizes the sum of weights of the Steiner trees chosen at each snapshot, then the optimal solution is simply to take $V' = V$ and the optimal Steiner trees at each snapshot. It is then equivalent to the direct extension given in Section 2.1.

Clearly, there is a compromise to accept between the size of the Steiner set and the overal cost of the chosen edges. Anyway, it seems natural to first compute the minimum size of a Steiner tree, and then to compute the solutions with minimum cost for a *fixed* value of the cardinality of V'.

Finding the (partially connected) Dynamic Minimum Steiner Set, denoted *DMSS*, is NP-hard in the static case, even for bipartite[12] or for chordal graphs[21]. 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.

In figure 1, there are two terminals. It is easy to see that $\{6, 7\} \cup \{1, 4\}$ is the Steiner set of minimum cardinality 4 Remember for the fully connected Steiner set variant, the optimal value is 5 for the same example. Note that it is easy to build examples for which the optimal cost for the second model is arbitrary larger than the optimal cost for the third model. In the next section, we consider the case of the Partially Connected Minimum Steiner tree, with two terminals, and we show that this problem is NP-hard. It is noted *DMSS2* for (Partially Connected) Dynamic Minimum Steiner Set with 2 terminals.

3 Partially Connected Dynamic Minimum Steiner Set with 2 Terminals

3.1 Definition

▶ **Definition 3** (Partially Connected Minimum Steiner Set with 2 Terminals)**.** *Let us consider some dynamic graph $\mathcal{G}$. For a given vertex set $S \subset V$ of terminals, with $|S| = 2$, find V' with $S \subset V' \subset V$, and $E'_i \subset E_i \; \forall i \leq L$ such that:*

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

3.2 Remarks

This problem is very easy when $T = 1$ as this is a shortest path problem in a graph where all edges have weight 1. But it is not so easy in the dynamic context, and some extreme examples can be given.

Figure 2 presents a dynamic graph on 4 time steps. The terminal vertices are the squared ones. They are directly connected by an edge at each time step, therefore no extra vertex is necessary to connect them.

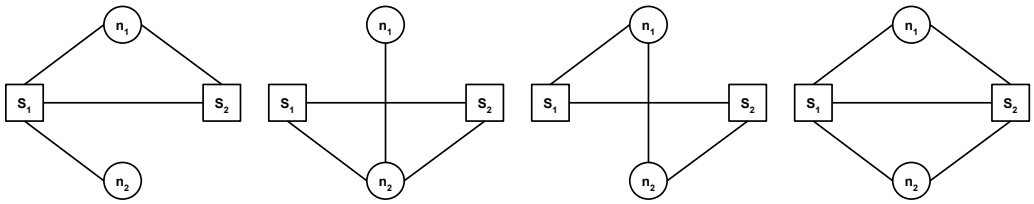
Figure 3 presents a dynamic graph on n time steps. In each G_i, there is exactly one path of length 2 between the two terminals going through a different vertex at each time step. In this case, every vertex of the graph must be in the Steiner Set to keep the terminals connected.

Figure 4 presents a dynamic graph on 4 time steps. At each time step, the terminals are connected by two distinct paths: one of length 3 and one of length at most 2. If we consider the vertices on the shortest path at each time step for the Steiner Set, we need 3 vertices to keep the terminals connected. But if we connect the terminals at each time step by a longer path (one of length 3), we only need 2 extra vertices for the Steiner set. This example also shows that the shortest path at each time step does not help getting an optimal solution, and that intersecting the graphs G_i does not help either. Indeed, the path of length 3 present at each time step goes through the same two vertices but not the same edges.

Even though this problem is easy to describe, it is difficult to solve. This is what we prove in Section 3.3.

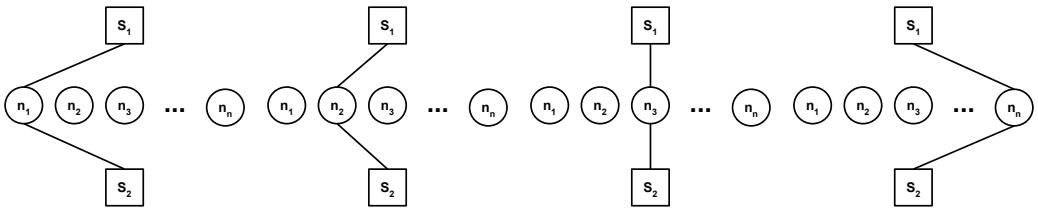
3.3 NP-Completeness for DMSS2 decision problem

Let us consider the associated decision problem. Note that for simplicity, we introduce the set V_s as the considered Steiner set minus the two terminals, denoted by a and b.



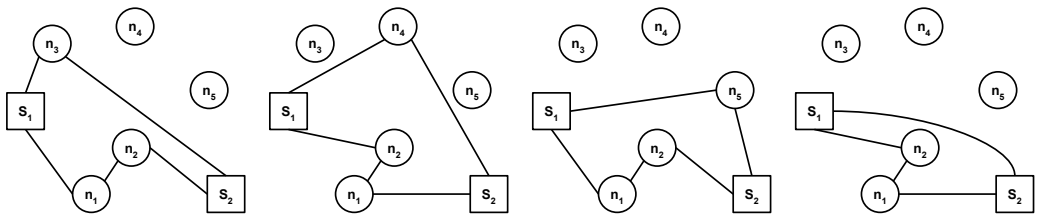
(a) G_1 **(b)** G_2 **(c)** G_3 **(d)** G_4

Figure 2 Dynamic graph on 4 time steps. The terminal vertices are the squared vertices.



(a) G_1 **(b)** G_2 **(c)** G_3 **(d)** G_n

Figure 3 Dynamic graph on n time steps. The terminal vertices are the squared vertices.



(a) G_1 **(b)** G_2 **(c)** G_3 **(d)** G_4

Figure 4 Dynamic graph on 4 time steps. The terminal vertices are the squared vertices.

▶ **Definition 4** (*DMSS2* decision problem). *Let $\mathcal{G} = (G_i)_{1 \leq i \leq L}$ be a dynamic graph such that:*

- *$G_i = (V, E_i)$, $i \leq L$*
- *$\cup_{i \leq T} E_i = E$*

Let $a \in V$ and $b \in V$ be two vertices called terminals. Let $V_s \subset V - \{a, b\}$ be a vertex set such that:

- *$\forall i \leq L$ there is a path between a and b in G_i using only vertices from V_s*

***Question**: for a given non-negative integer p, is there a vertex set V_s of size p?*

The NP-completeness of *DMSS2* decision problem is proven with a reduction from the vertex cover problem (VC). This problem is defined as follows.

▶ **Definition 5** (VC). *Let $G = (V, E)$ be a graph, and let $V_c \subset V$ be a vertex set such that $\forall (u, v) \in E$, $u \in V_c$ or $v \in V_c$.*

***Question**: for a given non-negative integer k, is there a vertex set (a cover set) V_c of size k?*

In order to prove that the *DMSS2* decision problem is NP-complete, we first need to show that there is a polynomial reduction from a Vertex Cover instance to a DMSS2 instance (Lemma 6). Then we need to show that a valid VC instance gives a valid DMSS2 instance (Lemma 7) and that a valid DMSS2 instance gives a valid VC instance (Lemma 8) using the polynomial reduction. An instance of a given decision problem is said valid if the answer to the question is YES for this instance.

▶ **Lemma 6.** *There is a polynomial reduction from VC to DMSS2.*

Proof. Let us build a *DMSS2* instance on a dynamic graph G^{DYN} from a VC instance on graph $G = (V, E)$.

- For each vertex u in G, there is a vertex u in the dynamic graph G^{DYN};
- G^{DYN} has two extra vertices a and b;
- For each edge $e = (u, v)$ in G, there is a corresponding time step i_e in G^{DYN} and $G^{DYN}_{i_e}$ has 4 edges: (a, u), (u, b), (a, v), (v, b).

Vertices a and b in G^{DYN} are the terminal vertices. Figure 5 displays an example of such transformation.

If graph G has n vertices and m edges, G^{DYN} has m time steps, $n + 2$ vertices, and each G^{DYN}_i has 4 edges.

The *DMSS2* instance is built in polynomial time ($O(m + n)$) from the VC instance. ◀

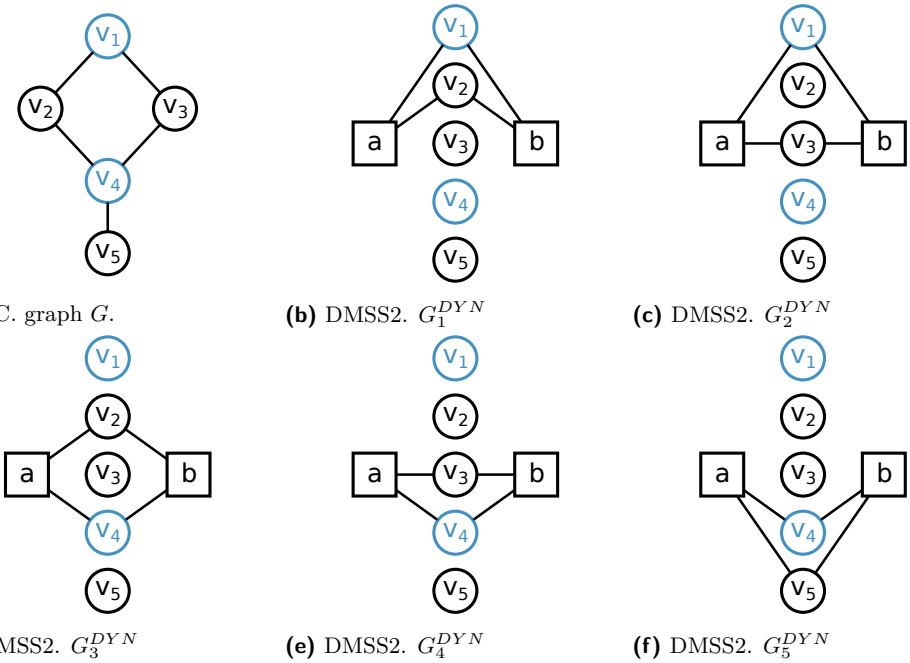
▶ **Lemma 7.** *If a* VC *instance is valid, then the corresponding* DMSS2 *instance is also valid.*

Proof. Consider a vertex cover V_c of size p, and let V_s be the vertex set of G^{DYN} with the same vertices. Therefore V_s also has size p.

Let us consider some time step i. The corresponding graph G^{DYN}_i contains exactly two paths of length 2 between a and b using some vertex u and some vertex v. By construction of G^{DYN}, (u, v) is an edge of G. Hence either $u \in V_c$ or $v \in V_c$ (or both). Therefore, in G^{DYN} either $u \in V_s$ or $v \in V_s$ (or both). As at least one of them is in V_s, there exists in G^{DYN}_i a path between a and b using only vertices from V_s.

This is true for each time step in G^{DYN}, so V_s is a solution of the *DMSS2* instance. ◀

▶ **Lemma 8.** *If a* DMSS2 *instance is valid, then the corresponding* VC *instance is also valid.*



(a) VC. graph G. **(b)** DMSS2. G_1^{DYN} **(c)** DMSS2. G_2^{DYN}

(d) DMSS2. G_3^{DYN} **(e)** DMSS2. G_4^{DYN} **(f)** DMSS2. G_5^{DYN}

Figure 5 Transforming a VC instance with 5 edges to a *DMSS2* instance with 5 time steps. The sets $V_C = V_S = \{v_1, v_4\}$ are in blue.

Proof. Consider the set V_s of G^{DYN} of size p, and let V_c be the vertex set of G with the same vertices. Therefore V_c also has size p.

Let $e = (u, v)$ be an edge of G. For the corresponding time step i_e, $G_{i_e}^{DYN}$ has two paths of length exactly 2 between a and b, with u and v the two vertices on those paths. As V_s is a solution of the *DMSS2* problem, it means that either u or v (or both) are in V_s. Therefore, as V_c and V_s contain the same vertices, u or v (or both) are in V_c.

This is true for all edges of G, hence V_c is a solution of the Vertex Cover problem. ◀

Figure 5 shows an example where both instances are valid for $p = 2$. The associated sets V_c and V_s are in blue. In Figure 5a, the blue vertices are a vertex cover. Indeed, each edge of the graph is adjacent to at least one blue vertex. In Figures 5b to 5f, the blue vertices, plus the terminal vertices, form a Steiner Set. Indeed, at each time step of the graph, vertex a and vertex b are connected by a path going only through blue vertices.

▶ **Theorem 9.** *The decision problem* DMSS2 *is NP-complete.*

Proof. Clearly, *DMSS2* is in *NP* as a proof of *YES* answer is obtained by providing one Steiner set and one tree per snapshot containing all terminal vertices and some other vertices from the Steiner set.

The remaining of the proof follows directly from lemmas 6, 7 and 8. ◀

Remember that 3 variants were presented in the previous section; we can remark that in the first variant Steiner sets are independently computed for each snapshot. Hence with 2 terminals it can be done through L breadth first searches, and the problem is polynomial. In the second variant, the whole Steiner set (including the two terminals) must be connected at each snapshot. The same proof than for partially connected DMSS can be used. Indeed, it suffices to modify slightly the reduction (see lemma 6) as follows: in G_i^{DYN} all vertices

but a and b form a clique. Thus, the solution corresponding to the vertex cover set remains the same as it still needs to contain at least one extremity of each edge of G to connect a and b. Because of the clique edges, the restriction of G_i^{DYN} to the solution set is connected for all i, so this vertex set is indeed a solution of the fully connected *DMSS*. So this variant is also NP-complete even with two terminals.

3.4 NP-Completeness for two time-steps

The dynamic Steiner optimization problem with two terminals is NP-hard. But in the reduction, the number of time-steps of the resulting dynamic graph is m, so at least $\Theta(n)$. What happens when the life-time is strictly smaller? In particular, the problem should be easier when the life-time is bounded by a constant. We now prove that the problem is still NP-hard, even when the lifetime is 2.

▶ **Definition 10** (*DMSS2x2* decision problem). *Let $\mathcal{G} = (G_1, G_2)$ be a dynamic graph with $G_1 = (V, E_1)$, $G_2 = (V, E_2)$. Let $a \in V$ and $b \in V$ be two vertices called terminals. Let $V_s \subset V \setminus \{a, b\}$ be a vertex set such that:*

- *$\forall i = 1, 2$ there is a path between a and b in G_i using only vertices from V_s*

***Question**: for a given non-negative integer p, is there a vertex set V_s of size p?*

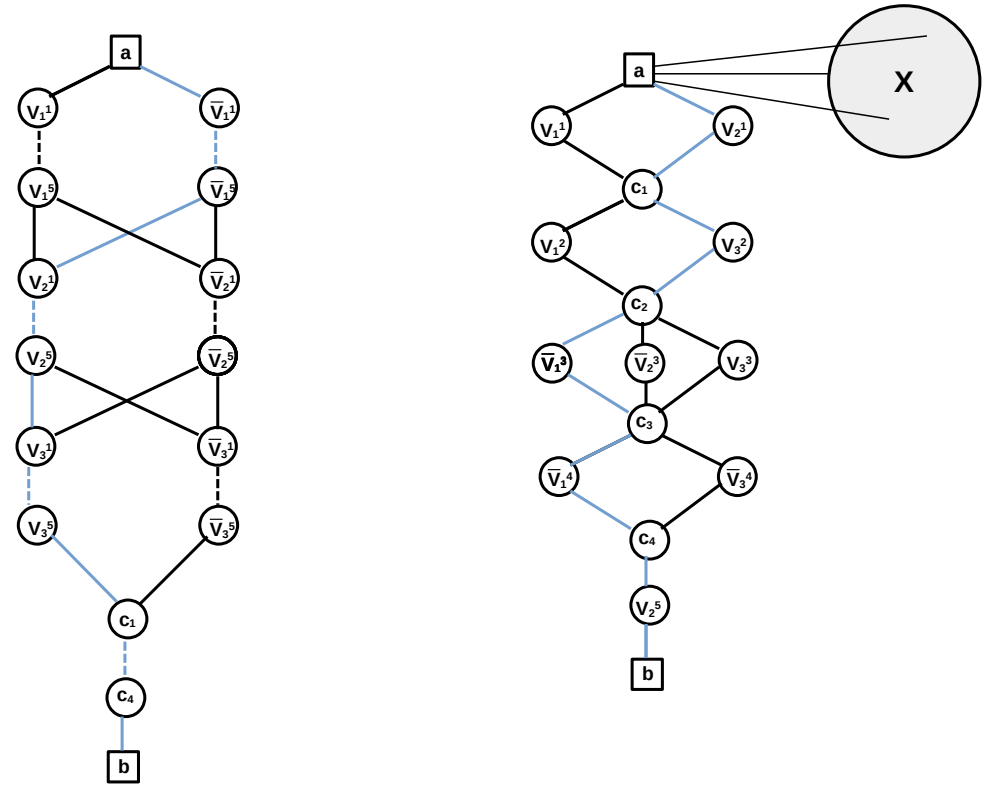
The NP-completeness of DMSS2x2 decision problem is proven with a reduction from the SAT problem. We first show that there is a polynomial reduction from a SAT instance to a DMSS2x2 instance (Lemma 11). Then we show that a valid SAT instance gives a valid DMSS2x2 instance (Lemma 12) and that a valid DMSS2x2 instance gives a valid SAT instance (Lemma 13) using the polynomial reduction.

▶ **Lemma 11.** *There is a polynomial reduction from SAT to* DMSS2x2.

Proof. Let us build a DMSS2x2 instance on a dynamic graph $G^{DYN} = (G_1 = (V, E_1), G_2 = (V, E_2))$ from a SAT instance. The SAT instance formula is given as a conjunctive normal form, with n variables and c clauses. Graph G_1 is first built as a level graph, one level per variable and two vertices by level. Then each initial vertex is replaced by a chain of c vertices. Some additional vertices are added in a straightforward way. Graph G_2 is also built as a level graph, one level per clause and c vertices by level. Vertices c_j are then used to separate these levels. The remaining vertices are connected to a to insure connexity. Thus, a shortest path in G_1 is associated to an interpretation of the formula (i.e. a boolean assignment for each variable), and a shortest path of G_2 to one litteral per clause. More formally:

- for each vertex variable x_i and clause C_j, there are two vertices v_i^j and $\bar{v}_i^j$ in V;
- for each clause C_j; except C_n $(j = 1 \to (n-1))$, there is a vertex c_j in V;
- G^{DYN} has two extra vertices a and b, also denoted c_0 and c_c;
- E_1 contains the following edges:
  $(a, v_1^1); (a, \bar{v}_1^1)$; $\forall i = 1 \to n, \forall j = 1 \to c-1, (v_i^j, v_i^{j+1}); (\bar{v}_i^j, \bar{v}_i^{j+1})$;
  $\forall i = 1 \to n-1, (v_i^c, v_{i+1}^1); (v_i^c, \bar{v}_{i+1}^1); (\bar{v}_i^c, v_{i+1}^1); (\bar{v}_i^c, \bar{v}_{i+1}^1)$;
  $(v_n^c, c_1); (\bar{v}_n^c, c_1)$; $\forall j = 1 \to c-1, (c_j, c_{j+1})$.
- E_2 contains the following edges:
  $\forall j = 1 \to c$, $\forall x_i \in C_j, (c_{j-1}, v_i^j); (v_i^j, c_j)$;
  $\forall j = 1 \to c$, $\forall \bar{x}_i \in C_j, (c_{j-1}, \bar{v}_i^j); (\bar{v}_i^j, c_j)$;
  $\forall j = 1 \to c, \forall x_i, \bar{x}_i \notin C_j, (a, v_i^j); (a, \bar{v}_i^j)$.

Vertices a and b in G^{DYN} are the terminal vertices. Figure 6 displays an example of such reduction. G^{DYN} has 2 time-steps, $2nc + c + 1$ vertices. E_1 has $(2n+1)(c-1) + 4n$ edges;



(a) Graph G_1 **(b)** Graph G_2

Figure 6 Example of reduction from instance 1 of SAT. Dashed lines represent a path with some inner vertices. X is the set of all vertices not present elsewere in G_2, all linked to a. In blue, both paths of a valid solution.

E_2 has $2cn + \sum_j s_j \leq 3cn$ (s_j is the number of litterals in clause C_j). Finally, the question for DMSS2x2 is: is it a vertex set V_s of size $p = nc + (c-1)$?

The DMSS2x2 instance is built in polynomial time $O(nc)$ from the SAT instance. ◀

Figure 6 shows a simple example of reduction. The SAT instance has 3 variables and 5 clauses. Its formula is:

$$(x_1 + x_2)(x_1 + x_3)(\bar{x}_1 + \bar{x}_2 + x_3)(\bar{x}_1 + \bar{x}_3)x_2. \quad (1)$$

The obtained instance of DMSS2x2 has 36 vertices. Both instances are valid. From the paths P_1 and P_2 of a solution V_s for DMSS2x2, one can easily get the corresponding solution for SAT: x_1 is false, x_2 and x_3 are true.

▶ **Lemma 12.** *If a* SAT *instance is valid, then the corresponding* DMSS2x2 *instance is also valid.*

Proof. Consider a solution of the SAT instance, that is an interpretation I that satisfies the formula. Let us consider the path P_1 from a to b in G_1 so that: if $x_i \in I$, then v_i^j belongs to P_1 for all j; and if $\bar{x}_i \in I$, then $\bar{v}_i^j$ belongs to P_1 for all j. The number of vertices of P_1 (a and b excluded) is exactly $nc + c - 1$ (and there are exactly 2^n shortest paths). Let us now consider G_2. As I is a solution of the SAT instance, it contains at least one litteral per clause C_j, say $\tilde{x}_{i_j}$. So in G_2, one can build an elementary path P_2 from a to b including the

corresponding vertex of this litteral (v_i^j or $\bar{v}_i^j$) for all C_j, plus the c_j. By construction, all vertices of P_2 belong to P_1. So we have built a feasible solution for DMSS2x2 of size exactly $p = nc + c - 1$. ◀

▶ **Lemma 13.** *If a* DMSS2x2 *instance is valid, then the corresponding* SAT *instance is also valid.*

Proof. From the construction of G^{DYN}, we know that all paths of G_1 contain at least $nc + c - 1$ vertices (as usual, a and b excluded), and all elementary paths of G_2 contain exactly $2c - 1$ vertices. Let us consider a set V_s of G^{DYN} of size $p = nc + c - 1$, supposed to be a valid solution, and denote P_1 and P_2 the corresponding paths. As V_s is a valid solution, P_1 contains exactly $nc + c - 1$ vertices, and all vertices of P_2 must be included into P_1.

Let us build an interpretation I of the SAT instance as follows: if some v_i^j belongs to P_2, then x_i belongs to I. If some $\bar{v}_i^j$ belongs to P_2, then $\bar{x}_i$ belongs to I. Note that it is impossible to have both v_i^j and $\bar{v}_i^{j'}$ in V_s for some i, as they belong to P_1 and by construction, for a given i either all v_i^j or all $\bar{v}_i^j$ belong to P_1. By construction of G_2, the path P_2 contains at least one vertex associated to one litteral of each clause. However, it is possible that for a given i it contains no vertex of the form v_i^j, and no vertex of the form $\bar{v}_i^j$. In that case, one may add to I either x_i or $\bar{x}_i$, indifferently. That way, interpretation I satisfies all clauses, so it is a valid SAT instance.

◀

▶ **Theorem 14.** *The decision problem* DMSS2x2 *is NP-complete.*

Proof. Clearly, DMSS2x2 is in NP as DMSS2 is.
The remaining of the proof follows directly from lemmas 11, 12 and 13. ◀

Note that in the proposed reduction, both snapshots G_1 and G_2 are connected. Furthermore, one may observe that all vertices of $V_s \cup \{a, b\}$, if V_s is a valid solution of DMSS2x2, are connected. This is true in particular for the vertices of X that are all directly connected to a in G_2. This choice was not mandatory to prove theorem 14, but was done to prove easily the following result.

▶ **Theorem 15.** *The decision problem associated to variant two: Fully Connected Minimum Steiner Set, with* 2 *terminals and* 2 *snapshots, is NP-complete.*

Proof. The proof is immediate. Indeed, in the second variant there is an additional condition on the Steiner set: its vertices must be connected through other vertices of the Steiner set. This property is verified for the reduction used in the above proof: in G_1, the chosen vertices form a path. In G_2, some of the chosen vertices form a path from a to b, and the others are directly connected to a. Therefore the proof directly applies to this variant. ◀

4 Algorithmic issues

This section focuses on efficiently solving the optimization problem associated with DMSS: costs on the edges are not considered, only the cardinality of the Steiner set matters. We first consider the problem of finding a Steiner set of 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.

Our objective is to design efficient (polynomial) algorithms when k is fixed.

4.1 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. Still, for dynamic graphs with a high number of edges in the underlying graph and a slow dynamic, this bound might be close. Note however that the problem on such graphs should not be very interesting from the applications point of view. Furthermore, a Steiner set of the given cardinality will be found with high probability by a simple, and fast, breadth first search on the intersection graph (intersection of all snapshots). We shall discuss in the next section on these questions relative to the instance types.

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. The process starts with all candidates for $i = 1$. It ends when $i = L$ with all Steiner sets of cardinality k. This principle is used in the algorithm below. In the remaining of the section, we shall precise the different steps and give some complexity issues.

Algorithm 1 FindSteinerFixedSize($\mathcal{G}$,k)

Input: $\mathcal{G} = (G_i)_{i\in\{1,\ldots,T\}} = (V, E_i)$; $k \leq |V|$
Output: all Steiner sets of cardinality k.

$PSS \leftarrow$ FindCandidates(G_1,k)
for i *from* 2 *to* L **do**
  $PSS \leftarrow$ Update(PSS,i);
  If ($PSS = \emptyset$) then STOP.
end
Return: PSS

4.2 Algorithm details and complexity

Method *FindCandidates*(G_1,k) computes all candidates for G_1. Unfortunately, there is no other way than enumerating all subsets of $V - S$ of cardinality k'. For each subset K, determining wether it is a candidate can be done in two steps. First, build the subgraph of G_1 generated by $S \cup K$. Second, perform some search from any vertex of S on this subgraph. Subset K is candidate if and only if all other vertices of S are found during this search.

Method *Update*(PSS,i) uses the same procedure as *FindCandidates*, but only on the elements of PSS, and on G_i.

So the core procedure of the algorithm is a search on a small subgraph. Its complexity is $\mathcal{O}(m')$ where m' is the number of edges of the subgraph. This number is bounded by k^2 and by m, the number of edges of the underlying graph.

In the worst case (see above), the number of candidates remains very high throughout the iteration process, it is majored by $\binom{n-s}{k'}$, hence by $\frac{(n-s)^{k'}}{k'!}$.

▶ **Theorem 16.** *The worst case complexity of algorithm 1 as a function of the number of time steps L, the number of vertices n, the number of terminals s, and the size of the computed Steiner sets k, is $\Theta(T \times (n-s)^{k-s} \times \frac{k^2}{(k-s)!})$. When k and s are fixed parameters, the complexity is $\Theta(T \times n^{k-s})$.*

Proof. The first part is immediate from the number of iterations, the maximum number of candidates at each iteration, and the complexity of the search for each candidate.
The best case is obtained when no Steiner set exists for G_1. In that case, the algorithm tests all candidates for G_1 then stops. ◀

These recent years, many studies focused on revisiting classical problems in the light of parameterized complexity, and this is of course true for Steiner problems. The main results for Steiner, with a presentation of the basics of parameterized complexity, are summarized in [14]. When the parameter is the number of terminals s, the minimum cardinality problem is FPT (Fixed Parameter Tractable). But it is W[2]-hard if the parameter is the size of the Steiner set (the problem is still FPT if the graph is planar). This means no algorithm solves the problem in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where $|I|$ is the instance size and f is an arbitrary function depending only of k, and this of course remains true in the dynamic case. Theorem 16 however shows that $DMSS$ is in the XP class when k (or $k' = k - s$) is considered, but leaves open whether it is FPT for parameter s.

4.3 Some additional remarks on the algorithms

The worst case complexity is the same as that of the naive method that computes all candidates at each iteration. It is expected that the average and experimental complexity will be much better.

If the snapshot graphs are sparse, it is expected that the number of kept candidates diminishes very fast as the number of iterations increases, and eventually equals 0. Then the algorithm stops and returns the empty set. One expects also for these instances that most of the work is done during first step. Finally, Note that a preprocessing phase accelerates greatly the tests (see next section): a graph search checks if a Steiner tree exists in the underlying graph. If not, the result is of course the empty set.

Algorithm 1 returns all Steiner sets of size k. To return Steiner sets of minimal size k^*, two approaches are possible. The direct one is to use a dichotomy search and repeatedly call the algorithm. The number of calls is then $\mathcal{O}(\log \kappa)$ where κ is an upper bound of $k'^* = k^* - s$ (at most $n - s$). The second approach consists in keeping at each step the candidates that are minimal for the inclusion (that is, removing any vertex removes the subset from the candidate set). Updating this set at each time step is more complicated, and it is not clear wether this approach is faster.

Finally, let us consider the problem of Steiner sets of fixed size and minimal cost. Due to the nature of the problem, a last step must be added to algorithm 1: for each final Steiner set, compute its weight. This implies to compute L minimum spanning trees (on a graph with k vertices) for each Steiner set, which costs at most $\mathcal{O}(T \times k^2 \times \log k)$ with Prim or Kruskal algorithms. Note that this complexity is polynomial in all parameters, and independent of n (but of course the number of Steiner sets is not independent of n, and not always polynomial).

5 Experimental study

In this section, we present an experimental study of our algorithm, both on randomly generated graphs and on real-world networks.

5.1 Generated graphs

The experimental study on randomly generated graphs was performed using the GraphStream Java library[2][9]. The virtual machines used for this experiment have 8 core processors with 64 GB of RAM.

5.1.1 Experimental settings

In order to run experiments on our algorithm, we generated dynamic graphs using a method previously presented in [20] to test connected component computation. First we generate the underlying graph, then we add dynamicity to the edges using a Markovian process.

The underlying graph is generated using generators from the GraphStream library. We test three different types of graphs that present specific features:

1. Random graphs, corresponding to the Erdős-Rényi model[10]. It is the most common way of randomly generating a static graph. GraphStream Random Graph generator is used. This generator adds a vertex and randomly connects it to the other vertices of the graph. This operation is repeated for each vertex added.
2. Regular graphs, which are generated using GraphStream Grid generator. This generator generates a torus with the given number of vertices, all with the same degree.
3. Scale-free graphs, which are used to model many social networks or web networks. Graphstream Barabasi-Albert generator is used. This generator adds a vertex to the graph and connects it to one or several vertices randomly chosen using Barabasi-Albert's [1] preferential attachment rule. This operation is repeated for each vertex added.

Once the underlying graph is generated, the dynamic is introduced through a Markov model (edge-Markovian dynamic graphs, introduced by [8]): edges appear and disappear independently, and from the model two parameters are derived. The *presence* is equal to the stationary probability of edge presence, often noted π_1 ($\pi_1 \in [0, 1]$). The *stability* of the dynamic graph may be defined as the average number of steps during which one edge remains present. A presence of 0.9 was chosen in our experiments (each edge is present 90% of the time), for a stability equal to 9.

Once the dynamic graph is generated, the terminal vertices are selected randomly in the vertex set.

Two experiments were run on randomly generated graphs. The first one aims at analyzing the behavior of the algorithm (both through the number of solutions found and the computation time) as it is computing time step by time step. The second experiment aims at analyzing the outcome of the algorithm depending on the number of vertices of the graph. Table 1 presents the parameters of the generated graphs for each experiment, where n is the number of vertices, k the size of the Steiner set, s the number of terminals and L the lifetime of the dynamic graph.

5.1.2 Experiment on the behavior of the algorithm with regard to the number of iterations

The curves are only presented for $s = 3$, but the results for other values are summarized in table 2.

Figure 7 presents the evolution of the number of solutions when the number of iterations (time steps) increases. The decrease is very fast for grids, and to a lesser extend for random

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

	First experiment	Second experiment
Graph type	Random, Scale-free, Regular	Random, Scale-free, Regular
n	49	36 ; 49 ; 64 ; 81 ; 100
k	10	10
S	3 ; 4 ; 5 ; 6	4
Presence	0.9	0.9
Stability	9	9
Average degree	4	4
T	50	50

Table 1 Graphs generation settings

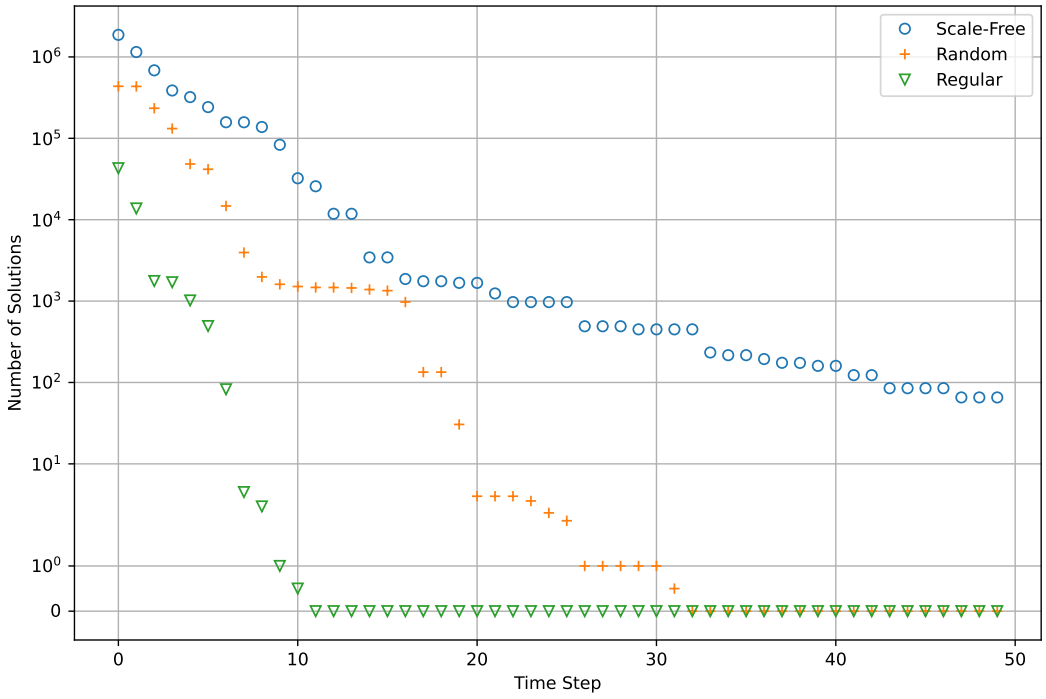


Figure 7 Evolution of the median number of Steiner sets with regard to the number of time steps, with 3 terminal vertices.

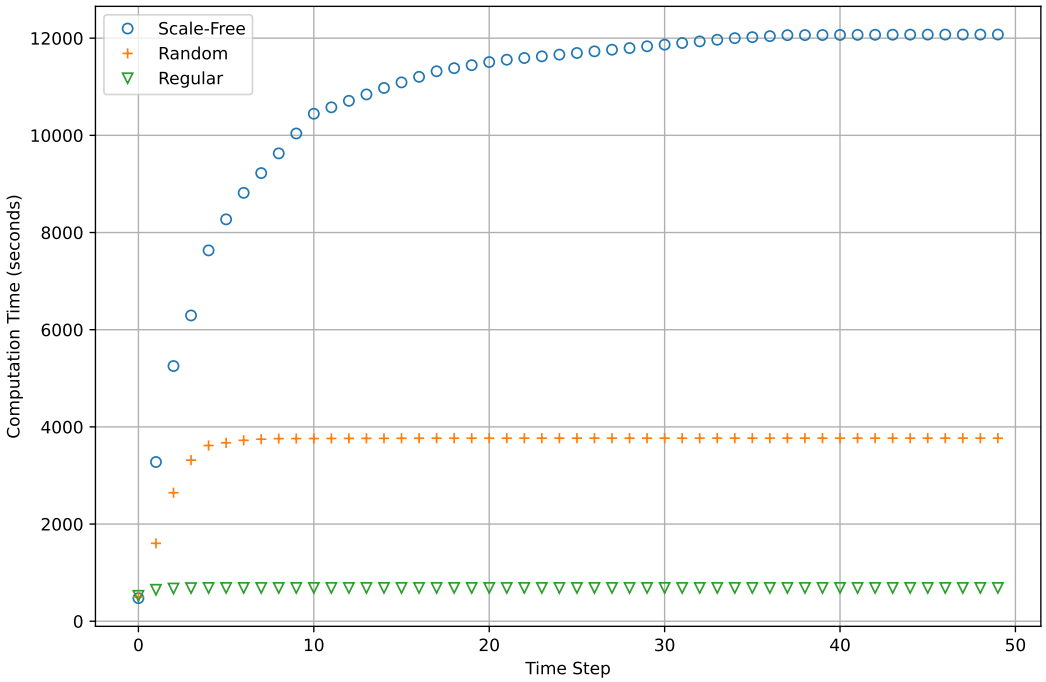


Figure 8 Evolution of the median computation time as it is being computed, with 3 terminal vertices.

graphs. Indeed, it is very difficult to keep large persistent connected components (vertex sets than remain connected) for these two classes of graphs, see[20]. So the number of Steiner sets rapidly reaches 0. For scale free graphs however, the number of Steiner sets decreases but much slower, and remains around 100 even for nearly 50 iterations. In these graphs, there are some hubs with large degree that are very good candidates for the Steiner set.

Figure 8 provides the evolution of the total computation time. It is quite logically a concave function that is constant as soon as the number of solution is 0. The theoretical results on the algorithm complexity are confirmed by figure 9 that shows the computation time per iteration varies linearly with the number of tested solutions.

5.1.3 Experiment on the runtime depending on the number of vertices of the graph

Figure 10 shows that the complexity curve fits well with the regression function n^{k-s}, as predicted by the algorithm analysis.

The main conclusion from this experimental study is that the algorithm can be applied with medium size instances despite the high theoretical complexity. Another conclusion concerns the number of possible Steiner sets. This number is highly dependent on the type of underlying graph (footprint) and might decrease very fast with the number of iterations if the graph is not scale free. These results were obtained for a moderate dynamic: a present edge has a probability 0.9 to remain present from one iteration to the next.

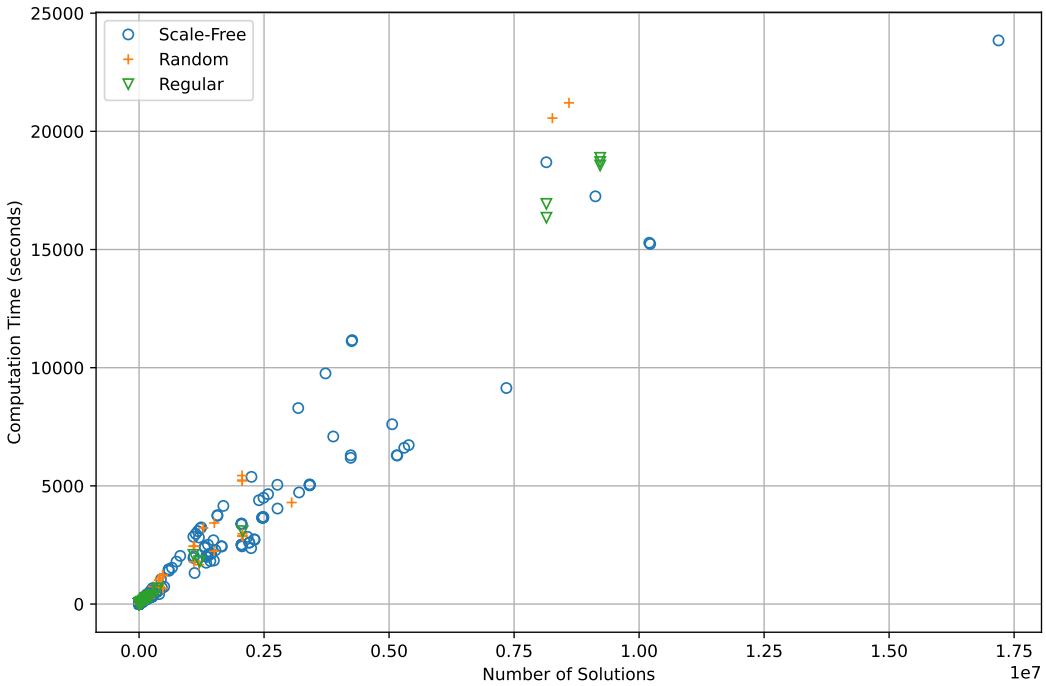


Figure 9 Median computation time per iteration according to the number of tested vertex subsets, with 3 terminal vertices.

Table 2 Number of solutions

Graph Type	S	Maximum Value (date $t = 0$)	Minimum Value	First snapshot with Minimum Value
Scale-free	3	1 867 447,5	65,5	47
	4	256 030,0	5,0	47
	5	2 626,5	0,0	13
	6	2 405,5	0.0	17
Random	3	435 853,5	0.0	32
	4	33 258,5	0.0	17
	5	1 373,5	0.0	8
	6	85,0	0.0	3
Regular	3	42 909,5	0.0	11
	4	7 587,5	0.0	9
	5	10.0	0.0	4
	6	0.0	0.0	0

Scale-Free	0,976
Random	0,999
Regular	0,998

Table 3 R^2 values of the regression functions from Figure 10.

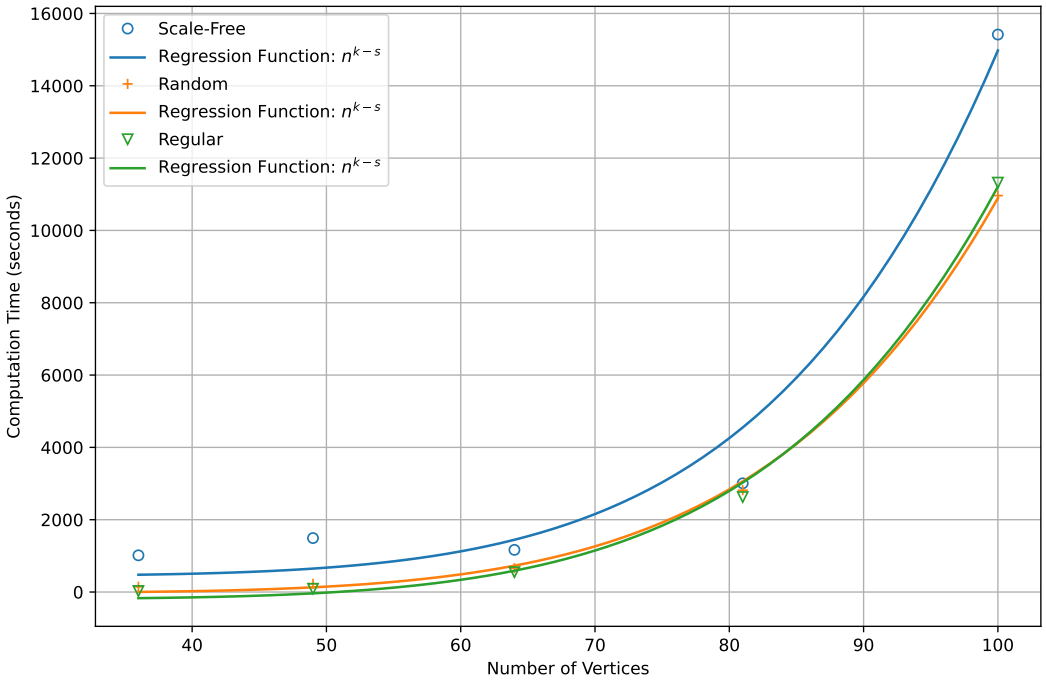


Figure 10 Median computation time according to the number of vertices in the graph, compared with a regression function of the form n^{k-S}, which is n^6 here.

5.2 Real graphs

5.2.1 Experimental settings

The CRAWDAD VT/MANIAC dataset[13], 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 dataset comprises log files providing information for each monitored device. These files can be interpreted as routing tables for each device on a per-second basis. Based on this information, a dynamic graph has been constructed, where nodes represent mobile stations and edges depict connections between two mobile stations. The edges are labeled with a list of timestamps indicating the dates when these connections are effective. In the presented results, the experiment involves 14 mobile stations for a duration of about 20 minutes. It translates into a graph with 14 nodes, 74 edges (for the footprint), and 1244 timesteps. On average, edges are present for 592 seconds (less than 10 minutes) during the experiment, which is 48% of the time. When present, edges remain up for 114 seconds (less than 2 minutes) on average (9.1% of the time). In comparison to the artificial graphs used in the previous experiment, the *presence* is lower here (48%) than before (90%).

If for some snapshot, the terminals can not be connected, the number of Steiner sets is of course 0. Hence it is useful to verify this first. An efficient algorithm to compute open Eternal Connected Components *oECC* is available, see[20]. After running this algorithm,

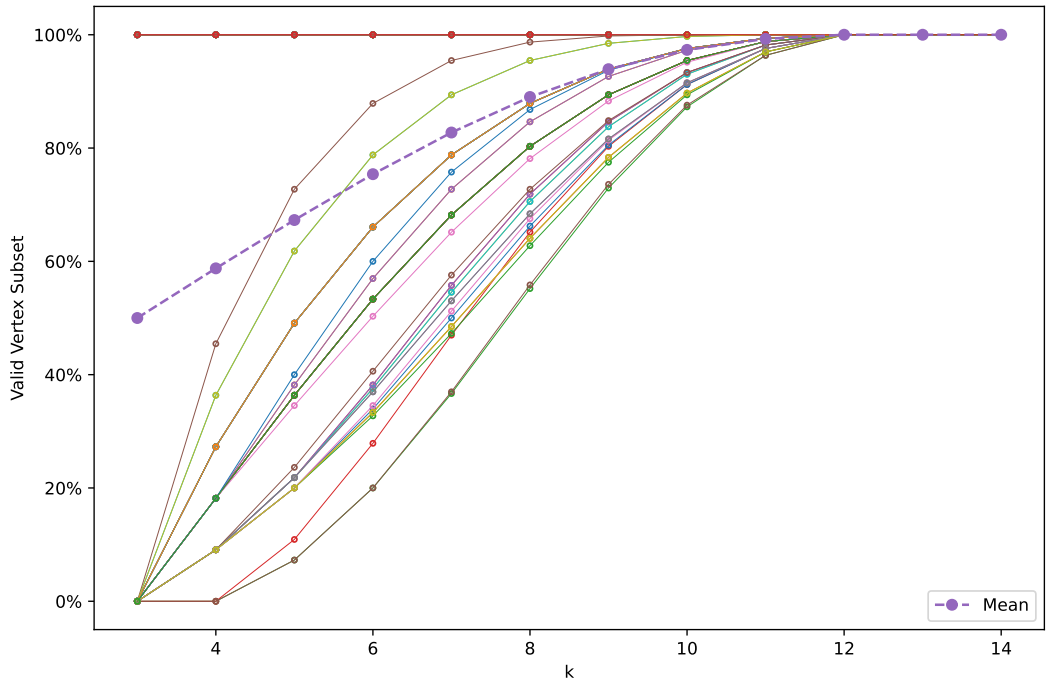


Figure 11 Real dataset: percentage of Steiner sets found for different values of k ($s = 4$).

an $oECC$ with 9 vertices has been found. If all terminal vertices are inside this ECC, it guarantees the existence of at least one Steiner set (of size $n - s$). Conversely, no Steiner set exists if the terminal set is not included into an $oECC$. Note however that a vertex outside the $oECC$ may still belong to some Steiner set, due to the open nature of the component. We decided to look only for vertices of the $oECC$ to build the terminal set, whereas the Steiner set candidates are included into V.

5.2.2 Results and discussion

Figure 11 shows the percentage of Steiner sets among all possible vertex sets of same size, for all possible terminal sets of size $s = 3$ (one line may represent different terminal sets, with the same behavior). The same kind of "banana" curve has been obtained for $s = 4, 5, 6$. Note the large value of the mean, hence the large number of cases where the terminal set is already connected (x-axis = 3, y-axis = 100%), due to the nature of the graph: a bit less than 50%. Conversely, there is almost always at least one solution (except in some cases when $k = s + 1 = 4$). The results are similar for larger values of s.

So it is nearly always possible to find a Steiner set in this example, for any value of s and k except specific cases with $k - s = 1$ or $k - s = 0$. The finding is very fast (a few seconds at most) even if the time horizon is very large.

6 Conclusion

In this paper, we presented several possible extensions of the Steiner Tree problem to the case of dynamic (or temporal) graphs. The goal is to maintain instantaneous connectivity between all terminal vertices for all snapshots. The chosen extension minimizes the size of

the Steiner set. It can be further extended to minimize the sum of the costs of the edges used at each time step as a the second criterion. We proved that the associated decision problem is NP-Complete even when the costs are not considered, and there are only two terminals. The proof is rather straightforward thanks to a reduction to Vertex Cover. However, we also prove that this remains true even when the lifetime is 2, through another and a bit more complicated reduction from SAT.

This paper proposed an exact algorithm that computes all Steiner sets of a given size, studied its complexity, and provided some experimental tests. The algorithm may be used on-line. First, randomly generated dynamic graphs of different types have been used. The algorithm was able to solve small to medium size instances. The number of solutions, hence the computation time, is highly dependent on the type of graphs. The results confirmed Steiner sets are much more numerous on scale free graphs. Second, the algorithm has been applied to a dynamic graph obtained from real data considering MANET's. The underlying graph is small, but the time horizon is large. The algorithm was able to find rapidly all Steiner sets, for different parameter values. Of course the efficiency will be very versatile according to the type of applications, that use graphs of very different kinds and very different sizes.

References

1 Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47–97, 2002.

2 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.

3 Stefan Balev, Eric Sanlaville, and Jason Schoeters. Temporally connected components. *Theoretical Computer Science*, 1013:114757, 2024.

4 Sandeep Bhadra and Afonso Ferreira. Computing multicast trees in dynamic networks and the complexity of connected components in evolving graphs. *Journal of Internet Services and Applications*, 3:269–275, 2012.

5 Robert Bredereck, Christian Komusiewicz, Stefan Kratsch, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Assessing the computational complexity of multilayer subgraph detection. *Network Science*, 7(2):215–241, 2019.

6 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.

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

8 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.

9 Antoine Dutot, Frédéric Guinand, Damien Olivier, and Yoann Pigné. Graphstream: A tool for bridging the gap between complex systems and dynamic graphs. In *Emergent Properties in Natural and Artificial Complex Systems. Satellite Conference within the 4th European Conference on Complex Systems (ECCS'2007)*, 2007.

10 Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

11 Lester R. Ford and Delbert R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6(3):419–433, 1958.

12 Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. Freeman San Francisco, 1979.

692 **13** Amr Hilal, Jawwad N Chattha, Vivek Srivastava, Michael S Thompson, Allen B MacKenzie,
693 Luiz A DaSilva, and Pallavi Saraswati. Crawdad vt/maniac, 2022. URL: `https://dx.doi.`
694 `org/10.15783/C7WG6T`, `doi:10.15783/C7WG6T`.
695 **14** Mark Jones, Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Ondřej Suchý.
696 Parameterized complexity of directed steiner tree on sparse graphs, 2012. `arXiv:1210.0260`.
697 **15** Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US,
698 Boston, MA, 1972. `doi:10.1007/978-1-4684-2001-2_9`.
699 **16** Nina Klobas, George B Mertzios, Hendrik Molter, and Paul G Spirakis. The complexity of
700 computing optimum labelings for temporal connectivity. *Journal of Computer and System*
701 *Sciences*, 146:103564, 2024.
702 **17** Othon Michail and Paul G Spirakis. Traveling salesman problems in temporal graphs.
703 *Theoretical Computer Science*, 634:1–23, 2016.
704 **18** Martin Skutella. An introduction to network flows over time. In *Research Trends in*
705 *Combinatorial Optimization*, pages 451–482. Springer, 2009.
706 **19** Mathilde Vernet, Maciej Drozdowski, Yoann Pigné, and Eric Sanlaville. A theoretical and
707 experimental study of a new algorithm for minimum cost flow in dynamic graphs. *Discrete*
708 *Applied Mathematics*, 296:203–216, 2021.
709 **20** Mathilde Vernet, Yoann Pigne, and Eric Sanlaville. A study of connectivity on dynamic
710 graphs: computing persistent connected components. *4OR*, 21(2):205–233, 2023.
711 **21** Kevin White, Martin Farber, and William Pulleyblank. Steiner trees, connected domination
712 and strongly chordal graphs. *Networks*, 15(1):109–124, 1985.
713 **22** B Bui Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost
714 journeys in dynamic networks. *International Journal of Foundations of Computer Science*,
715 14(02):267–285, 2003.