A Generic Framework for Computing Parameters of Sequence-based Dynamic Graphs*

Arnaud Casteigts[1], Ralf Klasing[1], Yessin M. Neggaz[2], and Joseph G. Peters[3]

[1] LaBRI, CNRS, University of Bordeaux, France
[2] IRIT - SMAC Team, University of Toulouse, France
[3] School of Computing Science, Simon Fraser University, Burnaby, BC, Canada

Abstract. We presented in [10] an algorithm for computing a parameter called T-*interval connectivity* of dynamic graphs which are given as a sequence of static graphs. This algorithm operates at a high level, manipulating the graphs in the sequence as atomic elements with two types of operations: a *composition* operation and a *test* operation. The algorithm is optimal in the sense that it uses only $O(\delta)$ composition and test operations, where δ is the length of the sequence. In this paper, we generalize this framework to use various composition and test operations, which allows us to compute other parameters using the same high-level strategy that we used for T-interval connectivity. We illustrate the framework through the study of three minimization problems which refer to various properties of dynamic graphs, namely Bounded-Realization-of-the-Footprint, Temporal-Connectivity, and Round-Trip-Temporal-Diameter.

Keywords: Dynamic networks, Property testing, Generic algorithms, Temporal connectivity.

1 Introduction

Dynamic networks consist of entities making contact over time with one another. The types of dynamics resulting from these interactions are varied in scale and nature. For instance, some of these networks remain connected at all times [21]; others are always disconnected [18] but still offer some kind of connectivity over time and space (*temporal* connectivity); others are recurrently connected, periodic, etc. All of these contexts can be represented as properties of dynamic graphs (also called time-varying graphs, evolving graphs, or temporal graphs). A number of such classes were identified in recent literature and organized into a hierarchy in [9]. Each of these classes corresponds to specific properties which

* Part of this work was done while Joseph Peters was visiting the LaBRI as a guest professor of the University of Bordeaux. This work was partially funded by the ANR projects DISPLEXITY (ANR-11-BS02-014) and ESTATE (ANR-16-CE25-0009-03). This study has been carried out in the frame of "The Investments for the Future" Programme IdEx Bordeaux CPU (ANR-10-IDEX-03-02). The work of Joseph Peters was partially supported by NSERC of Canada.

play a role either in the complexity or in the feasibility of distributed problems. For example, it was shown in [12] that if the edges are *recurrent* (i.e. if an edge appears once, then it will reappear infinitely often), denoted class $\mathcal{R}$, then such a property guarantees the feasibility of a certain type of optimal broadcast with termination detection (namely, *foremost* broadcast). However, it is not sufficient to satisfy other measures of optimality, such as *shortest* or *fastest* broadcast. Strengthening the assumption to having a *bound* on the reappearance time (class $\mathcal{B}$) makes it possible to achieve shortest broadcast, and the even stronger assumption of having *periodic* edges (class $\mathcal{P}$) enables fastest broadcast. These three classes have been shown to play a role in a variety of problems (see e.g. [1, 15, 22]). Another important class, which is less constrained (and thus more general) is the class of all graphs with recurrent temporal connectivity (i.e. all nodes can recurrently reach each other through journeys), corresponding to class $\mathcal{C}_5$ in the hierarchy of [9]. This property is very general, and it is used (implicitly or explicitly) in a number of recent studies addressing distributed problems in highly-dynamic environments [4–6, 14]. Interestingly, this property was considered more than three decades ago by Awerbuch and Even [2].

Given a dynamic graph, a natural question to ask is to which of the classes this graph belongs, or what related property it satisfies. These questions are interesting in several respects. Firstly, most of the known classes correspond to necessary or sufficient conditions for given distributed problems or algorithms (broadcast, election, spanning trees, token forwarding, etc.). Thus, being able to classify a graph in the hierarchy is useful for determining which problems can be solved on that graph. Furthermore, it is useful for choosing a good algorithm in settings where some properties are guaranteed (as in the above example with classes $\mathcal{R}, \mathcal{B}$, and $\mathcal{P}$). Hence, when targeting a given scenario from the real world, an algorithm designer may first record some topological traces from the target environment and then test which useful properties are satisfied. A growing amount of research is now focusing on testing properties (or computing structures) in dynamic graphs. A seminal example is the computation of foremost, shortest, or fastest journeys [7], the algorithms of which can also be used to test membership in a number of dynamic graph classes [8]. More recent examples include computing reachability graphs [3, 24], enumerating maximal cliques [23], and establishing the hardness of computing metrics like *temporal diameter* (that is, how long it takes in the worst case to communicate through journeys) when the evolution is not known in advance [17].

In a previous paper [10], we focused on a property called *T-interval connectivity* [20], which captures two aspects of a network, *stability* and *connectivity*, and was shown to play a role in several distributed problems, such as determining the size of a network or computing a function of the initial inputs of the nodes. T-interval connectivity (Class $\mathcal{C}_{10}$ in [9]) generalizes the class of dynamic graphs that are connected at all time instants [21] (Class $\mathcal{C}_9$ in [9]). The definition of T-interval connectivity is closely related to a representation of a network as a sequence of graphs $\mathcal{G} = (G_1, G_2, ..., G_\delta)$ which correspond to the state of the topology at increasing time instants (also called *untimed* evolving graphs [7]).

Informally, T-interval connectivity requires that, for every T consecutive graphs in $\mathcal{G}$, there exists a common connected spanning subgraph. In [10], we proposed a high-level algorithm for finding the largest T such that a given sequence $\mathcal{G}$ is T-interval connected. We also addressed the related decision problem of testing if $\mathcal{G}$ is T-interval connected for given T. The approach in [10] focuses on high-level strategies in which the graphs in the sequence are considered to be atomic elements and the algorithm only uses two high-level operations on these elements: the intersection of two graphs, and testing if a given graph is connected. We showed that both the maximization and decision versions of the problem can be solved using only a linear number (in the length δ of the sequence) of such operations. The technique is based on a walk in a hierarchy, the elements of which are graphs that represent the intersections of various subsequences of $\mathcal{G}$.

1.1 Contributions

In this paper, we show that the high-level logic of the algorithm from [10] is actually quite general and can be used to compute a number of parameters in addition to T-interval connectivity by replacing the *intersection* and *connectivity test* operations by other operations. We begin by abstracting the two operations into a *composition* operation, which defines the hierarchy of elements in which the walk is performed, and a *test* operation, which determines the choices made by the walk. We investigate both the maximization and minimization of graph parameters and illustrate our framework with four instantiations of the operations: one solves a maximization problem (T-interval connectivity) and three instantiations solve the following minimization problems concerning temporal properties of recognized importance.

First, we consider the class of dynamic graphs with *time-bounded reappearance of edges*. A graph has time-bounded edge reappearance with bound b if the time between two appearances of the same edge in the graph $\mathcal{G}$ is at most b. This property, together with the knowledge of n (the number of nodes) and b, allows the feasibility of shortest broadcast with termination detection [11]. We consider the problem Bounded-Realization-of-the-Footprint of finding the smallest bound b such that $\mathcal{G}$ has time-bounded edge reappearance, *i.e.* the smallest b such that every edge that appears in the sequence $\mathcal{G}$ appears at least once in every subsequence of length b of $\mathcal{G}$.

Then, we look at the class of dynamic graphs with *temporal connectivity* where a journey (temporal path) exists from any node to all other nodes. In this class of graphs, any node can perform a broadcast to all other nodes and can collect information from all the other nodes. The concept of temporal connectivity is relatively old and dates back at least to the article [2]. We consider the minimization problem Temporal-Diameter of finding the *temporal diameter* of a given dynamic graph $\mathcal{G}$, i.e. the smallest duration in which there exist journeys (temporal paths) from any node to all other nodes.

Finally, we are interested in the class of dynamic graphs with *round-trip temporal connectivity* meaning that a back-and-forth journey exists from any node to all other nodes. This class characterizes an important property of distributed solutions for information collection problems that require termination detection [9]. We investigate the problem ROUND-TRIP-TEMPORAL-DIAMETER of computing the *round trip diameter* of a given graph $\mathcal{G}$, i.e. the smallest duration in which there exist back-and-forth journeys from any node to all other nodes.

2 Definitions and Observations

Let $\mathcal{G}$ be a graph sequence $\{G_1, G_2, ..., G_\delta\}$ such that $G_i = (V, E_i)$ represents the network topology at time i. Note that V does not vary; only the edges change. We assume that the changes between two consecutive graphs in the sequence are arbitrary. Let P be a boolean predicate (hereafter called *property*) defined on a consecutive subsequence $\{G_i, G_{i+1}, ..., G_j\} \subseteq \mathcal{G}$.

Definition 1. *The* minimization problem on $\mathcal{G}$ with respect to P *is the problem of finding the smallest k such that $\forall i \in [1, \delta - k + 1]$, $\{G_i, G_{i+1}, ..., G_{i+k-1}\}$ has property P (in other words, any subsequence of $\mathcal{G}$ of length k satisfies P).*

Definition 2. *The* maximization problem on $\mathcal{G}$ with respect to P *is the problem of finding the largest k such that $\forall i \in [1, \delta - k + 1]$, $\{G_i, G_{i+1}, ..., G_{i+k-1}\}$ has property P.*

We present here a general strategy for minimization and maximization problems that relies on a *composition hierarchy* of elements which is computed on demand using a *composition operation.*

Definition 3 (Composition hierarchy and test operation). *An* element $G_{(i,j)} : i \leq j$ *of a composition hierarchy is a graph from which one can determine whether the sequence $\{G_i, G_{i+1}, ..., G_j\}$ satisfies a predicate P using a* test operation *which maps any element into $\{true, false\}$: $test(G_{(i,j)}) = true$ iff the sequence $\{G_i, G_{i+1}, ..., G_j\}$ satisfies P. The initial G_i's are not elements of the hierarchy themselves, but all $G_{(i,i)}$'s are.*

A hierarchy of elements consists of rows denoted $\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^\delta$ where $\mathcal{G}^k = \{G_{(1,k)}, G_{(2,k+1)}, \ldots, G_{(\delta-k+1,\delta)}\}$. We use $\mathcal{G}^k[i]$ to denote the i^{th} element of row $\mathcal{G}^k$, that is the element $G_{(i,i+k-1)}$. The first row $\mathcal{G}^1$ of the hierarchy corresponds to the graphs of the sequence $\mathcal{G}$ (or to simple transformations of these graphs); that is, $G_{(i,i)}$ corresponds to G_i. An example of a hierarchy in which elements are intersection graphs is shown in Figure 1.

Definition 4 (Composition operation). *A* composition operation $\circ$ *is a binary operation that maps two elements of the hierarchy into another element: $G_{(i,j)} \circ G_{(i',j')} = S$ where S is the element that relates to the sequence $\{G_i, G_{i+1}, ..., G_j, G_{i'}, G_{i'+1}, ..., G_{j'}\}$.*

Observation 1. *A minimization (resp. maximization) problem amounts to finding the lowest (highest) row $\mathcal{G}^k$ in which all elements $\{G_{(1,k)}, G_{(2,k+1)}, \ldots, G_{(\delta-k+1,\delta)}\}$ satisfy the test.*

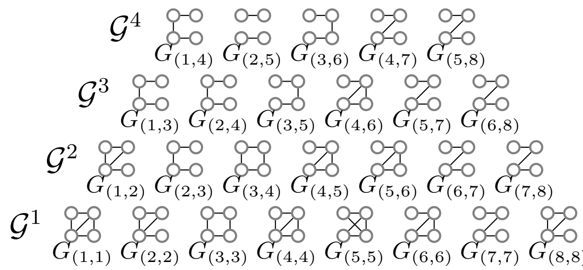


Fig. 1: Example of partial hierarchy with intersection graphs as elements.

The general framework that we propose makes it possible to solve minimization or maximization problems by focusing only on the composition and test operations, while the high-level logic of the algorithm remains the same. More precisely, there is one high-level algorithm for minimization problems, and another for maximization problems.

Observation 2 (Requirements). *For a minimization or a maximization problem relative to some property P to be solvable within our framework, the following conditions must hold on the composition operation $\circ$ and the test operation* `test`*:*

(1) $test(G_{(i,j)}) = true \Leftrightarrow \{G_i, G_{i+1}, \ldots, G_{j-1}, G_j\}$ *satisfies* P*;*

(2) The composition operation $\circ$ is associative, that is $(G_{(i,j)} \circ G_{(i',j')}) \circ G_{(i'',j'')} = G_{(i,j)} \circ (G_{(i',j')} \circ G_{(i'',j'')})$.

Only for maximization problems:

(3') If $test(G_{(i,j)}) = true$ *then* $test(G_{(i',j')}) = true$ *for all* $i' \geq i$ *and* $j' \leq j$.

Only for minimization problems:

(3") If $test(G_{(i,j)}) = true$ *then* $test(G_{(i',j')}) = true$ *for all* $i' \leq i$ *and* $j' \geq j$.

3 Generic algorithm

We propose a strategy based on the generic composition and test operations defined above. The algorithm is then instantiated in Section 4 to solve three specific minimization problems and one maximization problem by plugging in the appropriate operations. The strategy relies on the concept of *ladder*. Informally, a ladder is a sequence of elements that "climbs" the composition hierarchy bottom-up.

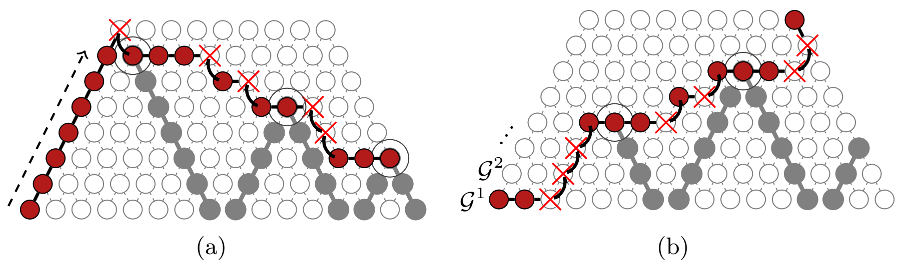


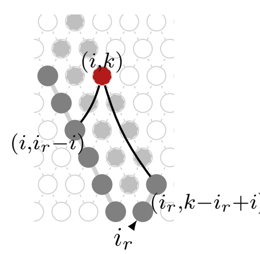
Fig. 2: Example of execution of the algorithm in (a) the maximization case (b) the minimization case.

Definition 5. *The* right ladder of length l at index i*, denoted by* $\mathcal{R}^l[i]$*, is the sequence of elements* $\{\mathcal{G}^k[i],\ k = 1, 2, \ldots, l\}$*. The* left ladder of length l at index i*, denoted by* $\mathcal{L}^l[i]$*, is the sequence* $\{\mathcal{G}^k[i-k+1],\ k = 1, 2, \ldots, l\}$*.*

Lemma 1 ([10]). *A ladder of length l can be computed using $l-1$ binary compositions by computing each element as the composition of the preceding element in the ladder and an element in* $\mathcal{G}^1$*.*

Lemma 2 ([10]). *Given a left ladder of length l_ℓ at index i_ℓ and a right ladder of length l_r at index $i_r = i_\ell + 1$. For any pair (i, k) such that $i_r - l_\ell \leq i < i_r$ and $i_r - i < k \leq i_r - i + l_r$, $\mathcal{G}^k[i]$ can be computed by a single composition operation, namely* $\mathcal{G}^k[i] = \mathcal{G}^{i_r - i}[i] \circ \mathcal{G}^{k - i_r + i}[i_r]$*.*

Informally, the constraints $i_r - l_\ell \leq i < i_r$ and $i_r - i < k \leq i_r - i + l_r$ in Lemma 2 define a rectangle delimited by two ladders and two lines that are parallel to the two ladders as shown in the figure to the right. The pairs (i, k) defined by the constraints, shown in light grey in the figure, include all pairs that are strictly inside the rectangle, and all pairs on the parallel lines, but pairs on the two ladders are excluded.



3.1 Informal description of the algorithm

We describe the algorithm with reference to Figures 2a and 2b that respectively show examples of executions in the maximization case and the minimization case (see Algorithm 1 for details).

The algorithm takes as input a boolean problem type `problem` $\in \{min, max\}$, a dynamic graph $\mathcal{G}$, a composition operation $\circ$, and a test operation `test`. It starts by computing the first element $\mathcal{G}^1[1]$ and then traverses the hierarchy from left to right by computing a new adjacent element at each step: the next element in the same row, or the element with the same index in the row above, or the next element in the row below, depending on `problem` and the result of the `test` operation on the current element. We will call this traversal process a *walk*.

In the maximization case, the walk starts at the element $\mathcal{G}^1[1]$ and builds a right ladder incrementally until the test is negative (first loop, lines 3 ff. of Algorithm 1). If $\mathcal{G}^\delta[1]$ is reached and $test(\mathcal{G}^\delta[1]) = true$, then the execution terminates returning δ. Otherwise, suppose that $test(\mathcal{G}^{k+1}[1]) = false$ for some k. Then k is an upper bound on the maximization parameter of $\mathcal{G}$ and the walk drops down a level to $\mathcal{G}^k[2]$ which is the next element in row k that needs to be tested. The walk proceeds rightward on row k by computing at each step a new element in the row while the test is true. However, every time the test is negative, the walk drops down by one row. If the walk eventually reaches the rightmost element $\mathcal{G}^k[\delta - k + 1]$ of some row k and $test(\mathcal{G}^k[\delta - k + 1]) = true$, then the algorithm terminates returning k. Otherwise the walk will terminate at an element $\mathcal{G}^1[i]$ that does not satisfy the test. In this case, the algorithm returns 0 indicating that the dynamic graph $\mathcal{G}$ does not have the property.

In the minimization case, the walk goes up in the composition hierarchy if the test is negative, otherwise it moves forward in the same row. If the walk hits the right side of the hierarchy and the last visited element $\mathcal{G}^k[\delta-k+1]$ in the row $\mathcal{G}^k$ satisfies the `test` operation, then it terminates and returns k. Otherwise, it terminates returning $k+1$ (Observation 2, requirement (3")). If the walk reaches $\mathcal{G}^1[\delta]$ and the test is negative, then the algorithm outputs 0 indicating that the dynamic graph $\mathcal{G}$ does not have the property.

Computing elements of the hierarchy (function *compute*()**).** The elements resulting from the walk (red/dark elements) are computed based on ladders (intermediate elements, in grey in Figure 2a and Figure 2b) as follows. When the walk moves one step forward in the same row, the next element is computed from a right ladder and a left ladder (*e.g.* $\mathcal{G}^4[6] = \mathcal{G}^2[6] \circ \mathcal{G}^2[8]$ in Figure 2b) or from the ladder to which it belongs and an adjacent bottom element (*e.g.* $\mathcal{G}^5[9] = \mathcal{G}^1[9] \circ \mathcal{G}^4[10]$ in Figure 2b). Intermediate elements *i.e. ladders* (in grey in Figure 2a and Figure 2b) are computed, according to Lemma 1, by incrementally composing an element $G_{(i,j)}$ with the adjacent bottom element $G_{(i-1,i-1)}$ (left ladder) or $G_{(j+1,j+1)}$ (right ladder), providing useful shortcuts in the construction. Suppose that $\mathcal{G}^k[i]$ is the first element to be computed where no element $\mathcal{G}^{k'}[i]$ with $k' < k$ has been computed. The first ladder built is $\mathcal{L}^k[k + i - 1]$ of length k ending at $\mathcal{G}^k[i]$ ($\mathcal{G}^7[2]$ in Figure 2a, $\mathcal{G}^4[4]$ in Figure 2b). Differently from left ladders, right ladders are constructed gradually as the walk proceeds. Each time that the walk moves right to a new index, the current right ladder is incremented (a new element is added to the ladder) and the new top element of this right ladder is used immediately to compute the element at the current index in the walk (using Lemma 2). This continues until the walk crosses the current right ladder, on an element $\mathcal{G}^k[i]$ ($\mathcal{G}^6[8]$ in Figure 2b), at which time a left ladder $\mathcal{L}^k[k + i - 1]$ is built to compute $\mathcal{G}^k[i]$ and to be used to compute the next elements on the walk.

This generic algorithm has the following property which is crucial for the correctness of two of the problems described in Section 4.

```
Input: problem ∈ {min, max}, 𝒢, composition operation ∘, test operation test
i ← 1                                              // current index in the row
k ← 1                                              // current row

if problem = max then
    compute(𝒢^k[i])
    while test(𝒢^k[i]) do
        if k = δ then
            return k
        else
            k++; compute(𝒢^k[i])
    k−−; i++
while 1 ≤ k ≤ δ do
    compute(𝒢^k[i])
    if test(𝒢^k[i]) then
        if i = δ − k + 1 then
            return k
        else
            i++
    else
        switch problem do
            case max do
                k−−; i++
            case min do
                if i = δ − k + 1 then
                    return k + 1
                else
                    k++
return 0
```

Algorithm 1: Generic algorithm for minimization and maximization problems

Lemma 3 (Disjoint sequences property). *If the algorithm performs a composition of two elements $G_{(i,j)}$ and $G_{(i',j')}$, then the corresponding sequences $\{G_i, G_{i+1}, \ldots, G_j\}$ and $\{G'_i, G_{i'+1}, \ldots, G_{j'}\}$ are disjoint and consecutive. That is, in any execution, $G_{(i,j')} = G_{(i,j)} \circ G_{(i',j')} \Rightarrow j = i' - 1$.*

Proof. According to the algorithm, each element of the composition hierarchy is computed from: 1) two elements of two different ladders, a left one and a right one, or 2) an element of a ladder and an element in the first row. In both cases the two sequences covered by the two elements used in the computation are disjoint and consecutive, so in any execution, $G_{(i,j')} = G_{(i,j)} \circ G_{(i',j')} \Rightarrow j = i' - 1$. □

Lemma 4. *Let $\mathcal{G}^k[\delta - k + 1]$ be the last visited element at the termination of the algorithm. If $test(\mathcal{G}^k[\delta - k + 1]) = true$, then $\forall i \in [1, \delta - k]$, $test(G^k[i]) = true$ (all elements in the row $\mathcal{G}^k$).*

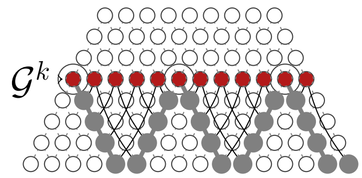


Fig. 3: Example of the execution of the algorithm for the decision variant.

Proof. According to the algorithm, for any element $G_{(i,j)}$ above (below) the walk in the minimization (maximization) case, there exists a computed element $G_{(i',j')}$ in the walk such that $i' \leq i \wedge j' \geq j$ ($i' \geq i \wedge j' \leq j$) and $test(G_{(i',j')}) = true$. According to Observation 2 (requirements (3') and (3")), any element above (below) the walk in the minimization (maximization) case satisfies the test operation. □

Decision variant. The algorithm for the decision variant of each problem (*i.e.* for a given k, answer *true* if any sequence of length k in the dynamic graph $\mathcal{G}$ has the property P, answer $false$ otherwise) can be deduced readily from the algorithm for the minimization/maximization variant. The algorithm gradually computes elements of row k from left to right, starting at $\mathcal{G}^k[1]$, as shown in Figure 3. If an element that does not satisfy the test operation is found, the algorithm returns $false$ and terminates. If the algorithm reaches the last element in the row, i.e. $\mathcal{G}^k[\delta - k + 1]$, and it satisfies the test operation, then it returns *true*. The elements $\mathcal{G}^k[1], \mathcal{G}^k[2], \ldots, \mathcal{G}^k[\delta - k + 1]$ are computed based on ladders.

Theorem 1. *The generic algorithm has a cost of O(δ) composition and test operations.*

Proof. The ranges of the indices covered by the left ladders that are constructed by the process are disjoint, so their total length is O(δ). With the computation of each new element in a right ladder, the walk moves closer to the right side of the hierarchy, so the total length of the right ladders is also O(δ). According to Lemma 2, any element can be computed using a single composition operation based on ladders. According to the algorithm, the number of elements computed by the walk is O(δ) and any computed element is tested at most once. This establishes that this algorithm has a cost of O(δ) composition operations and test operations. □

Online version. The generic algorithm can be adapted to an online setting in which the sequence of graphs $G_1, G_2, G_3, \ldots$ of a dynamic graph $\mathcal{G}$ is processed in the order that the graphs are received. For the decision problem, the algorithm cannot provide an answer until at least k graphs have been received. When the k^{th} graph is received, the algorithm builds the first left ladder using $k-1$ compositions. It can then perform a test and answer whether or not the sequence has the property so far. After this initial period, a test can be performed for the k most recently received graphs (by performing a test on the corresponding element in row T) after the receipt of each new graph. The same logic is followed for minimization and maximization problems.

Theorem 2. *The online generic algorithm has an amortized cost of O(1) composition and test operations per graph received.*

Proof. At no time during the execution of the algorithm does the number of compositions performed to build left ladders exceed the number of graphs received and the same is true for right ladders. The number of elements on the walk that are not on ladders never exceeds the number of graphs received, and each can be computed with one composition by Lemma 2. Only elements on the walk are tested. In summary, the amortized cost is O(1) composition and test operations for each graph received. □

4 Illustration of the Framework

We illustrate the general framework by solving one maximization problem: Interval-Connectivity and three minimization problems: Bounded-Realization-of-the-Footprint, Temporal-Diameter, and Round-Trip-Temporal-Diameter. We define each problem within the framework and provide the corresponding operations for composition and test.

4.1 *T*-Interval Connectivity (maximization)

A dynamic graph $\mathcal{G}$ is said to be *T-interval connected* if for any $t \in [1, \delta - T + 1]$ all graphs in $\{G_t, G_{t+1}, ..., G_{t+T-1}\}$ share a common connected spanning subgraph. We consider the problem Interval-Connectivity of finding the smallest duration T for which the dynamic graph $\mathcal{G}$ is T-interval connected.

Composition and test operations. By using the *intersection* of two elements as the composition operation (starting with $\{G_{(i,i)}\} = \{G_i\}$), a hierarchy of intersection graphs (Figure 1) as elements can be used to solve Interval-Connectivity which is the problem of finding the highest row $\mathcal{G}^T$ in which every element $\mathcal{G}^T[i]$, $i \in [1, \delta - T + 1]$, is connected. So, the composition operation is *intersection* and the test operation is *connectivity test*.

Observation 3 (Cost of the operations). *Using an adjacency list data structure, the binary intersection of two elements $G_{(i,j)}$ and $G_{(i',j')}$ can be computed in time $O(\min(|E(G_{(i,j)})|, |E(G_{(i',j')})|))$. Testing connectivity of an undirected graph can also be done in time $O(|E(G_{(i,j)})|)$ by building a depth-first search tree from an arbitrary node to test whether all nodes are reachable.*

4.2 Bounded Realization of the Footprint (minimization)

The *footprint* G of a dynamic graph $\mathcal{G}$ is the graph that contains all the edges that appear at least once, that is $\cup\{G_1, G_2, ..., G_\delta\}$. We consider the problem of finding the smallest duration b such that in any window of length b, all edges of G appear at least once (BOUNDED-REALIZATION-OF-THE-FOOTPRINT). The problem then amounts to finding the lowest row $\mathcal{G}^b$ in which every element $\mathcal{G}^b[i]$, $i \in [1, \delta - b + 1]$, equals the footprint G.

Composition and test operations. Finding these operations is straightforward. By taking the *union* of two elements as the composition operation (starting with $G_{(i,i)} = G_i$), it follows that the lowest row $\mathcal{G}^b$ such that all elements *equal* the footprint indicates, by definition, that the answer is b. So, the composition operation is *union* and the test operation is *equality to footprint*.

Observation 4 (Cost of the operations). *Using an adjacency matrix representation, the union operation and the equality test can be performed in $O(|V|^2)$ time.*

4.3 Temporal Diameter (minimization)

A dynamic graph might never be connected at one time, and yet offer a form of connectivity over time based on journeys (temporal paths). Informally, a journey is a path whose edges are crossed at non-decreasing (or increasing) times, with possible pauses at intermediate nodes. The edges need not be all present simultaneously. If at most one edge can be crossed at a time (*i.e.* the crossing times are strictly increasing), then we refer to the journey as being *strict*. Formally, journeys can be defined in various ways, depending on the graph formalism used. In sequence-based models like evolving graphs, it is defined as follows.

Definition 6 (Journey). *A* journey $\mathcal{J}$ *from u to v in $\mathcal{G}$ is a sequence of edges $e_1, e_2, \ldots, e_p$ connecting u to v through intermediate vertices and a corresponding sequence of non-decreasing indices $t_1, t_2, \ldots, t_p$ such that $e_i \in E(G_{t_i})$. In a* strict journey*, the sequence $t_1, t_2, \ldots, t_p$ is strictly increasing. The existence of a journey from u to v is denoted $u \leadsto v$. We note* $\text{departure}(\mathcal{J}) = t_1$ *and* $\text{arrival}(\mathcal{J}) = t_p$.

The distinction between strict and non-strict journeys actually boils down to deciding if the latency of communication is neglected or not. In either case, one can define the concept of *temporal diameter* (at time t) as the smallest d such that for all nodes u and v, there exists a journey from u to v in the sequence $\{G_t, G_{t+1}, ..., G_{t+d-1}\}$. We consider here the problem TEMPORAL-DIAMETER of finding the smallest d such that the *temporal diameter* of $\mathcal{G}$ is less than or equal to d at *every* time $t \le \delta - d + 1$. In other words, any subsequence of $\mathcal{G}$ of length d is temporally connected. Several solutions exist for this and similar problems (see e.g. [24]), which operate at a lower level of abstraction. Here, we show how the problem fits elegantly within our proposed framework. More specifically, we consider the case of *non-strict* journeys, which is slightly more difficult and contains as a subproblem the case of strict journeys.

Definition 7 (Transitive closure). *The* transitive closure *of the dynamic graph $\mathcal{G}$ is the static directed graph $\mathcal{G}^* = (V, E^*)$ such that $(u, v) \in E^* \Leftrightarrow u \leadsto v$.*

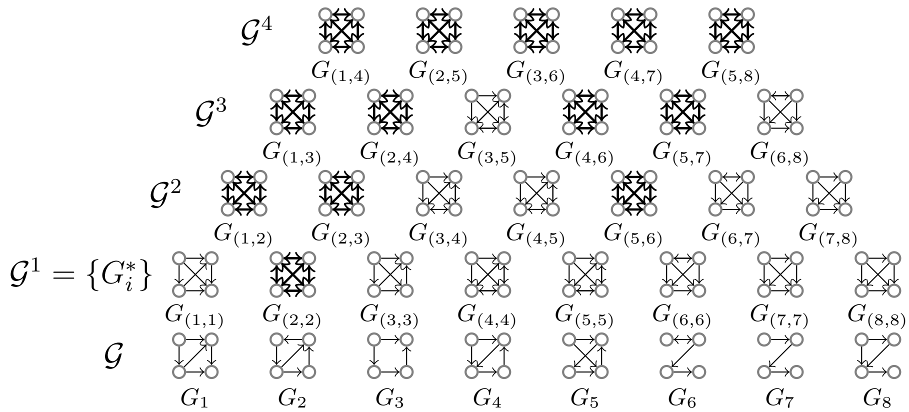


Fig. 4: Example of a transitive closure hierarchy for a given dynamic graph $\mathcal{G}$ of length $\delta = 8$.

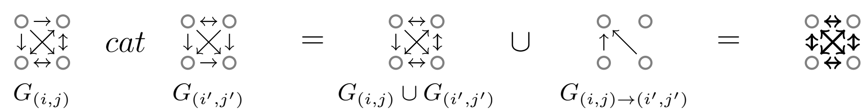


Fig. 5: Example of concatenation of transitive closures. *Edges in* $G_{(i,j)\to(i',j')}$ *are added after the union.*

The composition hierarchy built here is one of *transitive closures* of journeys. Figure 4 shows an example. For this problem, each bottom element $G_{(i,i)}$ is not equal to G_i; instead, it corresponds to the "classical" *transitive closure* of G_i, i.e. the graph G_i^* built on the same vertex set as G_i, such that an *edge* exists between u and v in G_i^* if and only if a *path* exists between u and v in G_i (the $G_{(i,i)}$'s are computed gradually as the algorithm progresses). Then, the answer is the smallest d such that every element in row $\mathcal{G}^d$ of the hierarchy is a complete graph (i.e. every subsequence of $\mathcal{G}$ of length d is temporally connected).

Composition and test operations. The composition hierarchy is built using *concatenation* of transitive closures, $cat(G_{(i,j)}, G_{(i',j')})$, with the restriction that $i' = j + 1$ (Lemma 3), defined as follows. First compute the union of both elements, then add an *additional edge* (u, v) if there exists a node w such that $(u, w) \in E(G_{(i,j)})$ and $(w, v) \in E(G_{(i',j')})$. See Figure 5 for an example. Then, the *test* operation consists of determining if an element of the hierarchy (transitive closure) is a complete graph.

Observation 5 (Cost of the operations). *The union of two transitive closures* $G_{(i,j)}$ *and* $G_{(i',j')}$ *can be computed in time* $O(\max(|E(G_{(i,j)})|, |E(G_{(i',j')})|))$ *using an adjacency list data structure. The cost of the concatenation operation is dominated by the computation of the additional edges which costs* $O(|E(G_{(i',j')})| \cdot |V|)$*. The completeness test of a transitive closure* $G_{(i,j)}$ *can* *be done in constant time by checking $|E(G_{(i,j)})|$ which is maintained during the construction of the transitive closure graph.*

4.4 Round-Trip Temporal Diameter (minimization)

We address here the more complex property of *round-trip temporal connectivity* defined by the existence of a back-and-forth journey from any node to all other nodes. The *round-trip temporal diameter* of a graph $\mathcal{G}$ at time t is the smallest d such that, in the sequence $\{G_t, G_{t+1}, ..., G_{t+d-1}\}$, there is a journey $\mathcal{J}(u,v)$ from any node u in the graph to any other node v *and* a journey $\mathcal{J}'(v,u)$ from v to u which starts after the arrival of the journey $\mathcal{J}(u,v)$. This does not mean that there is simply a succession of two temporally connected sequences. A back-and-forth journey from a node u to a node v can finish before a back-and-forth journey from a node u' to a node v' starts. Also, the time intervals of the two back-and-forth journeys can overlap. We consider the problem Round-Trip-Temporal-Diameter of finding the smallest d such that the *round-trip temporal diameter* of $\mathcal{G}$ is less than or equal to d at any time $t \leq \delta - d + 1$. For this problem, we consider the case of non-strict journeys.

Definition 8 (Round trip transitive closure). *A* round trip transitive closure $G_{(i,j)}$ *is the directed graph where $(u,v) \in G_{(i,j)}$ iff at least one journey $u \rightsquigarrow v$ exists in the sequence $\{G_i, G_{i+1}, ..., G_j\}$. The edges $\{(u,v) \in E(G_{(i,j)})\}$ are labelled with two times:* $\mathrm{arrival}(u,v,G_{(i,j)})$ *is the earliest arrival of any journey in the sequence and* $\mathrm{departure}(u,v,G_{(i,j)})$ *is the latest departure of any journey in the sequence. Labels on the same edge may or may not be the departure and arrival times of the same journey. Note that $departure(u,v,G_{(i,i)}) = i$ and $arrival(u,v,G_{(i,i)}) = i$.*

The composition hierarchy built for this problem is one of *round trip transitive closures* of journeys. Figure 6 shows an example of a round trip transitive closure hierarchy of a dynamic graph $\mathcal{G}$ of length $\delta = 3$. Labels *arr* and *dep* on an edge $u \xrightarrow{arr,dep} v$ (label on the destination/head end) represent respectively $arrival(u,v,G_{(i,j)})$ and $departure(u,v,G_{(i,j)})$. As for Temporal-Diameter, each bottom element $G_{(i,i)}$ corresponds to the "classical" *transitive closure* of G_i, i.e. the graph G_i^* built on the same vertex set as G_i, such that an *edge* exists between u and v in G_i^* if and only if a *path* exists between u and v in G_i. The labels of the edges are initialized with "i,i", which corresponds to the arrival and departure times of the corresponding journey(s). Then, the answer is the smallest d such that every element in row $\mathcal{G}^d$ is a complete graph (i.e. every subsequence of $\mathcal{G}$ of length d is round-trip temporally connected).

Composition operation. The composition operation in this case is the *concatenation of round trip transitive closures* $rtcat(G_{(i,j)}, G_{(i',j')})$ with the restriction that $i' = j + 1$ (Lemma 3). A composition is computed as follows. First, compute the graph $G^{\cup\circlearrowleft} = G_{(i,j)} \cup^{\circlearrowleft} G_{(i',j')}$ which is the union graph $G_{(i,j)} \cup G_{(i',j')}$ with $arrival(u,v,G^{\cup\circlearrowleft}) = \min(arrival(u,v,G_{(i,j)}), arrival(u,v,G_{(i',j')}))$ and $departure(u,v,G^{\cup\circlearrowleft}) = \max(departure(u,v,G_{(i,j)}), departure(u,v,G_{(i',j')})$ if

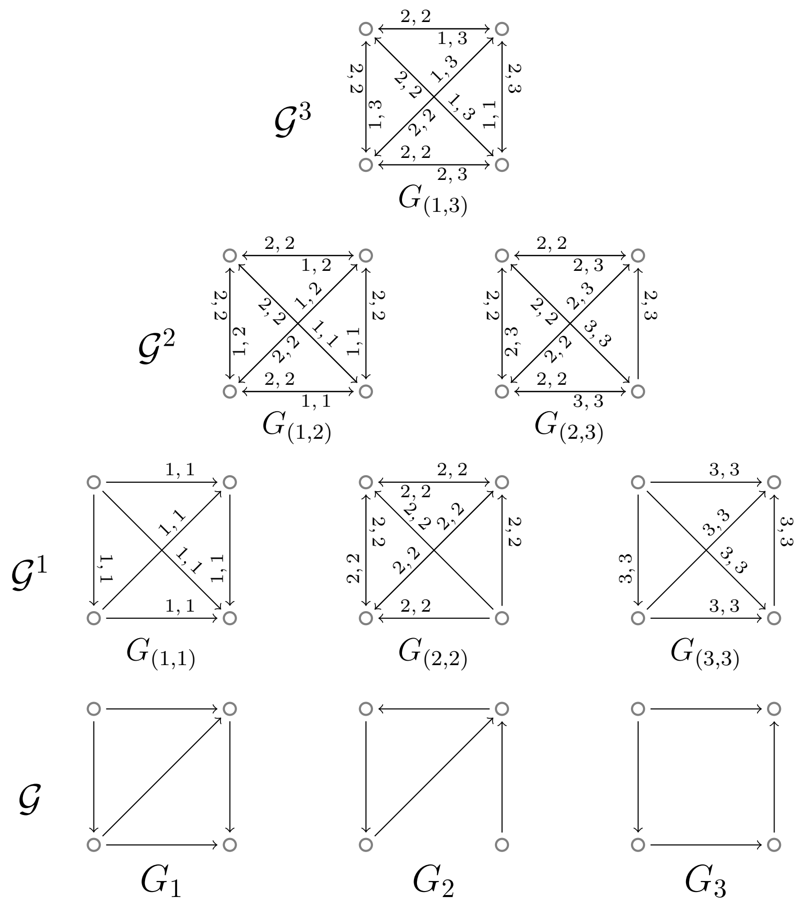


Fig. 6: Example of a round trip transitive closure hierarchy of a dynamic graph $\mathcal{G}$ of length $\delta = 3$. (*Arrival and departure times are on the head ends of the arrows.*)

$(u, v) \in G_{(i,j)} \cap G_{(i',j')}$. Otherwise, the edge is added with the initial arrival and departure times. A graph of *extra edges* $G_{(i,j)\to(i',j')}$ is then computed as follows: $(u, v) \in G_{(i,j)\to(i',j')}$ iff there exists a non-empty set of nodes $extra = \{w : (u, w) \in E(G_{(i,j)}) \text{ and } (w, v) \in E(G_{(i',j')})\}$. The labels on an extra edge are $arrival(u, v, G_{(i,j)\to(i',j')}) = \min_{w \in extra}\{arrival(w, v, G_{(i,j)})\}$ and $departure(u, v, G_{(i,j)\to(i',j')}) = \max_{w \in extra}\{departure(u, w, G_{(i,j)})\}$. Finally, the round trip transitive closure $rtcat(G_{(i,j)}, G_{(i',j')}) = G^{\cup\circlearrowleft} \cup^{\circlearrowleft} G_{(i,j)\to(i',j')}$ (see Figure 7).

Test operation. The test operation used for this problem is the *round trip completeness test*, that is, test if the graph is complete and if $arrival(u, v, G_{(i,j)}) \leq departure(v, u, G_{(i,j)})$ for every edge (u, v) in the graph.

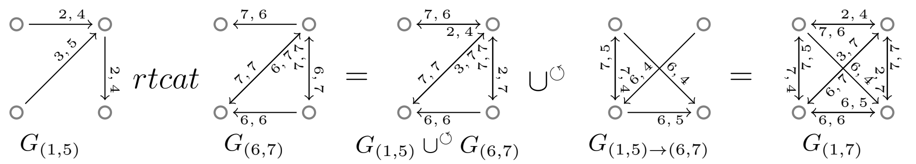


Fig. 7: Example of round trip transitive closures concatenation. (*Arrival and departure times are on the head ends of the arrows.*)

Observation 6 (Cost of the operations). *As for the concatenation operation for* Temporal-Diameter*, the concatenation of two round trip transitive closures* $G_{(i,j)}$ *and* $G_{(i',j')}$ *can be computed in time* $O(|E(G_{(i',j')})| \cdot |V|)$*. The completeness test can be done in time* $O(|E(G_{(i,j)})|)$ *by verifying the condition on the times for each pair of edges* (u, v)*,* (v, u)*.*

4.5 Parallel Version

We define a subset of particular minimization and maximization problems that we call *symmetric problems* as follows.

Definition 9 (Symmetric problems). *A minimization or maximization problem is* symmetric *if it can be solved using a composition hierarchy of elements and a composition operation* $\circ$ *such that* $G_{(i,j)} \circ G_{(i',j')} = G_{(i,j')}$ *for all* $1 \leq i \leq i' \leq j \leq j' \leq \delta$*.*

Bounded-Realization-of-the-Footprint and T-Interval-Connectivity are examples of *symmetric problems*. We now present a strategy for *symmetric problems* that can be parallelized on a PRAM. We first describe the algorithms for a sequential machine (RAM). The general strategy is to compute only some of the rows of the composition hierarchy based on the following lemma. The proofs of Lemma 5 and Lemma 6 are straightforward generaliztions of proofs in [10].

Lemma 5. *If some row* $\mathcal{G}^k$ *is already computed, then any row* $\mathcal{G}^\ell$ *for* $k + 1 \leq \ell \leq 2k$ *can be computed with* $O(\delta)$ *composition and test operations.*

Decision variant. Using Lemma 5, for a given k, we can incrementally compute rows $\mathcal{G}^{2^i}$ ("power rows") for all i from 1 to $\lceil \log_2 k \rceil - 1$ without computing the intermediate rows. Then, we compute row $\mathcal{G}^k$ directly from row $\mathcal{G}^{2^{\lceil \log_2 k \rceil - 1}}$ (again using Lemma 5). This way, we compute $\lceil \log_2 k \rceil = O(\log \delta)$ rows using $O(\delta \log \delta)$ composition operations, after which we perform $O(\delta)$ tests.

Minimization and maximization variants. For the maximization case, we incrementally compute rows $\mathcal{G}^{2^i}$ until we find a row that contains an element that does not satisfy the test operation (thus, a test is performed after each composition). By Lemma 5, each of these rows can be computed using $O(\delta)$ compositions. Suppose that row $\mathcal{G}^{2^{j+1}}$ is the first power row that contains an element that does not satisfy the test, and that $\mathcal{G}^{2^j}$ is the row computed before $\mathcal{G}^{2^{j+1}}$. Next, we do a binary search among the rows between $\mathcal{G}^{2^j}$ and $\mathcal{G}^{2^{j+1}}$ to find the highest row $\mathcal{G}^k$ such that all elements on this row satisfy the test. See Figure 8 (left) for an illustration of the algorithm. The computation of each of these rows is based on row $\mathcal{G}^{2^j}$ and uses $O(\delta)$ compositions by Lemma 5. Overall, we compute at most $2\lceil \log_2 k \rceil = O(\log \delta)$ rows using $O(\delta \log \delta)$ compositions and the same number of tests.

For the minimization case, we follow the same principle. This time, we incrementally compute rows $\mathcal{G}^{2^i}$ while each row contains an element that does not satisfy the test. Suppose that row $\mathcal{G}^{2^{j+1}}$ is the first power row such that all elements on this row satisfy the test. Then, we do a binary search among the rows between $\mathcal{G}^{2^j}$ and $\mathcal{G}^{2^{j+1}}$ to find the lowest row $\mathcal{G}^k$ such that all elements on this row satisfy the test. See Figure 8 (right) for an illustration of the algorithm.

Lemma 6. *If some row $\mathcal{G}^k$ is already computed, then any row between $\mathcal{G}^{k+1}$ and $\mathcal{G}^{2k}$ can be computed in $O(1)$ time on an EREW PRAM with $O(\delta)$ processors.*

Parallel version for the decision problems on an EREW PRAM. The sequential algorithm for this problem computes $O(\log \delta)$ rows. By Lemma 6, each of these rows can be computed in $O(1)$ time on an EREW PRAM with $O(\delta)$ processors. Therefore, all of the rows (and hence all necessary compositions) can be computed in $O(\log \delta)$ time with $O(\delta)$ processors. The $O(\delta)$ tests for row $\mathcal{G}^k$ can be done in $O(1)$ time with $O(\delta)$ processors. Then, the processors can establish whether or not all elements in row $\mathcal{G}^k$ satisfy the test operation by computing the logical AND of the results of the $O(\delta)$ tests in time $O(\log \delta)$ on a EREW PRAM with $O(\delta)$ processors using standard techniques (see [16, 19]). The total time is $O(\log \delta)$ on an EREW PRAM with $O(\delta)$ processors.

Parallel version for maximization and minimization problems on an EREW PRAM. The sequential algorithm for this problem computes $O(\log \delta)$ rows. Differently from the decision version, a test is done for each of the computed elements (rather than just those of the last row) and it has to be determined for each computed row whether or not all of the elements satisfy the test. This takes $O(\log \delta)$ time for each of the $O(\log \delta)$ computed rows using the same techniques as for the decision version. The total time is $O(\log^2 \delta)$ on an EREW PRAM with $O(\delta)$ processors.

5 Conclusions

In this paper, we generalized the framework and the algorithm for Interval-Connectivity [10] to solve other problems on dynamic graphs. We studied the minimization problems of finding the temporal diameter and the round trip temporal diameter of a given dynamic graph $\mathcal{G} = \{G_1, G_2, ..., G_\delta\}$, and the problem of finding a bound on the footprint realization of $\mathcal{G}$. We proposed algorithms for these problems within the same framework.

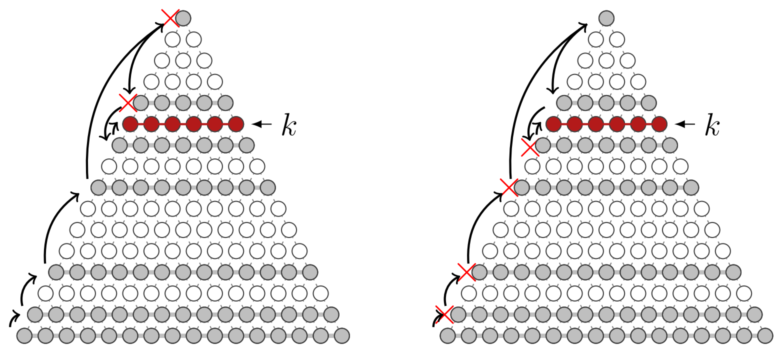


Fig. 8: Examples of the execution of the parallel version of the algorithm; *maximization case on the left and minimization case on the right*.

In our study, we focused on algorithms using only two elementary operations, *composition* and *test* operations. This approach is suitable for a high-level study of these problems when the details of changes between successive graphs in a sequence are arbitrary. If the evolution of the dynamic graph is constrained in some ways (e.g., bounded number of changes between graphs), then one could benefit from the use of more sophisticated data structures to reduce the complexity of the algorithms.

A natural extension of our investigation would be a similar study for other classes and properties of dynamic networks, as identified in [9].

References

1. Aaron, E., Krizanc, D., Meyerson, E.: DMVP: foremost waypoint coverage of time-varying graphs. In: International Workshop on Graph-Theoretic Concepts in Computer Science. pp. 29–41. Springer (2014)
2. Awerbuch, B., Even, S.: Efficient and reliable broadcast is achievable in an eventually connected network. In: Proceedings of the third annual ACM symposium on Principles of distributed computing (PODC). pp. 278–281. ACM (1984)
3. Barjon, M., Casteigts, A., Chaumette, S., Johnen, C., Neggaz, Y.M.: Testing temporal connectivity in sparse dynamic graphs. CoRR abs/1404.7634, 8p (2014), (A French version appeared in Proc. of ALGOTEL 2014.)
4. Bournat, M., Datta, A., Dubois, S.: Self-stabilizing robots in highly dynamic environments. In: SSS 2016 - 18th International Symposium Stabilization, Safety, and Security of Distributed Systems. Lecture Notes in Computer Science, vol. 10083, pp. 54–69. Springer (2016)
5. Bramas, Q., Tixeuil, S.: The complexity of data aggregation in static and dynamic wireless sensor networks. Information and Computation (2016)
6. Braud-Santoni, N., Dubois, S., Kaaouachi, M.H., Petit, F.: The next 700 impossibility results in time-varying graphs. International Journal of Networking and Computing 6(1), 27–41 (2016)
7. Bui-Xuan, B., Ferreira, A., Jarry, A.: Computing shortest, fastest, and foremost journeys in dynamic networks. Int. J. of Foundations of Computer Science 14(2), 267–285 (April 2003)

8. Casteigts, A., Chaumette, S., Ferreira, A.: Characterizing topological assumptions of distributed algorithms in dynamic networks. In: Proc. 16th Int. Colloquium on Structural Information and Communication Complexity (SIROCCO 2009). Lecture Notes in Computer Science, vol. 5869, pp. 126–140. Springer (2009), (Full version in *CoRR, abs/1102.5529*.)
9. Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and dynamic networks. Int. J. of Parallel, Emergent and Distributed Systems 27(5), 387–408 (2012)
10. Casteigts, A., Klasing, R., Neggaz, Y., Peters, J.: Efficiently testing T-interval connectivity in dynamic graphs. In: Proc. 9th Int. Conf. on Algorithms and Complexity (CIAC 2015). Lecture Notes in Computer Science, vol. 9079, pp. 89–100. Springer (2015)
11. Casteigts, A., Flocchini, P., Mans, B., Santoro, N.: Measuring temporal lags in delay-tolerant networks. IEEE Transactions on Computers 63(2), 397–410 (2014)
12. Casteigts, A., Flocchini, P., Mans, B., Santoro, N.: Shortest, fastest, and foremost broadcast in dynamic networks. International Journal of Foundations of Computer Science 26(4), 499–522 (2015)
13. Casteigts, A., Klasing, R., Neggaz, Y.M., Peters, J.G.: Calcul de Paramètres Minimaux dans les Graphes Dynamiques. In: 19èmes Rencontres Francophones sur les Aspects Algorithmiques de Télécommunications (ALGOTEL) (2017)
14. Dubois, S., Kaaouachi, M.H., Petit, F.: Enabling minimal dominating set in highly dynamic distributed systems. In: Symposium on Self-Stabilizing Systems. pp. 51–66. Springer (2015)
15. Flocchini, P., Mans, B., Santoro, N.: On the exploration of time-varying networks. Theoretical Computer Science 469, 53–68 (2013)
16. Gibbons, A., Rytter, W.: Efficient parallel algorithms. Cambridge University Press (1988)
17. Godard, E., Mazauric, D.: Computing the dynamic diameter of non-deterministic dynamic networks is hard. In: Proc. 10th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS 2014). Lecture Notes in Computer Science, vol. 8847, pp. 88–102. Springer (2014)
18. Jain, S., Fall, K., Patra, R.: Routing in a delay tolerant network. In: Proc. of SIGCOMM. pp. 145–158 (2004)
19. JáJá, J.: An introduction to parallel algorithms. Addison-Wesley (1992)
20. Kuhn, F., Lynch, N., Oshman, R.: Distributed computation in dynamic networks. In: Proc. of STOC. pp. 513–522. ACM (2010)
21. O'Dell, R., Wattenhofer, R.: Information dissemination in highly dynamic graphs. In: Proc. of DIALM-POMC. pp. 104–110. ACM (2005)
22. Raynal, M., Stainer, J., Cao, J., Wu, W.: A simple broadcast algorithm for recurrent dynamic systems. In: Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on. pp. 933–939. IEEE (2014)
23. Viard, T., Latapy, M., Magnien, C.: Computing maximal cliques in link streams. Theoretical Computer Science 609, 245–252 (2016)
24. Whitbeck, J., Dias de Amorim, M., Conan, V., Guillaume, J.L.: Temporal reachability graphs. In: Proc. of MOBICOM. pp. 377–388. ACM (2012)