Computing Parameters of Sequence-Based Dynamic Graphs

Arnaud Casteigts, Ralf Klasing, Yessin M Neggaz, Joseph Peters

Theory of Computing Systems manuscript No.
(will be inserted by the editor)

Computing Parameters of Sequence-based Dynamic Graphs

Arnaud Casteigts · Ralf Klasing · Yessin M. Neggaz · Joseph G. Peters

Received: date / Accepted: date

Abstract We present a general framework for computing parameters of dynamic networks which are modelled as a sequence $\mathcal{G} = (G_1, G_2, \ldots, G_\delta)$ of static graphs such that $G_i = (V, E_i)$ represents the network topology at time i and changes between consecutive static graphs are arbitrary. The framework 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 framework allows us to compute different parameters of dynamic graphs using a common high-level strategy by using composition and test operations that are specific to the parameter. The resulting algorithms are optimal in the sense that they use only $O(\delta)$ composition and test operations, where δ is the length of the sequence. We illustrate our framework with three minimization problems, *bounded realization of the footprint, temporal diameter,* and *round trip temporal diameter,* and with *T-interval connectivity* which is a maximization problem. We prove that the problems are in **NC** by presenting polylogarithmic-time parallel versions of the algorithms. Finally, we show that the algorithms can operate online with amortized complexity $\Theta(1)$ composition and test operations for each graph in the sequence.

Preliminary results concerning this work have been presented at CIAC 2015 [13] and at SIROCCO 2017 [14]. Part of this work was done while Joseph Peters was visiting the LaBRI as a guest professor of the University of Bordeaux (IdEx Bordeaux - ANR-10-IDEX-03-02). This work was also partially funded by ANR projects DISPLEXITY (ANR-11-BS02-014), ESTATE (ANR-16-CE25-0009-03), and NSERC of Canada.

A. Casteigts · R. Klasing
LaBRI, CNRS, University of Bordeaux, France
E-mail: arnaud.casteigts@labri.fr, ralf.klasing@labri.fr

Y. M. Neggaz
LAAS-CNRS, INSA-Toulouse, Toulouse, France
E-mail: yessin.neggaz@gmail.com

J. G. Peters
School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
E-mail: peters@cs.sfu.ca

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 [24]; others are always disconnected [19] 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 [12]. Each of these classes corresponds to specific properties which play a role either in the complexity or in the feasibility of distributed problems. For example, it was shown in [11] that if the edges are *recurrent* (*i.e.* if an edge appears once, then it will reappear infinitely often), denoted class $\mathcal{R}$, then this property guarantees the feasibility of a certain type of optimal broadcast with termination detection called *foremost* broadcast (the time of delivery of the message at each node is minimized). However, it is not sufficient to satisfy other measures of optimality, such as *shortest* broadcast (the number of hops for the message to reach each node is minimized), or *fastest* broadcast (the time for the completion of the broadcast is minimized). Strengthening the property to require a *bound* on the reappearance time (class $\mathcal{B}$) makes it possible to achieve shortest broadcast, and the even stronger property of having *periodic* edges (class $\mathcal{P}$) enables fastest broadcast. These three classes have been shown to play roles in a variety of problems (see *e.g.* [1, 16, 25]). 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 temporal paths called *journeys*), corresponding to class $\mathcal{C}_5$ in the hierarchy of [12]. 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 [5–7, 15]. 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 which related properties 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 focusing on testing properties (or computing structures) in dynamic graphs. A seminal example is the computation of foremost,

shortest, and fastest journeys [8]. The algorithm for foremost journeys in [8] is a temporal adaptation of Dijkstra's algorithm. Such an algorithm can be used to test membership in a number of dynamic graph classes [9] that are characterized by the existence of journeys including the class of temporally connected networks. Other algorithms for testing temporal connectivity include [3] and [28], both of which compute reachability graphs (transitive closure of journeys), the second being more general than the first, but also more costly. The algorithm in [3] is closer in spirit to the work in this paper in that the considered model is a sequence of graphs processed one after the other.

One of the four metrics that we consider in this paper, *temporal diameter* (*i.e.,* how long it takes in the worst case to communicate through journeys), is a generalization of temporal connectivity. Godard and Mazauric [18] showed that computing the temporal diameter is hard when the sequence of static graphs is unknown but each graph is required to be sampled from a given family of graphs, with various levels of (in)approximability depending on restrictions of the family. In contrast, each of our algorithms processes a known sequence of static graphs and has a tractable complexity for the considered metrics. Other examples of work related to the general problem of testing properties of dynamic graphs include enumerating maximal cliques [27] and treewidth-based algorithms [22].

In this paper, we present a general framework for computing parameters of dynamic networks which are modelled as a sequence $\mathcal{G} = (G_1, G_2, \ldots, G_\delta)$ of static graphs such that $G_i = (V, E_i)$ represents the network topology at time i and changes between consecutive static graphs are arbitrary. The framework 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 framework allows us to compute different parameters of dynamic graphs using a common high-level strategy by using composition and test operations that are specific to the parameter. The resulting algorithms are optimal in the sense that they use only $O(\delta)$ composition and test operations, where δ is the length of the sequence. The technique is based on a walk in a *composition hierarchy*, the elements of which are graphs that represent the compositions of various subsequences of $\mathcal{G}$. The *composition operation* defines a *composition hierarchy* in which the walk is performed, and the *test operation* is used by the algorithm to construct 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 minimization problems concerning temporal properties of recognized importance.

The maximization problem we study is the class of dynamic graphs which are *T-interval connected*. *T-interval connectivity* [21] 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 [12]) generalizes the class of dynamic graphs that are connected at all times [24] (Class $\mathcal{C}_9$ in [12]). The definition of T-interval connectivity is closely related to a representation of a network as a sequence of graphs $\mathcal{G} = (G_1, G_2, \ldots, G_\delta)$ which correspond to the state of the topology at increasing times (also called *untimed* evolving graphs [8]). Informally, T-interval connectivity requires that, for every T consecutive graphs in

$\mathcal{G}$, there exists a common connected spanning subgraph. We consider the maximization problem of finding the largest T such that a given dynamic graph $\mathcal{G}$ is T-interval connected. We also address the related decision problem of testing if $\mathcal{G}$ is T-interval connected for a given T.

The first minimization problem that we consider is 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 [10]. We consider the problem 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 dynamic graph $\mathcal{G}$ 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* in which 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 other nodes. The concept of temporal connectivity is relatively old and dates back at least to the article [2]. We consider the minimization problem of finding the *temporal diameter* of a given dynamic graph $\mathcal{G}$, *i.e.* the shortest duration in which there exist journeys (temporal paths) from any node to all other nodes.

The third minimization problem concerns the class of dynamic graphs with *round-trip temporal connectivity* meaning that back-and-forth journeys exist between every pair of nodes. This class characterizes an important property of distributed solutions for information collection problems that require termination detection [12]. We investigate the problem of computing the *round trip temporal diameter* of a given graph $\mathcal{G}$, *i.e.* the shortest duration in which there exist back-and-forth journeys between every pair of nodes.

The paper is organized as follows. Section 2 contains definitions and some basic results including observations about the structure of the problems and simple upper and lower bounds on the numbers of composition and test operations. The general framework and its properties are presented in Section 3. The framework is applied to solve four problems in Section 4. Finally, Section 5 discusses parallel versions of the algorithms and establishes that the problems are in **NC** (solvable in polylogarithmic-time on a parallel machine).

2 Definitions and Observations

Let $\mathcal{G}$ be a dynamic network modelled as a sequence $\mathcal{G} = \{G_1, G_2, \ldots, G_\delta\}$ of static graphs such that $G_i = (V, E_i)$ represents the network topology at time i. We assume that the changes between two consecutive graphs in the sequence are arbitrary. The parameter δ is called the *length* or *lifetime* of the dynamic graph $\mathcal{G}$. Observe that V is invariant; only the set of edges varies.

Let P be a boolean predicate (hereafter called *property*) defined on a consecutive subsequence $\{G_i, G_{i+1}, \ldots, G_j\} \subseteq \mathcal{G}$.

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

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

Definition 3 The *decision problem on $\mathcal{G}$ with respect to P* and a fixed value k is the problem of deciding whether $\{G_i, G_{i+1}, \ldots, G_{i+k-1}\}$ satisfies property P, $\forall i \in [1, \delta - k + 1]$.

We will use a general framework for computing parameters of dynamic graphs. The static graphs in the sequence are manipulated as atomic elements with two types of operations: a *composition operation* and a *test operation*. Algorithms to compute specific parameters are obtained by using appropriate composition and test operations in the framework. The strategy is based on a walk in a *composition hierarchy of elements* which are computed on demand using the composition operation. The test operation is used to determine which elements of the hierarchy are computed during construction of the walk. In this paper, we will focus mainly on maximization and minimization problems. Algorithms for decision problems are easily derived from the related optimization algorithms. Unless otherwise stated, the edges of the static graphs are *undirected*. However, given appropriate composition and test operations, exactly the same framework can be applied to directed static graphs, and the static graphs in two of the examples in Section 4 are directed.

Definition 4 (Composition operation) A *composition operation* $\circ$ is a binary operation that maps two graphs G' and G'' with the same vertex set into another graph $G' \circ G''$. A composition operation $\circ$ is *associative* if $(G' \circ G'') \circ G''' = G' \circ (G'' \circ G''')$ for all graphs G', G'', G'''. Given an associative composition operation $\circ$ and a subsequence $\{G_i, G_{i+1}, \ldots, G_j\}$, $i \leq j$ of a dynamic graph, we use $G_{(i,j)}$ to denote the graph $G_i \circ G_{i+1} \circ \ldots \circ G_j$.

Definition 5 (Test operation) A *test operation* with respect to a property P maps a graph $G_{(i,j)}, i \leq j$, into $\{true, false\}$ such that $test(G_{(i,j)}) = true$ iff the sequence $\{G_i, G_{i+1}, \ldots, G_j\}$ satisfies property P.

Definition 6 (Composition hierarchy) The elements of a *composition hierarchy* $\mathcal{H}$ for a dynamic graph $\mathcal{G} = \{G_1, G_2, \ldots, G_\delta\}$ and associative composition operation $\circ$ are graphs $G_{(i,j)}$, $i \leq j$, arranged into rows $\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^\delta$ 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 $\mathcal{G}^k[i] = G_{(i,i+k-1)}$. The first row $\mathcal{G}^1 = \{G_{(1,1)}, G_{(2,2)}, \ldots, G_{(\delta,\delta)}\}$ of $\mathcal{H}$ corresponds to the static graphs of the sequence $\mathcal{G}$ (or to simple transformations of these graphs); that is, $G_{(i,i)}$ corresponds to G_i.

Note that each graph $G_{(i,i+k-1)}$ in a hierarchy can be computed as $G_{(i,i)} \circ G_{(i+1,i+k-1)}$, and that $G_{(i,j)} \circ G_{(i',j')} = G_i \circ G_{i+1} \circ \ldots \circ G_j \circ G_{i'} \circ G_{i'+1} \circ \ldots \circ G_{j'}$, $i \leq j$ and $i' \leq j'$. An example of a hierarchy for which the composition operation

is graph intersection is shown in Figure 1. Other examples of composition operations used in this paper are the *union* of graphs and the *concatenation* of transitive closures (see Section 4).

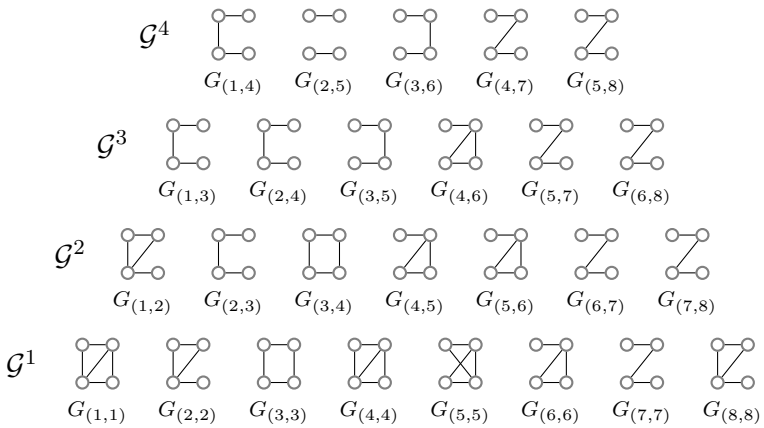


Fig. 1: First four rows of a composition hierarchy of intersection graphs.

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

Our framework requires the composition and test operations to meet certain conditions.

Observation 2 (Requirements) *For a maximization or a minimization problem with respect to some property P to be solvable within our framework, the following conditions must hold on the test and composition operations:*

(1) $test(G_{(i,j)}) = true$ *iff* $\{G_i, G_{i+1}, \ldots, G_{j-1}, G_j\}$ *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* $j' \geq j$.

General Framework. One advantage of using our general framework is that the high-level logic of the algorithms becomes elegant and simple, and we can solve problems by focusing only on the composition and test operations. In most cases, the

operations are highly generic and thus could benefit from dedicated circuits (*e.g.* similar to what has been done for all-pairs shortest-paths computations with FPGAs [4]) or optimized code. Our approach is suitable for dynamic graphs in which the details of changes between successive graphs in a sequence are arbitrary. If more structural information about the evolution of the dynamic graphs were known, for example, if it were known that the number of changes between each pair of consecutive static graphs is bounded by a constant, then algorithms could benefit from the use of sophisticated data structures and a lower-level approach than ours might be more appropriate.

Naive Upper Bound. It is not hard to show that any problem that can be solved using the framework can be solved using $O(\delta^2)$ composition and test operations based on a naive strategy. A naive algorithm computes all of the rows of the composition hierarchy $\mathcal{H}$ using the fact that each element $G_{(i,j)}$ can be obtained from the composition of two elements below it in $\mathcal{H}$, *i.e.* $G_{(i,j)} = G_{(i,i)} \circ G_{(i+1,j)}$. For instance, $G_{(3,6)} = G_{(3,3)} \cap G_{(4,6)}$ in Figure 1. Since there are $O(\delta^2)$ elements in $\mathcal{H}$, the total number of composition operations is $O(\delta^2)$. Furthermore, at most one test operation is needed for each element in $\mathcal{H}$.

Lemma 1 (Lower bound) *A total of δ composition and test operations are necessary to solve a problem using the framework.*

Proof (by contradiction) Let $\mathcal{A}$ be an algorithm that uses only composition and test operations and that solves a problem with respect to a property P with less than δ operations. Then, for any dynamic graph $\mathcal{G}$ of length δ, at least one element in $\mathcal{G}^1$ is never accessed by $\mathcal{A}$ using either a composition or a test operation. Suppose that $\mathcal{A}$ solves the problem for $\mathcal{G}$ without accessing $\mathcal{G}^1[i]$. Let $\mathcal{G}'$ be a dynamic graph that is identical to $\mathcal{G}$ except graph G_i is replaced by a graph G'_i such that one of the corresponding elements $\mathcal{G}^1[i]$ and $\mathcal{G}^1[i]'$ satisfies property P and the other does not. Since $\mathcal{G}^1[i]'$ is never accessed, the executions of $\mathcal{A}$ on $\mathcal{G}$ and $\mathcal{G}'$ are identical and $\mathcal{A}$ can return an incorrect result for $\mathcal{G}'$. □

3 Generic algorithm

We now introduce a strategy that uses the generic composition and test operations defined in Section 2. This generic algorithm will be instantiated in Section 4 to solve three specific minimization problems and one maximization problem by substituting appropriate composition and test operations for the generic operations. Examples of composition operations used in this paper are the *intersection* of graphs, the *union* of graphs, and the *concatenation* of transitive closures. Examples of test operations used in this paper are the *connectivity test* of a graph and the *equality to footprint* of a dynamic graph.

The strategy relies on the concept of *ladder*. Informally, a ladder "climbs" a sequence of elements in a composition hierarchy starting from the first row of the hierarchy.

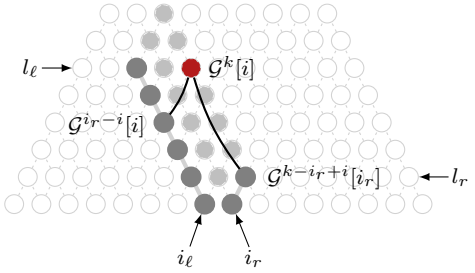


Fig. 2: Composition based on left and right ladders.

Definition 7 The *right ladder of length l at index i* in the first row of a hierarchy $\mathcal{H}$, denoted $\mathcal{R}^l[i]$, is the sequence of elements $\{\mathcal{G}^k[i],\ k = 1, 2, \ldots, l\}$. The *left ladder of length l at index i*, denoted $\mathcal{L}^l[i]$, is the sequence $\{\mathcal{G}^k[i-k+1],\ k = 1, 2, \ldots, l\}$.

Lemma 2 *A ladder of length l can be computed using $l-1$ binary compositions.*

Proof Consider a right ladder $\mathcal{R}^l[i]$. For any $k \in [2, l]$ it holds that $\mathcal{G}^k[i] = \mathcal{G}^{k-1}[i] \circ \mathcal{G}^1[i+k-1]$. Indeed, by definition, $\mathcal{G}^{k-1}[i] = \mathcal{G}^1[i] \circ \mathcal{G}^1[i+1] \circ \ldots \circ \mathcal{G}^1[i+k-2]$. The ladder can thus be built bottom-up using a single new composition in each row (in particular, the composition of the previous element in the ladder and one of the elements in the first row of the hierarchy).

Consider a left ladder $\mathcal{L}^l[i]$. For any $k \in [2, l]$ it holds that $\mathcal{G}^k[i-k+1] = \mathcal{G}^1[i-k+1] \circ \mathcal{G}^{k-1}[i-k+2]$. Indeed, by definition, $\mathcal{G}^{k-1}[i-k+2] = \mathcal{G}^1[i-k+2] \circ \mathcal{G}^1[i-k+3] \circ \ldots \circ \mathcal{G}^1[i]$. The ladder can thus be built bottom-up using a single new composition in each row. □

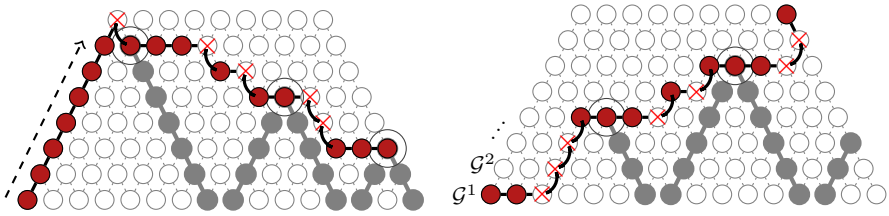
Lemma 3 *Given a left ladder $\mathcal{L}^{l_\ell}[i_\ell]$ of length l_ℓ at index i_ℓ and a right ladder $\mathcal{R}^{l_r}[i_r]$ of length l_r at index $i_r = i_\ell + 1$ in a composition hierarchy $\mathcal{H}$. For any index i and row k in $\mathcal{H}$ such that $i_r - l_\ell \leq i < i_r$ and $i_r - i < k \leq i_r - i + l_r$, $\mathcal{G}^k[i]$ 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 3 define a rectangle of elements in the hierarchy delimited by two ladders, and two lines each of which is parallel to one of the ladders, as shown in Figure 2. The elements $\mathcal{G}^k[i]$ defined by the constraints, shown in light grey in the figure, include all elements that are strictly inside the rectangle, and all elements on the parallel lines, but elements on the two ladders (dark grey) are excluded.

Proof (of Lemma 3) By definition, $\mathcal{G}^k[i] = \mathcal{G}^1[i] \circ \mathcal{G}^1[i+1] \circ \ldots \circ \mathcal{G}^1[i+k-1]$ and $\mathcal{G}^{i_r-i}[i] = \mathcal{G}^1[i] \circ \mathcal{G}^1[i+1] \circ \ldots \circ \mathcal{G}^1[i_r-1]$ and $\mathcal{G}^{k-i_r+i}[i_r] = \mathcal{G}^1[i_r] \circ \mathcal{G}^1[i_r+1] \circ \ldots \circ \mathcal{G}^1[i+k-1]$. It follows that $\mathcal{G}^k[i] = \mathcal{G}^{i_r-i}[i] \circ \mathcal{G}^{k-i_r+i}[i_r]$. By definition, $\mathcal{G}^{i_r-i}[i] \in \mathcal{L}^{l_\ell}[i_\ell]$ and $\mathcal{G}^{k-i_r+i}[i_r] \in \mathcal{R}^{l_r}[i_r]$, so only a single binary composition is needed. □

Generic algorithm. We describe the algorithm with reference to Figures 3a and 3b which show examples of executions of the algorithm in the maximization case and the

minimization case respectively (see Algorithm 1 for details). The algorithm takes as input a dynamic graph $\mathcal{G}$, a boolean problem type $\texttt{problem} \in \{min, max\}$, a composition operation $\circ$ which is used by the function `compute` to compute elements of the hierarchy, and a test operation `test`. It starts by computing the first element $\mathcal{G}^1[1]$ of the hierarchy $\mathcal{H}$ and then traverses $\mathcal{H}$ from left to right, computing a new adjacent element at each step. If $\mathcal{G}^i[j]$ is the most recently computed element, then the next element can be the element $\mathcal{G}^i[j+1]$ with the next index in the same row, or the element $\mathcal{G}^{i+1}[j]$ with the same index in the row above, or the element $\mathcal{G}^{i-1}[j+1]$ with the next index 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*. The elements on the walks in Figure 3 are the red/dark elements and the elements that are crossed out.



(a) (b)

Fig. 3: Examples of execution of the algorithm in (a) the maximization case (b) the minimization case.

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 3-10 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 $test(\mathcal{G}^k[1]) = true$ and $test(\mathcal{G}^{k+1}[1]) = false$ for some k. Then k is an upper bound on the maximization parameter of $\mathcal{G}$ and the walk drops down a row from $\mathcal{G}^{k+1}[1]$ to $\mathcal{G}^k[2]$ which is the next element in row k that needs to be tested. The walk proceeds rightward on the current row of $\mathcal{H}$, computing a new element in the row at each step while the test is positive. However, every time that the test is negative, the walk drops down one row in $\mathcal{H}$. Conceptually, the dropping down operation from a node $\mathcal{G}^k[i]$ visits two nodes (curved line in Figure 3a, line 21 of Algorithm 1). First the walk drops down to $\mathcal{G}^{k-1}[i]$ and then it moves right to $\mathcal{G}^{k-1}[i+1]$. Necessarily $test(\mathcal{G}^{k-1}[i]) = true$ because $test(\mathcal{G}^k[i-1]) = true$, and $\mathcal{G}^{k-1}[i]$ is not needed for later compositions, so neither a composition nor a test operation is needed here. 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 (line 15 of Algorithm 1). Otherwise the walk will terminate at an element $\mathcal{G}^1[i]$ that does not satisfy the test. In this case, the algorithm returns $\perp$ indicating that the dynamic graph $\mathcal{G}$ does not have the property.

In the minimization case, the walk starts at the element $\mathcal{G}^1[1]$ and goes up one row in $\mathcal{H}$ when a test is negative and right in the same row when a test is positive. If the walk reaches the right side of $\mathcal{H}$ at an element $\mathcal{G}^k[\delta-k+1]$ and $test(\mathcal{G}^k[\delta-k+1]) = true$, then the algorithm terminates and returns k. If the test is negative, it terminates and returns $k+1$ (by Observation 2, requirement $(3'')$, line 24 of Algorithm 1). Note that $\mathcal{G}^{k+1}[\delta-k]$ is the last element on the walk in this case but it is not necessary to compute or test this element. If the walk reaches $\mathcal{G}^\delta[1]$ and the test is negative, then the algorithm returns $\perp$ to indicate that the dynamic graph $\mathcal{G}$ does not have the property.

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

 3 if problem=max then
 4 |   compute(𝒢^k[i])
 5 |   while test(𝒢^k[i]) do
 6 |   |   if k = δ then
 7 |   |   |   return k
 8 |   |   else
 9 |   |   |   k++; compute(𝒢^k[i])
10 |   k−−; i++
11 while 1 ≤ k ≤ δ do
12 |   compute(𝒢^k[i])
13 |   if test(𝒢^k[i]) then
14 |   |   if i = δ − k + 1 then
15 |   |   |   return k
16 |   |   else
17 |   |   |   i++
18 |   else
19 |   |   switch problem do
20 |   |   |   case max do
21 |   |   |   |   k−−; i++
22 |   |   |   case min do
23 |   |   |   |   if i = δ − k + 1 and k ≠ δ then
24 |   |   |   |   |   return k + 1
25 |   |   |   |   else
26 |   |   |   |   |   k++
27 return ⊥
```

Algorithm 1: Generic algorithm for maximization and minimization problems

Computing elements of the hierarchy (function `compute`). The elements of the hierarchy that are on the walk (red/dark elements in Figure 3) are computed using ladders (grey elements in Figure 3) as follows. When the walk moves right in $\mathcal{H}$ and the next element of the walk to be computed is between a left ladder and a right ladder, then it is computed using one element of each ladder by Lemma 3 (*e.g.* $\mathcal{G}^7[4] = \mathcal{G}^5[4] \circ \mathcal{G}^2[9]$ in Figure 3a and $\mathcal{G}^5[6] = \mathcal{G}^2[6] \circ \mathcal{G}^3[8]$ in Figure 3b). An element of a walk that is not between two ladders will be on a ladder and is computed as a ladder

element using Lemma 2 as described below. This is the case for all elements on the first right ladder (*e.g.* elements $\mathcal{G}^1[1], \mathcal{G}^2[1], \ldots, \mathcal{G}^8[1]$ in Figure 3a, and elements $\mathcal{G}^1[3], \mathcal{G}^2[3], \mathcal{G}^3[3], \mathcal{G}^4[3]$ in Figure 3b), and the top element of each left ladder (*e.g.* element $\mathcal{G}^5[9]$ in Figure 3a, and element $\mathcal{G}^4[4]$ in Figure 3b).

Ladders provide useful shortcuts in the construction of a walk and are the basis of the efficiency of the algorithms. An element of a ladder (in grey in Figure 3, or red/dark if it is also an element of the walk), are computed, according to Lemma 2, by incrementally composing an element $G_{(i,j)}$ with an adjacent bottom element $G_{(i-1,i-1)}$ (left ladder) or $G_{(j+1,j+1)}$ (right ladder). Suppose that $\mathcal{G}^k[i]$ is the first element of the walk to be computed where no element $\mathcal{G}^{k'}[i]$ with $k' < k$ has been computed. This results in the construction of the first left ladder $\mathcal{L}^k[k+i-1]$ of length k ending at $\mathcal{G}^k[i]$ ($\mathcal{G}^7[2]$ in Figure 3a, $\mathcal{G}^4[4]$ in Figure 3b). 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 3). This continues until the walk crosses the current right ladder at an element $\mathcal{G}^k[i]$ ($\mathcal{G}^6[8]$ in Figure 3b), at which time a left ladder $\mathcal{L}^k[k+i-1]$ is built to be used to compute the next elements on the walk.

Note that when the walk goes up in the minimization case, it is sometimes possible to compute the next composition without incrementing the current right ladder. For example, in Figure 3b, $\mathcal{G}^5[6]$ could be constructed by composing $\mathcal{G}^4[6]$ and $\mathcal{G}^1[10]$. However, we continue to increment the right ladder because it could be useful later if the walk moves right. For example, in Figure 3b, $\mathcal{G}^3[8]$ is not needed to compute $\mathcal{G}^5[6]$, but $\mathcal{G}^4[8]$ is needed to compute $\mathcal{G}^5[7]$.

This generic algorithm has an important property concerning disjoint sequences that is required for some optimization problems. We prove it here for use in Section 4 (see Oberservation 5 for details). Note that this property is guaranteed by the algorithm, in any execution, but is not required to solve any maximization or minimization problem.

Lemma 4 (Disjoint sequences property) *If the algorithm performs a composition of two elements $G_{(i,j)}$ and $G_{(i',j')}$ of a hierarchy, 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 hierarchy is computed from either an element of a left ladder and an element of a right ladder, or an element of a ladder and an element in the first row of the hierarchy. In both cases the two sequences covered by the two elements used in the computation are disjoint and consecutive, so in any execution, $G_{(i,j')} = G_{(i,j)} \circ G_{(i',j')} \Rightarrow j = i' - 1$. □

Theorem 1 *The generic algorithm returns the correct value.*

Proof If the algorithm returns $\perp$ for a maximization problem on a dynamic graph $\mathcal{G}$ with respect to a property P, then the walk terminated at some element $\mathcal{G}^1[i]$ of

the hierarchy $\mathcal{H}$ that does not satisfy the test operation. Thus there is no $k \geq 1$ such that all subsequences of $\mathcal{G}$ of length k satisfy P. If the algorithm returns $\perp$ for a minimization problem then the element $\mathcal{G}^{\delta}[1]$ does not satisfy the test operation and there is no $k \geq 1$ such that all subsequences of $\mathcal{G}$ of length k satisfy P.

Now suppose that the algorithm returns a value different from $\perp$, and let $\mathcal{G}^{k}[\delta - k + 1]$ be the last element of $\mathcal{H}$ that is computed before the algorithm terminates. We claim that if $test(\mathcal{G}^{k}[\delta - k + 1]) = true$, then $\forall i \in [1, \delta - k]$, $test(\mathcal{G}^{k}[i]) = true$ (*i.e.* all elements in row $\mathcal{G}^{k}$ satisfy the test operation).

First consider the case of a maximization problem. If an element $\mathcal{G}^{k}[i]$ is on the walk computed by the algorithm, then the algorithm tests it directly. Otherwise, $\mathcal{G}^{k}[i]$ must be below the walk (because the walk never goes up in $\mathcal{H}$ after the first right ladder) and there is some element $\mathcal{G}^{k'}[i']$ with $i' \leq i$ and $k' \geq k$ on the walk that satisfies the test operation. Then $\mathcal{G}^{k}[i]$ satisfies the test operation by requirement $(3')$ of Observation 2.

If the problem is a minimization problem and $test(\mathcal{G}^{k}[\delta - k + 1]) = true$, then any element $\mathcal{G}^{k}[i]$ on the walk computed by the algorithm is tested directly. Otherwise, if $\mathcal{G}^{k}[i]$ is not on the walk, then it must be above the walk (because the walk never goes down in $\mathcal{H}$) and there is some element $\mathcal{G}^{k'}[i']$ with $i' \geq i$ and $k' \leq k$ on the walk that satisfies the test operation. Then $\mathcal{G}^{k}[i]$ satisfies the test operation by requirement $(3'')$ of Observation 2. If $test(\mathcal{G}^{k}[\delta - k + 1]) = false$, then the algorithm correctly returns $k + 1$. This is also guaranteed by requirement $(3'')$ of Observation 2: the ascending walk ensures that there exists a graph $\mathcal{G}^{k'}[\delta - k]$, $k' < k$, that satisfies the test operation. □

Theorem 2 *The generic algorithm has a cost of $\Theta(\delta)$ composition and test operations.*

Proof The ranges of the indices covered by the left ladders that are constructed by the algorithm are disjoint, so their total length is $O(\delta)$. 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 $O(\delta)$. According to Lemmas 2 and 3, any element can be computed using a single composition operation based on ladders. The number of elements on the walk is also $O(\delta)$. In the minimization case, the walk moves either right or up in the hierarchy after each static graph is received, so there are δ elements on the walk if the last element satisfies the test and $\delta + 1$ elements otherwise. In the maximization case, each time that the walk moves up (during construction of the first right ladder) or right corresponds to the reception of a new static graph. When the walk goes down, it does not correspond to the reception of a new static graph, but the number of times that this happens is no greater than the height of the first right ladder. Thus, the total number of elements computed is $O(\delta)$. Only elements on the walk are tested. This establishes that the algorithm has a cost of $O(\delta)$ composition and test operations which matches the lower bound of Lemma 1. □

Online algorithm. 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 as the graphs are received and the algorithm can determine the value of the parameter

based on the sequence of graphs received so far. The only difference from the offline algorithm is that the online algorithm can report a sequence of values of the parameter as the graphs are received.

Theorem 3 *The online generic algorithm has an amortized cost of $\Theta(1)$ 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 static 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 3. Only elements on the walk are tested. In summary, the amortized cost is $O(1)$ composition and test operations for each graph received. By arguments similar to the proof of Lemma 1, each graph received must be examined, so the amortized cost is optimal. □

Decision problems. Recall from Definition 3 that the *decision problem* on $\mathcal{G}$ with respect to P and a fixed value k is the problem of deciding if $\{G_i, G_{i+1}, \ldots, G_{i+k-1}\}$ has property P, $\forall i \in [1, \delta - k + 1]$. A generic algorithm for decision problems can compute the elements in row $\mathcal{G}^k$ of the hierarchy $\mathcal{H}$ for $\mathcal{G}$ and verify that all of them satisfy property P. We describe an efficient implementation of a generic decision algorithm based on some of the same techniques that we used for the generic optimization algorithm. The algorithm is the same for decision problems associated with both maximization and minimization problems.

The decision algorithm computes the elements in row $\mathcal{G}^k$ of $\mathcal{H}$ from left to right, starting at $\mathcal{G}^k[1]$. Each element is tested immediately after it is computed and if $test(\mathcal{G}^k[i]) = false$ for some $\mathcal{G}^k[i]$, the algorithm returns $false$ and terminates. If the walk reaches the rightmost element $\mathcal{G}^k[\delta - k + 1]$ in row $\mathcal{G}^k$ and $test(\mathcal{G}^k[\delta - k + 1]) = true$, then the algorithm returns $true$. An efficient implementation of the walk using ladders is simpler than for the optimization version. The algorithm starts by building a left ladder $\mathcal{L}^k[k]$ after the first k graphs $\{G_1, G_2, \ldots, G_k\}$ in the sequence are received and then element $\mathcal{G}^k[1]$ is tested. Then a right ladder $\mathcal{R}^{k-1}[k+1]$ is constructed one element at a time as the graphs $\{G_{k+1}, G_{k+2}, \ldots, G_{2k-1}\}$ are received and each element of the ladder is used immediately to compute the next element on row $\mathcal{G}^k$ which is then tested. If more elements on row $\mathcal{G}^k$ need to be tested, then left and right ladders are alternately constructed until either an element does not satisfy property P, in which case the algorithm returns $false$ and terminates, or the rightmost element $\mathcal{G}^k[\delta - k + 1]$ in row $\mathcal{G}^k$ is reached and $test(\mathcal{G}^k[\delta - k + 1]) = true$, in which case the algorithm returns $true$ and terminates. Two examples are shown in Figure 4.

Proofs that the generic decision algorithm returns the correct answer and that it has cost $\Theta(\delta)$ composition and test operations follow by similar arguments to the proofs of Theorems 1 and 2. An online decision algorithm cannot report any values of the parameter until the first k graphs $G_1, G_2, G_3, \ldots, G_k$ have been received. Thereafter, the amortized cost is $\Theta(1)$ composition and test operations per graph received by similar arguments to the proof of Theorem 3.

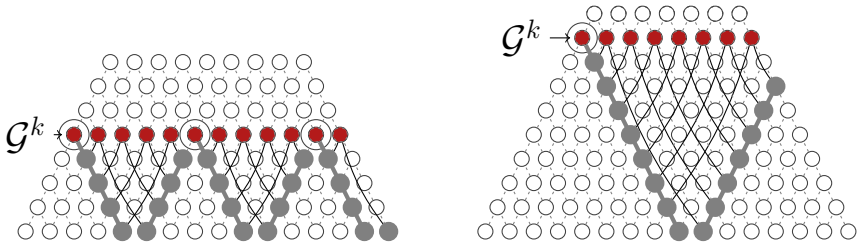


Fig. 4: Examples of the execution of the algorithm for decision problems.

4 Applications of the Framework

In this section, we apply 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 *T-interval connected* if for any $t \in [1, \delta - T + 1]$ all graphs in $\{G_t, G_{t+1}, \ldots, G_{t+T-1}\}$ share a common connected spanning subgraph. We consider the problem INTERVAL-CONNECTIVITY of finding the largest 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 $G_{(i,i)} = G_i$, $1 \leq i \leq \delta$), a hierarchy with intersection graphs as elements (Figure 1) 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 a sorted adjacency list in which the neighbours of each node are sorted, a binary intersection of two elements $G_{(i,j)}$ and $G_{(i',j')}$ can be computed in linear time in the number of edges: $O(\min(|E(G_{(i,j)})|, |E(G_{(i',j')})|))$. Testing the connectivity of an element can also be done in linear time - $O(|E(G_{(i,j)})|)$ - by building a depth-first search tree from an arbitrary root node and testing whether all nodes are reachable from the root node. If the elements have directed edges, then Tarjan's algorithm for strongly connected components [26] can be used. Hence, both the* intersection *operation and the* connectivity testing *operation have similar costs.*

4.2 Bounded Realization of the Footprint (minimization)

The *footprint* G of a dynamic graph $\mathcal{G}$ is the graph that contains all of the edges that appear at least once, that is $G = \cup\{G_1, G_2, \ldots, G_\delta\}$. We consider the problem BOUNDED-REALIZATION-OF-THE-FOOTPRINT of finding the shortest duration b such that in any window of length b, every edge of G appears at least once. The problem can be solved by 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 $G_{(i,i)} = G_i$, $1 \leq i \leq \delta$), it follows that the lowest row $\mathcal{G}^b$ in which all elements *equal* the footprint determines, 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 (for both the undirected and directed cases), the union operation and the equality test can be performed in $O(|V|^2)$ time, by traversing the two involved adjacency matrices element by element.*

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 in each static graph (*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 dynamic graphs, it is defined as follows.

Definition 8 (Journey) A *journey* from u to v in $\mathcal{G}$, denoted $u \rightsquigarrow v$, 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 departure time of $u \rightsquigarrow v$ is $departure(u, v) = t_1$ and the arrival time of $u \rightsquigarrow v$ is $arrival(u, v) = t_p$.

For both non-strict and strict journeys, one can define the concept of *temporal diameter* (at time t) as the smallest d such that for all nodes u and v, a journey $u \rightsquigarrow v$ exists 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 $t \leq \delta - d + 1$. 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. [28]), 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 the case of strict journeys as a subproblem.

Definition 9 (Transitive closure) The transitive closure of the dynamic graph $\mathcal{G}$ is the static directed graph $\mathcal{G}^* = (V, E^*)$ such that $(u, v) \in E^* \Leftrightarrow \exists u \rightsquigarrow v$.

The composition hierarchy built here is one of *transitive closures* of journeys. Figure 5 shows an example. For this problem, each bottom element $G_{(i,i)}$ is not equal to G_i; instead, it is similar to a "classical" *transitive closure* of G_i. The graph $G_{(i,i)}$ is built on the same vertex set as G_i and there is a *directed edge* from u to v in $G_{(i,i)}$ if and only if a *journey* $u \rightsquigarrow v$ exists in G_i. The elements $G_{(i,i)}$, $1 \leq i \leq \delta$, 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 directed graph (*i.e.* every subsequence of $\mathcal{G}$ of length d is temporally connected). Note that the static graphs in Figure 5 are directed. The algorithm is similar for undirected static graphs except each undirected edge in a static graph is treated as a pair of directed edges when the elements $G_{(i,i)}$, $1 \leq i \leq \delta$, are computed.

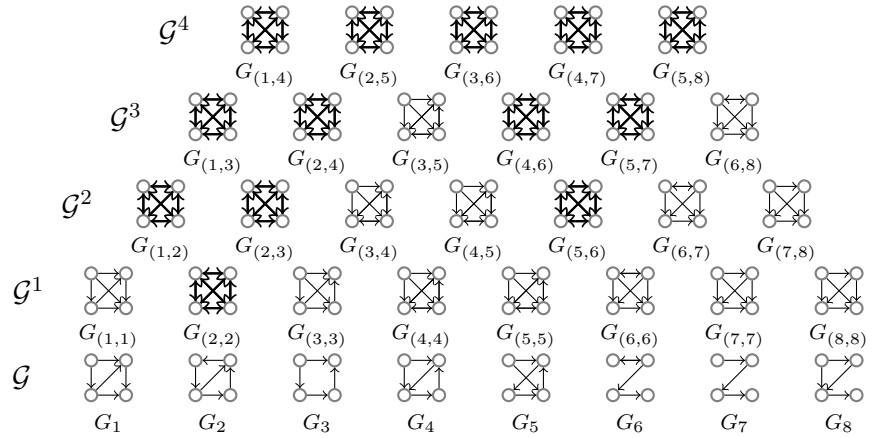


Fig. 5: Example of a transitive closure hierarchy for a given dynamic graph $\mathcal{G}$ of length $\delta = 8$.

Composition and test operations. The composition hierarchy is built using *concatenation* of transitive closures, $cat(G_{(i,j)}, G_{(i',j')})$. It is 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 6 for an example. Then, the *test* operation consists of determining if an element of the hierarchy (transitive closure) is a complete directed graph.

Observation 5 *The concatenation operation presented above decides whether a journey exists in a sequence $\{G_i, G_{i+1}, ..., G_{j'}\}$ from existing journeys in two sequences $\{G_i, G_{i+1}, ..., G_j\}$ and $\{G_{i'}, G_{i'+1}, ..., G_{j'}\}$ using the computation of extra-edges. In order for the concatenation operation to be consistent (the existence of an edge (journey) in $G_{(i,j)}$ and an edge in $G_{(i',j')}$ implies the existence of a new*

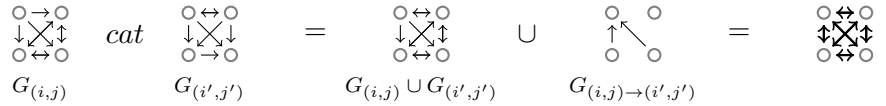


Fig. 6: Example of concatenation of transitive closures. *Edges in $G_{(i,j)\to(i',j')}$ are added after the union.*

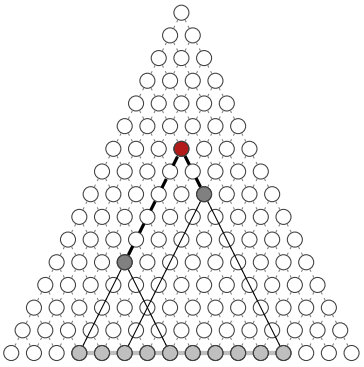


Fig. 7: Example of a potentially incorrect computation of the transitive closure $\mathcal{G}_{(4,13)}$ from $\mathcal{G}_{(4,8)}$ and $\mathcal{G}_{(6,13)}$.

*edge in $G_{(i,j')}$), the two used sequences $\{G_i, G_{i+1}, ..., G_j\}$ and $\{G_{i'}, G_{i'+1}, ..., G_{j'}\}$ must neither overlap nor be separated, we should have $j = i' - 1$ or $j = i'$. Otherwise, the computation of a transitive closure does not always allow a correct result. Figure 7 shows an example where the concatenation of two transitive closures of journeys $G_{(4,8)}$ and $G_{(6,13)}$ of the two sequences $\{G_4, G_5, ..., G_8\}$ and $\{G_6, G_7, ..., G_{13}\}$ does not always give a correct transitive closure of journeys $G_{(4,13)}$ of the sequence $\{G_4, G_5, ..., G_{13}\}$. Assume that only one node w exists such that $(u, w) \in E(G_{(4,8)})$ and $(w, v) \in E(G_{(6,13)})$ and that (u, w) corresponds to a journey $u \rightsquigarrow w$ whose $arrival = 8$ and (w, v) represents the existence of a journey $w \rightsquigarrow v$ with $departure = 6$. In this case the concatenation operation adds an edge (u, v) to $G_{(4,13)}$ even if no journey is implied by the existence of the two latter ones. Actually, the concatenation operation computes in this case the transitive closure of journeys in the sequence $\{G_4, G_5, ..., G_8, G_6, G_7, ..., G_{13}\}$. With the restriction that $i' = j + 1$ (*disjoint sequences property, Lemma 4), any computed transitive closure $G_{(i,j)}$ corresponds to the right sequence $\{G_i, G_{i+1}, ..., G_j\}$.*

Lemma 5 *Any computed transitive closure is correct: $\forall i, j, i \leq j \leq \delta, G_{(i,j)} = G_i \circ G_{i+1} \circ ... \circ G_j$.*

Proof According to the *disjoint sequences property* (Lemma 4) guaranteed by the algorithm, in any execution, $G_{(i,j')} = cat(G_{(i,j)}, G_{(i',j')}) \Rightarrow j = i' - 1$. So, any computed transitive closure is correct. □

Observation 6 (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 ad-*

jacency list data structure, by merging for each node the two lists of neighbours. The cost of the concatenation operation is dominated by the computation of the additional edges which costs $O(|E(G_{(i,j)})| \cdot |V|)$, by adding for each $(u,w) \in E(G_{(i,j)})$ and each $(w,v) \in E(G_{(i',j')})$ the edge (u,v) to the adjaceny list of the concatenation. (This is done by traversing the edges (u,w) in $G_{(i,j)}$ and traversing the neighbours of w in the adjacency list of $G_{(i',j')}$.) The completeness test of a transitive closure $G_{(i,j)}$ can be done in constant time by checking the number of edges $|E(G_{(i,j)})|$ (and by maintaining this number during the construction of the transitive closure graphs).

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, for every pair of nodes u, v, there is a journey $u \rightsquigarrow v$ in the sequence $\{G_t, G_{t+1}, ..., G_{t+d-1}\}$ *and* a journey $v \rightsquigarrow u$ in $\{G_t, G_{t+1}, ..., G_{t+d-1}\}$ which starts after the arrival of the journey $u \rightsquigarrow v$. 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 10 (Round trip transitive closure) A *round trip transitive closure* $G_{(i,j)}$ is the static directed graph where $(u,v) \in G_{(i,j)}$ iff at least one journey $u \rightsquigarrow v$ exists in the sequence $\{G_i, G_{i+1}, \ldots, G_j\}$. Each directed edge $(u,v) \in E(G_{(i,j)})$ is labelled with two times: $arrival(u, v, G_{(i,j)})$ is the earliest arrival of any journey in the sequence and $departure(u, v, G_{(i,j)})$ is the latest departure of any journey in the sequence. Labels on the same edge may or may not be the arrival and departure times of the same journey. Note that $arrival(u, v, G_{(i,i)}) = i$ and $departure(u, v, G_{(i,i)}) = i$.

The composition hierarchy built for this problem is one of *round trip transitive closures* of journeys. Figure 8 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 $u \xrightarrow{arr,dep} v$ (labels 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 $G_{(i,i)}$ is similar to a "classical" *transitive closure* of G_i. The graph $G_{(i,i)}$ is built on the same vertex set as G_i and there is a *directed edge* from u to v in $G_{(i,i)}$ if and only if a *journey* $u \rightsquigarrow v$ exists in G_i. The labels on the edges of $G_{(i,i)}$ are "i, i", which correspond to the arrival and departure times of the corresponding journey(s). Then, the answer is the smallest d such that every element in row $\mathcal{G}^d$ of the hierarchy is a complete directed graph (i.e. every subsequence of $\mathcal{G}$ of length d is round-trip temporally connected), where for every edge (u,v) in the graph

$arrival(u, v, G_{(i,j)}) \leq departure(v, u, G_{(i,j)})$. The static graphs in Figure 8 are directed. As for TEMPORAL-DIAMETER, the algorithm is similar for undirected static graphs except each undirected edge in a static graph is treated as a pair of directed edges when the elements $G_{(i,i)}$, $1 \leq i \leq \delta$, are computed.

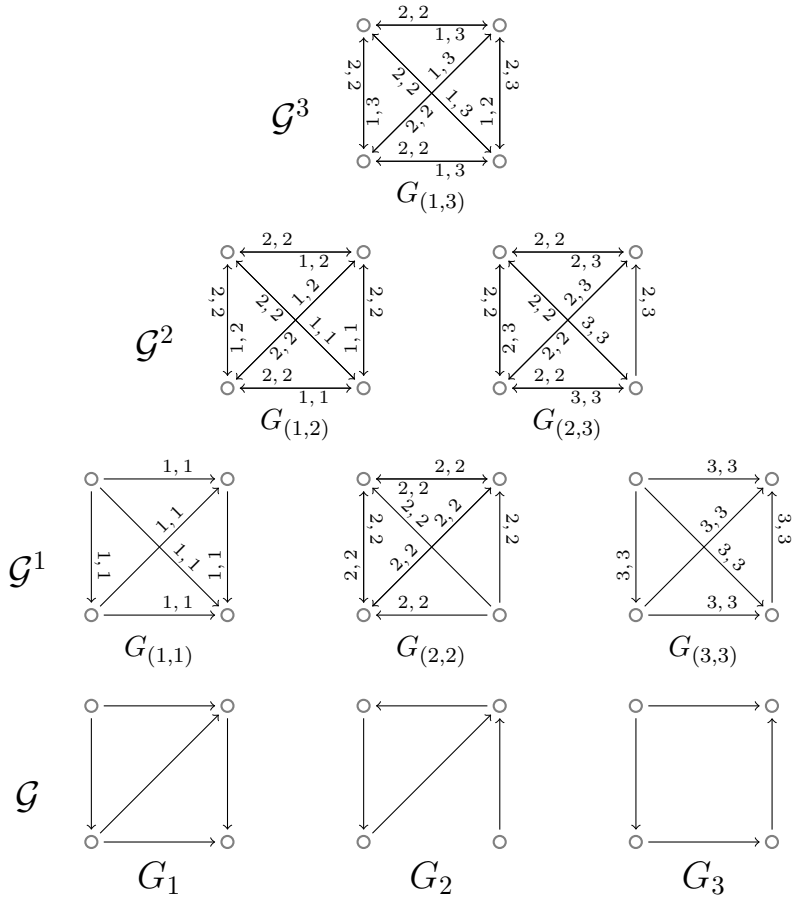


Fig. 8: Example of a round trip transitive closure hierarchy of a dynamic graph $\mathcal{G}$ of length $\delta = 3$. (*Arrival and departure times are on the head ends of the arrows.*)

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 $i' = j+1$ (*disjoint sequences property*, Lemma 4). A composition is computed as follows. First, compute the graph $G^{\cup\circlearrowleft} = G_{(i,j)} \cup^{\circlearrowleft} G_{(i',j')}$ from the union graph $G_{(i,j)} \cup G_{(i',j')}$. For each edge $(u, v) \in G_{(i,j)} \cup G_{(i',j')}$ such that $(u, v) \in G_{(i,j)} \cap G_{(i',j')}$, set $arrival(u, v, G^{\cup\circlearrowleft}) = \min(arrival(u, v, G_{(i,j)}), arrival(u, v, G_{(i',j')}))$ and $departure(u, v, G^{\cup\circlearrowleft}) = \max(departure(u, v, G_{(i,j)}), departure(u, v, G_{(i',j')})$ in

$G^{\cup\circlearrowleft}$. If $(u,v) \in G_{(i,j)}$ or $(u,v) \in G_{(i',j')}$, but not both, then label the edge in $G^{\cup\circlearrowleft}$ with its 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 $rtcat(G_{(i,j)},G_{(i',j')}) = G^{\cup\circlearrowleft}\cup^{\circlearrowleft} G_{(i,j)\to(i',j')}$. Figure 9 shows an example of an $rtcat$ composition operation. Also, in Figure 8, in the label (1,2) on the rightmost edge of $G_{(1,3)}$, the arrival time 1 comes from the corresponding edge in $G_{(1,2)}$, but the departure time 2 comes from extra edges.

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.

Observation 7 (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 $O(|E(G_{(i,j)})| \cdot |V|)$ using an adcancency list data structure. The completeness test can be done in time $O(|E(G_{(i,j)})|)$ by verifying the condition on the arrival and departure times for each pair of edges (u,v), (v,u) (and by maintaining these times for each edge of the computed transitive closure graphs).*

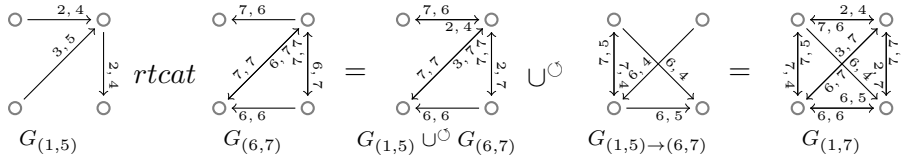


Fig. 9: Example of round trip transitive closures concatenation. (*Arrival and departure times are on the head ends of the arrows.*)

5 Parallel Algorithms

In this section, we present a row-based strategy that uses $O(\delta \log \delta)$ composition and test operations for problems that have a hereditary structure. While it is less efficient than the algorithms presented in previous sections, it has the advantage that it can be parallelized. This allows us to show that problems solvable with this strategy are in **NC**, *i.e.* parallelizable on a PRAM with a polylogarithmic running time [17,20].

Definition 11 (Hereditary property) A maximization, minimization, or decision problem is *hereditary* 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 *hereditary problems*. The general idea of the row-based strategy is to compute only some of the rows of the composition hierarchy based on the following lemma.

Lemma 6 *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 operations.*

Proof Assume that row $\mathcal{G}^k$ is already computed and that one wants to compute row $\mathcal{G}^\ell$ for some $k+1 \leq \ell \leq 2k$. Note that row $\mathcal{G}^\ell$ consists of the entries $\mathcal{G}^\ell[1], \ldots, \mathcal{G}^\ell[\delta - \ell + 1]$. Now, observe that for any $1 \leq i \leq \delta - \ell + 1$, $\mathcal{G}^\ell[i] = \mathcal{G}^k[i] \circ \mathcal{G}^k[i + \ell - k]$. Hence, $\delta - \ell + 1 = O(\delta)$ composition operations are sufficient to compute all of the entries of row $\mathcal{G}^\ell$. □

Maximization and minimization algorithms. For the maximization problem on $\mathcal{G}$ with respect to a property P, we incrementally compute rows $\mathcal{G}^{2^i}$ ("power rows") for $i = 1, 2, \ldots$ until we find a row that contains an element that does not satisfy P (thus, a test operation is performed after each composition). By Lemma 6, each of these rows can be computed using $O(\delta)$ composition operations. Suppose that row $\mathcal{G}^{2^{j+1}}$ is the first power row that contains an element that does not satisfy P, 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 P. The computation of each of these rows is based on row $\mathcal{G}^{2^j}$ and uses $O(\delta)$ composition operations by Lemma 6. Overall, we compute at most $2\lceil \log_2 k \rceil = O(\log \delta)$ rows using $O(\delta \log \delta)$ composition operations and the same number of test operations.

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 P. Suppose that row $\mathcal{G}^{2^{j+1}}$ is the first power row such that all elements on this row satisfy P. 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 P. See Figure 10 for illustrations of these algorithms.

Decision problems. An algorithm for a decision problem on $\mathcal{G}$ with respect to a property P and a fixed value k returns *true* if all elements in row $\mathcal{G}^k$ satisfy P and *false* otherwise. Using Lemma 6, for a given k, we can incrementally compute rows $\mathcal{G}^{2^i}$ 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 6). This way, we compute $\lceil \log_2 k \rceil = O(\log \delta)$ rows using $O(\delta \log \delta)$ composition operations, after which we perform $O(\delta)$ test operations.

Now we establish that these problems are in **NC** by showing that the row-based algorithms are efficiently parallelizable.

Lemma 7 *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.*

Proof Assume that row $\mathcal{G}^k$ is already computed, and that one wants to compute row $\mathcal{G}^\ell$, consisting of the entries $\mathcal{G}^\ell[1], \ldots, \mathcal{G}^\ell[\delta - \ell + 1]$, for some $k + 1 \leq \ell \leq 2k$. Since $\mathcal{G}^\ell[i] = \mathcal{G}^k[i] \circ \mathcal{G}^k[i + \ell - k]$, $1 \leq i \leq \delta - \ell + 1$, the computation of row $\mathcal{G}^\ell$ can be implemented on an EREW PRAM with $\delta - \ell + 1$ processors in two rounds as follows. Let P_i, $1 \leq i \leq \delta - \ell + 1$, be the processor dedicated to computing $\mathcal{G}^\ell[i]$. In the first round P_i reads $\mathcal{G}^k[i]$, and in the second round P_i reads $\mathcal{G}^k[i + \ell - k]$. This guarantees that each P_i has exclusive access to the entries of row $\mathcal{G}^k$ that it needs for its computation. Hence, row $\mathcal{G}^\ell$ can be computed in $O(1)$ time on an EREW PRAM using $O(\delta)$ processors. □

Parallel maximization and minimization algorithms for an EREW PRAM. The sequential algorithm for maximization and minimization problems computes $O(\log \delta)$ rows of the composition hierarchy. By Lemma 7, 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. After the computation of each row, it must be determined whether or not all of the elements in the row satisfy property P. The $O(\delta)$ test operations for a row can be done in $O(1)$ time with $O(\delta)$ processors. Then, the processors can establish whether or not all elements in the row satisfy P 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 [17,20]). The total time is $O(\log^2 \delta)$ on an EREW PRAM with $O(\delta)$ processors.

Parallel decision algorithms for an EREW PRAM. The sequential algorithm for decision problems computes $O(\log \delta)$ rows of the composition hierarchy. By Lemma 7, each of these rows can be computed in $O(1)$ time on an EREW PRAM with $O(\delta)$ processors. Differently from maximization and minimization problems, tests are only required for one row $\mathcal{G}^k$. This can be done in $O(\log \delta)$ time using the same techniques as the maximization and minimization algorithms. The total time is $O(\log \delta)$ on an EREW PRAM with $O(\delta)$ processors.

6 Conclusions

In this paper, we presented a general framework for computing parameters of dynamic networks which are modelled as a sequence $\mathcal{G} = (G_1, G_2, \ldots, G_\delta)$ of static graphs. We studied three minimization problems (*bounded realization of the footprint, temporal diameter,* and *round trip temporal diameter),* and one maximization problem (*T-interval connectivity*). We proposed algorithms for these problems within the same framework.

In our study, we focused on algorithms that use only *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.

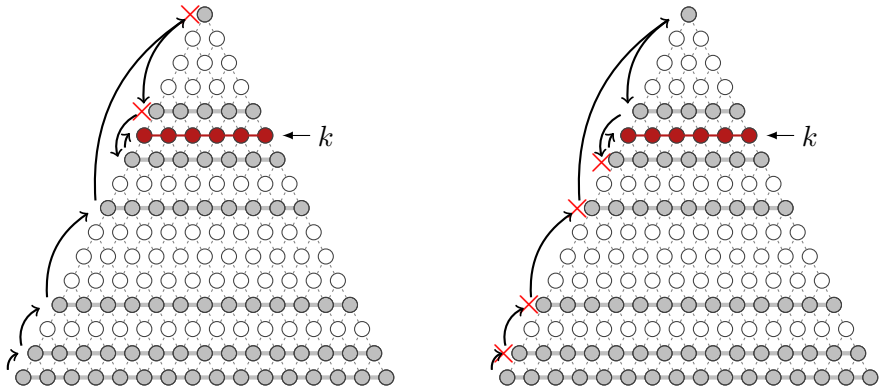


Fig. 10: Examples of the row-based algorithm; *maximization case* on the left and *minimization case* on the right.

A natural extension of our investigation would be a similar study for other classes and properties of dynamic networks, as identified in [12].

Acknowledgements. We thank the anonymous referees for their careful reading and valuable comments which helped to improve the presentation of the paper.

References

1. Aaron, E., Krizanc, D., Meyerson, E.: DMVP: foremost waypoint coverage of time-varying graphs. In: Proc. 40th Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG 2014), *LNCS*, vol. 8747, 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** (2014). (A French version appeared in Proc. of ALGOTEL 2014.)
4. Bondhugula, U., Devulapalli, A., Fernando, J., Wyckoff, P., Sadayappan, P.: Parallel FPGA-based all-pairs shortest-paths in a directed graph. In: Proc. 20th International Parallel and Distributed Processing Symposium. IEEE (2006)
5. 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, *LNCS*, vol. 10083, pp. 54–69. Springer (2016)
6. Bramas, Q., Tixeuil, S.: The complexity of data aggregation in static and dynamic wireless sensor networks. Information and Computation **255**, 369–383 (2017)
7. 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)
8. 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 (2003)
9. 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), *LNCS*, vol. 5869, pp. 126–140. Springer (2009)
10. Casteigts, A., Flocchini, P., Mans, B., Santoro, N.: Measuring temporal lags in delay-tolerant networks. IEEE Transactions on Computers **63**(2), 397–410 (2014)

11. 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)
12. Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and dynamic networks. International Journal of Parallel, Emergent and Distributed Systems **27**(5), 387–408 (2012)
13. Casteigts, A., Klasing, R., Neggaz, Y.M., Peters, J.G.: Efficiently testing T-interval connectivity in dynamic graphs. In: Proc. 9th International Conference on Algorithms and Complexity (CIAC 2015), *LNCS*, vol. 9079, pp. 89–100. Springer (2015)
14. Casteigts, A., Klasing, R., Neggaz, Y.M., Peters, J.G.: A generic framework for computing parameters of sequence-based dynamic graphs. In: Proc. 24th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2017), *LNCS*, vol. 10641, pp. 321–338. Springer (2017)
15. 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)
16. Flocchini, P., Mans, B., Santoro, N.: On the exploration of time-varying networks. Theoretical Computer Science **469**, 53–68 (2013)
17. Gibbons, A., Rytter, W.: Efficient parallel algorithms. Cambridge University Press (1988)
18. 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), *LNCS*, vol. 8847, pp. 88–102. Springer (2014)
19. Jain, S., Fall, K., Patra, R.: Routing in a delay tolerant network. In: Proc. of SIGCOMM, pp. 145–158 (2004)
20. JáJá, J.: An introduction to parallel algorithms. Addison-Wesley (1992)
21. Kuhn, F., Lynch, N., Oshman, R.: Distributed computation in dynamic networks. In: Proc. of STOC, pp. 513–522. ACM (2010)
22. Mans, B., Mathieson, L.: On the treewidth of dynamic graphs. Theoretical Computer Science **554**, 217–228 (2014)
23. Neggaz, Y.M.: Automatic classification of dynamic graphs. Ph.D. thesis, University of Bordeaux (October 2015. https://hal.archives-ouvertes.fr/tel-01419691v1)
24. O'Dell, R., Wattenhofer, R.: Information dissemination in highly dynamic graphs. In: Proc. of DIALM-POMC, pp. 104–110. ACM (2005)
25. Raynal, M., Stainer, J., Cao, J., Wu, W.: A simple broadcast algorithm for recurrent dynamic systems. In: Proc. 28th IEEE International Conference on Advanced Information Networking and Applications (AINA 2014), pp. 933–939. IEEE (2014)
26. Tarjan, R.E.: Depth first search and linear graph algorithms. SIAM Journal on Computing **1**(2), 146–160 (1972)
27. Viard, T., Latapy, M., Magnien, C.: Computing maximal cliques in link streams. Theoretical Computer Science **609**, 245–252 (2016)
28. Whitbeck, J., Dias de Amorim, M., Conan, V., Guillaume, J.L.: Temporal reachability graphs. In: Proc. of MOBICOM, pp. 377–388. ACM (2012)