Calcul de paramètres minimaux dans les graphes dynamiques.

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

Calcul de paramètres minimaux dans les graphes dynamiques. †

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

[1] *LaBRI, CNRS, Université de Bordeaux, France*
[2] *School of Computing Science, Simon Fraser University, Burnaby, BC, Canada*

Dans un travail précédent [2], nous avons présenté un algorithme permettant de calculer un paramètre appelé T-*interval connectivity* dans les graphes dynamiques qui se présentent sous forme d'une suite de graphes $G_1, G_2, ..., G_\delta$. Cet algorithme considère les graphes de la suite comme des atomes et les manipule via deux opérations élémentaires : une opération de *composition* (de deux graphes) et une opération de *test* (sur un graphe). Cet algorithme est optimal dans le sens où il n'utilise que $O(\delta)$ opérations de ce type. Dans cet article, nous généralisons cette approche, en montrant notamment qu'il suffit de définir différemment les opérations de composition et de test pour résoudre immédiatement d'autres problèmes. Nous illustrons cela par l'étude de trois problèmes de minimisation, à savoir BOUNDED-REALIZATION-OF-THE-FOOTPRINT, TEMPORAL-DIAMETER, et ROUND-TRIP-TEMPORAL-DIAMETER, chacun faisant référence à une propriété temporelle importante dans les réseaux dynamiques.

Keywords: Réseaux dynamiques, Test de propriétés, Généricité, Connexité temporelle.

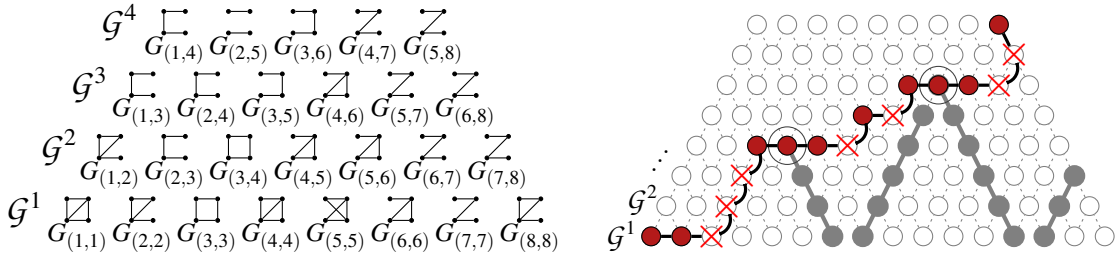
1 Introduction

Over the last decade, research has highlighted the importance of characterizing and testing properties in dynamic networks, studying patterns, and analyzing their dynamics. These networks can be modelled in various ways. When time is discrete, a common model is to represent the network as a sequence of static graphs $\mathcal{G} = \{G_1, G_2, ..., G_\delta\}$, where each $G_t = (V, E_t)$ represents the state of the network at a time t (*a.k.a.* untimed evolving graphs). In [2], we presented an optimal solution to the problem of computing a stability parameter called T-interval connectivity, i.e. the largest T such that all graphs in every consecutive subsequence of length T share a common connected spanning subgraph. The 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. Using *intersection* of two graphs as the composition operation and *connectivity* of a graph as the test operation, a hierarchy of intersection graphs is built such that the whole sequence is T-interval connected if and only if all the composed graphs in row T of the hierarchy are *connected* (see Figure 1a for an illustration). The algorithm "walks" within this hierarchy building the required graphs on demand. The walk strategy is optimal in that the algorithm computes the maximum value of T using only $O(\delta)$ intersections and connectivity tests.

In this paper, we generalize this framework to use various composition and test operations, which yields solutions to other problems using the same strategy that we used for T-interval connectivity. We formulate a *minimization* version of the strategy (T-interval connectivity is a *maximization* problem) and illustrate it with three pairs of composition and test operations. These operations make it possible to compute three parameters that relate to specific properties of dynamic networks discussed in [1]. They are (1) BOUNDED-REALIZATION-OF-THE-FOOTPRINT, where the goal is to find the smallest duration d such that if any edge is present in some G_i (with $i \leq \delta - d$), then it must be present again in some $G_{i'}$ with $i' \leq i + d$; (2) TEMPORAL-DIAMETER, where the goal is to find the smallest duration d such that a temporal path (journey) exists between every pair of nodes in *every* window of size d (see e.g. [3]); and (3) ROUND-TRIP-TEMPORAL-DIAMETER, which asks the same question for *back and forth* journeys (which is not the same as solving the second problem twice).

By using the proposed general framework, we can reuse most of the high-level logic of the algorithm and focus instead on the specific aspects of each problem through the composition and testing operations. This approach is

†This work has been supported by ANR projects ESTATE (ANR-16-CE25-0009-03) and by "the Investments for the future" Programme IdEx Bordeaux CPU (ANR-10-IDEX-03-02).



(a) (b)

Figure 1: (a) Example of partial hierarchy from [2] using *intersection* as the composition operation. (b) Example of execution of a minimization algorithm (not related to (a) which corresponds to a maximization problem).

especially relevant when the changes between two consecutive graphs are arbitrary. If that is not the case, or if the graphs are sparse, then using low level dedicated data structure might be a more relevant approach.

2 Overview and main definitions

Let $\mathcal{G}$ be a graph sequence $\{G_1, G_2, ..., G_\delta\}$ and let P be a boolean predicate (hereafter called *property*) defined on a consecutive subsequence $\{G_i, G_{i+1}, ..., G_j\} \subseteq \mathcal{G}$. The *minimization (resp. maximization) problem on $\mathcal{G}$ with respect to P* is the problem of finding the smallest (largest) 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). We present here a general strategy that relies on a virtual hierarchy of graphs which is computed on demand. The hierarchy consists of 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} graph of row $\mathcal{G}^k$, that is the graph $G_{(i,i+k-1)}$. 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. Then, the hierarchy is defined by applying some *composition* operation $\circ$ that computes a graph in a higher row from two graphs in lower rows. By doing this, the graph $G_{(i,j)}$ is the (iterated) composition of the graphs in the original subsequence $\{G_i, G_{i+1}, \ldots, G_{j-1}, G_j\}$. Observe that the graphs in row $\mathcal{G}^k$ are the (iterated) composition of subsequences of length k in the original sequence. An example of the hierarchy based on the *intersection* operation from [2] is shown in Figure 1a. Finally, in addition to the composition operation, we consider a *test* operation which maps any graph of the hierarchy into $\{true, false\}$. For example, in [2] this operation tests whether the given graph is *connected*.

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. Due to space limitation, we describe only the minimization version here, and refer the reader to [2] for an example of a maximization algorithm, albeit described in a less general way, and to Chapter 4 of [4] for further details (e.g. proofs) about the minimization version.

For a minimization 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) operation $\circ$ is associative, that is $(G_{(i,j)} \circ G_{(i',j')}) \circ G_{(i'',j'')} = G_{(i,j)} \circ (G_{(i',j')} \circ G_{(i'',j'')})$; and (3) it holds that $[G_{(i,j')} = G_{(i,j)} \circ G_{(i',j')} \ \wedge (test(G_{(i,j)}) = true \ \vee \ test(G_{(i',j')}) = true)] \Rightarrow test(G_{(i,j')}) = true$. Then the problem amounts to finding the lowest row in which all graphs satisfy the test.

3 High-level Strategy for Minimization Problems

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 by plugging in the appropriate operations. We describe the algorithm with reference to Figure 1b that shows an example of execution. The algorithm starts by computing the first graph $\mathcal{G}^1[1]$ and then traverses the hierarchy from left to right by computing a new adjacent graph at each step: the next graph in the same row or the graph with the same index in the row above, depending on the result of the test operation on the current graph. We call this traversal process a *walk*. The walk goes up in the 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 graph $\mathcal{G}^k[\delta-k+1]$ in the row $\mathcal{G}^k$ satisfies the test operation, then it terminates and outputs k. Otherwise, it terminates and outputs $k+1$. 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.

Graphs computation: Intermediate graphs called *ladders* (in grey in Figure 1b) are computed by composing incrementally a graph $G_{(i,j)}$ with the adjacent bottom graph $G_{(i-1,i-1)}$ (left ladder) or $G_{(j+1,j+1)}$ (right ladder), providing useful shortcuts in the construction. Then, the graphs resulting from the walk (red/dark graphs in Figure 1b) are computed as follows: When the walk moves one step forward in the same row, the next graph is computed from the right and the left ladders (*e.g.* $\mathcal{G}^4[6] = \mathcal{G}^2[6] \circ \mathcal{G}^2[8]$) or from the ladder to which it belongs and an adjacent bottom graph (*e.g.* $\mathcal{G}^4[4] = \mathcal{G}^1[4] \circ \mathcal{G}^3[5]$). If the walk climbs a step (moves up) in the hierarchy, then the next graph is computed from the preceding graph in the walk and the next graph in $\mathcal{G}^1$ (instead of using ladders). Although this computation does not require the use of ladders, the process continues to build a right ladder as the walk goes up for later use (if the walk later moves forward on the same row).

As it turns out, this general algorithm for minimization problems has the following convenient property which is crucial for correctness of two of the problems described in Section 4.

Lemma 1 (Disjoint sequences property)**.** *If the algorithm performs a composition of two graphs $G_{(i,j)}$ 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, $G_{(i,j')} = G_{(i,j)} \circ G_{(i',j')} \Rightarrow j = i' - 1$.*

Theorem 1. *This minimization algorithm has a cost of $\Theta(\delta)$ composition operations and test operations (see [2] for proof).*

4 Illustration of the Framework

We illustrate the general framework by solving 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. The reader is referred to Chapter 4 of [4] for more details.

4.1 Bounded Realization of the Footprint

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 have appeared at least once (BOUNDED-REALIZATION-OF-THE-FOOTPRINT).

Composition and test operations: Finding these operations is straightforward. By taking the *union* of two graphs as composition operation (starting with $\{G_{(i,i)}\} = \{G_i\}$), it follows that the lowest row $\mathcal{G}^k$ such that all graphs *equal* the footprint indicate, by definition, the answer k. So, the composition operation is *union* and the test operation is *equality to footprint* (with labelled nodes, not to be mistaken with isomorphism).

4.2 Temporal Diameter

A dynamic graph might never be connected at one time, and yet offer a form of connectivity over time and space based on journeys (or temporal paths). A journey is a sequence of adjacent edges which are available at increasing dates (with possible intermediate pauses). This concept has different definitions depending on the model. In a sequence of graphs, one may consider that at most one hop can be performed in each G_i for each journey. Other definitions are possible, but we consider this one here. Then, the *temporal diameter* of $\mathcal{G}$ at time t is the smallest duration d such that for all nodes u and v, u can reach v through a journey before time $t+d$. We consider the problem of finding the smallest duration d such that the *temporal diameter* of $\mathcal{G}$ is less or equal to d at any time $t < \delta - d$ (problem TEMPORAL-DIAMETER). In other words, any subsequence of length d is temporally connected. Several solutions exist to this or similar problems (e.g. [3, 5]), which operate at a lower level of abstraction. Here, we show how the problem fits elegantly within the proposed framework.

The hierarchy built here is one of *transitive closures* of journeys. Therefore, each $G_{(i,j)}$ is a *directed* graph such that an edge (u,v) exists if and only if u can reach v in $\mathcal{G}$ by starting after (or at) i and arriving before (or at) j. As a special case, $G_{(i,i)}$ is the directed (symmetric) version of G_i. Then, the answer is the smallest k such that every graph in row $\mathcal{G}^k$ is a complete graph (i.e. every subsequence of length k is temporally connected).

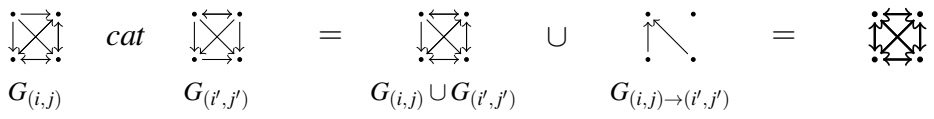


Figure 2: Example of concatenation of transitive closures. *Edges in $G_{(i,j)\to(i',j')}$ are added after the union.*

Composition and test operations: The hierarchy is built through a *concatenation* of transitive closures, $cat(G_{(i,j)}, G_{(i',j')})$, with the restriction that $i' = j+1$ (Lemma 1), defined as follows. First make the union of both graphs, 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 2 for an example. Then, the *test* operation consists of verifying if the graph is complete.

4.3 Round-trip Temporal Diameter

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 duration d such that, between time t and $t+d$, 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 $\mathcal{J}(u,v)$. 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 or equal to d at any time $t \leq \delta - d$.

Composition operation: The composition operation in our case is the *concatenation of round trip transitive closures* $rtcat(G_{(i,j)}, G_{(i',j')})$ with the restriction that $i' = j+1$ (Lemma 1), that computes the *round trip transitive closure* of journeys $G_{(i,j')}$ *i.e.* $(u,v) \in G_{(i,j')}$ iff at least one journey from u to v exists in the sequence $\{G_i, G_{i+1}, ..., G_{j'}\}$, and edges $\{(u,v) \in E(G_{(i,j')})\}$ are labelled with two dates: $departure(u,v,G_{(i,j')})$ the latest departure of any journey in the sequence, and $arrival(u,v,G_{(i,j')})$ the earliest arrival of any journey in the sequence. Note that labels on the same edge may or may not be departure and arrival of the same journey and that $departure(u,v,G_{(i,i)}) = i$ and $arrival(u,v,G_{(i,i)}) = i$.

The composition is 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 $(u,v) \in G_{(i,j)} \cap G_{(i',j')}$. Otherwise, the edge is added with the initial dates. 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 a set of nodes $extra = \{w : (u,w) \in E(G_{(i,j)})$ and $(w,v) \in E(G_{(i',j')})\}$ exists (not empty). $arrival(u,v,G_{(i,j)\to(i',j')}) = min\{arrival(w,v,G_{(i,j)})\}$ and $departure(u,v,G_{(i,j)\to(i',j')}) = max\{departure(u,w,G_{(i,j)})\} : w \in extra$. 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 3).

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, for all edges $\{(u,v)\}$ in the graph, $arrival(u,v,G_{(i,j)}) \leq departure(v,u,G_{(i,j)})$.

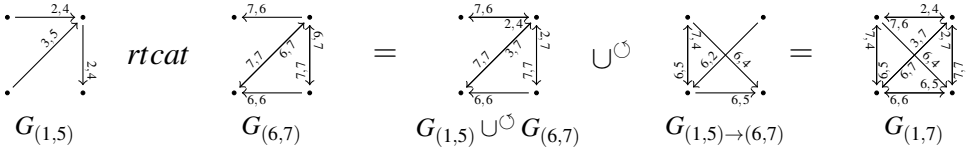


Figure 3: Example of round trip transitive closures concatenation. *Labels arr and dep on an edge $u \xrightarrow{arr,\ dep} v$ (shifted toward the tip) represent respectively $arrival(u,v,G_{(i,j)})$ and $departure(u,v,G_{(i,j)})$.*

References

[1] A. Casteigts, P. Flocchini, W. Quattrociocchi, N. Santoro. Time-varying graphs and dynamic networks. *IJPEDS*, 27(5):387–408, 2012.

[2] A. Casteigts, R. Klasing, Y. M. Neggaz, J. G. Peters. Tester efficacement la T-intervalle connexité dans les graphes dynamiques. *CIAC* 2015 (English) and *ALGOTEL* 2015 (French).

[3] C. Huyghues-Despointes, B.-M. Bui-Xuan, C. Magnien. Forte Δ-connexité dans les flots de liens. *ALGOTEL* 2016.

[4] Y. M. Neggaz. *Automatic Classification of Dynamic Graphs*. Ph.D. Thesis, University of Bordeaux, October 2016.

[5] J. Whitbeck, M. Dias de Amorim, V. Conan, J.-L. Guillaume. Temporal reachability graphs. *MOBICOM* 2012.