Journal of Advances in Science and Technology
Vol. 19, Issue No. 3, September-2022, ISSN 2230-9659

DOI: https://doi.org/10.29070/wgnvd437

Check for updates

Dynamic Communication Networks- Evolving Graphs

Dr. J. K. Navis Vigilia [1] , Dr. V. Margret Ponrani [2]

1. Assistant Professor, Department Of Mathematics, Jyoti Nivas College Autonomous, Bengaluru., India ,
2. Assistant Professor, Department Of Mathematics, Jyoti Nivas College Autonomous, Bengaluru, India

Abstract: New technologies and the deployment of mobile and nomadic services are driving theemergence of complex communications networks, that have a highly dynamic behavior. Modeling suchdynamics, and designing algorithms that take it into account, received considerable attention recently. Inthis paper, we introduce the evolving graphs, a simple model which aims at harnessing the complexity ofan evolving setting as yielded by dynamic communication networks. We exemplify its use through thecomputation of shortest paths under different hypotheses in fixed-schedule dynamic networks.

Keywords: Dynamic networks, models, algorithms, evolving graphs, radio networks, shortest paths

---------------------------X---------------------------

INTRODUCTION

The advent of new technological communication networks, such as the Internet and ad-hoc radio networks, highly motivates the study of several facets of the dynamic behavior of such networks. The underlying mobility of users and/or relays is just one of the factors that contribute to their dynamics. Others include varying link congestion, node and link faults, and components addition and deletion.

The related domain of dynamic graphs has been extensively studied in the past decade (see [DFMSN01] and references therein). In this case, some property (e.g. a minimum spanning tree) of an input graph is computed, and algorithms are given in order to maintain such a property after the graph is modified (typically an addition/deletion of a vertex/edge). One could say that the discrete step is one modification undergone by the graph.

The field of communications in dynamic networks, i.e., networks with evolving characteristics, also receives considerable attention [GAGPK01, KRR+00, LLS01], and even several research projects are already tackling this important issue [ARC, COL, Eur]. Most papers in this area define a network as a graph, and let p be a fault probability for each edge. At each time step, each edge is kept independently at random with probability p. Problems studied under such a random-fault model are, for instance, fault-tolerant routing and the computation of large fault-free connected components [Sch02]. A good example of stochastic mod- els for dynamic networks appears in papers presenting research on the graph of the Web, like in [KRR+00]. In [Vie01], motivated by the modeling of networks of moving robots, a modified Dijkstra algorithm was used to find shortest paths in a dynamic graph, described with the help of an edge-events list, coding the additions/deletions of edges. An excellent state of the art on models and techniques for communication in such networks can be found in [Sch02]. We refer the interested reader to its extensive bibliography, where issues concerning connectivity, routing, and admission control are addressed.

Our work in this area focuses on the design of models and algorithmic techniques that can harness the complexity of an evolving setting as yielded by dynamic communication networks. In this paper, we give a simple but powerful model that captures most characteristics of such networks. The notion of evolving graphs, introduced here, basically consists in formalizing a time domain in graphs. Surprisingly, this leads to a plethora of interesting questions in (algorithmic) graph theory, some of which we investigate here-inafter, through the computation of shortest paths under several settings. This paper is organized as follows. The formal definitions of evolving graphs and of some of their main parameters are given in the next section. Then, in Section 3, we exemplify their use through the computation of shortest paths under different hypotheses in fixed-schedule dynamic networks. We close this paper with concluding remarks and ways for further research.

EVOLVING GRAPHS

In this section, we formally define a model capturing most of the characteristics of dynamic networks, in which to study graph-theoretic properties and algorithms.

Definition 1 (Evolving Graphs) Let a digraph G(V; E) be given, along with an ordered sequence of its subgraphs, SG = G, G1…., GT , T € N. Then, the system G = (G; SG) is called an evolving graph.

Let I = [0, T] . If we consider I as a time interval, where Gi is the subgraph at time instant i, then G = (G, SG) can be seen as a simple time-dependent discrete dynamical system, running during I

We now define some of the main parameters of an evolving graph. Let EG = U Ei, and VG =U Vi. It is clear that M = | EG | ≤|E | = M and that N = |VG |≤ |V | = N.

Two vertices are said to be adjacent in G if and only if they are adjacent in some Gi. The degree of a vertex in G is defined as its degree in EG .

Let P be a path in Gi, under the usual definition. Let F(P) be its source, L(P) be its destination, and |P| be its length. We define a path in G between two vertices u and v of VG as a sequence PG (u; v) = Pt1 ,Pt2 ,…, Ptk , with t1 < t2 <… < tk, such that Pti is a (usually defined) path in Gti with F(Pt1) = u, L(Ptk) = v, and for all i < k it holds that L(Pti) = F(Pti+1). Note that this definition implies that there are no paths in G going to the "past".

A circuit in G is a path in G , PG , such that L(PG) = F(PG).

The length of a path in G, PG(u,v)= Pt1, Pt2….Ptk is | PG(u,v)|= | a path in G between two vertices u and v, with minimum length among all paths in G between u and v, is called timely.

We define the distance in G between two vertices u and v as dG (u, v) = min |PG (u, v)|, taken over all paths in G between u and v. We also define a notion of "distance" in the time domain, as follows.

Let a path in G between two vertices u and v, PG (u, v) = Pt1 , Pt2 ,…; Ptk , be such that tk is minimum. Then, we define stride(u, v) = tk. Roughly speaking, stride(u, v) gives the minimum number of time steps required to go from u to v in G .

We define the diameter of G as D(G) = max dG (u, v), taken over all pairs of vertices in VG . Analogously to the case above, a notion of diameter in the time domain can also be introduced [FV02].

As usual, a tree in G is defined as a connected induced subgraph of VG with no circuits in G . However, one such a tree would not be very helpful when studying connectivity issues, since it does not take into account the total order of the subgraphs in G , and the restrictions it imposes on paths in G . Therefore, we define a valid tree in G as a tree in G where each and all directed paths in the tree are paths in G . Likewise, a valid rooted tree in G is a rooted directed tree where all paths from the root to the leaves are paths in G .

It is interesting to note that the above definitions can be restated including time constraints, like “after time instant t”, “during time interval I1”, etc.

Notice that in the evolving graphs model above, it is made clear that between two subsequent time steps, any changes may happen, with the possible *creation and/or deletion of any number of vertices and arcs.*

Fixed-schedule dynamic networks

A dynamic network can be seen as a potentially infinite - , sequence R =…, Rt-1,Rt ,Rt+1,…. of networks over time. Some of these networks have predictable changes in their topologies ([FV02]), like LEO satellite networks ([FGP02]), or even explicitly fixed schedules of their topologies, like transport networks ([Cou02]). Such fixed-schedule dynamic networks (FSDN) could be seen as a dynamic network which has a presence matrix PE|(u; v), i |, indicating whether (u, v) is present at time step ti, for each link (u, v) of R , and another presence matrix PV |u, i|, indicating whether u is present at time step ti, for each node u of R . The network at time ti is then represented by the subnetwork Rti of R , which is obtained by taking the nodes and links of R for which their corresponding P[i]’s indicate they are to be present.

In order to model a fixed-schedule dynamic network by an evolving graph, it suffices to be given a time window W of size T , and to work with G = (U Ri|i € W, FSDN|W).

COMPUTING SHORTEST PATHS IN EVOLVING GRAPHS

To exemplify the use of evolving graphs, we show in this section how to compute the shortest paths from a source node A to all other nodes. Further interesting questions are proposed in the next section.

We assume the input evolving graph is given as linked adjacency lists, with the sorted arc schedule attached to each neighbor, given as time intervals indicating the time steps where that arc is alive. The head of each list is a vertex with its own sorted node schedule list attached, also given as time intervals. The space used is proportional to the size of the adjacency linked lists, plus the size of the arcs and nodes schedule lists. Therefore, the total size of the lists is O(M + (M + N)T) = O(M T) in the worst case. Note that other ways exist to code a dynamic network ([Vie01, FV02]).

First, remind that the usual Dijkstra’s algorithm [CLR90] proceeds by building a set S of closed vertices, for which the shortest paths have already been computed, then choosing a vertex u not in S whose shortest path estimate, d(u), is minimum, and adding u to S, i.e., closing u. At this point, all arcs from u to V-S are opened, i.e., they are examined and the respective shortest path estimate, d, is updated for all end-points. In order to have quick access to the best shortest path estimate, the algorithm keeps a min-heap priority queue Q with all vertices in V-S, with key d. Note that d is initialized to ∞ for all vertices but for A, which has d = 0.

The main problem to implement such an algorithm in an evolving setting is how to keep the correct values of the shortest path estimate at the time we will need them. Indeed, we need to know the vertex with least estimated distance to the set S only at the appropriate time step. Furthermore, since we are dealing with communication networks, several hypotheses may be made with respect to the actual communication of information. A basic one, valid throughout this text is that we shall consider packet networks. Hence, transmitting one piece of information means transmitting one packet over one arc. Other hypotheses will be specified case by case.

Scenario 1 Packet transmission time is normalized so as to coincide with the duration of a time step $\Delta t) = t_{i+1} - t_i$, which is constant for all i. There is information conservation, i.e., if a vertex disappears and then reappears, it still has the received informations.

This is the easiest case, because all the weights on the arcs can be seen as being unitary. Notwithstanding, in opposition to Dijkstra's algorithm, we may have to open arcs from arbitrary closed nodes in case they appear in an upcoming Gi. The evolution of (discrete) time will be represented by a counter i. Below, we give an efficient algorithm to compute the single-source shortest paths in evolving graphs under Scenario 1.

Algorithm 1

```
1. Make all $d(v) = \infty$ but for $d(A) = 0$. Initialize a min-heap Q with a record (A; key(A) = 1) in the root. Put in Q a dummy record (dummy; key(dummy) = $\infty$).
2. $i \leftarrow 1$.
3.   While key(root(Q)) $\neq \infty$ do
(a)    While key(root(Q)) = i do
i.       Extract x, the vertex at root(Q).
ii.      Delete root(Q).
iii.     Traverse the adjacency list of x, and for each open neighbor v do: compute $f_a(v)$ (the first valid arc schedule time greater or equal to current time step i), and insert v in Q if it was not there already.
iv.      If $f_a(v) < d(v)$ then update d(v) with $f_a(v)$, and key(v) with $f_a(v) + 1$.
v.       Close x and insert it in the shortest paths tree.
vi.      Update Q.
(b)    $i \leftarrow i + 1$.
```

Comments. The variable $f_a(v)$ indicates the earliest that node x can transmit the message to its neighbor

If this is early enough, then the distance from the source A to v becomes $f_a(v)$. The key to the heap key(v) indicates the first moment v can retransmit the received message (in this case, where transmission time is normalized, it would be the time step right after $f_a(v)$). Since i is counting the time steps, once a node is closed, no new path in the future can decrease its distance to A.

The shortest path is found by traversing the shortest paths tree back from a destination to A. In case two successive labels differ by more than 1, this implies that the shortest path yields a forced stay of the information in that vertex for a number of steps, until the connection is established to its successor in the tree.

Analysis. We can see that, starting from A, the algorithm examines all its out-neighbors (┌+(A)), and for each one there is one table look-up to find the valid schedule times, plus a heap update. Therefore, for each closed vertex, the algorithm performs O(log T + log N) operations. Hence, the total number of operations is at most O(∑v V [┌+(v)(log T + log N)]) = O(M (log T + log N)).

As we said, the hypothesis above represents the easiest setting. We will now try and relax it point by point.

Scenario 2 One time step (t) allows for the traversal of k arcs in the graphs.

Actually, relaxing the traversal time constraint in this sense does not cause a big problem. It is like each column of the arc presence matrix had been copied k times. A simple counting strategy could be used to implement it.

Analysis. The same as above, but for the parameter k.

Scenario 3 Crossing an arc takes arbitrary time, given as a positive cost c(u;v) associated to them, butthe subgraphs Gi are only allowed to grow (i.e., no arc disappears).

Under the assumption that no arc disappears [Sti01], then the algorithm above correctly computes all shortest paths, since a vertex x is closed at time step tk if and only if d(x)≤ tk, ensuring that no shorter path can be found for it. The only modification would be when updating the values of all d(v), v ┌+(x), in order to take c(x,v) into account.

Analysis. Same as for Scenario 1.

Now, we look at the interesting case where an arc may disappear before the information had enough time to cross it.

Scenario 4 Crossing an arc takes arbitrary time, given as a positive cost c(u,v) associated to them. Arcs may disappear.

One solution for this problem is to first pre-compute the time intervals (a, b) for each arc (u, v), and only keep those pairs for which b - a ≥ c(u,v). Then, run Algorithm 1 with this new arcs schedule list, checking only whether the transmission fits in the first arc time interval found, using the next one otherwise. A final modification would be when updating the values of all fa(v), v Є ┌+(x), in order to take c(x,v) into account.

Analysis. Same as for Scenario 1.

Scenario 5 There is no longer information conservation, i.e., if a vertex disappears and then reappears, it may have lost the received message

In this case, a node may have to be reached several times before we find out that it was part of a shortest path to another node. Recall that Algorithm 1 first extracts the current root of the heap Q – say vertex x –, then traverses its adjacency list, and finally closes it. In order to cope with the loss of information, we need to change the computation of fa(v) during the adjacency list traversal. For that, let PV (v) =[tl1, tr1], [tl2, tr2],… be the sorted schedule list of node v, with the intervals representing the time intervals in which node v is alive. Let also Left (PV (v)) (respectively, Right(PV (v))) denote the ordered list of all left (respectively, right) limits of such intervals. We will further need a variable, called flag, indicating the time interval in which a node has been closed, in order to avoid it receiving the same information again, while still alive.

We give below the algorithm for this scenario. For the sake of simplicity, we assume again that packet transmission time is normalized so as to coincide with the duration of a time step.

Algorithm 2

```
1.  Make all $d(v) = \infty$ but for $d(A) = 0$. Initialize a min-heap Q with a record (A; key(A) = 1) in the
    root. Put in Q a dummy record (dummy; key(dummy) = $\infty$).
2.  i $\leftarrow$1.
3.  While key(root(Q)) $\neq \infty$ do
    (a)  While key(root(Q)) = i do
         i.    Extract x, the vertex at root(Q).
         ii.   Delete root(Q).
         iii.  If x is not alive at time step i then skip and go to step (3(a)viii).
         iv.   Let [a, b] Є $P_V$ (x) denote the current time interval where x is alive. Traverse the adja-
               cency list of x, and perform the following. For each neighbor v do: if (v is closed and
               flag(v)  [a,b]≠0 then skip. Otherwise do: Locate i into Left(s(v)), getting tjl and tlj+1 such that tjl $\leq$ I <
               tlj+1. If i< rjj, if fa(v) $\leftarrow$ i. Otherwise if tlj+1$\leq$ b, then fa (v) $\leftarrow$ tlj+1
               Insert v in Q if it was not there already.
         v.    Update all open neighbors' d(v) with $f_a(v)$, if needed, and all neighbors' key(v) with $f_a(v) + 1$.
         vi.   Close x with flag(x) = [a, b], and insert it in the shortest paths digraph.
```

```
vii.    Update Q.
    (b)    i ← i + 1.
```

Comments. As before, the variable fa(v) indicates the earliest that node x can transmit the message to its neighbor v. If this is early enough, then the distance from the source A to v becomes fa(v). The key to the heap key(v) indicates the first moment v can retransmit the received message. However, the node may not be alive at that moment implying that only the heap update should be performed when it comes to the root of the heap (test at step 3(a)iii). Moreover, now a closed vertex (i.e., a vertex whose distance to A has already been computed) may become part of a shortest path to another node, because it disappeared from the network and lost the message before being informed again. Thus, we have to keep considering closed nodes, but ensure that it does not receive the message again before it disappears (whence the test on the flags of closed neighbors at step 3(a)iv).

Notice that this algorithm will not produce a shortest path tree, since a node may have to be reached several times in order to ensure information delivery. Therefore, we should build a digraph with labels in the arcs, indicating the time step such an arc was used for transmission. This should be done at step 3(a)vii, when closing the root of Q and inserting it in the shortest paths digraph. In order to compute the shortest paths, we take the inverse path in this digraph from any destination back to A. In each vertex we take the inverse arc with the highest label which is smaller than the label of the arc just used, and mark the arc as done.

Analysis. It is easy to see that the cost of the whole procedure now depends on the number of steps each vertex reappears in the network. This number may be as large as O(T), the schedule length. Therefore, Algorithm 2 performs at most O(T M (log T + log N)) operations. Notice, however, that this number is bounded by the actual size of the schedule lists, which measures the number of changes in the network topology during the time window W.

CONCLUSION AND PERSPECTIVES

In this paper we introduced the notion of evolving graphs, defined as an ordered sequence of subgraphs of a given graph, where paths are not allowed to counter the given order. We then defined some of its parameters and presented shortest paths algorithms that can be applied in fixed-schedule dynamic networks. Scenarios 1 through 5 have been implemented in C++, and a dynamic interface was implemented in GTK [Fas02].

We have two main concluding remarks. First, just like in standard graph algorithms, the assumption on the a priori knowledge about the subgraphs topology can be dropped, giving raise to on-line evolving graph algorithms. Likewise, the assumption on the a priori centralized knowledge can also be dropped, in order to instigate the design of distributed evolving graph algorithms.

Second, it seemed clearer to us to present evolving graphs as a discrete system. We note, however, that the sequence of subgraphs could actually be a continuous system that is discretized according to the application in hand, like by the traversal time of an arc, for instance. In [FV02], an algorithm is given for continuous time.

Turning our attention to perspectives, several open problems in the area of dynamic communication networks are given in [Sch02]. With respect to evolving graphs, we think that many interesting ways for further research are worth pursuing. Even in the simple setting used in this paper, how would more intricate algorithms behave, like multi-packet transmissions or flow algorithms? What happens when the presence matrices represent a probabilistic process? What would be the implications on hard problems of restricted hypotheses over the graph G or over its subgraphs Gi? Not mentioning all the questions arising in the case where a distributed or on-line setting, even partial, is assumed.

ACKNOWLEDGMENTS

The author is grateful to D. Coudert, G. Fasoli, F. Havet, K. Marcus, M. Morvan, S. Perennes, H. Rivano, and L. Viennot, for fruitful discussions, and to the anonymous referees for their insightful comments, which helped improve the presentation of this paper.

References

1. ARC INRIA. Algorithmique et analyse des graphes dynamiques, www.hipercom.inria.fr/soleil.
2. T. Cormen, C. Leiserson, and R. Rivest. Introduction to Algorithms. The MIT Press, 1990.
3. COLOR INRIA Sophia-Antipolis. Proble`mes de dynamicite´ dans les re´seaux.
4. D. Coudert. Private communication, February 2002.
5. C. Demetrescu, D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Maintaining shortest paths in digraphs with arbitrary arc weights: An experimental study. In Proceedings of the 4th Workshop on Algorithm Engineering (WAE) 2000, volume 1982 of LNCS, pages 218–229, Saarbru¨cken, Germany, September 2001.
6. European FET project. Critical Resource Sharing for Cooperation in Complex Systems.
7. G. Fasoli. Graphes e´volutifs et re´seaux radio-mobiles : simulation et visualisation d’algorithmes de plus court chemins. Rapport de stage, EPF & INRIA Sophia Antipolis, avril 2002.
8. A. Ferreira, J. Galtier, and P. Penna. Topological design, routing and handover in satellite networks. In I. Stojmenovic, editor, Handbook of Wireless Networks and Mobile Computing, pages 473–493. John Wiley and Sons, 2002.
9. A. Ferreira and L. Viennot. A note on models, algorithms, and data structures for dynamic communication networks. Technical Report 4403, INRIA, 2002.
10. T. Goff, N. Abu-Ghazaleh, D. Phatak, and R. Kahvecioglu. Preemptive routing in ad hoc networks. In Proceedings of the seventh International Conference on Mobile Computing and Networking (MobiCom’01), pages 43–52, Rome, Italy, July 2001. ACM.

11. R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochas- tic models for the web graph. In Proceedings of the IEEE Symposium on Foundations of Computer Science,, 2000.

12. X. Lin, M. Lakshdisi, and I. Stojmenovic. Location based localize alternate, disjoint, multi- path and component routing schemes for wireless networks. In Proceedings of the 2001 ACM International Symposium on Mobile ad hoc Networking and Computing (MobiHoc'01), pages 287–290, October 2001.

13. C. Scheideler. Models and techniques for communication in dynamic networks. In H. Alt and A. Ferreira, editors, Proceedings of the 19th STACS, volume 2285 of LNCS, pages 27–49, Juan-les-Pins, France, March 2002. Springer-Verlag.

14. V. Stix. Finding all maximal cliques in evolving graphs. Technical Report TR2001-05, De- partment of Statistics and Decision Support Systems, University of Vienna, February 2001.

15. L. Viennot. Routage entre robots dont les de´placements sont connus – Un exemple de graphe dynamique. Re´union TAROT, ENST, Paris, Novembre 2001.