# Searching Shortest Paths on Weakly Dynamic Graphs

Jean-Yves COLIN[1], Moustafa NAKECHBANDI[1]
[1]LITIS, Le Havre University, Le Havre, France
{moustafa.nakechbandi,jean-yves.colin}@univ-lehavre.fr
Ahmed Salem OULD CHEIKH[2]
[2]Nouakchott University, Nouakchott, Mauritania
ahdsalem@gmx.fr

***Abstract***. In this paper, we study weakly dynamic graphs, and we propose an efficient polynomial algorithm that computes in advance shortest paths for all possible configurations. No additional computation is needed after any change in the problem because shortest paths are already known in all cases. We apply this result to a dynamic routing problem. In this problem, messages must be sent from some components (captors for example) to a specific one (a processor for example) as quickly as possible. The actual network is a mesh and the problem can represented by a weighted directed acyclic graph. One known arc has unreliable performances.

**Keywords**: shortest paths; dynamic graphs ; route planning

## 1 Inroduction

In complex systems, dynamic graphs in which some of their values or even their topologies may change from one moment to another offer new research opportunities.

One of the most famous algorithm, and one of the most used on static graphs, is the shortest-paths algorithm from E. W. Dijkstra [1]. For the more complex dynamic graphs, several models are proposed, probabilistic [4, 2] or non-probabilistic[7,5]. In non-probabilistic models, most algorithms update previously computed data after each change in a value or after the addition or removal of an edge [8,10]. In probabilistic models, the optimal paths definition of a shortest (or longest) path is different according to the published papers. The most usual definition sets as optimal a path that maximizes the expected value of an utility function chosen by the authors [4, 2]. The problem itself is usually NP-hard.

Among used metaheuristics, one can find ant colony algorithms [9] and other swarm intelligence algorithms. These algorithms are very general and try too to adapt their results following changes in the problem.

In this paper, we study weakly dynamic graphs, and we propose an efficient polynomial algorithm that computes in advance shortest paths for all possible configurations. No computation is needed after any change in the problem because shortest paths are already known in all cases.

We apply this result to one delivery problem for trucks from one regional storehouse to several local stores when one possible point has a variable traversal duration. We apply too this result to the problem of rerouting delivery trucks toward their final destinations when there is a change in the traversal duration of one known point.

## 2 Problem statement

We first define weakly dynamic graphs.

**Definition 2.1**. A weakly dynamic graph is a graph in which there is an unstable valuated edge (in an undirected graph) or valuated arc (in a directed graph) between two known vertices $x_1$ and $x_2$ of the graph. That edge or arc has an unknown positive value $x$ that may change at any time. All other edges or arcs are stable and their values never change.

In the rest of this paper, we will study a weakly dynamic directed acyclic graph $G(V, E)$ with $V$ being the set of vertices of $G$ and E being the set of arcs. Each arc $(i,j)$ of $G$ is valuated by a positive stable value $p_{ij}$, except for two known vertices $x_1$ and $x_2$ of the graph. This arc $(x_1, x_2)$ from $x_1$ to $x_2$ is valuated by an unknown and unstable value x that may change at any time. The length of a path at a given moment is the sum of the values of all the used arcs.

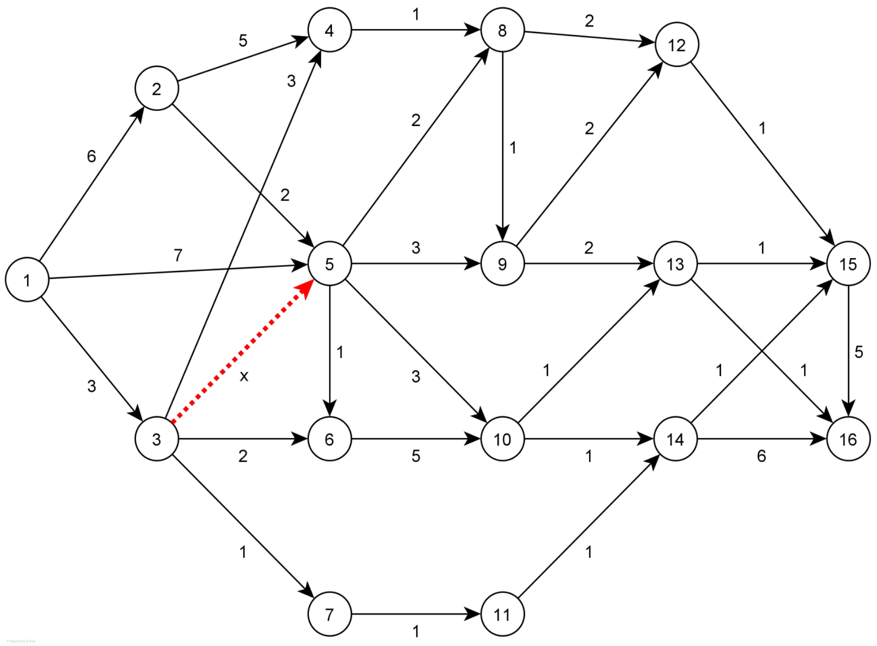

**Fig. 1.** Example of a Weakly Dynamic Graphs The arrow in dotted lines represents the non steady bow

We now want to find in $G$ shortest paths between a given vertex and all other vertices, or alternatively shortest paths from all vertices to a given vertex.

## 3 Main results

### 3.1 The proposed algorithm.

In the following, we study only the problem of finding a shortest path between a given vertex $s_0$ and all other vertices. Finding a shortest path from all vertices to a given vertex is a similar problem that can be solved by using predecessors instead of successors in our algorithm.

The algorithm works in four successive phases:

1. it first builds a set of the vertices that can be reached from the starting vertex using arc $(x_1, x_2)$. Only the successors of $x_2$ are considered because $G$ is a directed graph.
2. next, it computes the length $d(x_2, s_n)$ of the shortest path from vertex $x_2$ to all vertices of the above computed set.
3. then it computes the length $ds(s_0, s_n)$ of the shortest path that does not use arc $(x_1, x_2)$, from vertex $s_0$ to all other vertices of the graph.
4. finally, it computes the length $d(s_0, s_n)$ of the shortest path from vertex $s_0$ to all other vertices of the graph, by comparing

- the length $ds(s_0, x_1) + x + d(x_2, s_n)$ of the shortest path that uses arc $(x_1, x_2)$,
- and the length $ds(s_0, s_n)$ of the shortest path that does not use arc $(x_1, x_2)$. Thus,

$$\begin{aligned} d(s_0, s_n) &= ds(s_0, x_1) + x + d(x_2, s_n), \text{ if } ds(s_0, s_n) > ds(s_0, x_1) + x + d(x_2, s_n) \\ \text{else} &= ds(s_0, s_n) \end{aligned}$$

Because the length of any shortest path that uses arc $(x_1, x_2)$ depends on the value of $x$, the length $d(s_0, s_k)$ of the shortest path from vertex $s_0$ to another vertex $s_k$, will also depends on $x$. In most cases for a vertex $s_k$, there will be one value $x(s_k)$ of $x$ such that, if $x$ is inferior to this value $x(s_k)$, the shortest path will use arc (($x_1, x_2$), else it will not use it. Each path itself to each vertex $s_n$ is computed during the lengths computations.

**Example 3.1. :** We now apply this algorithm on the graph of Figure 1.

**Step 1 :** the set of direct or indirect successors of $x_2$ is {6, 8, 9, 10, 12, 13, 14, 15, 16}. The sub-graph corresponding is:

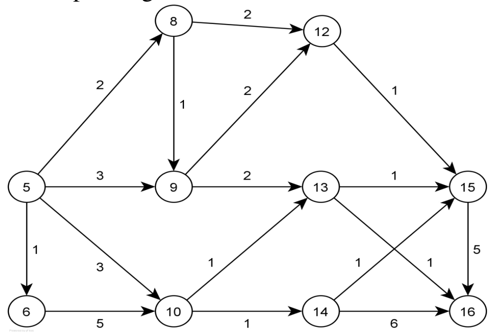

**Fig. 2.** Sub-graph generate by step1

**Step 2 :** the computed shortest paths from $x_2$ to these successors $s_n$ have lengths $d(x_2, s_n)$ of

| $s_n$ | 5 | 6 | 8 | 9 | 10 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|
| $ds(x_2,s_n)$ | 0 | 1 | 2 | 3 | 3 | 4 | 5 | 4 | 5 | 6 |

**Step 3 :** the computed lengths $ds(s_0, s_n)$ of shortest paths that do not use arc $(x_1, x_2)$, from vertex $s_0$ (with $s_0$ being vertex 1) to all other vertices of the graph, are

| $s_n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $ds(x_2,s_n)$ | 0 | 6 | 3 | 6 | 7 | 5 | 4 | 7 | 8 |

| $s_n$ | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|
| $ds(x_2,s_n)$ | 10 | 5 | 9 | 10 | 6 | 7 | 11 |

**Step 4 :** the lengths $d(x_1, x_2)$ of the shortest paths for all vertices is the minimum of the two values not using arc $(x_1, x_2)$ and using arc $(x_1, x_2)$.

| $s_n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| *If not using arc* $(x_1, x_2)$ | 0 | 6 | 3 | 6 | 7 | 5 | 4 | 7 | 8 |
| *if using arc* $(x_1, x_2)$ | - | - | - | - | 3+x | 4+x | - | 5+x | 6+x |

| $s_n$ | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|
| *If not using arc* $(x_1, x_2)$ | 10 | 5 | 9 | 10 | 6 | 7 |
| *if using arc* $(x_1, x_2)$ | 6+x | - | 7+x | 8+x | 6+x | 7+x |

Thus, for some vertices and some values of $x$, the shortest path uses arc $(x_1, x_2)$, and for these vertices and some larger values of $x$, the shortest path does not use arc $(x_1, x_2)$.

## 3.2 Some proprieties of the algorithm

**Theorem 3.2.1**. The paths computed by the algorithm are shortest paths.

**Theorem 3.2.2** The algorithm complexity is $O(n^2)$.

**Definition 3.2.1** we call critical value of $x$ for a vertex $s_k$ the value $x(s_m)$ such that, if x is inferior to this value, the shortest path from vertex $s_0$ to vertex $s_k$ will use arc $(x_1, x_2)$ and will have a length that depends on $x$, else it will not use it and its length will be constant.

**Theorem 3.2.3** each vertex $s_m$ that is a direct or indirect successor of vertex $x_2$ has 0 or 1 critical value for the computation of a shortest path from $s_0$ to this vertex.

A corollary of the last theorem is that the number of critical values is a finite number. Furthermore, if we sort in ascending order the critical values of all vertices of the graph, one can remark that the computed set of shortest paths from $s_0$ to all other vertices in the graph is the same for all values of $x$ between two consecutive values critical values. So the proposed algorithm can be used to efficiently compute shortest paths for all possible values of $x$ from a given vertex to all other vertices. It can be used too to efficiently compute shortest paths from all vertices of a graph to a given target vertex, by using the predecessors instead of the successors during the computation.

## 4 Application to a routing problem

We now apply the algorithm to the following dynamic routing problem. Messages must be sent from some components (captors for example) to a specific one (a processor for example) as quickly as possible. The actual network is a mesh and can represented in this problem by a weighted directed acyclic graph [12]. One arc is known to have unreliable performances for some reason. This problem is thus a weakly dynamic graph problem with vertex $s_n$ being the destination, and arc $(x_1, x_2)$ being the unreliable arc (cf. Fig. 3).

As said above, we may use a version of the proposed algorithm that uses the predecessors instead of the successors. This allows us to compute shortest paths from all other vertices to vertex $s_n$. In the example graph of Figure 4, there then are two critical values, 2 and 4, giving 3 intervals. Each interval has its own (inverted tree) of shortest paths spanning the whole graph and leading to vertex $s_n$ (cf. Fig. 4). Each gives then a routing policy that is optimal in its interval. These routing policies are then all precalculated and stored in the vertices with their relevant critical values of validity. The current value of x is next used to decide what routing will be used. The unreliable arc is then monitored. When its value changes, the new value is transmitted to all vertices. Each vertex compares it to the critical values and may immediately decide to keep using the currently used local routing policy, or switch to the already computed one best suited to the new value of *x*, just by checking in which interval the value x is now in. Thus, no time is lost recomputing new optimal paths.

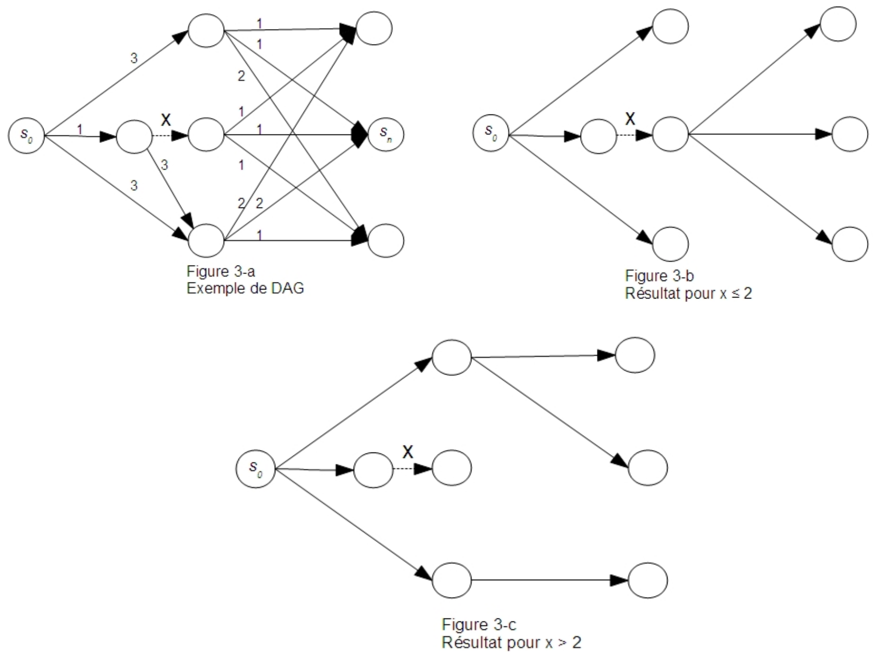

**Fig. 3.** Application 1
Trees of shorter path from vertex s0 to all others

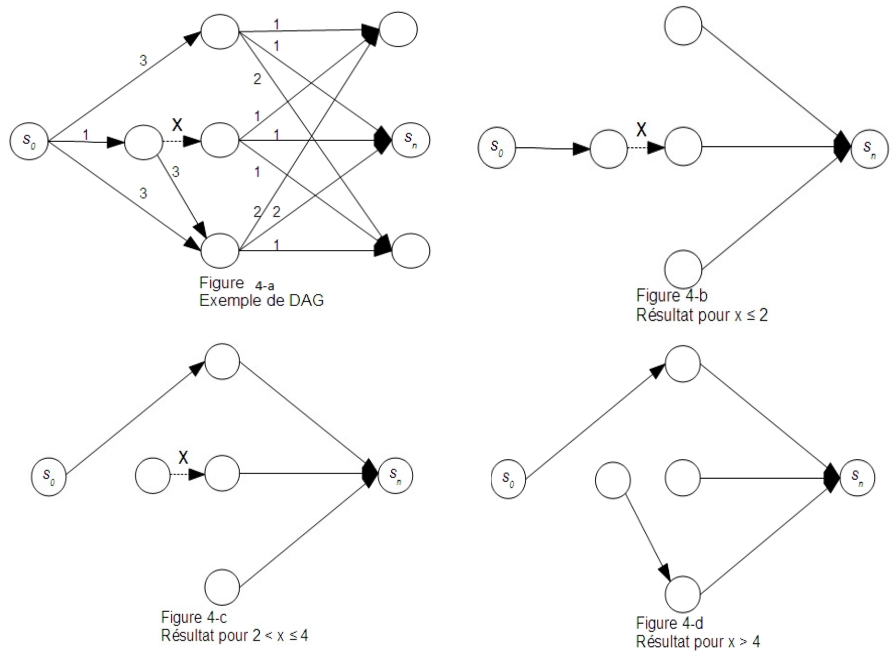

**Fig. 4.** Application 2
Trees of shorter path from all vertex to $s_n$
Here one uses the graph of the previous application(figure 3-a)

## 5 Conclusions

In this paper, we proposed a new graph model we call weakly dynamic graph, and we presented an algorithm to compute shortest paths for all possible cases in a given graph. This algorithm has a polynomial complexity.

We intend to extend this study to weakly dynamic non oriented graphs, to graphs with 2 or more variable arcs, to logistical routing problems and to the longest paths algorithms used in scheduling and project management.

## References

1. E.W. Dijkstra. A note on two problems in connexion with graphs. Numerishe Matematik, 1959, 1:269 271 (1959).
2. D. Fulkerson, Expected critical path lengths in PERT networks. Operations Research 1962, 10 808 - 817 (1962).
3. M. Minoux. Structures algébriques généralisées des problèmes de cheminement dans les graphes: théorèmes, algorithmes et applications. RAIRO, Recherche opérationnelle,1976, vol.10 n°6, pp. 33–62, (1976).
4. P.B Mirchandani et H. Soroush. Optimal paths in probabilistic networks. A case with temporary preferences . Computers & Operations Research, 1985, 12:365 - 383 (1985).

5. A. Orda, R. Rom. Shortest-path and minimum-delay algorithms in networks with time dependent edgelength. J. ACM, 1990, 37(3), 607–625 (1990).
6. G. Ramalingam, T. Reps. On the computational complexity of dynamic graph problems, Theoret. Comput. Sci. 1996, 158 233-277 (1996).
7. F. Kuhn and R. Oshman. Dynamic networks: models and algorithms. SIGACT News, 2011, 42(1):82–96 (2011).
8. C. Demetrescu and G. F. Italiano. A new approach to dynamic all pairs shortest paths. J. ACM, 2004, 51(6):968–992 (2004).
9. S. Balev and F. Guinand and Y. Pigné Maintaining Shortest Paths in Dynamic Graphs. In proceedings of the International Conference on Non-Convex Programming: Local and Global Approaches Theory, Algorithms and Applications (NCP'O7). 2007, December 17-21, Rouen (2007).
10. H. Mao, Pathfinding Algorithms for Mutating Graphs. Computer Systems Lab 2007-2008 (2008).
11. M. Alivand, A.A. Alesheikh and M.R. Malek, New method for finding optimal path in dynamic networks. World Applied Sci. J.,2008, 3: 25-33 (2008).
12. H. Bast, E. Carlsson, A. Eigenwillig, R. Geisberger, Ch. Harrelson, V. Raychev, F. Viger. Fast Routing in Very Large Public Transportation Networks Using Transfer Patterns. 2008, ESA (1), 290-301, (2010).