A theoretical and experimental study of a new algorithm for minimum cost flow in dynamic graphs

Mathilde Vernet, Maciej Drozdowski, Yoann Pigné, Eric Sanlaville

A Theoretical and Experimental Study of a New Algorithm for Minimum Cost Flow in Dynamic Graphs

Mathilde Vernet Maciej Drozdowski Yoann Pigné
Eric Sanlaville

Abstract

This work focuses on the minimum cost flow problem in dynamic graphs. The model here proposed does not have travel time nor storage but all other parameters are time-dependent. The most popular method to solve this kind of problem uses the time-expanded graph but due to the large size of this graph, this solution technique is not manageable on a long time horizon. We propose a new approach that does not use the time-expanded graph. Our algorithm is based on the computation of successive shortest paths on static graphs, which are much smaller compared to the time-expanded graph. It has a significantly better computational complexity. Our method was implemented using the GraphStream library and experiments showed that it is also much faster than the classical method in practice.

1 Introduction

Graphs model interactions between entities, and are widely used in real world applications. Unfortunately static graphs are not always appropriate when time is an issue. We want our graphs to be able to represent a system whose entities' interactions may evolve with time. For instance, transportation networks can change over time when roads are blocked for a certain amount of time. Communication networks and social networks are not constant over time because the represented relations vary. A molecular structure

may evolve when its environment (temperature or pressure) changes. This is a reason to study dynamic graphs and graph problems in a dynamic context.

Dynamic graphs are an extension of graphs where there is dynamicity on vertex and/or arc sets: they are modified over time. Vertices and arcs can appear or disappear. Weights on vertices or arcs can also be time-dependent.

The time dimension in flow networks opened a new area of modeling. Since the pioneer works of Ford and Fulkerson (1958, 1962), the method mainly used to solve a flow problem on a dynamic graph consists in building the time-expanded graph. It allows the direct use of algorithms developed for the static case. This method has a major drawback: the time and space complexity is large. We present a new method using the Successive Shortest Path (SSP) algorithm of smaller complexity to solve the minimum cost flow problem in a specific dynamic context. Our method outperforms theoretically and experimentally an algorithm using the time-expanded graph.

Further organization of the paper is the following: Section 2 presents an overview of works done on dynamic graphs both from modeling and algorithmic point of view, and in particular works on minimum cost flow in a dynamic context. The model used and the problem studied are introduced in Section 3. Section 4 describes our method, and experimental results are presented in Section 5. Concluding remarks are given in Section 6.

2 State of the art

The literature on dynamic graphs has been very prolific these last years. This section focuses on the first seminal works on dynamic graphs and then on papers dealing with flow problems on dynamic graphs.

2.1 Graphs and Temporal Dynamicity

Dynamic graphs have been extensively studied for decades by different communities producing works on many application domains, with various models and terminologies. The consequence is that several formalisms and many different models can be found in the literature. Models differ with respect to which parameters change (vertices, arcs, weights) and how they change (continuously and discretely). For a complete overview, see Holme (2015).

Holme (2015) uses the term of temporal networks. He lists different domains and systems that can be modeled using temporal networks such as

communication, transportation or biological networks. Different ways used to represent and study temporal networks and their structure are described.

Casteigts et al. (2012) propose the term of time-varying graphs (TVG). They define different concepts that can be found in dynamic graphs, such as journeys (time-respecting paths). They detail three different points of view on graph evolution. The first one is the edge-centric evolution where the presence of each edge is defined as a union of time intervals. The second point of view is the vertex-centric evolution where, for each vertex, the set of its neighbors at a specific time is defined. The last point of view is the graph-centric evolution where the dynamic graph is seen as a sequence of static graphs. Casteigts et al. (2012) also describe classes of TVGs based on the graph structure or on the properties of the graph. They propose ways to build graphs with the given properties. Note that our paper chooses the graph centric evolution point of view.

Michail (2016) uses the term temporal graphs. His work presents properties about the graph structure and works that were done on graph problems applied to a dynamic context.

In all the works considered so far, time takes discrete values but it can also be considered as continuous values (Anderson et al., 1982). In this case however, discretization is necessary to solve optimization problems.

Different optimization problems have been studied on dynamic graphs. Many papers consider the shortest path problem, see for instance (Xuan et al., 2003), especially with applications to large-scale road traffic networks (Nannicini et al., 2010). The minimum spanning tree (Huang et al., 2015) and the traveling salesman problem (Michail and Spirakis, 2016) have also been investigated. Due to their practical importance, flows have also been widely studied.

2.2 Flows in dynamic graphs

2.2.1 Dynamic Flows

Skutella (2009) gives important definitions and major results obtained in the research on dynamic flows.

Flows in dynamic graphs can be defined on different dynamic graph models. Figure 1 shows all possible models depending on the different parameters, whether they are defined or not, time-varying or not.

Furthermore, different flow problems can be addressed. The maximum

flow problem, as in the static case, aims at sending as much flow as possible through the graph respecting the constraints of the graph and the time horizon. In the dynamic case, the minimum cost flow problem minimizes arc travel costs and storage costs. Another flow problem, specific to the dynamic case, and widely studied, is the quickest flow problem where the goal is to send a fixed amount of flow through the graph minimizing the latest arrival at the sink.

Flows in dynamic graphs have been studied since the 1950s beginning with the works of Ford and Fulkerson (1958, 1962). In their models, a time dimension is added to the graphs but all parameters are constant over time. The flows are defined over time using travel time that is added on arcs. The same kind of models is used by Wilkinson (1971); Hoppe and Tardos (1994, 2000).

Minieka (1973) is the first to work on a model where parameters are not constant over time. In his work, arcs capacities and arcs travel times are time-dependent.

Baumann and Köhler (2007) propose a model where the arc capacity is constant over time but the arc travel time depends on the flow over this arc. They give approximation results for the earliest arrival flow problem (also known as universal maximum flow problem), where the flow arriving at the sink at each time step must be maximum.

Most works use the time-expanded graph to solve their problems. The time-expanded graph is a static graph in which each vertex corresponds to a vertex of the dynamic graph at one specific time step. It carries the same information as the dynamic graph (more details about the time-expanded graph are given in Section 3.3). Halpern (1979) is the first to propose a new method that does not use the time-expanded graph. It is inspired by (Ford and Fulkerson, 1956). In the case of a highly dynamic graph, the author himself admits that this method might not be as efficient as using the time-expanded graph. Akrida et al. (2019) propose a model in which each arc has a set of labels indicating the time steps of availability of the arc. The arc capacity is constant in all the time steps of arc availability. Storage is allowed and a constant storage capacity is defined. The case when storage is unlimited is also studied. They define a temporal cut and prove that the maximum temporal flow is equal to the minimal temporal cut. The authors describe a simplified time-expanded graph (STEG) which size is linear on the input size of the dynamic graph unlike the time-expanded graph used in the literature to solve such problems. In their paper, the authors also address the

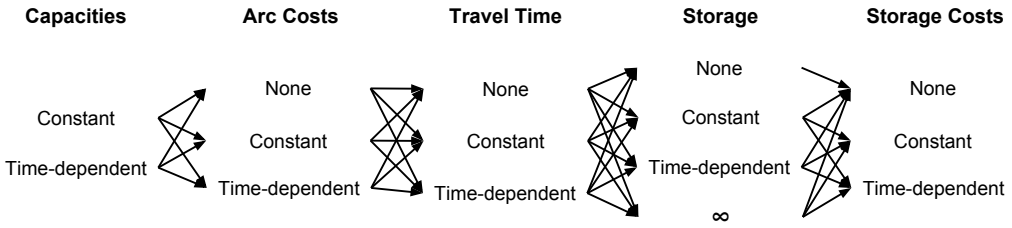


Figure 1: All possible models of dynamic networks for flows.

problem of expected maximum temporal flow where the labels of presence are randomly chosen for a fixed subset of edges in the graph.

Some works consider time to be continuous rather than discrete. Therefore, flow is seen as a continuous function of time. Anderson et al. (1982), or more recently Hashemi and Nasrabadi (2012), work on dynamic graphs with continuous time but solving the problem implies a discretization of time. In the following, we focus on the discrete case.

2.2.2 Dynamic Minimum Cost Flow Algorithms

The minimum cost flow problem over time is NP-hard (Skutella, 2009) when the number of time steps T is a parameter and the capacity, travel time and costs either change at a finite subset of time steps or do not change at all. If these parameters are allowed to change at each time step, then the size of the data is linear in T and not logarithmic in T. Therefore, the problem, solved by an algorithm polynomial in T, is polynomial. Indeed, the classical algorithms on the time-expanded graph are polynomial in the number of vertices n, the number of arcs m and the number of time steps T. However, most works addressing this problem present either a pseudo-polynomial time algorithm or an approximation algorithm. Note that many theoretical studies on minimum cost flow problem in a dynamic context were carried on, but most of them do not offer computational experiments to validate their results.

Many works are conducted on models with constant parameters, while time dimension is present through travel time, usually also constant. Orlin (1984) works on the minimum cost flow problem with an infinite time horizon. No supplies, demands, sources or sinks are defined in this model. Arc travel times are constant over time, arc costs are defined as convex functions of flow. A polynomial time algorithm is described to solve the problem on this infinite model. Fleischer and Skutella (2003) work on a model with con-

stant parameters. They define a condensed time-expanded graph where it is not necessary to copy every vertex for every time step, under some conditions on arc travel time, or by rounding up the travel time. The authors propose an approximation algorithm to solve the minimum cost flow problem. Rostami and Ebrahimnejad (2014) work on a similar model and also propose an approximation algorithm. Klinz and Woeginger (2004) work on a model with constant parameters and infinite storage. They address the minimum cost flow problem and present two variants of this problem. First, the minimum cost maximum dynamic flow problem where the number of flow units to send is fixed to the maximum flow on this graph under the time horizon T. The second is the minimum cost quickest flow problem where the end of the time horizon T is fixed to the minimum time such that the defined amount of flow can be sent. They define a class of graphs for which a greedy algorithm optimally solves the minimum cost flow problem. Lin and Jaillet (2015) work on the quickest flow problem on a graph with constant parameters and propose a method to solve their problem using a minimum cost flow. A recent work, conducted by Grande et al. (2018), proposes a column generation method to solve the minimum cost flow problem. They work on a model with constant parameters, without storage, with multiple sources and sinks. Computational experiments are also presented.

The minimum cost flow problem is also studied on models with time-varying parameters. Cai et al. (2001) propose an algorithm based on SSP. They design a shortest dynamic path algorithm with complexity $O(n \cdot m \cdot T^2)$. Note that this complexity can be achieved by the Bellman-Ford algorithm on the time-expanded graph. Miller-Hooks and Patterson (2004) work on a dynamic model where arc capacities and arc travel times are time-dependent. Infinite storage is allowed on vertices. Supplies and demands are not defined globally for the whole period the graph is studied on, but they are defined at each time step. Their goal is to solve the time-dependent quickest flow problem. In order to achieve this, they define the time-dependent minimum time dynamic flow problem, which is a minimum cost flow problem where the arc cost is the arc travel time. An optimal solution to this problem gives an optimal solution to the quickest flow problem. Their method does not use the time-expanded graph and has complexity $O(U \cdot n^2 \cdot T^2)$ where U is the number of flow units to send. Using the successive shortest path algorithm, with Dijkstra algorithm used at each iteration, on the time-expanded graph would take $O(U \cdot T \cdot (m + n \cdot \log(n \cdot T)))$. Parpalea and Ciurea (2011) give an algorithm to obtain the maximum flow of minimum cost on graphs where capacities and

travel times are time-dependent and storage on vertices is forbidden. Their algorithm, based on successive shortest paths, and first described in (Ahuja et al., 1993), works on the time-expanded graph. Nasrabadi and Hashemi (2010) work on a very general model where arc capacities, arc costs, arc travel time, storage capacities, storage costs and vertex supplies and demands are time-dependent. They solve the associated minimum cost flow problem. Note that the successive shortest path algorithm can be used directly on the time-expanded graph in this case and it has complexity $O(B \cdot n^2 \cdot T^2)$ where B is an upper bound on the total supply. However, they propose an algorithm based on the successive shortest path algorithm on a compact residual graph that has complexity $O(B \cdot n \cdot T \cdot (n + T))$, which is strictly better. Hashemi and Nasrabadi (2012) also work on a model with continuous time and solve the minimum cost flow problem through discretization.

3 Problem Description

3.1 Definitions

We work on dynamic graphs that evolve through time in a discrete way. The graphs, with n vertices and m arcs, are defined on a set of time steps $\mathcal{T} = \{1, \ldots, T\}$, where T is called the time horizon. The dynamic graph G is defined as a succession of static graphs: $G = (G_\theta)_{\theta \in \mathcal{T}}$, where G_θ, called *t-graph*, is the graph at time step θ.

We assume a flow network model without travel time. Flow units travel instantaneously from a vertex to its neighbors. Storage on vertices is not allowed. A flow unit arriving at a vertex has to leave this vertex immediately and is not allowed to "wait" on it before going to a neighbor. Arc capacities, noted u, are time-dependent and supposed integers. The capacity of arc ij at time θ is noted $u_\theta(ij)$. Arc costs, noted c, are also time-dependent and supposed integers. The cost for one unit of flow to go through arc ij at time θ is noted $c_\theta(ij)$. Without loss of generality, vertices and arcs are present over all the time steps in $\mathcal{T}$. Their accessibility is determined by arc capacities. If an arc capacity is 0, this arc cannot be used to transfer flow units. If every incoming arc of a vertex has 0 capacity, then this vertex is unreachable.

We consider the minimum cost flow problem. A source and a sink are defined in our graphs. The source, as well as the sink, is supposed unique but it is always possible to model several sources and sinks with a unique source

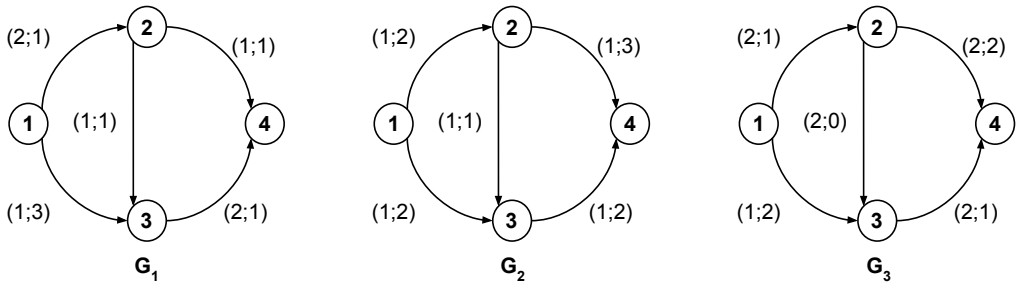


Figure 2: Dynamic Graph over 3 time steps. Each time step is represented. On arc ij, couple $(u(ij); c(ij))$ represents arc ij capacity and cost, respectively. 4 flow units have to be sent on this graph.

and a unique sink. The source has a supply that needs to be evacuated to the sink before the end of the time horizon at the minimum possible cost. This global supply is defined for the whole set $\mathcal{T}$ and not for each time step separately. The global supply is what makes this problem relevant to our model. Indeed, a supply defined at each time step implies solving the problem independently at each time step. In our model, the minimum cost flow at each time step does not give a solution for the minimum cost flow on $\mathcal{T}$. In order to get the optimum solution, we might need to skip a few time steps that are too expensive (see example in Section 3.2).

3.2 Example

Figure 2 presents the evolution of a dynamic graph over 3 time steps ($\mathcal{T} = \{1, 2, 3\}$). Vertex 1 is the source, vertex 4 is the sink. We have to send 4 flow units through this graph. The optimum solution here is to send 1 unit in G_1 on path $(1, 2, 4)$ at cost 2, 1 unit in G_1 on path $(1, 2, 3, 4)$ at cost 3 and 2 units in G_3 on path $(1, 2, 3, 4)$ at cost 4. The minimum cost flow of value 4 has cost 9. The optimum solution avoids G_2.

Remark that, in the same graph, a flow of value 4 cannot be achieved in any of the t-graphs. The maximum flow in G_1 has value 3 and costs 9. The maximum flow in G_2 has value 2 and costs 9. The maximum flow in G_3 has value 3 and costs 8. The minimum cost flow on the whole set $\mathcal{T}$ cannot be deduced from the minimum cost flow in each t-graph.

3.3 Classical Method

The most popular method to solve flow problems on dynamic graphs involves building the time-expanded graph. The time-expanded graph, noted G^{EXP}, gives a static representation of a dynamic graph without loss of information. The time-expanded version of a dynamic graph $G = (G_\theta)_{\theta \in \mathcal{T}}$ has a vertex for each vertex of G at each time step θ. An arc is added in G^{EXP} from vertex i_θ (corresponding to vertex i of G at time θ) to vertex $j_{\theta+\delta}$ (corresponding vertex j of G at time $\theta + \delta$) if there exists an arc from i to j in G at time step θ having travel time δ. In our specific case, there are no travel times. Therefore $\delta = 0$.

The time-expanded graph is a static graph, therefore, it can be used directly to solve the optimization problems with classical algorithms developed for static graphs (Minieka, 1973; Parpalea and Ciurea, 2011). The solution obtained can be easily transformed into a solution of the dynamic graph. The major issue with this method is the increased size of the instance.

To solve the minimum cost flow problem on a dynamic graph, we can build the time-expanded graph, add a *supersource* with outgoing arcs to the sources of the graph at each time step and a *supersink* with incoming arcs from the sinks of the graph at each time step. This time-expanded graph has exactly $T \cdot n + 2$ vertices and $T \cdot m + 2 \cdot T$ arcs. Figure 3 represents the time-expanded graph for our model corresponding to the example from Figure 2. The dynamic graph has 4 vertices, 5 arcs, 3 time steps. The corresponding time-expanded graph is a static graph with 14 vertices, 21 arcs, and carries the same information as the dynamic graph.

The Successive Shortest Path algorithm (SSP) described by Ahuja et al. (1993) has complexity $O(U \cdot (m + n \cdot \log(n)))$. It uses Dijsktra's algorithm to solve the shortest path problem, hence the part $m + n \cdot \log(n)$ in the complexity. In the general case, U is the maximum supply. When considering a unique source, U is the source supply.

Because of the higher number of vertices and arcs of the time-expanded graph, applying SSP on this graph would have complexity $O(U \cdot T \cdot (m + n \cdot \log(n \cdot T)))$. Let us name this method *Time-Expanded SSP*.

The simple question addressed in this paper is: can we achieve a better complexity?

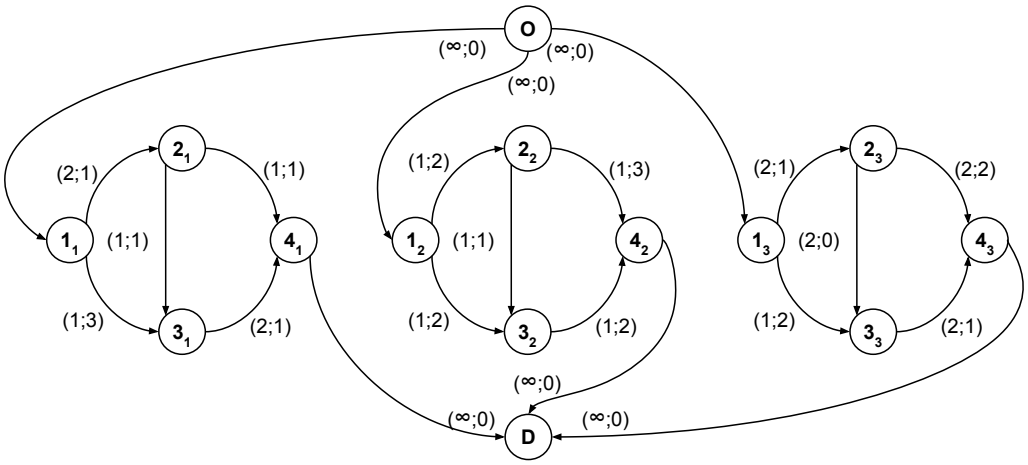


Figure 3: Time-expanded graph of the dynamic graph given in Figure 2. We add a *supersource* O connected to the source of each t-graph G_t and a *supersink* D connected to the sink of each t-graph G_t. The new arcs all have infinite capacity and null cost.

3.4 Applications

As stated earlier, the objective of this paper is to provide a general result on the complexity of finding a minimum cost flow on a dynamic network. Though our time-varying network model and the related minimum cost flow problem may seem basic, they have many practical applications.

Since our model associates no travel durations on the arcs, a common denominator of the applications is the lack of travel time, or travel time negligible compared to the network dynamics. This is indeed the case in communication networks, in logistic networks or in power grids.

In the first case, transporting large amount of data is a major issue, and graph models and their classical algorithms have been used for many years, see for instance the book by Bertsekas et al. (1992), or the one edited by Koster and Muñoz (2009). More specifically, let us look at the problem of sending a large amount of data to a server. This data is partitioned into smaller units (packets), but cannot be stored on intermediate nodes for the lack of storage resources. A single packet travel time is short compared to the whole file transfer time. The capacity of each part of the network depends on what is allowed by the network provider. The transfer cost is also defined by the provider and both can be modified over time depending

on the provider's needs and possibilities. The optimal way to transfer the data can be computed thanks to our model. Notice that this generic scenario applies to scientific data transfers in wide area networks, content streaming, peer to peer networks or sensor networks.

Let us consider logistic networks. Whatever the transportation mode, the links are subject to changes (capacity, availability) because of new or disappearing maritime lines, road works, railway schedule changes, to name just a few. Financial costs may also vary (*e.g.*, tolls according to the season, or fuel prices). These changes, however, do not appear that frequently and may also be anticipated, so that it rarely happens during a transportation (specially for long time horizon and appropriate time step). Moreover, intermediate storage might be too costly and should often be avoided. Démare et al. (2017), for instance, present a model to simulate traffic of goods along the Seine valley.

In deregulated electricity markets, electric energy is sold and bought in day-ahead markets with hourly intervals (Weron, 2014). Then, the energy transfers in the grid are known one day in advance. However, some contractors may under-contract (i.e. contract too little energy) and in order to keep grid stable the shortage of energy must be delivered from alternative sources on a short notice from the transmission system operator. Thus, the network is dynamic because the grid lines free capacity changes hourly, costs of the energy also change over day, travel time is immaterial, storage (in the network itself) is impossible. A minimum cost flow according to our model can construct a minimum cost balancing plan for the grid.

4 New Approach: Dynamic SSP

4.1 Presentation

Solving the minimum cost flow problem with the time-expanded graph as presented in Section 3.3 might not be very efficient. That is the reason we propose a method that does not use the time-expanded graph. Our method, which we named *Dynamic SSP* (DSSP), adapts the SSP algorithm. It uses the fact that in the dynamic graph, the t-graphs are independent because there is no travel time and storage is forbidden. A major feature is that, even though all t-graphs are considered, a complete minimum cost flow computation is not performed on each of them.

The classical SSP algorithm is based on successive searches of shortest paths in the residual graph with Dijkstra algorithm and successive updates of the residual graph. It is pseudo-polynomial but quite efficient in practice. Furthermore, it presents some nice features that are useful in this paper setting:

- Starting from the null flow, it increases at each iteration the value of the obtained flow. And these successive flows are of minimum cost for each of their values (amount of flow sent from source to sink).
- It computes shortest paths on the residual graph, considering the reduced costs associated to the current flow. From the optimality of the successive flows, one directly deduce that the reduced costs are non negative, which allows to use Dijkstra's algorithm.
- A potential $\pi(i)$ is associated to each vertex i and updated at each iteration, as such: $\pi(i) = \pi(i) - d(i)$, where $d(i)$ is the distance from the source to vertex i given by Dijkstra's algorithm. The reduced cost $\bar{c}(ij)$ of arc ij is computed as such: $\bar{c}(ij) = c(ij) + \pi(j) - \pi(i)$ where $c(ij)$ is the unit cost of arc ij. For more details, see (Ahuja et al., 1993).
- Another key feature implied by this optimality is that the successive shortest paths have non-decreasing unit costs (the cost to send one unit of flow along that path). This comes directly from the reduced cost definition. This condition is used below to prove the optimality of our algorithm.

In our version, we compute one iteration of SSP on one t-graph at each iteration of our algorithm. As there is no travel time on arcs or storage on vertices, the t-graphs can be treated independently.

It works as such:

1. Execute the Dijkstra algorithm using the cost as arc length on each t-graph G_θ, $\theta \in \mathcal{T}$.
2. Sort those T shortest paths according to their costs.
3. Let $\bar{\theta}$ be the time step which has the shortest path from the source to the sink.

4. Update current flow and the residual graph $G_{\bar{\theta}}$: reduced costs, vertex potentials and capacities.

5. If there are flow units left to be sent, execute Dijkstra on the residual graph $G_{\bar{\theta}}$ using the reduced cost as arc length, insert the obtained path among the others using its real cost (if the maximum flow value is reached on $G_{\bar{\theta}}$, set the associated real cost to ∞) and go back to step 3.

6. Return the current flow.

Step 1 initializes the algorithm by finding, in each static t-graph G_θ, the shortest path from the source to the sink using the standard Dijkstra algorithm.

In step 2, the shortest paths found in the previous step are sorted using a binary search tree.

Step 3 is the first iteration step. The best shortest path and its corresponding time step $\bar{\theta}$ are easily retrieved thanks to the tree.

In step 4, the updating procedure from SSP is done on the static graph $G_{\bar{\theta}}$ where the best shortest path was. As much flow as possible is sent on this path. The residual graph of $G_{\bar{\theta}}$ is updated as in standard SSP, see (Ahuja et al., 1993).

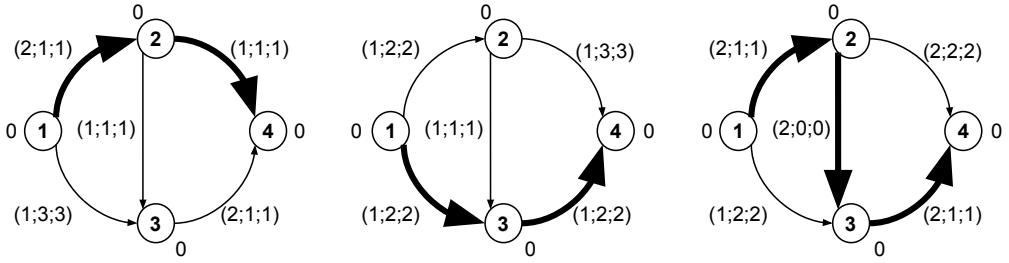
In step 5, Dijkstra is executed on the residual t-graph that was just updated at the previous step using the reduced costs, provided that the current flow value is lower than U. Then this new path is inserted in the tree using its real cost. If no new path can be found on this graph, it means the maximum flow on $G_{\bar{\theta}}$ is reached and no extra flow can be sent at this time step. In this case, the cost of the shortest path for this time step is set to ∞ so $G_{\bar{\theta}}$ can never be selected again. This step is the last iteration step.

Step 6 is reached when the current flow equals U, and it is returned.

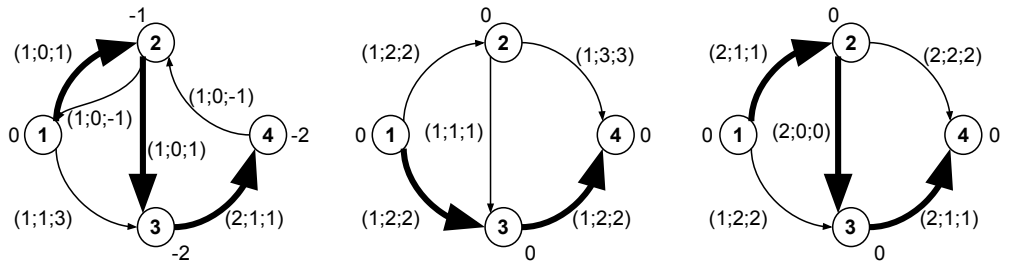
4.2 Example

Let us look at the execution of Dynamic SSP on the example in Figure 2. Figure 4 shows the evolution of the residual graph during the execution of the algorithm and gives the values of reduced costs and vertex potentials.

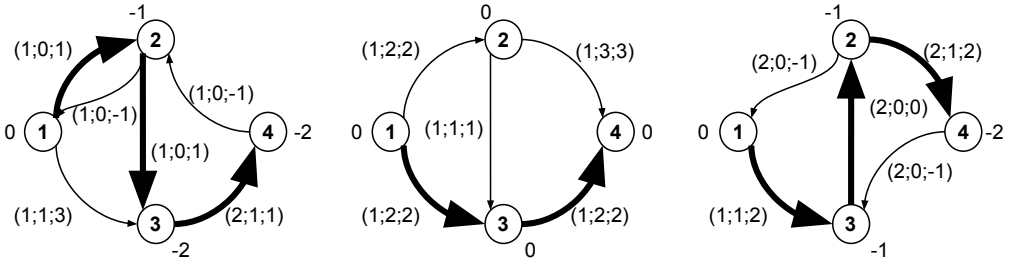
Step 1 is the initialization phase in which a shortest path is found using Dijkstra in each t-graph. Those shortest paths appear in bold in Figure 4a.



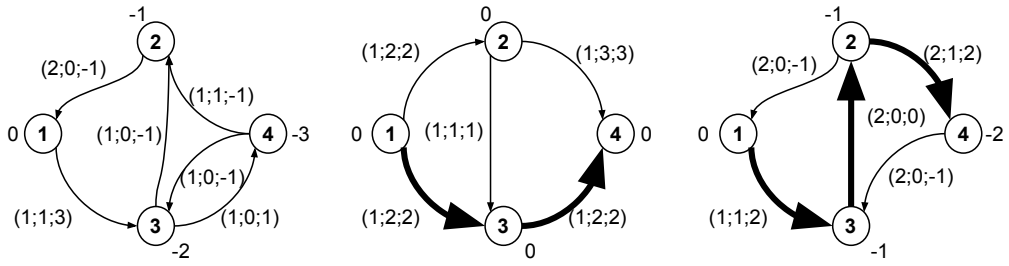
(a) End of the initialization phase (step 1 of the algorithm)



(b) End of the first iteration



(c) End of the second iteration



(d) End of the third iteration

Figure 4: Evolution of the residual graph during execution of Dynamic SSP on example from Figure 2. On arc ij, triple $(u(ij); \bar{c}(ij); c(ij))$ represents, respectively, the capacity of arc ij, the reduced cost of arc ij and the original cost of arc ij. On vertex i, λ_i represents the potential of vertex i. On each residual t-graph, the shortest path given by the Dijkstra algorithm is represented by bold arcs.

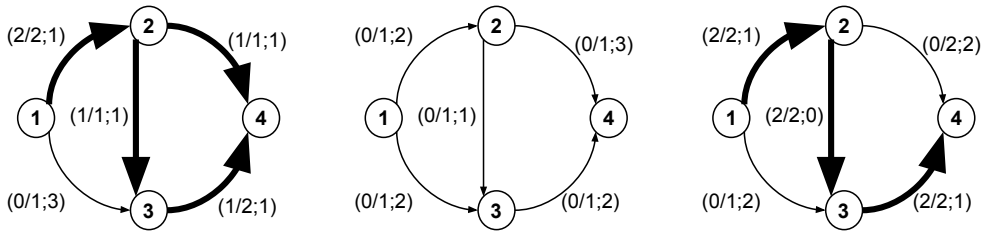


Figure 5: Minimum Cost Flow on the example from Figure 2. On arc ij, couple $(f(ij)/u(ij); c(ij))$ represents the flow on arc ij over the capacity of arc ij and the cost of arc ij respectively. Bold arcs are the ones where flow is sent (where $f(ij) \neq 0$).

In each iteration of the algorithm, a shortest path is chosen. In the first iteration, the real cost of the shortest path for the first t-graph is 2. The real cost of the shortest path for the second t-graph is 4 and the real cost of the shortest path for the third t-graph is 2. We can choose either the first or the third t-graph. In this example, the first t-graph was arbitrarily chosen. As the capacity of this path is 1, we send 1 flow unit through this path at cost 2 and we update this t-graph. The algorithm is not finished because we want to send 4 flow units, so we compute another Dijkstra on the first t-graph in order to know the next shortest path in this graph. The state of the residual graph at this point of the algorithm is shown in Figure 4b.

In the second iteration, the real cost of the shortest path for the first t-graph is 3, the real cost of the shortest path for the second t-graph is 4 and the real cost of the shortest path for the third t-graph is 2. The shortest one is the one in the last t-graph, with a capacity 2. Two flow units are sent through this path at cost 4. We update this t-graph. The algorithm is not finished because we still have 1 flow unit to send, so we compute another Dijkstra on the last t-graph in order to know the next shortest path in this graph. The state of the residual graph at this point of the algorithm is shown in Figure 4c.

In the third iteration, the real cost of the shortest path for the first t-graph is 3, the real cost of the shortest path for the second t-graph is 4 and the real cost of the shortest path for the third t-graph is 4. The shortest one is the one in the first t-graph, with a capacity 1. The last flow unit is sent through this path at cost 3. We update this t-graph. We have sent the 4

flow units we wanted to send so we do not compute another Dijkstra on the first t-graph. The state of the residual graph at this point of the algorithm is shown in Figure 4d.

We sent 2 units at the first time step, one at cost 2 and one at cost 3, and 2 at the last time step at cost 2 for each unit. The units were sent for a total cost 9. The minimum cost flow is presented in Figure 5 in bold.

4.3 Correctness

Theorem 1. *The algorithm terminates in a finite number of iterations bounded by U, the obtained flow has the fixed value U and its total cost is the minimum total cost.*

Proof. Remember that the capacities and unit costs are supposed to be integers.

Assume that U is reachable (otherwise, the algorithm can easily be modified to detect it).

At each iteration, a shortest path in the residual network is considered (steps 1 and 5). According to the properties of the SSP algorithm, their residual capacities are integer and positive, and the resulting successive flows are also integer. At each iteration, one of these paths is chosen, and the total flow value is increased by at least one. The algorithm may stop in two cases. In the first case, the maximum flow value has been reached for all t-graphs, it means that the fixed total flow value U targeted is unreachable (larger than the sum of all maximum flow values). This case is discarded by the hypotheses. In the second case (step 6), U is reached. Hence the algorithm terminates in a finite number of iterations bounded by U.

Let us consider some optimal flow $f^* = (f_1^*, \ldots, f_\theta^*, \ldots, f_T^*)$ where f_θ^* is the flow on t-graph G_θ, and some flow $f = (f_1, \ldots, f_\theta, \ldots, f_T)$ obtained by the algorithm. Both flows have the same value U. For each flow, this value can be decomposed according to the flow amounts for each t-graph. Hence we can write the associated vectors $v^* = (v_1^*, \ldots, v_\theta^*, \ldots, v_T^*)$ and $v = (v_1, \ldots, v_\theta, \ldots, v_T)$, called flow value vectors, where v_θ^* and v_θ are the values of flows f_θ^* and f_θ respectively. Note that as f^* is globally optimal, each f_θ^* is optimal for G_θ and the value v_θ^*. Hence it is possible to consider that each of these flows is obtained by SSP.

Suppose first that f^* and f have the same flow value vector ($v^* = v$, that is, $\forall\ \theta,\ v_\theta^* = v_\theta$). From the remark above, f and f^* are identical on each

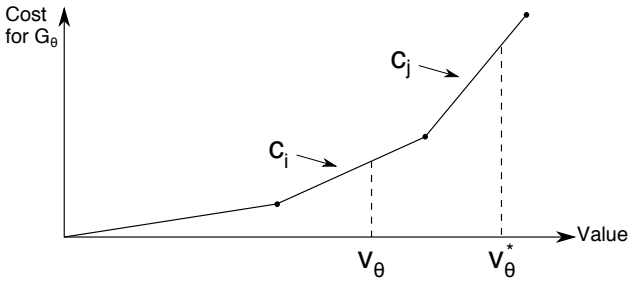


Figure 6: Chart showing the cost of the flow on a graph G_θ depending on the value of this flow.

t-graph, and f is optimal.

Suppose now that there exists some θ such that $v^*_\theta > v_\theta$. Hence there exists some θ' such that $v^*_{\theta'} < v_{\theta'}$. Figure 6 illustrates the flow values and their costs considered in the following. Consider executing SSP on G_θ. The successive unit costs of the computed paths are $c_1 \leq \ldots \leq c_i \leq \ldots \leq c_j$, where c_i is the last unit cost for f and c_j is the last unit cost for f^*. In the same manner, the successive unit costs for $G_{\theta'}$ are $c'_1 \leq \ldots \leq c'_k \leq \ldots \leq c'_l$, where c'_k is the last unit cost for f^* and c'_l is the last unit cost for f.
Suppose first that $i = j$. As $v^*_\theta > v_\theta$, it follows that DSSP do not use the full capacity of the associated path. This is only possible if this path is used during the last iteration of DSSP (when U is reached). Hence $c'_l \leq c_j$, so $c'_k \leq c_j$. Now suppose $i < j$. The path of unit cost c_{i+1} is never used by DSSP, this means $c_{i+1} \geq c'_l$, hence $c_j \geq c_{i+1} \geq c'_l \geq c'_k$. So in both cases $c_j \geq c'_k$. It is possible to remove one unit of flow from f^*_θ and to add this flow unit to $f^*_{\theta'}$. The resulting flow for the dynamic graph is of non-increasing cost, so it is still optimal.

This swapping operation can be performed until the vectors v^* and v are equal, thus f is of minimum cost and the result is proved. □

4.4 Complexity

Theorem 2. *The algorithm Dynamic SSP has worst case complexity* $O((U + T) \cdot (m + n \cdot \log(n) + \log(T)))$.

Proof. In order to efficiently select the cheapest path at each iteration, a tree structure is used to store the paths according to their real costs.

The initialization phase (steps 1 and 2) executes Dijkstra algorithm on T graphs with n vertices and m arcs. Therefore it needs $O(T \cdot (m + n \cdot \log(n)))$ time. It is then necessary, at step 2, to sort the T paths according to their real costs, this is done in $O(T \cdot \log(T))$.

In one iteration (steps 3 to 5), the cheapest path is selected using the costs of T paths and this is done in constant time. Updating the residual t-graph $G_{\bar{\theta}}$ at step 4 is done in $O(n + m)$. Dijkstra algorithm on the t-graph $G_{\bar{\theta}}$ at step 5 is done in $O(m + n \cdot \log(n))$. The new path computed is then inserted at the right place in the structure in $O(\log(T))$. In total, the complexity of one iteration is $O(m + n \cdot \log(n) + \log(T))$.

At each iteration, the flow is increased by at least 1 unit, so the number of iterations is bounded by U, which is the source supply. The worst case complexity of the whole algorithm is $O((U+T) \cdot (m+n \cdot \log(n)+\log(T)))$. □

It should be noted that the worst case complexity of Dynamic SSP is better by an order of magnitude than the worst case complexity of SSP on the time-expanded graph.

5 Numerical Tests

In order to evaluate the algorithm we propose, we implemented it using the GraphStream Java library[1] (Dutot et al., 2007).

The machine used for this experiment has an Intel Core 64 bits processor with 8 cores, 4 MB cache size, 2 GHz frequency and 96 GB of RAM. The OpenJDK 1.8 Runtime Environment is used.

5.1 Experiment Setting

5.1.1 Graph Generation

We implemented a dynamic network generator using the GraphStream Random Euclidean Generator (to build graphs also known as random geometric graphs). This class of graphs is appropriate when, for instance, arc existence is linked to relative spatial positions of the vertices.

The vertices are placed randomly in a finite space $[0, 1] \times [0, 1]$ and are connected by a randomly directed arc if their distance is below a given threshold

[1] http://grahpstream-project.org

	Capacity Vectors	Cost Vectors
min	0	0
max	100	100
inc	10	10
p_{inc}	0.25	0.25
dec	10	10
p_{dec}	0.25	0.25

Table 1: Parameters used to generate capacity vectors and cost vectors

th. There is a direct relation between the threshold and the graph density. The source and sink vertices are designated as the vertices which are the closest to a given position.

Once the graph arc set is built, a capacity vector and a cost vector of size T are randomly generated for each arc. Both vectors are generated by a random process using the same six parameters: min, max, inc, p_{inc}, dec, p_{dec}. The first value in time, of the cost or capacity, is randomly chosen between min and max, and for each time step, the value is increased by inc with a probability p_{inc} or is decreased by dec with a probability p_{dec}. The values cannot go beyond $[min, max]$ range.

The graphs are generated with $th = 0.08$, which corresponds to 1% density approximately. The source and sink are positioned at $(0.25; 0.75)$ and $(0.75; 0.25)$ respectively. The parameters used to generate the capacity vectors and cost vectors are given in Table 1. The idea is to have significant dynamicity but also to avoid ending up with a sequence of independent t-graphs. We want to test the algorithm on a more realistic setup where each t-graph evolves from the previous one.

The parameters we chose allow capacities to be null.

5.1.2 Source Supply

The source supply, in other words the amount of flow to send in the network, needs to be determined. Our choice is to send an amount corresponding to approximately 80% of the maximum flow that can be expected in such a graph.

To determine this amount, experiments were run where a certain number of graphs were built the way explained previously, with one time step. On those graphs, the maximum flow was computed. The median value obtained

gave the maximum flow value that can be expected on one time step which was multiplied by T to get the maximum flow that can be expected on the whole graph. As our t-graphs are independent and built in the same way, the maximum flow for T time steps is close to T times the maximum flow for one time-step. We then kept 80% of this value.

Therefore, the supply grows linearly with T and experiments showed that it also grows linearly with n.

5.1.3 Algorithm Tests

One run of an experiment consists in generating a random dynamic graph and the corresponding time-expanded graph then computing the minimum cost flow using respectively the Dynamic SSP and Time-Expanded SSP. The goal is to compare the time necessary to compute the minimum cost flow in both cases depending on the parameters of the graph. For each set of parameters, 15 runs of the experiment were done. A regression function is computed using the median computation time values. The theoretical behavior defines the form of the function. The regression is evaluated using the R^2 value computed as such $R^2 = 1 - \frac{\sum_{i=1}^{k}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{k}(y_i - \bar{y})^2}$, where k is the number of data values, y_i is the i^{th} data value, $\hat{y}_i$ is the i^{th} predicted value according to the regression function and $\bar{y}$ is the mean data value.

5.2 Run Time Dependence on T

5.2.1 Experiment

For the first experiment, we fix every parameter but T. The number of vertices n is fixed to 500. T takes values in {10; 50; 100; 200; 300; 400; 500; 600; 700; 800; 900; 1000}.

Time-Expanded SSP has worst case complexity $O(U \cdot T \cdot (m + n \cdot \log(n \cdot T)))$, U being the source supply. In our experiment setting, U is linear in T, n is fixed, and as th is also fixed, then m is constant. So the complexity becomes $O(T^2 \cdot \log(T))$.

Dynamic SSP has worst case complexity $O((U + T) \cdot (m + n \cdot \log(n) + \log(T)))$. T is the only parameter that is not constant, so the complexity becomes $O(T \cdot \log(T))$.

The complexity ratio between the Time-Expanded and the Dynamic SSP is T.

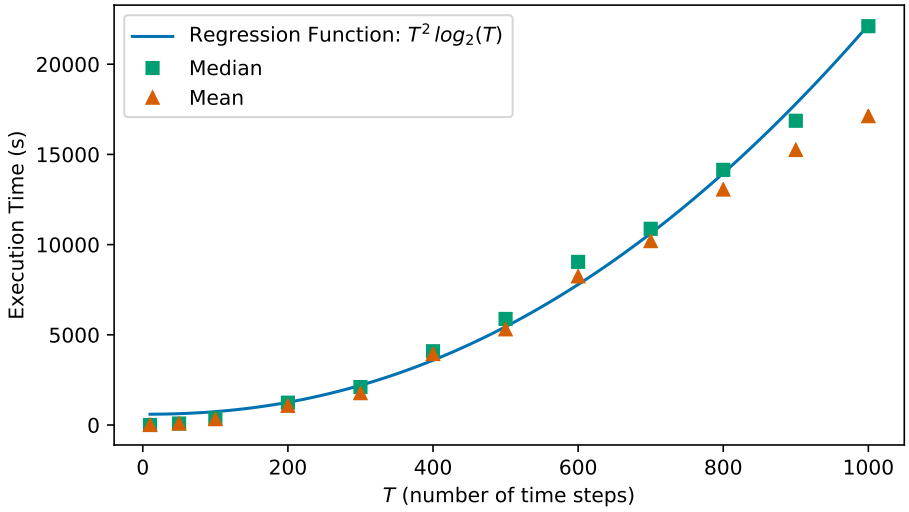


Figure 7: Time-Expanded SSP computation time depending on T, expressed in seconds.

5.2.2 Results

Figure 7 presents the mean and median values of the Time-Expanded SSP computation time depending on T. A regression function of the form $T^2 \cdot \log(T)$ is also represented. The R^2 value of the regression is 0.99. While the computation is almost instantaneous for 10 time steps, it takes up to 20000 seconds (almost 6 hours) for 1000 time steps.

Figure 8 presents the mean and median values of the Dynamic SSP computation time depending on T. A regression function of the form $T \cdot \log(T)$ is also represented and its R^2 value is 0.99. The computation time goes from no more than 1 second (for 10 time steps) to less than 20 seconds (for 1000 time steps).

Figure 9 presents the computation time ratio between the Time-Expanded and the Dynamic SSP depending on T. The mean and median values are shown as well as a linear regression function. The regression R^2 value is 0.94.

5.2.3 Discussion

In Figures 7, 8 and 9, the R^2 values give a strong confidence that the regression function fits the data obtained. It confirms the theoretical behavior

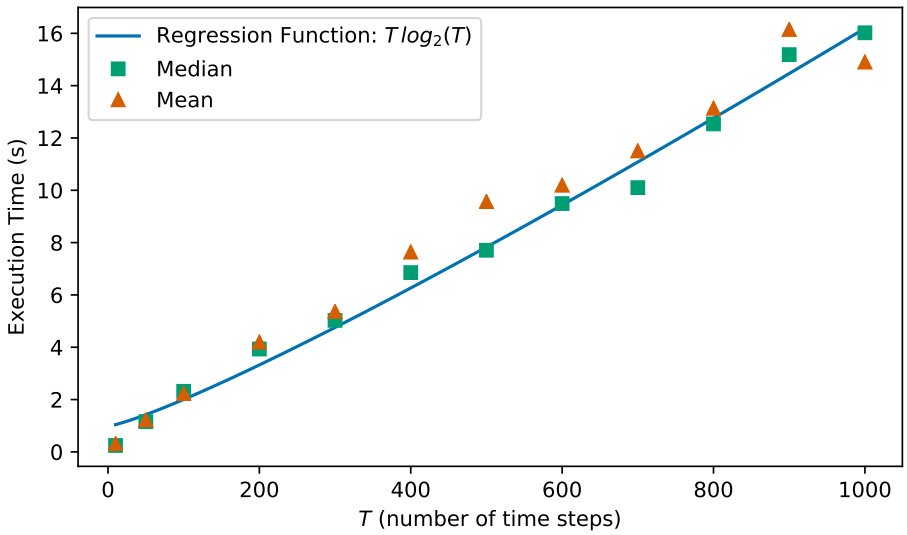


Figure 8: Dynamic SSP computation time depending on T, expressed in seconds.

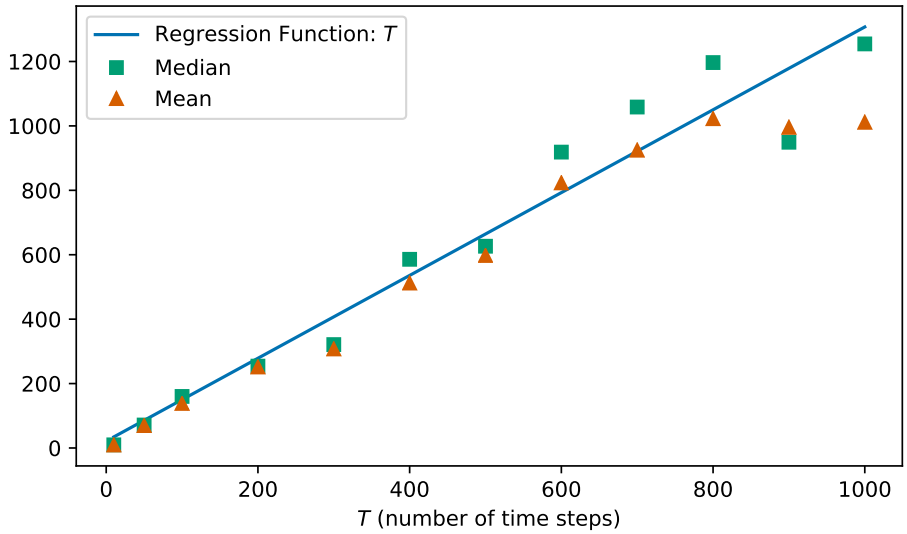


Figure 9: Computation time ratio vs T.

and the fact that the order of average complexity is the same as the order of worst case one.

Dynamic SSP has a better complexity than Time-Expanded SSP. This experiment shows that it is also much more efficient in practice. We can compare the Time-Expanded and the Dynamic SSP computation time (see Figure 9). We observe that it is about 600 times faster with 500 time steps and over 1000 times faster with 1000 time steps.

5.3 Run Time Dependence on n

5.3.1 Experiment

For the second experiment, T is fixed to 100 and n takes values in {500; 700; 900; 1000; 1100; 1300; 1500; 1700; 1900; 2000}.

Time-Expanded SSP has worst case complexity $O(U \cdot T \cdot (m + n \cdot \log(n \cdot T)))$. In the experiment setting, U is linear in n, m is in the order of n^2 and T is constant. The complexity becomes $O(n^3)$.

Dynamic SSP has complexity $O((U + T) \cdot (m + n \cdot \log(n) + \log(T)))$. The parameters are defined as previously explained, so the complexity also becomes $O(n^3)$.

In this case, the complexity ratio between the Time-Expanded and the Dynamic SSP is constant over n.

5.3.2 Results

Figure 10 presents the mean and median values of the Time-Expanded SSP computation time depending on n. A regression function of the form n^3 is also represented and its R^2 value is 0.98. For 500 vertices, the computation is almost instantaneous. For 2000 vertices, it takes up to 120000 seconds (over 33 hours).

Figure 11 presents the mean and median values of the Dynamic SSP computation time depending on n as well as a regression function of the form n^3. The regression R^2 value is 0.98. The computation is almost instantaneous for 500 vertices and takes over 700 seconds (less than 12 minutes) for 2000 vertices.

Figure 12 presents the computation time ratio between the Time-Expanded and the Dynamic SSP depending on n. The mean and median values and a

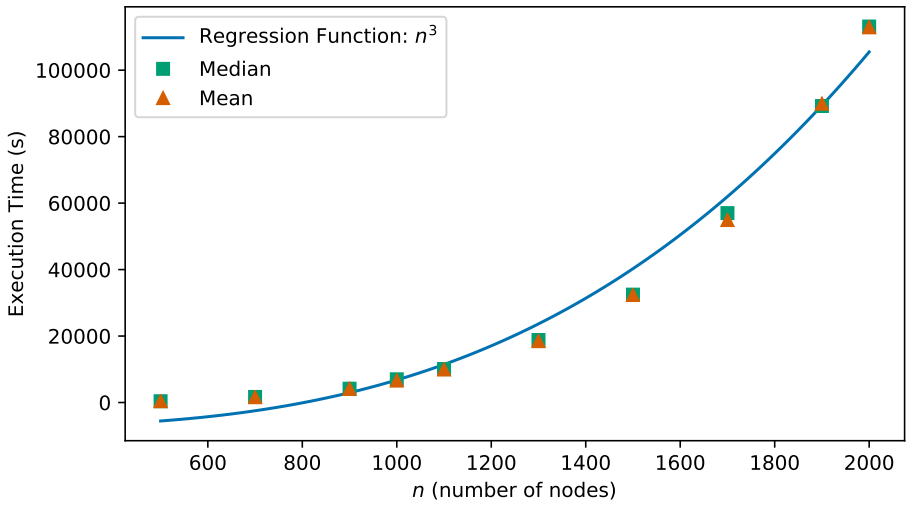


Figure 10: Time-Expanded SSP computation time depending on n, expressed in seconds.

regression function are shown. The regression function is a constant equal to around 157 which is exactly the mean value.

5.3.3 Discussion

The regression functions in Figures 10, 11, 12 fit the data obtained during the experiment, as indicated by the R^2 values. In this specific case, for the ratio regression, the R^2 value is irrelevant because a constant regression will always give a null R^2. The predicted behavior is confirmed by the experiment results. There again, the experimental results fit the worst case complexity.

In this case where every parameter but n (and m) is fixed, the worst-case complexity of Dynamic SSP is equivalent to the one of Time-Expanded SSP. In practice, Dynamic SSP is much faster. The computation time ratio between the Time-Expanded and the Dynamic SSP is constant in theory and in practice, as shown in Figure 12. This ratio is about 157, which shows that Dynamic SSP is in practice much faster on the random graphs built. This ratio was found during the first experiment described in Section 5.2 (see Figure 9 where $T = 100$ and $n = 500$ fixed). This second experiment shows it does not depend on n.

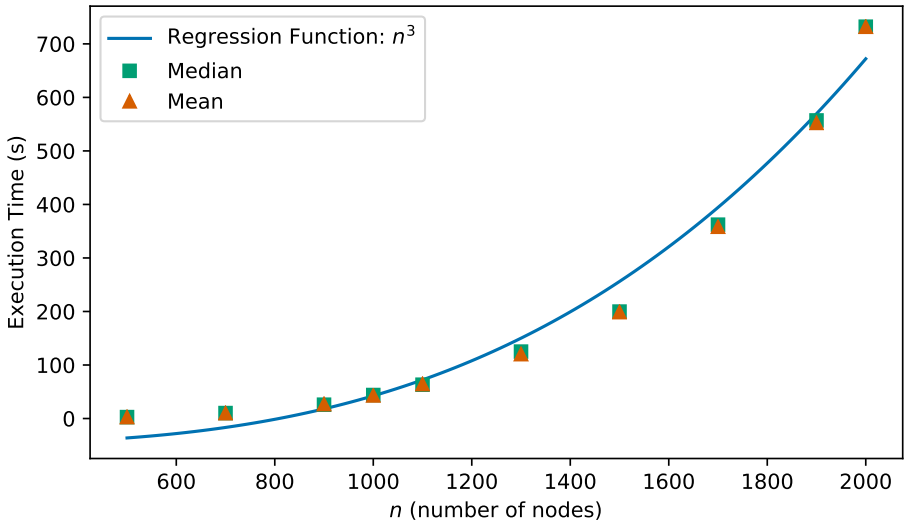


Figure 11: Dynamic SSP computation time depending on n, expressed in seconds.

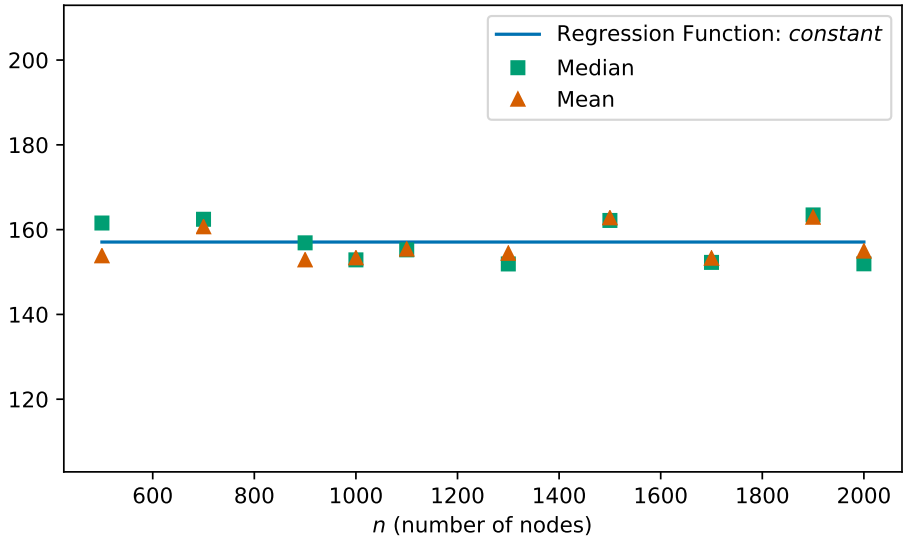


Figure 12: Computation time ratio vs n, *constant* ≈ 157.

6 Conclusion

In this paper, we considered the minimum cost flow problem on dynamic graphs. The most popular method to solve this problem is to expand the graph and then use a regular static algorithm on it. The obvious advantage of this method is that the standard algorithm can be used. On the down side, the algorithm runs on a much bigger graph. Even though it is polynomial in n, m and T, this method cannot manage large graphs in practice.

We presented Dynamic SSP, a new algorithm working on dynamic graphs without travel time or storage. Dynamic SSP adapts the SSP algorithm for this context. We improved significantly the theoretical complexity compared to the standard method.

This paper did not consider multi-source and multi-sink case. However, it can be treated by adding a *supersource* and a *supersink* implying a modification in the algorithm that consists in maintaining the current supplies and demands throughout the algorithm's execution. Such a modification is out of scope of this paper.

We ran experiments in which n, then T, varied. In both cases, Dynamic SSP computes the minimum cost flow much faster than Time-Expanded SSP, i.e. the classical SSP algorithm applied to the time-expanded graph. Thanks to the new method, it is possible to solve much larger dynamic flow problems in practice.

Acknowledgment

The project is co-financed by the European Union with the European regional development fund (ERDF) and by the Normandie Regional Council (CLASSE2 project). The French-Polish collaboration is possible thanks to the Polonium Project 37819RB.

The authors would like to thank the anonymous reviewers for their useful comments.

References

Ahuja, R. K., Magnanti, T. L., Orlin, J. B., 1993. Network flows: theory, algorithms, and applications.

Akrida, E. C., Czyzowicz, J., Gąsieniec, L., Kuszner, Ł., Spirakis, P. G., 2019. Temporal flows in temporal networks. Journal of Computer and System Sciences 103, 46 – 60.

Anderson, E. J., Nash, P., Philpott, A. B., 1982. A class of continuous network flow problems. Mathematics of Operations Research 7 (4), 501–514.

Baumann, N., Köhler, E., 2007. Approximating earliest arrival flows with flow-dependent transit times. Discrete Applied Mathematics 155 (2), 161–171.

Bertsekas, D. P., Gallager, R. G., Humblet, P., 1992. Data networks. Vol. 2. Prentice-Hall International New Jersey.

Cai, X., Sha, D., Wong, C., 2001. Time-varying minimum cost flow problems. European Journal of Operational Research 131 (2), 352 – 374, artificial Intelligence on Transportation Systems and Science.

Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N., 2012. Time-varying graphs and dynamic networks. International Journal of Parallel, Emergent and Distributed Systems 27 (5), 387–408.

Démare, T., Bertelle, C., Dutot, A., Lévêque, L., 2017. Modeling logistic systems with an agent-based model and dynamic graphs. Journal of Transport Geography 62, 51 – 65.

Dutot, A., Guinand, F., Olivier, D., Pigné, Y., 2007. Graphstream: A tool for bridging the gap between complex systems and dynamic graphs. In: Emergent Properties in Natural and Artificial Complex Systems. Satellite Conference within the 4th European Conference on Complex Systems (ECCS'2007).

Fleischer, L., Skutella, M., 2003. Minimum cost flows over time without intermediate storage. In: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, pp. 66–75.

Ford, L. R., Fulkerson, D. R., 1956. Maximal flow through a network. Canadian journal of Mathematics 8 (3), 399–404.

Ford, L. R., Fulkerson, D. R., 1958. Constructing maximal dynamic flows from static flows. Operations Research 6 (3), 419–433.

Ford, L. R., Fulkerson, D. R., 1962. Flows in networks. Princeton University Press.

Grande, E., Nicosia, G., Pacifici, A., Roselli, V., 2018. An exact algorithm for a multicommodity min-cost flow over time problem. Electronic Notes in Discrete Mathematics 64, 125–134.

Halpern, J., 1979. A generalized dynamic flows problem. Networks 9 (2), 133–167.

Hashemi, S. M., Nasrabadi, E., 2012. On solving continuous-time dynamic network flows. Journal of Global Optimization, 1–28.

Holme, P., 2015. Modern temporal network theory: a colloquium. The European Physical Journal B 88 (9), 234.

Hoppe, B., Tardos, É., 1994. Polynomial time algorithms for some evacuation problems. In: Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. Vol. 94. pp. 433–441.

Hoppe, B., Tardos, É., 2000. The quickest transshipment problem. Mathematics of Operations Research 25 (1), 36–62.

Huang, S., Fu, A. W.-C., Liu, R., 2015. Minimum spanning trees in temporal graphs. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM, pp. 419–430.

Klinz, B., Woeginger, G. J., 2004. Minimum-cost dynamic flows: The series-parallel case. Networks 43 (3), 153–162.

Koster, A., Muñoz, X., 2009. Graphs and algorithms in communication networks: studies in broadband, optical, wireless and ad hoc networks. Springer Science & Business Media.

Lin, M., Jaillet, P., 2015. On the quickest flow problem in dynamic networks: a parametric min-cost flow approach. In: Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, pp. 1343–1356.

Michail, O., 2016. An introduction to temporal graphs: An algorithmic perspective. Internet Mathematics 12 (4), 239–280.

Michail, O., Spirakis, P. G., 2016. Traveling salesman problems in temporal graphs. Theoretical Computer Science 634, 1–23.

Miller-Hooks, E., Patterson, S. S., 2004. On solving quickest time problems in time-dependent, dynamic networks. Journal of Mathematical Modelling and Algorithms 3 (1), 39–71.

Minieka, E., 1973. Maximal, lexicographic, and dynamic network flows. Operations Research 21 (2), 517–527.

Nannicini, G., Baptiste, P., Barbier, G., Krob, D., Liberti, L., 2010. Fast paths in large-scale dynamic road networks. Computational Optimization and Applications 45 (1), 143–158.

Nasrabadi, E., Hashemi, S. M., 2010. Minimum cost time-varying network flow problems. Optimization Methods & Software 25 (3), 429–447.

Orlin, J. B., 1984. Minimum convex cost dynamic network flows. Mathematics of Operations Research 9 (2), 190–207.

Parpalea, M., Ciurea, E., 2011. The quickest maximum dynamic flow of minimum cost. International Journal of Applied Mathematics and Informatics 3 (5), 266–274.

Rostami, R., Ebrahimnejad, A., 2014. An approximation algorithm for discrete minimum cost flows over time problem. International Journal of Operational Research 20 (2), 226–239.

Skutella, M., 2009. An introduction to network flows over time. In: Research Trends in Combinatorial Optimization. Springer, pp. 451–482.

Weron, R., 2014. Electricity price forecasting: A review of the state-of-the-art with a look into the future. International Journal of Forecasting 30 (4), 1030–1081.

Wilkinson, W. L., 1971. An algorithm for universal maximal dynamic flows in a network. Operations Research 19 (7), 1602–1612.

Xuan, B. B., Ferreira, A., Jarry, A., 2003. Computing shortest, fastest, and foremost journeys in dynamic networks. International Journal of Foundations of Computer Science 14 (02), 267–285.