The multiple pairs shortest path problem for sparse graphs: Exact algorithms

Roland Grappe
LIPN
Université Sorbonne Paris Nord
Paris, France
roland.grappe@lipn.univ-paris13.fr
0000-0002-7093-2175

Mathieu Lacroix
LIPN
Université Sorbonne Paris Nord
Paris, France
lacroix@lipn.univ-paris13.fr
0000-0001-8385-3890

Sébastien Martin
Huaweï Technologies & Co
Boulogne-Billancourt, France
sebastien.martin@huawei.com
0000-0002-8528-6698

***Abstract*—In this paper, we propose two exact algorithms based on the computation of the Dijkstra tree to solve the multiple pairs shortest path problem. Traditionally, to solve this kind of problems, algorithms are based on distance matrices. For sparse graphs, the computation of these matrices is too costly.**

The two approaches that we propose allow tackling this issue by computing a small number of Dijkstra trees. We test our algorithms on telecommunication network instances and random instances, and we discuss the dependence of the obtained results on the structure of sources and destinations of commodities.

We also propose an extension of the Bi-Dijkstra algorithm to consider several destinations together.

***Index Terms*—Dijkstra tree, shortest path, sparse graph**

I. Introduction

The shortest path problem is one of the most studied optimization problems and has many applications in telecommunication networks. This problem consists in finding a path of minimum length from a source to a destination. In telecommunication networks, one needs to find a shortest path for each commodity, the length or weight of a link being traditionally its IGP cost [14] which is inversely proportional to its capacity. This implies that the more capacity a link has, the smaller its weight is, and thus the link attracts more traffic. When several commodities are considered at the same time with the same weights on the network, one can hope to reduce the complexity of solving the shortest path problem for each commodity by aggregating some operations. This problem is called the Multiple Pairs Shortest Path problem (MPSP).

The extremal cases of the MPSP are two well-known shortest path problems: the single-source shortest path (SSSP) on the one hand, and the all pairs shortest path (APSP) problems on the other hand. The SSSP asks for computing a shortest path tree for a single source node and is typically well solved by a graph theoretic approach combined with a suitable queueing strategy such as Dijkstra's algorithm [1]. The APSP is the problem of computing shortest paths for all the node pairs. Algebraic shortest path algorithms such as Floyd-Warshall [3], [10], [13] are more suitable for the APSP. These methods are called algebraic because they involve matrices encoding the distances between pairs of nodes, and calculations on these matrices (typically, calculations in the $(\max, +)$ algebra). They can also be viewed as dynamic programming algorithms.

In the literature, the known approaches to solve the MPSP rely on algebraic considerations, as in [11], [12]. Such approaches involve calculations with matrices whose size corresponds to that of the underlying graph, which makes them competitive when the graph is dense. However, for sparse graphs, the storage of these matrices is too costly. In this paper, we focus on sparse graphs, that represent most of the real-life telecommunication network instances.

To overcome the storage of these matrices, we develop a different strategy to solve the MPSP using a suitable number of Dijkstra trees in order to get the desired shortest paths.

The paper is organized as follows. In Section II we formally present the problem and provide two use cases of the MPSP. Section III introduces and investigates variants of how to compute a minimum number of Dijkstra trees to get all the desired shortest paths, and experimental comparisons between these methods. Almost all of these methods are beaten by the classical Bi-Dijkstra algorithm. In Section IV, we present what we call Multi-Dijkstra, that extends Bi-Dijkstra to a more general setting, and adapts consequently our previous approaches. Based on experimental computations, we discuss the settings in which these new approaches are better. Finally, Section V provides some conclusions and perspectives.

II. Problem definition and applications

In this section, we provide a formal definition of the MPSP and two use cases explaining the interest in the approach for telecommunication network applications.

A. MPSP definition

Let $D = (V, A)$ be a weighted directed graph, where V is the set of nodes, A is the set of arcs, and w_a is the weight of arc a for all $a \in A$. Let K be a set of commodities, where each commodity $k \in K$ is defined by a couple (s_k, t_k). The node s_k is called the *source* of commodity k, and t_k is its *destination*.

The associated *multiple pairs shortest path problem (MPSP)* consists in finding a shortest path from each source s_k to each

destination t_k. Here, shortest path means minimum in terms of the sum of the weights of the arcs of the path.

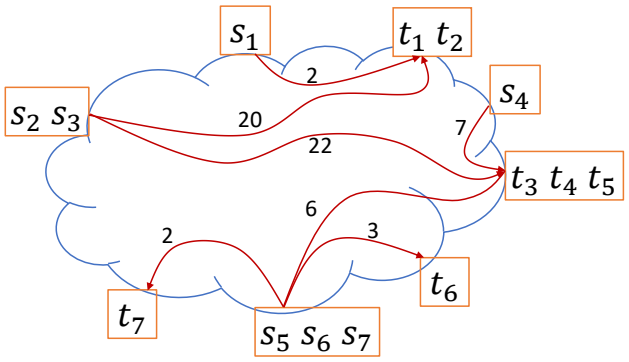

Fig. 1. An instance of the MPSP (the graph is implicit and depicted in blue) with seven commodities and a solution in red. The number on each path from s_k to t_k corresponds to the number of iterations in a Dijkstra algorithm with initial node s_k and final node t_k or with initial node t_k and final node s_k (for simplicity, we consider symmetric values), see Section III-B.

B. One-shot application

The first application is when the MPSP is used to compute simultaneously several shortest paths in an online context. In some distributed protocols, like Open Shortest Path First (OSPF) [14], each router computes the Dijkstra tree to find the next hop to route traffic for each destination. In this context, packets are routed on the network with the destination encoded in the header and the routing can be done hop by hop. If a link fails or becomes congested, then a Link State Advertisement message is sent on the network to change the routing strategy. This message can be used to change the weights of some links and thus change the routing of traffic. For this kind of network, a controller needs to monitor the routes and the link utilization when the weights of the network change, for logs analysis or to change the routing strategy.

In this context, the weights can change over time, and thus at each change, we obtain a new instance of the MPSP.

C. Inside Column Generation application

The second application considers the case of planning the routes of several commodities, to solve the multi-commodity flow problem (MCFP), which is NP-hard [2]. Let us consider a set of commodities K where each commodity $k \in K$ is defined by a triplet $\{s_k, t_k, b_k\}$, where s_k is the source, t_k the destination, and b_k the size of the commodity. Consider also for each arc $a \in A$ a unitary routing cost r_a and a capacity c_a. A solution to the MCFP is a path from the origin s_k to the destination t_k for each commodity $k \in K$ such that for each arc $a \in A$, the sum of sizes of the commodities whose paths contain arc a is no more than c_a. Its cost is the sum of the costs of the paths of the commodities, the latter being defined as the sum of the unitary routing costs over the arcs of the path times the size of the commodity. The MCFP then consists in finding a solution of minimum cost.

One way to solve this problem is to model the problem using the following integer linear programming formulation. For each commodity $k \in K$, let P_k be the set of all possible paths[1] from s_k to t_k. Let x_p^k be a binary variable equal to 1 if the path p is selected for the commodity k and 0 otherwise, for each $k \in K$ and $p \in P_k$.

$$\min \sum_{k \in K} \sum_{p \in P^k} \sum_{a \in p} b_k r_a x_p^k \tag{1}$$

$$\sum_{k \in K} \sum_{p \in P^k : a \in p} b_k x_p^k \leq c_a \qquad \forall a \in A, \tag{2}$$

$$\text{s.t.} \sum_{p \in P^k} x_p^k = 1 \qquad \forall k \in K, \tag{3}$$

$$x_p^k \in \{0,1\} \qquad \forall k \in K, p \in P^k. \tag{4}$$

The objective function (1) minimizes the total cost, constraints (2) are the capacity constraints, and constraints (3) are the convexity constraint ensuring that one path is selected for each commodity.

Remark that the number of variables is in exponential number and thus, to solve its linear relaxation, we use a column generation algorithm [4], [5] to dynamically generate the good variables. More precisely, this method consists in iteratively solving a restricted master problem corresponding to the linear relaxation of (1)-(4) in which only a small subset of paths for each commodity is considered. It is followed by a pricing phase which looks for paths with negative reduced cost. If no such path is found, the solution of the restricted master problem is an optimal solution to the linear relaxation of (1)-(4). Otherwise, the found paths are added to the restricted master problem and this process is repeated one more time.

The pricing problem can be decomposed by commodity and reduces for a commodity $k \in K$ to find whether there exists a path $p \in P^k$ satisfying $b_k \left(\sum_{a \in p}(r_a - \alpha_a)\right) < \alpha_k$, where α_a and α_k are the dual variables associated with constraints (2) and (3), respectively. This can be solved by computing for each commodity $k \in K$ a shortest $s_k t_k$-path with arc costs $r_a - \alpha_a$.

The costs of the arcs do not depend on the commodities so, at each iteration of the column generation algorithm, we need to solve a MPSP with the same commodities.

D. Instances description

For the comparison of the MPSP algorithms, we consider two families of instances. The first ones are realistic IP-RAN telecommunication network instances described in [6]. Among those, we only consider the large IP-RAN instances in terms of nodes. These networks are composed of a core network that connects multiple domains together. Domains are composed of access networks connected to aggregation networks linked to the core. The core network is a random graph, the aggregation networks are rings with some shortcuts and the nodes of the access networks (CSG nodes)

[1]Since graphs are simple, a path is identified to its set of arcs.

are connected to the aggregation network using one or two links. The large IP-RAN instances describe in [6] have 5000 nodes and 6000 links. This kind of instances has a very low density (less than 0.05%). For the commodities, we consider either commodities from [6] called "realistic" in the rest of the paper, 600 commodities, or random commodities characterized by the number of commodities $|K|$ where sources and destinations are drawn randomly. We consider 15 different values of $|K| \in \{100, 200, 300, 400, 500, 750, 1000, 1500, 2000, 2500, 5000, 7500, 10000, 15000, 20000\}$. In the figures presented in the rest of the paper for IP-RAN instances, the y-axis represents the total computational time in seconds to solve the MPSP. The x-axis represents instances grouped into the realistic ones when the original commodities are considered, and by their number of commodities for the instances with the random commodities described above.

The second set of instances are completely random instances, where arcs are generated randomly with a fixed number of nodes. We consider five different values for the number of nodes, $|V| \in \{500, 1000, 1500, 2000, 2500\}$. For each number of nodes, we consider two different densities, namely an amount of arcs of 10% and 1% of the maximum number of arcs. For each graph, we generate three sets of commodities with respectively 2500, 5000, and 7500 random commodities. This provides us 30 random instances. In the figures presented in the rest of the paper for random instances, the y-axis represents the total computational time in seconds to solve the MPSP. The x-axis represents instances grouped by the number of nodes.

III. Selection of Dijkstra trees

Dijkstra's algorithm starts with an initial node and iteratively marks every node. At each iteration, it marks among the non marked nodes the one which has the minimum distance and updates the distances. In the end, it returns a tree giving the distance from the initial node to every other node.

Since the arc weights do not depend on the commodities, applying a Dijkstra algorithm from a node v gives a shortest path for every commodity whose source is v. Hence, the MPSP can be solved by computing a Dijkstra algorithm on every node corresponding to a source. In the instance given in Figure 1, this corresponds to calling Dijkstra four times (with initial nodes s_1, s_2, s_4 and s_6).

A. Relationship with Vertex Cover in a Bipartite Graph.

Let $\overleftarrow{D} = (V, \overleftarrow{A})$ denote the *reversed graph* of D, obtained from D by changing the direction of each arc ($(u, v) \in \overleftarrow{A}$ if and only if $(v, u) \in A$), keeping the same cost.

Note that one can also compute in the reverse graph in a single application of Dijkstra with initial vertex v a shortest path for every commodity whose destination is v. Thanks to this remark, one can decrease the number of Dijkstra calls to solve the MPSP as follows (see also [12]).

We define the *bipartite demand graph* to be a bipartite graph $B = (S \cup T, F)$ where S represents the set of sources of commodities and T the set of destinations of commodities[2]. In the graph B, a source node $s \in S$ is connected to a destination node $t \in T$ (that is, $st \in F$) if there exists a commodity $k \in K$ with $s_k = s$ and $t_k = t$. The goal is to find a set of nodes $X \subseteq S \cup T$ of minimum cardinality such that each edge of F has an extremity in X. This problem is the vertex cover problem and since B is a bipartite graph, it can be solved in polynomial time as it reduces to compute a maximum matching [8].

Since each edge of F represents a shortest path to be computed, computing $|X|$ Dijkstra trees as follows will compute the desired set of shortest paths:

- For each $u \in X \cap S$, compute the Dijkstra tree in D with initial node u,
- For each $u \in X \cap T$, compute the Dijkstra tree in $\overleftarrow{D}$ with initial node u.

Determining the minimum cover in the bipartite demand graph associated with the instance of Figure 1, we deduce that only three calls of Dijkstra (with initial node s_5 in D, and initial nodes t_1 and t_3 in the reverse graph) are sufficient.

B. Relationship with Substar Cover

When minimizing the number of Dijkstra calls using a vertex cover of the bipartite demand graph B, it is implicitly assumed that the complexity does not depend on the initial node in the algorithm. If it is true when computing a spanning Dijkstra tree on a connected graph, this is no more the case when only a shortest path from the initial node to a node subset W of $V \setminus \{v\}$, hereafter called *final nodes*, is needed. In this case, one can adapt the Dijkstra algorithm to stop once each node of W has been marked. The number of iterations of the algorithm, and thus, its efficiency, is then the number of nodes that are closer to the initial node than the farthest node of W.

Hence, to speed up the resolution of the MPSP, one can look for a series of Dijkstra trees computations minimizing the overall number of iterations. Suppose that one already knows, for each commodity $k \in K$, the number of iterations $\overrightarrow{f_k}$ (or at least approximated values) when applying Dijkstra with initial vertex s_k and final node t_k, and conversely $\overleftarrow{f_k}$ in the reverse graph with initial vertex t_k and final node s_k. The objective is to determine the number of Dijkstra calls, together with the initial node and the final ones for each call, to minimize the overall number of iterations. This is nothing but considering, in the graph B introduced in the previous section, a set of substars covering each edge of F (a *substar* being a set of edges that are all incident to a given node). Each substar $H \subseteq \delta(v)$ with $v \in S$ (resp. $v \in T$) corresponds to a Dijsktra call in D (resp. $\overleftarrow{D}$) with initial node v and final nodes those covered by the edges of the substar but v, and its number of iterations is the substar cost $\sum_{e \in H} \overrightarrow{f_e}$ (resp $\sum_{e \in H} \overleftarrow{f_e}$). The objective is to minimize the sum of the costs of the substars covering F.

Applying this to the instance of Figure 1, we determine that it is enough to compute four Dijkstra trees:

[2]If a node is both the source and the destination of commodities, two copies of these nodes are considered.

- one in D with initial node s_1 and final node t_1,
- one in D with initial node s_2 and final nodes t_2 and t_3,
- one in $\overrightarrow{D}$ with initial node s_5 and final nodes t_6 and t_7,
- one in $\overleftarrow{D}$ with initial node t_3 and final nodes s_4 and s_5,

for an overall number of iterations of 34. Note that computing a Dijkstra tree for all nodes that are source of a commodity yields a solution in 37 iterations. This gives 47 iterations if one chooses the nodes of the minimum cover of the bipartite demand graph.

Obviously, it is unlikely to have the values $\overleftarrow{f_k}$ and $\overrightarrow{f_k}$ for every $k \in K$, since having them means that the MPSP is already solved. However, it is possible to have approximations of them. One could use some oracle (*e.g.*, heuristic or machine learning model based on some historical collected data). In the case of column generation (see Section II-C), it is also possible to consider for $\overleftarrow{f_k}$ and $\overrightarrow{f_k}$ the number of iterations performed by Dijkstra algorithm when computing a shortest path from s_k to t_k in the previous iteration of the column generation (*i.e.*, in the previous pricing call).

The complexity of determining the covering of F by substars minimizing the overall number of iterations is not known. We solve it to optimality using the following integer linear program. For each edge $st \in F$ with $s \in S$ and $t \in T$, we consider a binary variable x_{st} which is equal to 1 if the shortest path is computed by a Dijkstra with initial node $s \in S$, and 0 if the initial node is t.

$$
\begin{aligned}
(subSC) \quad \min \quad & \sum_{v \in S \cup T} y_v & \\
\text{s.t.} \quad & \overrightarrow{f_{st}}\, x_{st} \leq y_s & \forall (s,t) \in F, s \in S, t \in T, \\
& \overleftarrow{f_{st}}\, (1 - x_{st}) \leq y_t & \forall (s,t) \in F, s \in S, t \in T, \\
& x_{st} \in \{0,1\} & \forall (s,t) \in F, \\
& y_v \geq 0 & \forall u \in S \cup T.
\end{aligned}
$$

C. Experimental results

First, we compare three different algorithms.

- All sources: where the Dijkstra algorithm is run on all sources S and for each source, we consider all associated destinations.
- Min cover: where the Dijkstra algorithm is run on the set of vertices provided by the min vertex cover algorithm, and for each source (resp. destination), we consider all associated destinations (resp. sources).
- Min iteration: where the Dijkstra algorithm is run on the sources/destinations provided by the resolution of ($subSC$), and for each source (resp. destination), we consider destinations (resp. sources) provided by ($subSC$).

In Figure 2, we observe that on IP-RAN instances, on average, Min cover performs better than All sources. Furthermore, the Min iteration algorithm always reduces the computational time in comparison with the Min cover and All sources algorithms except for one instance. That shows the capability of the new method to reduce computational time in this kind of instances.

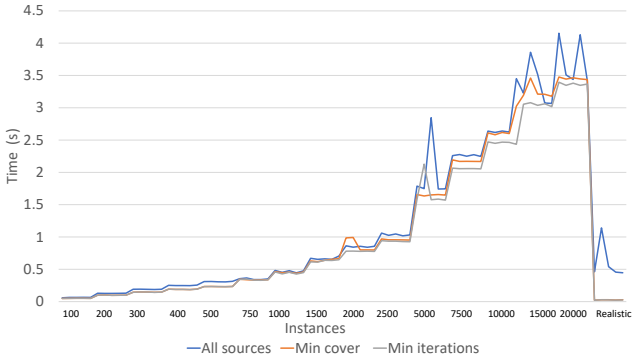

Fig. 2. Computational time comparison to solve the MPSP using Dijkstra algorithm for IP-RAN instances.

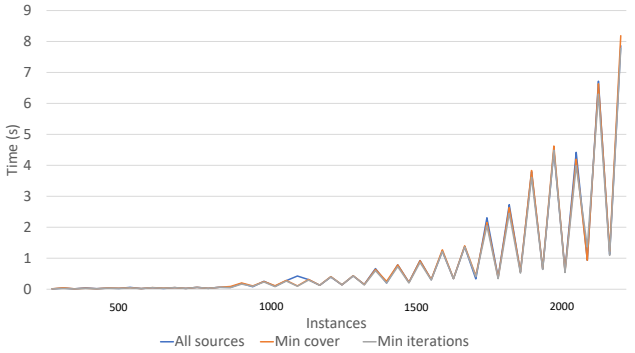

Fig. 3. Computational time comparison to solve the MPSP using Dijkstra algorithm for random instances.

In comparison, in Figure 3, we remark that the gain in computational time is really small. This implies that the structure of the graph has an impact on the quality of our solution.

The best-known algorithm to compute a single shortest path between a source and a destination in the IP-RAN instances is the Bi-Dijkstra algorithm described in [9]. The Bi-Dijkstra algorithm consists in marking alternatively the nodes from the source and from the destination, *i.e.* opening two trees simultaneously. The algorithm stops when the two trees, from the source and from the destination reach the same node. To compare our approach to this classical algorithm, we run it on each commodity. In Figure 4 the Bi-Dijkstra outperformed our approach for IP-RAN instances. We can remark, in Figure 5 that the gain of using Bi-Dijkstra algorithm in comparison to the Min-iteration algorithm is less important.

In the next section, we propose an extension of the Bi-Dijkstra and show the efficiency of this approach.

IV. Multi-Dijkstra algorithm an extension of Bi-Dijkstra algorithm

We extend the Bi-Dijkstra algorithm into what we call *Multi-Dijkstra* to consider one source and several destinations.

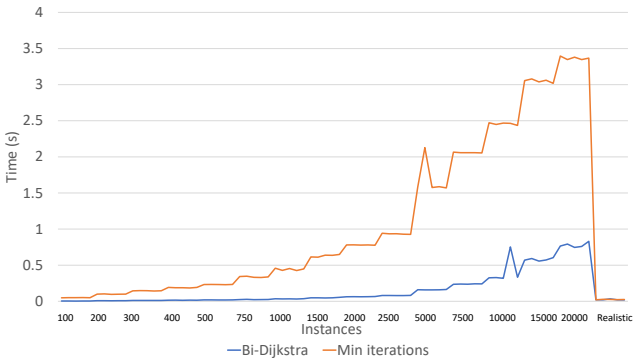

Fig. 4. Computational time comparison to solve the MPSP with Bi-Dijkstra for IP-RAN instances

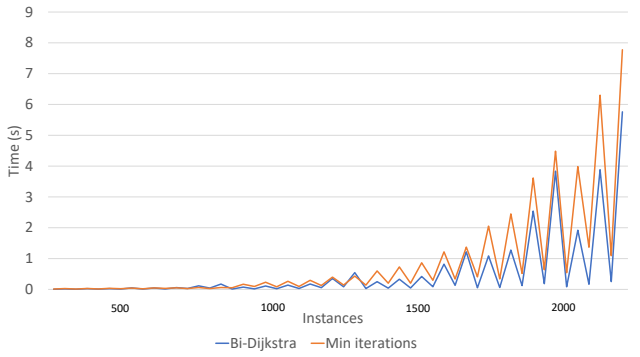

Fig. 5. Computational time comparison to solve the MPSP with Bi-Dijkstra for random instances

The idea of Multi-Dijkstra is to mark alternatively the nodes from the source and from each destination, *i.e.* opening the number of final nodes plus one trees simultaneously. The algorithm stops when the tree associated with the source reached each tree associated with the final nodes. Remark that if the tree of a final node reaches the tree of the source then we stop to mark nodes on the tree from this final node.

In the following figures, the compared algorithms are described as follows.

- Multi all sources: where the Multi-Dijkstra algorithm is run on all sources S and for each source, we consider all associated destinations.
- Multi min cover: where the Multi-Dijkstra algorithm is run on the set of vertices provided by the min vertex cover algorithm, and for each source (resp. destination), we consider all associated destinations (resp. sources).
- Multi min iterations: where the Multi-Dijkstra algorithm is run on the sources/destinations provided by the resolution of ($subSC$), and for each source (resp. destination), we consider destinations (resp. sources) provided by ($subSC$).

In Figure 6, we can remark that the Multi all sources algorithm has a computational time higher than the Bi-Dijkstra algorithm. But both of its optimized versions, Multi min cover and Multi min iterations, allow to reduce the computational time and to obtain better results on the set of commodities generated randomly, even compared to Bi-Dijkstra. On the realistic instances, the Multi min cover and the Multi min iterations approaches are not that efficient. This is probably due to the specific structure of the commodities in those instances, where all sources and destinations are all grouped in a specific area of the network.

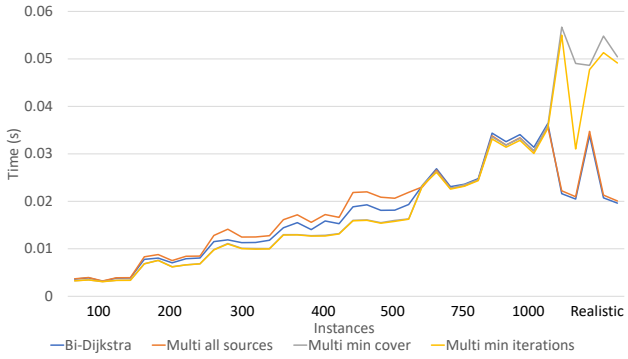

Fig. 6. Computational time comparison to solve the MPSP with Multi-Dijkstra for IP-RAN instances with a small number of commodities

In Figure 7, we remark that, when the number of commodities increases, Multi min cover and Multi min iterations are the best approaches in terms of total computational time. Furthermore, in some instances the impact of Multi min iterations in comparison to Multi min cover is interesting. This happens in instances with 10000 commodities, and this means that number of commodities has an impact on the interest to use the Multi min iterations approach.

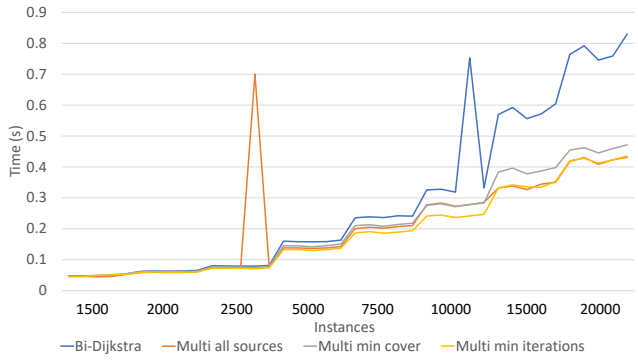

Fig. 7. Computational time comparison to solve the MPSP with Multi-Dijkstra for IP-RAN instances with a large number of commodities

Figure 8 shows the computational results for random instances. We see that all our three approaches based on Multi-Dijkstra show similar performances. In particular, they are all better than the classical Bi-Dijkstra algorithm. Beside, for random instances, on the contrary to some of the instances studied in Figure 7, the use of Multi min cover or Multi min iteration approach does not seem to have a huge impact on the computational time.

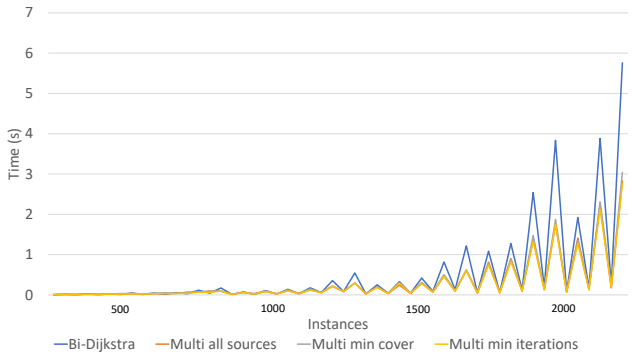

Fig. 8. Computational time comparison to solve the MPSP with Multi-Dijkstra for random instances

V. Conclusion and perspectives

In this work, we present different algorithms based on Dijkstra computations for solving the Multi Pairs Shortest Paths problem in sparse directed graphs, and provide an experimental comparison of these algorithms. From these experiments, we draw conclusions.

First, if the exploration in Dijkstra is only made from the initial node, computing Dijkstra with initial node and final nodes given by a substar cover gives the best results. Nevertheless, two ingredients of this approach still remain to be investigated. Firstly, the question of how to obtain beforehand good approximations on the number of iterations of Dijkstra. Secondly, this approach requires to solve the minimum substar cover as a preprocessing, and the complexity of this problem is not known. Note that in all experiments, we have not included in the running time the one relative to the preprocessing step. It is not an issue for the minimum vertex cover preprocessing since it is a polynomial problem[3] that can be efficiently solved just once, even if the MPSP must be solved several times on the same graph and the set of commodities but with different arc weights like in column generation. However, the substar cover preprocessing must be performed when the values $\overleftarrow{f}_k$ and $\overrightarrow{f}_k$ change and the resolution of this step using a MIP makes it clearly non competitive, but these experiments show that it might be a promising approach. So the next research question is the design of efficient heuristics for this preprocessing step.

Then, we propose a natural extension of the Bi-Dijkstra algorithm to consider one source and several destinations. Our experiments suggest that, depending on the structure of the instances, the minimum substar cover approach might be interesting. The weights that we used in Min cover and Min iterations are well designed for the Dijkstra algorithm, and a natural question is how to obtain a better approximation of these weights when the Multi-Dijkstra algorithm is used. Future investigations remain to be done to find good approximations, according to the multi-paths algorithm for these weights, and good heuristics to overcome these difficulties.

In this line of work, another interesting feature that we aim at would be to incorporate our approaches within a column generation algorithm. In this setting, it might be that the previous iteration contains enough information to get weights that are a good approximation for the current step.

[3]Note that this preprocessing step in no polynomial if the graph D is undirected because in this case, a single Dijkstra call with initial nodes v can be used to compute a shortest path for every commodity whose source or destination is v. Hence, vertices that are both source and destination of some commodities are not duplicated and the minimum vertex cover has to be computed on a demand graph which may not be bipartite, destroying the polynomiality of the problem [7].

References

[1] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec 1959.
[2] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.
[3] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, jun 1962.
[4] P. C. Gilmore and R. E. Gomory. A Linear Programming Approach to the Cutting-Stock Problem. *Operations Research*, 9(6):849–859, December 1961.
[5] P. C. Gilmore and R. E. Gomory. A Linear Programming Approach to the Cutting Stock Problem—Part II. *Operations Research*, 11(6):863–888, December 1963.
[6] Nicolas Huin, Jérémie Leguay, Sébastien Martin, and Paolo Medagliani. Routing and slot allocation in 5g hard slicing. *Computer Communications*, 2023.
[7] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.
[8] D. Köning. Graphs and matrices (in hungarian). *Mat. Fiz. Lapok*, 38:116–119, 1931.
[9] Michael Luby and Prabhakar Ragde. A bidirectional shortest-path algorithm with good average-case behavior. *Algorithmica*, 4:551–567, 1989.
[10] B. Roy. Transitivité et connexité. *C. R. Acad. Sci. Paris*, 249:216–218, 1959.
[11] I-Lin Wang. An algebraic decomposed algorithm for all pairs shortest paths. *Pacific Journal of Optimization*, 10:561–576, 2014.
[12] I-Lin Wang, Ellis L. Johnson, and Joel Sokol. A Multiple Pairs Shortest Path Algorithm. *Transp. Sci.*, 39:465–476, 2005.
[13] Stephen Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, jan 1962.
[14] W.V. Wollman and Y. Barsoum. Overview of open shortest path first, version 2 (ospf v2) routing in the tactical environment. 3:925–930 vol.3, 1995.