

Efficient Labeling Algorithms for Adjacent Quadratic Shortest Paths

João Marcos Dusi Vilella

Pontifical Catholic University of Rio de Janeiro

Department of Industrial Engineering

Rua Marquês de São de Vicente, 225 – Gávea, Rio de Janeiro, RJ – Brazil – 22541-900

jvilela@aluno.puc-rio.br

Advisors: Bruno Fanzeres and Rafael Martinelli

Pontifical Catholic University of Rio de Janeiro

Department of Industrial Engineering

Rua Marquês de São de Vicente, 225 – Gávea, Rio de Janeiro, RJ – Brazil – 22541-900

{bruno.santos, martinelli}@puc-rio.br

ABSTRACT

The main focus of this dissertation is to study the Adjacent Quadratic Shortest Path Problem (AQSP), which consists in finding the shortest path on a directed graph when its total weight component also includes the impact of consecutive arcs. We provide a formal description of the AQSP and propose an extension of Dijkstra's algorithm for solving AQSPs in polynomial-time, providing a proof of its correctness under mild assumptions. We introduce an adjacent quadratic A^* algorithm (that we denote aqA^*) with a backward search for cost-to-go estimation to speed up the search. We assess the performance of both algorithms by comparing their relative performance with benchmark algorithms from the scientific literature and carry out a thorough collection of sensitivity analyses of the methods on a set of problem characteristics using randomly generated graphs. Numerical results suggest that aqA^* outperforms all other algorithms, with performance significantly superior than the considered alternatives.

KEYWORDS. Adjacent Quadratic Shortest Path Problem. α -Cycle. Backward Cost-to-Go Estimation. Binary Quadratic Problem. Dijkstra Algorithm. Network Flow Problems.

TAG – Teoria e Algoritmos em Grafos, OC – Otimização Combinatória, PM – Programação Matemática

RESUMO

O foco principal desta dissertação é estudar o *Adjacent Quadratic Shortest Path Problem* (AQSP), que consiste em encontrar o caminho mais curto em um grafo direcionado quando sua componente de peso total também inclui o impacto de arcos consecutivos. Fornecemos uma descrição formal do AQSP e propomos uma extensão do algoritmo de Dijkstra para resolver AQSPs em tempo polinomial, fornecendo uma prova de sua correção. Introduzimos uma melhoria no algoritmo, que denotamos aqA^* , que se baseia em uma busca regressiva para estimar o custo total. Avaliamos o desempenho de ambos os algoritmos comparando seu desempenho relativo com algoritmos de referência da literatura científica e realizamos uma coleção completa de análise de sensibilidade dos métodos em um conjunto de instâncias do problema usando grafos gerados aleatoriamente. Os resultados numéricos sugerem que aqA^* supera todos os outros algoritmos, com desempenho significativamente superior às alternativas consideradas.

PALAVRAS CHAVE. Adjacent Quadratic Shortest Path Problem. Algoritmo de Dijkstra. α -Cycle. Backward Cost-to-Go Estimation. Problemas de Fluxo em Rede. Problema Quadrático Binário.

TAG – Teoria e Algoritmos em Grafos, OC – Otimização Combinatória, PM – Programação Matemática

1. Introduction

Finding the least-weighted path between two vertices in a graph is a well-studied problem in computer science and has been deeply explored in the technical literature both from a methodological and applied perspectives. This problem is usually referred to as the *Shortest Path Problem* (SPP) and aims at finding a path P on a weighted graph $G = (V, A, c)$ that connects a source node $s \in V$ with a target node $t \in V$ of minimal total weight. Structurally, the SPP can be formulated as an integer programming problem with a linear objective function $c^\top x$, where $c \in \mathbb{R}^{|A|}$ represents an edges-wise weight vector and $x \in \{0, 1\}^{|A|}$ indicates a binary vector of arc selections, constrained to a network-flow-based feasible region, that builds up the shortest path [Newman, 2010]. Many procedures have been developed over the past decades to solve the SPP efficiently (e.g., *Bellman-Ford*, *Floyd-Warshall*, and A^* algorithms), with the most well-known approach due to Dijkstra [1959]. Although the linear nature of SPPs can accommodate a large collection of problem structures, some applications demand more complex interactions and cost-impacts between edges within the path, as, for instance, the problems of establishing network protocols [Murakami e Kim, 1997], optimizing vehicle routing [Martinelli e Contardo, 2015], and identifying variance-constrained shortest paths [Sen et al., 2001], all intrinsically following a quadratic order. In this context, the *Quadratic Shortest Path Problem* (QSPP) emerges as a powerful modeling tool to properly characterize a broader class of problems since it extends the standard SPP structure by identifying a path that jointly minimizes both the total path weight and the sum of interaction costs over distinct pairs of edges on the path. Formally, the QSPP can be stated over a weighted digraph $G = (V, A, c, q)$ as a quadratic integer programming problem [Gill e Wong, 2015] with objective function defined as $x^\top qx + c^\top x$, where $q \in \mathbb{R}^{|A|} \times \mathbb{R}^{|A|}$ is a square matrix that maps the weight impact of pair of arcs and (c, x) follows similarly to the SPP formulation. In this work, we focus on a particular structure of QSPPs, referred to in technical literature as the *Adjacent Quadratic Shortest Path Problem* (AQSP), which follows the same quadratic structure, but in which the matrix q only accounts for interactions among adjacent arcs.

The theory and applicability of QSPP/AQSP have been studied through the seminal work of Amaldi et al. [2011] and further extended by Rostami et al. [2015, 2018], Hu e Sotirov [2018], Buchheim e Traversi [2018]. Amaldi et al. [2011] investigated the issues over complexity and approximability in problems that seek for optimum paths, tours, and flows under the so-called reload costs, defined as the cost incurred by the path when two adjacent arcs are of “different types”. Moreover, Buchheim e Traversi [2018] explores the more general framework of *Binary Quadratic Problems* (BQPs) and propose an exact algorithm for this class of optimization problems based on computing quadratic global underestimators and embedding the resulting lower bounds into branch-and-bound techniques. Rostami et al. [2015] provides an in-depth mathematical description of QSPPs and proposes an algorithm to solve AQSPs in polynomial-time based on a procedure that reduces the AQSP to an instance of the SPP by modifying the original graph. Furthermore, Hu e Sotirov [2018] also characterize QSPPs mathematically and derive an algorithm that examines whether QSPP instances of directed grid graphs are “linearizable”. More precisely, the authors evaluate when a grid graph with quadratic cost can be transformed into a graph with linear costs. They also reviewed the developments presented by Rostami et al. [2015] and prove that, despite stated, their algorithm does not find optimal solutions in polynomial time for general digraphs, but rather only for the particular case of acyclic digraphs. Therefore, following this latter observation and technical issue, in this work, we study computationally efficient solution methods to solve AQSPs on a broader class of graphs, the ones that do not possess improving α -cycles, defined as follows.

Definition 1.1. An α -cycle is a walk $(u_1, \dots, u_n)$ in the underlying graph with $n \geq 5$ such that $u_2 = u_{n-1}$. An α -cycle is said to be *improving* if its total compounded weight is lower than that of the shorter sequence (u_1, u_2, u_n) .

Note that an improving α -cycle may exist even without negative weights due to their quadratic

nature. We argue that graphs with no improving α -cycles are more general than acyclic ones since, by definition, the latter do not possess α -cycles, but the opposite does not hold. These types of graphs are found in several practical applications, as, for instance, on graphs that represent spatial attributes (terrain, distance, slope, etc.), widely used on route-design problems (see, for instance, [Piveteau, 2017] and the references therein). This work aims to contribute to the scientific literature on AQSPs by designing an extension of Dijkstra's algorithm (from now on referred to as aqD) to solve AQSP on graphs with no improving α -cycles. More specifically, we demonstrate that our algorithm finds optimal walks (paths possibly with cycles) in the absence of negative weights and acyclic paths when the graph is free of improving α -cycles. An acceleration to the algorithm is also proposed: an adjacent quadratic A^* with a backward search for cost-to-go estimation (hereinafter referred to as aqA^*). We perform a thorough computational campaign to analyze the proposed algorithms' performances. Due to page-limit constraints, we focus on presenting the analysis for high-demanding instances, where we measure the relative speed of aqD and aqA^* against the linearization-based algorithm designed in Rostami et al. [2015, 2018] on randomly generated instances using the Erdős e Rényi [1959] and Configuration Model [Newman, 2010] methods.

2. Problem Definition

The AQSP is defined over a weighted digraph $G = (V, A, c, q)$, where (V, A) denote the sets of nodes and arcs of the network, respectively, $c \in \mathbb{R}_+^{|A|}$ stands for the vector of linear costs, and $q \in \mathbb{R}_+^{|A| \times |A|}$ represents the respective matrix of quadratic costs, where we assume that $q(a_1, a_2) = 0$ if $h(a_1) \neq t(a_2)$, where the functions $t(\cdot)$ and $h(\cdot)$ stand for the *head* and the *tail* of an arc, respectively. This assumption allows us to, eventually, overload the notation and denote $q(u, v, w)$ instead of $q((u, v), (v, w))$ for every pair of consecutive arcs in $A \times A$. In addition to G , we are also given a source node $s \in V$ and a target node $t \in V \setminus \{s\}$. For every node $u \in V$ we denote $\delta^+(u) = \{a \in A : t(a) = u\}$ and $\delta^-(u) = \{a \in A : h(a) = u\}$. The AQSP can be formulated as a binary quadratic program (BQP), as presented in (1)–(3). We are given, for every arc $a \in A$, a binary variable x_a that takes the value 1 if a is part of the shortest path. The following BQP is valid for the AQSP:

$$\min_{\mathbf{x}} \sum_{a \in A} c(a)x_a + \sum_{(a,b) \in A \times A} x_a q(a, b)x_b \quad (1)$$

subject to:

$$\sum_{a \in \delta^+(u)} x_a - \sum_{a \in \delta^-(u)} x_a = \begin{cases} 1 & u = s \\ -1 & u = t \\ 0 & u \in V \setminus \{s, t\} \end{cases} \quad \forall u \in V \quad (2)$$

$$x_a \in \{0, 1\} \quad \forall a \in A. \quad (3)$$

It should be highlighted that using BQP techniques to solve the AQSP seems a reasonable approach and has already been explored in the scientific literature [Buchheim e Traversi, 2018]. However, we argue that as the size of the graph grows, using off-the-shelf solvers without exploiting the underlying shortest path structure of the problem may quickly become prohibitive. Memory issues that raise from building up a very heavy optimization problem can easily occur and prevent the solver from providing meaningful results. Additionally, the performance of BQP solvers in large instances can quickly deteriorate. These limitations motivate us to study this problem from an algorithmic perspective, to ultimately introduce two algorithms to solve AQSPs, described next.

3. Adjacent Quadratic Dijkstra

In this section, we describe a dynamic programming algorithm for the AQSP, referred to in this work as the *Adjacent Quadratic Dijkstra* (aqD). The algorithm is based on the concept of node

labels and can adequately address adjacent costs. The adjacent quadratic Dijkstra algorithm proposed in this work follows a similar structure to the standard Dijkstra's algorithm. More specifically, for a given source node $s \in V$ and an arc $a = (i, j) \in A$, we define a s - a -walk as a tuple $(v_0, v_1, \dots, v_k)$ such that $v_0 = s, v_{k-1} = i, v_k = j$, and let us denote by D_a the cost of the best s - a -walk. This notation allows computing adjacent costs when extending a walk that ends at arc a . We present a pseudo-code of the solution method in Algorithm 1. The iteration over the arcs is controlled by a priority queue Q , where the element with highest priority is always positioned on top of the stack, forcing it to be the first to be removed. In this case, the labels with smallest accumulated costs should be prioritized since they indicate shorter paths and should be positioned on top of the stack. At each iteration, the arc $a \in A$ with the smallest cumulative cost D_a is removed from Q and used as a reference to explore the neighboring nodes. The algorithm stops when Q becomes empty. The following results demonstrate the correctness of our method to compute optimal walks in the absence of negative costs and optimal acyclic paths if the graph is free of improving α -cycles.

Algorithm 1 Adjacent Quadratic Dijkstra

```

1: function  $aqDijkstra(V, A, c, q, s)$ :
2:
3: Let  $D_a \leftarrow +\infty$  for every  $a \in A$ 
4: Let  $D_{(s,s)} \leftarrow 0, Q \leftarrow \{(s, s)\}$ 
5: while  $Q \neq \emptyset$  do
6:   Let  $a = (i, j) \in \arg \min\{D_a : a \in Q\}$ 
7:   Set  $Q \leftarrow Q \setminus \{a\}$ 
8:   for each  $k$  in neighbors of  $j$  do
9:     if  $D_a + c_{jk} + q_{ijk} < D_{(j,k)}$  then
10:       Set  $D_{(j,k)} \leftarrow D_a + c_{jk} + q_{ijk}$ 
11:       Set  $Q \leftarrow Q \cup \{(j, k)\}$ 
12: return  $D$ 

```

Proposition 1. If $c, q \geq 0$, the algorithm ends in a finite number of iterations and computes, for every arc $a = (i, j) \in A$, the cost of a minimum s - a -walk.

Proof. The number of iterations of the algorithm is governed by the times an arc $a \in \arg \min\{D_a : a \in Q\}$ is selected at Line 6. We denote a_t the arc selected at this line at iteration t of the algorithm. Let us define $R_0 = \emptyset$ and for every $t \geq 1$, $R_t \leftarrow R_{t-1} \cup \{a_t\}$. With these definitions, we have that $a_1 = (s, s), R_1 = \{(s, s)\}$. We will first demonstrate that, for every $t \geq 1$, $a_t \notin R_{t-1}$. The first time an arc $a_t = a^*$ is added to R_t , it satisfies $D_{a^*} \leq D_a$ for every $a \in Q$. Since the components of c, q are non-negative by assumption, it follows that any extension of any walk can only increase its cost. By the time another walk is extended using the arc a^* , its cost can only be $\geq D_{a^*}$. This means the cost of D_{a^*} will never be updated, and therefore a^* will never enter the priority queue (Q) again. We will now prove that, at iteration t , D_{a_t} is the cost of the shortest s - a_t -walk. The result is obviously true for $t = 1$. Let us assume by induction that the result holds for t , and let us assume by contradiction that there exists a walk $w = (w_1 = s, \dots, w_{l-1} = i_{t+1}, w_l = j_{t+1})$ such that $c(w) < D_{a_{t+1}}$, where $a_{t+1} = (i_{t+1}, j_{t+1})$. Let k be the first index in this walk such that $(w_k, w_{k+1}) \notin R_t$ (this is well defined since a_{t+1} satisfies this property). Let $pw_k = (w_1 = s, \dots, w_k)$ be the k -th prefix of walk w . Since the components of c, q are non-negative by assumption, it follows that $c(pw_k) + c_{w_k w_{k+1}} + q_{w_{k-1} w_k w_{k+1}} \leq c(w)$. Also, because $(w_{k-1}, w_k) \in R_t$, the induction hypothesis implies that $D_{w_{k-1} w_k} \leq c(pw_k)$. Now, because w_{k+1} is adjacent to (w_{k-1}, w_k) in w , it follows from the labeling algorithm that $D_{w_k w_{k+1}} \leq D_{w_{k-1} w_k} + c_{w_k w_{k+1}} + q_{w_{k-1} w_k w_{k+1}}$. Finally, because a_{t+1} was picked at iteration $t + 1$ of the algorithm and because

$(w_k, w_{k+1}) \notin R_t$, it follows that $D_{a_{t+1}} \leq D_{w_k w_{k+1}}$. We put all these inequalities together to obtain:

$$D_{a_{t+1}} \leq D_{w_k w_{k+1}} \leq c(pw_k) + c_{w_k w_{k+1}} + q_{w_{k-1} w_k w_{k+1}} \leq c(w) < D_{a_{t+1}},$$

which is a contradiction, proving that $D_{a_{t+1}}$ is the minimum cost among all s - a_{t+1} -walks. $\square$

Proposition 2. In the absence of improving α -cycles, $\min\{D_a : a \in \delta^-(t)\}$ as computed by Algorithm 1 corresponds to the minimum cost of a s - t -path in G .

Proof. Let us assume by contradiction that, for a given arc $a = (u, t)$, D_a corresponds to the cost of a s - a -walk with a cycle and that no s - a -path exists with the same cost. This walk can be written as $(w_i)_{i=1}^n$ with $w_1 = s, w_{n-1} = u, w_n = t$ and $(w_i, w_{i+1}) \in A$ for every $i = 1 \dots n-1$. If $w_l = w_k$ for $1 < l < k < n$, one can replace the α -cycle $(w_i)_{i=l-1}^{k+1}$ by (w_{l-1}, w_l, w_{k+1}) to obtain a walk with one less cycle, at the same cost or less (because the α -cycle is non-improving). One can repeat this argument as many times as necessary to ultimately obtain a s - a -path of the same cost or less, which contradicts our assumption. It follows that D_a corresponds to the cost of a s - a -path. $\square$

These two results characterize a nice feature of our method. One does not need to incur into an expensive procedure to check for the existence of improving α -cycles. Instead, one can check for the non-negativity of the graph weights and execute the aqD regardless of the existence of α -cycles. One can *a posteriori* check for the existence of improving α -cycles once the algorithm has computed an optimal s - t -walk.

3.1. Complexity of aqD

The number of neighboring arcs that are inspected in Line 9 is bounded by the number of quadratic arcs of the form $((i, j), (j, k)) \in A \times A$, that we denote by $|A_Q|$. Removing the arc of highest priority from Q and rearranging its elements to maintain its structure can be done in $\mathcal{O}(\log |A|)$. The total number of iterations (governed by the main loop in Line 5 of the pseudo-code) is $\mathcal{O}(|A|)$. This leads to a total complexity of $\mathcal{O}(|A| \log |A| + |A_Q|)$. For dense graphs (those for which $\mathcal{O}(|A_Q|) = \mathcal{O}(V^3)$ and $\mathcal{O}(|A|) = \mathcal{O}(V^2)$), the time complexity is of $\mathcal{O}(V^3)$. On sparse graphs (those for which $\mathcal{O}(|A|) = \mathcal{O}(|A_Q|) = \mathcal{O}(|V|)$), the time complexity is reduced to $\mathcal{O}(|V| \log |V|)$. Note that the standard Dijkstra method runs in $\mathcal{O}((|A| + |V|) \log |V|)$ time using a heap, which reduces to $\mathcal{O}(|V|^2 \log |V|)$ for dense graphs, and to $\mathcal{O}(|V| \log |V|)$ for sparse ones. Hence our algorithm shares the same worst-case complexity as the standard Dijkstra method on sparse graphs, but is slower on dense graphs.

4. A* with backward search for cost-to-go estimation

In this section, we introduce a bidirectional labeling algorithm to increase the search speed. This scheme consists in solving the problem in two steps: (i) a backward search using a (linear) SPP, starting from the target node $t \in V$, to find cost-to-go estimates; (ii) a forward search for the original AQSP, starting from the source node $s \in V$, but considering the cost-to-go estimates as lower bounds to guide the search. This approach is inspired by the work of Thomas et al. [2019], where the authors extend the seminal work of Hart et al. [1968] to develop a bidirectional search for resource-constrained shortest path problems. The idea is built upon Hart et al.'s demonstration that A* only terminates with an optimal path when the estimates of the cost-to-go are actual lower bounds. In the context of this work, for the case of a quadratic shortest path search, linear estimators of the costs-to-go are actual lower bounds for the quadratic problem provided that the quadratic terms are reduced — or as in this case, ignored.

The fundamental idea is that these linear estimations can be calculated through a backward search from the target node $t \in V$ to all other nodes by solving the standard (linear) SPP. Structurally, starting from the sink node $t \in V$, one finds, for every node $u \in V$, the shortest distance from u to t using the linear costs c . We denote $(B_u)_{u \in V}$ the costs computed using this procedure. Note that the algorithm not only is fast even for dense graphs (it relies on solving the standard SPP using Dijkstra's method) but also occupies only linear space to store the information. Furthermore, this method also provides an upper

bound for the AQSP by evaluating the s - t -path solution provided by considering the quadratic costs. In the second step, we perform a forward search using the proposed aqD method but taking into account the cost-to-go estimators to guide the search. More specifically, for every arc $a = (i, j) \in A$, $D_a + B_j$ represents a lower bound on the cost that would be achieved by extending the current s - a -walk to $t \in V$. These values are then used to select the next arc to process in the aqD method. By estimating the cost of arriving at $t \in V$, we can significantly reduce the size of the search tree and speed up the procedure. The proposed method is presented on Algorithm 2.

Algorithm 2 Adjacent Quadratic A^* with backward cost-to-go estimation

```

1: function  $aqA^*(G, c, q, s, t)$ :
2:
3: Let  $(B_u)_{u \in V} \leftarrow Dijkstra(G, c, t)$ 
4: Let  $D_a \leftarrow +\infty$  for every  $a \in A$ 
5: Let  $D_{(s,s)} \leftarrow 0$ ,  $Q \leftarrow \{(s, s)\}$ 
6: while  $Q \neq \emptyset$  do
7:   Let  $a = (i, j) \in \arg \min \{D_a + B_j : a = (i, j) \in Q\}$ 
8:   Set  $Q \leftarrow Q \setminus \{a\}$ 
9:   for each  $k$  in neighbors of  $j$  do
10:    if  $D_a + c_{jk} + q_{ijk} < D_{(j,k)}$  then
11:      Set  $D_{(j,k)} \leftarrow D_a + c_{jk} + q_{ijk}$ 
12:      Set  $Q \leftarrow Q \cup \{(j, k)\}$ 
13: return  $D$ 

```

5. Computational Experiments

In this section, we focus on evaluating the performance of the proposed algorithms, presenting results from a set of computational experiments. Due to page-limit constraints, we focus on presenting the analysis for high-demanding instances. The computational analyses evaluate the total search time required by the proposed algorithms against the method of Rostami et al. [2015]; Hu e Sotirov [2018] that solves AQSPs for directed acyclic graphs, referred, hereinafter, as Lin . The results from solving the BQP using off-the-shelf solvers are not presented due to the solver's inability to handle even the smallest instance tested in our experiments. Finally, computational calculations were made on an Intel® Core i7-10700K 4.8GHz with 64 GB of RAM machine, using one single thread. We study the algorithms' performance considering randomly generated graph instances built by the methods described in [Erdős e Rényi, 1959] and [Newman, 2010] (termed *Configuration Model*).

The Erdős-Rényi method defines a graph where the arcs are added between pairs of nodes according to a given probability $p \in [0, 1]$ – the greater p is, greater is the number of arcs, thus the denser the graph. For expository purposes, in this model, we define p as 0.8 to obtain highly connected instances and range the number of nodes $|V| \in \{100, 150, 200, 250, 300, 350, 400\}$. For each configuration $(|V|, p)$, 100 random instances are generated. Costs for arcs and quadratic arcs are also randomly generated using a standard Uniform distribution, i.e., $(q(i, j, k), c((i, j))) \sim Unif(0, 1)$. Figure 1 presents compiled results from this experiment, showcasing the minimum, average, and maximum search times for each algorithm and size tested. Results for aqA^* and aqD show reduced and similar search-times throughout the instances. This behavior can be explained by the high connectivity of the Erdős-Rényi instances considered in this experiment. Since p was set to 0.8, there is a high probability that the source node is close-connected to the target node, thus the shortest path contains only a few nodes. Consequently, the number of nodes needed to be visited becomes small, shortening the search process. Furthermore, we highlight the difference in the search-time magnitude for the benchmark Lin approach, in comparison to aqA^* and aqD . This difference is primarily due to the time spent building the linearized graph. Since the number of quadratic arcs is large, performing the linearization is very time-consuming and becomes a drawback to this approach.

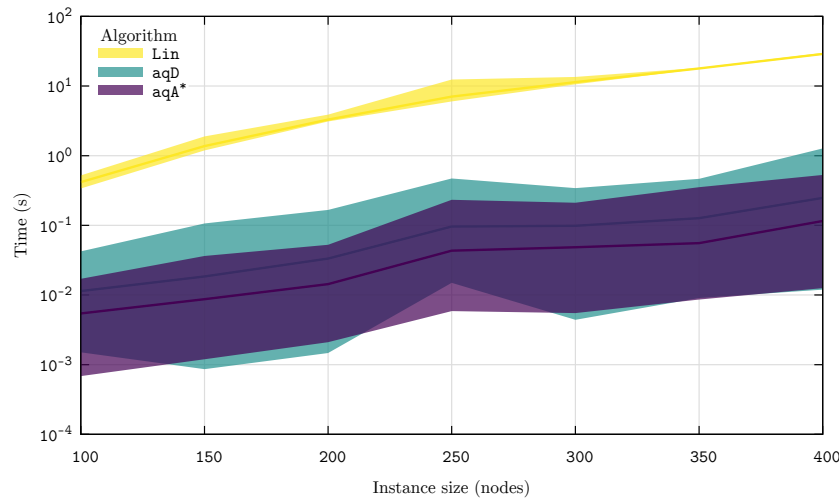


Figure 1: Results for Erdős-Rényi instances

The Configuration Model generates a random graph G from a given degree sequence $\kappa = \{\kappa_i\}_{i=1}^{|V|}$. A degree defines the number of connections a given node has, therefore, as κ increases, so does the number of arcs in G . To build the graph, the method randomly samples κ_i nodes from G and connects them to node $i \in V$. For this model, we fix the nodes' degrees to 8 and vary the number of nodes $|V| \in \{150 \times 10^3, 200 \times 10^3, 250 \times 10^3, 300 \times 10^3\}$. Similarly, 100 random instances are generated with $c((i, j))$ and $q(i, j, k)$ drawn from an $Unif(0, 1)$. Results from the Configuration Model, presented in Figure 2, suggest that the speed ups remain nearly constant, with our method aqA^* performing around 32 times faster on average than Lin . Additionally, aqD is about 5 times faster than Lin . Furthermore, the presented results indicate that the aqA^* exhibits less variance than aqD , resulting in a more consistent and predictable search time for instances generated with the Configuration Model. These results reaffirm the role of the backward step on providing reasonable cost-to-go estimates and consistently speeding up the search process.

6. Conclusion

This study provides a theoretical description of Adjacent Quadratic Shortest Problems and proposes an extension of Dijkstra's algorithm (aqD) to solve AQSPs on graphs with no improving α -cycles. An enhancement procedure for the primary approach is also presented, i.e., an adjacent quadratic version of A^* with backward cost-to-go estimation. From the theoretical perspective, we present evidence that aqD can solve AQSPs in polynomial time to proven optimality. We also show through computational experiments that aqD outperforms the algorithm proposed by Rostami et al. [2015]. This behavior is also valid for the proposed algorithm aqA^* , which was the fastest approach among the ones studied, outperforming aqD search speed by about the same amount on the benchmark analysis.

Bibliographical References

- Amaldi, E., Galbiati, G., e Maffioli, F. (2011). On Minimum Reload Cost Paths, Tours, and Flows. *Networks*, 57(3):254–260.
- Buchheim, C. e Traversi, E. (2018). Quadratic Combinatorial Optimization Using Separable Underestimators. *INFORMS Journal on Computing*, 30(3):424–437.
- Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271.

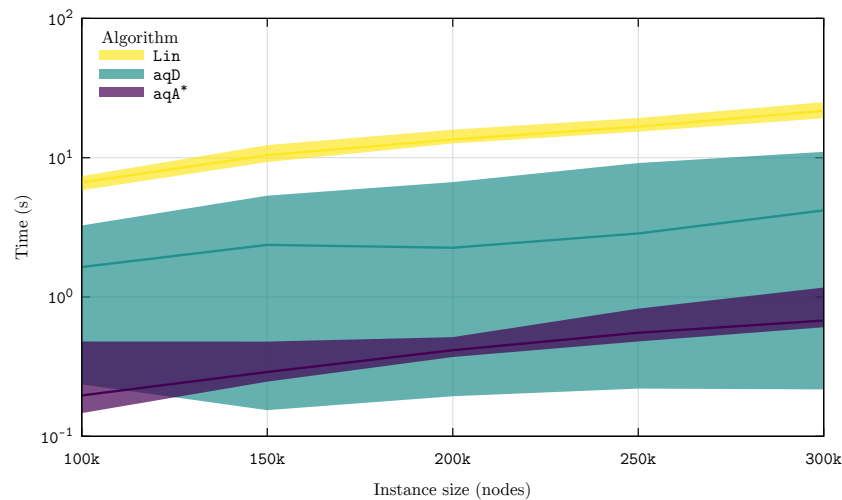


Figure 2: Results for Configuration Model instances

- Erdős, P. e Rényi, A. (1959). On Random Graphs. *Publicationes Mathematicae*, 6:290–297.
- Gill, P. E. e Wong, E. (2015). Methods for Convex and General Quadratic Programming. *Mathematical Programming Computation*, 7(1):71–112.
- Hart, P. E., Nilsson, N. J., e Raphael, B. (1968). Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.
- Hu, H. e Sotirov, R. (2018). A Polynomial Time Algorithm for the Linearization Problem of the QSPP and its Applications. *Optimization Online*, p. 1–19.
- Martinelli, R. e Contardo, C. (2015). Exact and Heuristic Algorithms for Capacitated Vehicle Routing Problems with Quadratic Costs Structure. *INFORMS Journal on Computing*, 27(4):658–676.
- Murakami, K. e Kim, H. S. (1997). Comparative Study on Restoration Schemes of Survivable ATM Networks. In *Proceedings of IEEE INFOCOM '97*, p. 345–352.
- Newman, M. (2010). *Networks: An Introduction*. Oxford University Press, USA, 1st edition.
- Piveteau, N. (2017). A Novel Approach to the Routing Problem of Overhead Transmission Lines. Master's thesis, University of Zurich.
- Rostami, B., Chassein, A., Hopf, M., Frey, D., Buchheim, C., Malucelli, F., e Goerigk, M. (2018). The Quadratic Shortest Path Problem: Complexity, Approximability, and Solution Methods. *European Journal of Operational Research*, 268(2):473–485.
- Rostami, B., Malucelli, F., Frey, D., e Buchheim, C. (2015). On the Quadratic Shortest Path Problem. In *International Symposium on Experimental Algorithms*, p. 379–390. Springer.
- Sen, S., Pillai, R., Joshi, S., e Rathi, A. K. (2001). A Mean-Variance Model for Route Guidance in Advanced Traveler Information Systems. *Transportation Science*, 35(1):37–49.
- Thomas, B. W., Calogiuri, T., e Hewitt, M. (2019). An Exact Bidirectional A* Approach for Solving Resource-Constrained Shortest Path Problems. *Networks*, 73(2):187–205.