Received: 28 August 2019 | Accepted: 30 August 2019

DOI: 10.1002/net.21909

SPECIAL ISSUE ARTICLE

WILEY

An exact method for a class of robust shortest path problems with scenarios

Daniel Duque | **Andrés L. Medaglia**

Centro para la Optimización y Probabilidad Aplicada (COPA), Departamento de Ingeniería Industrial, Universidad de los Andes, Bogotá, Colombia

Correspondence
Andrés L. Medaglia, Centro para la Optimización y Probabilidad Aplicada (COPA), Departamento de Ingeniería Industrial, Universidad de los Andes, Cra. 1E No. 19A-10, ML730, Bogotá, Colombia.
Email: amedagli@uniandes.edu.co

Present address
Daniel Duque, Industrial Engineering and Management Sciences, Northwestern University, 2145 Sheridan Rd, Evanston, IL.

Abstract

In this variant of the robust shortest path problem, the cost of traversing an arc is given by a discrete set of scenarios. The problem is then to find a (robust) path that takes into account the information arising from the multiple cost realizations of the possible scenarios. To account for a robust path, we adopt the *bw*-robustness criterion, which ameliorates the dramatic role played by worst-case approaches. Under this criterion, the parameter b represents a desirable upper bound for the cost that the decision maker wants for most of the scenarios; while parameter w strictly bounds the cost and represents a value that the decision maker is not willing to exceed in any scenario. To solve the problem, we extend the pulse algorithm, a general-purpose solution strategy that has been used on shortest path problems with side constraints. The proposed algorithm compares favorably against an integer programming approach both in terms of speed and scalability on networks with up to 39 377 nodes and 192 094 arcs.

KEYWORDS

bw-robustness, data-driven robust optimization, robustness, routing, shortest path, pulse algorithm

1 | INTRODUCTION

The robust shortest path (RSP) problem is a generalization of the well-known shortest path problem that takes into account the uncertainty on the parameters. Particularly, the cost of traversing an arc is no longer a fixed value, because it might not be determined accurately in practical applications. Given this source of uncertainty, it is convenient to look for a robust solution that accounts for the inherent variation of the cost of a path. There is no agreement among the concept of a robust path because there are several ways to model the parameters' uncertainty and to address their variability. Roy [22] presents different definitions of robustness and discusses how these variants can be used in the wide area of operations research. Kasperski and Zieliński [11] review recent results on robust discrete optimization.

Despite of the modeling approach or the robustness criterion, the RSP is known to be NP-hard [1, 8, 19, 25]. Murthy and Her [18] proposed one of the first approaches to solve the RSP. To model uncertainty, they adopt a scenario approach in which the cost of traversing an arc is given by a discrete set of scenarios. Using this characterization, they used dynamic programming (DP) to solve the min-max shortest path problem—also known as the absolute robust shortest path (ARSP) problem—with a label correcting algorithm that looks for a path that minimizes the maximum path cost among the scenarios. Yu and Yang [25] studied the ARSP and the robust deviation shortest path problem (RDSP). In their work on the RDSP, they look for a path that minimizes the maximum deviation between the cost of the path in a particular scenario and the cost of the shortest path of the corresponding scenario. Montemanni and Gambardella [15] studied the RDSP, but they characterize the uncertainty over the costs with interval data. Under this approach, the cost of each arc belongs to a non-negative interval of continuous values. To solve the problem they proposed a ranking algorithm [15] followed by a sequel work where they used Benders' decomposition [16]. The work by Montemanni and Gambardella [15, 16] is based on the min-max and min-max regret view of robustness for

combinatorial optimization problems which is thoroughly reviewed by Aissi et al. [1]. Gabrel et al. [8] reviewed both modeling approaches (scenario approach and interval data) and adapted the *bw*-robustness criterion in the context of RSP (*bw*-RSP). It is noteworthy to mention that the *bw*-robustness criterion was proposed earlier by Roy [22] in the wide context of operations research and decision making. Under this criterion, the objective is to find a path with a cost that never exceeds an upper bound (w) while complying with a desirable (not necessarily strict) cost bound (b) for most of the scenarios. They proposed a mathematical formulation for each modeling approach and solved it with CPLEX. Pascoal and Resende [19] proposed three algorithms for the RDSP, namely, a labeling, a ranking, and a hybrid algorithm (mixing both labeling and ranking). As an alternative way to account for a robust solution, there are also bicriteria approaches that deal with the uncertain parameters. Hasuike [10] uses fuzzy goals to find a path that minimizes the (mean) cost of a path and maximizes the range of a confidence interval for the cost. The confidence interval is modeled with the mean and variance of the path, regardless of the underlying probability distribution. Recently, Goerigk et al. [9] introduced two methods based on the concept of ranking robustness, an approach based on a preference ranking of solutions. In solution ranking, every solution is assigned a degree of preference in every scenario; whereas in objective ranking, solutions are ranked according to their objective value. They applied these concepts to the shortest path problem on a real-world street network in the context of evacuation planning. Chassein et al. [4] used the idea of data-driven robust optimization to find paths that optimize the worst-case performance over a wide range of uncertainty sets containing arc cost scenarios. They present a case study with real traffic data from the city of Chicago (United States).

In this work, we adopt the *bw*-robustness criterion adapted by Gabrel et al. [8] from Roy [22], by using a discrete set of scenarios to model uncertain parameters (i.e., costs). To solve the *bw*-RSP, we extended a general-purpose framework for hard shortest path problems [7]. The algorithm behind the framework was initially developed to handle (exactly) the constrained shortest path problem [13], yet later extended to address other shortest path variants and applications [3, 6, 7, 12, 21]. To test the algorithm, we conducted several experiments on networks with up to 39 377 nodes and 192 094 arcs, while considering up to 10 000 scenarios for each network. We performed a scalability experiment in which the algorithm compares favorably against the integer-programming approach used by Gabrel et al. [8], providing ways to use the *bw*-robustness criterion in large-scale instances under a data-driven robust optimization framework.

The rest of the article is organized as follows. Section 2 formally describes the RSP problem. Section 3 describes the algorithm that solves (exactly) the RSP problem and shows in detail its core components. Section 4 presents the computational experiments against the state-of-the-art benchmark while addressing scalability. Finally, Section 5 concludes and outlines future work.

2 | PROBLEM STATEMENT

Under a scenario approach, the RSP problem is defined over a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A}, \mathcal{S})$, where $\mathcal{N} = \{v_1, \dots, v_n\}$ is the set of nodes, $\mathcal{A} = \{(i,j) | v_i \in \mathcal{N}, v_j \in \mathcal{N}, i \neq j\}$ is the set of arcs, and $\mathcal{S} = \{1, \dots, r\}$ is the set of scenarios. For each arc $(i,j) \in \mathcal{A}$, the non-negative weight c_{ij}^k is the cost in scenario $k \in \mathcal{S}$ of traversing arc $(i,j) \in \mathcal{A}$. The RSP is the problem of finding a path $\mathcal{P}$ from the start node $v_s \in \mathcal{N}$ to the end node $v_e \in \mathcal{N}$, that achieves a given robustness criterion. Henceforth, we use function $c^k(\mathcal{P})$ to represent the cost of path $\mathcal{P}$ in scenario $k \in \mathcal{S}$, and $\mathbf{c}(\mathcal{P}) = (c^1(\mathcal{P}), \dots, c^r(\mathcal{P}))$ is the vector comprised of the costs of all scenarios. We denote by $\mathcal{P}_{ij} = \{v_i, \dots, v_{(p)}, \dots, v_j\}$ a partial path that starts at node $v_i \in \mathcal{N}$, ends at node $v_j \in \mathcal{N}$, and $v_{(p)}$ represents the node at the p-th position of the sequence.

In this work, we adopted the *bw*-robustness criterion introduced by Roy [22] and applied by Gabrel et al. [8] to shortest path applications. The idea behind this criterion is to take into account information revealed a priori by the decision maker or fed by a data-driven framework. Under *bw*-robustness, the parameter b represents a desirable upper bound for the cost that the decision maker wants for most of the scenarios; while parameter w strictly bounds the cost and represents a value that the decision maker is not willing to exceed in any scenario (so it holds that $b < w$). Gabrel et al. [8] proposed the following 0-1 integer program (IP) formulation for the *bw*-RSP:

$$\max \sum_{k \in \mathcal{S}} y_k \tag{1}$$

subject to,

$$\sum_{(i,j) \in \mathcal{A}} c_{ij}^k x_{ij} \leq w(1 - y_k) + b y_k \qquad \forall k \in \mathcal{S}, \tag{2}$$

$$\sum_{\{j | (i,j) \in \mathcal{A}\}} x_{ij} - \sum_{\{j | (j,i) \in \mathcal{A}\}} x_{ji} = \begin{cases} 1, & i = s \\ 0, & i \in \mathcal{N} \setminus \{s, e\} \\ -1, & i = e \end{cases}, \tag{3}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A}, \tag{4}$$

$$y_k \in \{0,1\} \qquad \forall k \in \mathcal{S}, \tag{5}$$

where y_k is a binary variable that takes the value of 1 if the cost of the path is less than or equal to b in scenario $k \in \mathcal{S}$ and takes the value of 0, otherwise; and x_{ij} is the arc flow through arc $(i,j) \in \mathcal{A}$. The objective function (1) aims to maximize the number of scenarios in which the cost of the robust path is below the target threshold b. Constraints (2) use variables y_k to bound the cost of the path for each scenario. Constraints (3) are the flow balance equations. Finally, constraints (4) and (5) state the binary nature of the decision variables.

3 | SOLUTION METHOD

To solve this problem we extended the pulse framework for hard shortest path problems presented in Duque et al. [7]. The underlying algorithm in the framework follows a depth-first search recursive exploration of the graph [13]. To avoid a complete enumeration of all paths in the graph, the algorithm uses pruning strategies to discard partial solutions (i.e., partial paths) that would not lead to a feasible or an improved solution at early stages of the execution. This pulse propagating idea was initially motivated by the constrained shortest path (CSP) problem [13], but it has been successfully extended to other hard shortest path variants [3, 6, 7, 12, 21]. In this section, we present an overview of the algorithm and the pruning strategies that we extended and developed to solve the RSP problem.

3.1 | An overview of the pulse algorithm

The input of the pulse algorithm for the RSP are the graph $\mathcal{G}$, a start node v_s, an end node v_e, and the robustness parameters b and w; and its output is a bw-robust path, if such a path exists. Algorithm 1 presents a pseudocode of the procedure. The `initialization` procedure involves several executions (one per scenario) of a one-to-all single-objective shortest path algorithm (see Section 3.2.1) and the `pulse` function invokes the recursive exploration accelerated by the pruning strategies.

Algorithm 1 Pulse algorithm.

Require: $\mathcal{G}$, directed graph; v_s, start node; v_e, end node; b, robustness target value; w, robustness upper bound.
Ensure: $\mathcal{P}^*$, a bw-robust path.

1: $\mathcal{P} \leftarrow \{v_s\}$
2: $\mathbf{c}(\mathcal{P}) \leftarrow \mathbf{0}$
3: `initialization`($\mathcal{G}$) ▷ *see Section 3.2.1*
4: `pulse`($v_s, \mathcal{P}, \mathbf{c}(\mathcal{P})$)
5: **return** $\mathcal{P}^*$

Given that the algorithm is based on the idea of a complete enumeration, an optimal solution is always found. By a *pulse* we refer to a partial path $\mathcal{P}_{si}$ that holds the information related to the cumulative cost $c^k(\mathcal{P}_{si})$ for each scenario $k \in \mathcal{S}$. Due to its recursive nature, the algorithm explores the network looking for feasible paths from the start node v_s to the end node v_e following a depth-first search exploration. Every time a complete feasible solution is found, the primal bound (denoted by $\underline{y}$ and globally stored) is updated and the algorithm backtracks to explore other paths. Similarly, every time that a partial solution is pruned, the algorithm backtracks to the previous node in the partial path and continues the exploration. Aside from its simple intuition, the efficiency of the algorithm lies on its pruning strategies that allows us to perform an implicit enumeration by discarding suboptimal and infeasible solutions in advance. In particular, for the bw-RSP we propose two new pruning strategies based on the bw-robustness criterion and extend the dominance pruning strategy from the pulse framework. Algorithm 2 presents the pulse recursive function which receives as input parameters the node being visited v_i, the partial path $\mathcal{P}_{si}$, and the cumulative cost of $\mathcal{P}_{si}$ in all scenarios $\mathbf{c}(\mathcal{P}_{si})$. Lines 1-3 of Algorithm 2 apply all the pruning strategies to the incoming pulse (see Section 3.2); if the pulse is not pruned (i.e., all Boolean function checks return false), line 4 stores (if applicable) the information of the partial path (see Section 3.2.3) and in lines 5-8, the pulse propagates over all nodes $v_j \in \Gamma^+(v_i)$, where $\Gamma^+(v_i)$ is the set of head nodes for the outgoing arcs from v_i, adding $\mathbf{c}_{ij}$ to the cumulative costs, where $\mathbf{c}_{ij} = (c_{ij}^1, \ldots, c_{ij}^r)$.

Algorithm 2 Pulse function.

Require: v_i, current node; $\mathcal{P}_{si}$, current path, $\mathbf{c}(\mathcal{P}_{si})$, cumulative costs for all scenarios.
Ensure: `void`

```
1: if ¬check_w_Robustness(v_i, c(P_si)) then                    ▷ see Section 3.2.1
2:     if ¬check_b_Robustness(v_i, P_si, c(P_si)) then          ▷ see Section 3.2.2
3:         if ¬checkLabels(v_i, c(P_si)) then                   ▷ see Section 3.2.3
4:             store(c(P_si))
5:             for v_j ∈ Γ+(v_i) do
6:                 P'_sj ← P_si ∪ {v_j}
7:                 c(P'_sj) ← c(P_si) + c_ij
8:                 pulse(v_j, P'_sj, c(P'_sj))
9:             end for
10:        end if
11:    end if
12: end if
13: return void
```

Algorithm 3 presents the pulse function when it is invoked over the end node. Once a pulse reaches the end node, a new solution has been found, so the algorithm verifies if the new solution is better than the current lower bound $\underline{y}$ and updates the current optimal solution $\mathcal{P}^*$ accordingly. Note that $1_{\mathcal{B}_e^k}(\mathcal{P})$ is an indicator function that takes the value of 1 if the cost of the path $\mathcal{P}$ (ending at node v_e) is less than or equal to b in scenario $k \in \mathcal{S}$ and takes the value of 0, otherwise. In Section 3.2.2 we extend this indicator function for partial solutions, that is, partial paths ending at any node v_i.

Algorithm 3 Pulse function at the end node.

Require: v_e, current node; $\mathcal{P}$, current path; $\mathbf{c}(\mathcal{P})$, cumulative costs for all scenarios.
Ensure: `void`

```
1: if Σ_{k∈S} 1_{B_e^k}(P) > y then
2:     y ← Σ_{k∈S} 1_{B_e^k}(P)
3:     P* ← P
4: end if
5: return void
```

3.2 | Pruning strategies for the *bw*-robust criterion

3.2.1 | Pruning by *w*-robustness

Parameter w is related with the feasibility of any path in the graph. The set of constraints (2) states that the cost of the path is either below the target b (if y_k takes the value of 1) or the bound w (if y_k takes the value of 0). However, since $b < w$, the cost of a path is bounded by w in every scenario. Based on the parameter w, we can prune a partial path $\mathcal{P}_{si}$ if $c^k(\mathcal{P}_{si}) > w$ for any scenario $k \in \mathcal{S}$, because the path will not reach the end node under the allowable cost bound. Moreover, we can prune this path earlier by incorporating lower bounds on the cost for every scenario. To do so, we calculate the minimum cost it takes from any node v_i to reach the end node v_e for each scenario. These lower bounds are denoted by $\underline{c}^k(v_i)$ and are computed by running a one-to-all shortest path algorithm per scenario from the end node to all nodes on a reversed graph (i.e., $\mathcal{G}\prime = (\mathcal{N}, \mathcal{A}\prime, \mathcal{S}) | \mathcal{A}\prime = \{(j,i) | (i,j) \in \mathcal{A}\}, c'^k_{ji} = c^k_{ij}, v'_s = v_e$). Once we compute these lower bounds, we can safely prune a partial path $\mathcal{P}_{si}$ if $c^k(\mathcal{P}_{si}) + \underline{c}^k(v_i) > w$ for any scenario, because completing this path with the best possible outcome is infeasible. Algorithm 4 shows the w-robustness verification.

Algorithm 4 Pruning by w-robustness: `check_w_Robustness`$(v_i, \mathbf{c}(\mathcal{P}_{si}))$.

Require: v_i, current node; $\mathbf{c}(\mathcal{P}_{si})$, cumulative costs for all scenarios.
Ensure: `boolean`

```
1: prune ←false
2: for k ∈ S and ¬prune do
```

3: **if** $c^k(\mathcal{P}_{si}) + \underline{c}^k(v_i) > w$ **then**
4: prune ←`true`
5: **end if**
6: **end for**
7: **return** prune

This idea is an extension of the infeasibility pruning strategy presented by Lozano and Medaglia [13] for the CSP. Other authors have proposed similar preprocessing procedures in the context of DP to solve other shortest path variants [5, 14, 17, 23, 24].

3.2.2 | Pruning by *b*-robustness

Under the *bw*-robustness criterion, the objective is to maximize the number of scenarios in which the cost of the path is less than or equal to b. For this reason, the algorithm keeps track of the best solution encountered so far and computes lower and upper bounds on the objective function to determine whether a partial path is suboptimal or not. These bounds always relate to the number of scenarios for which the cost target b is met.

Let $\underline{y}$ be a lower bound for the objective function (1), where $0 \leq \underline{y} \leq |S|$. Also, let $1_{\mathcal{B}_i^k}(\mathcal{P}_{si})$ be an indicator function that takes the value of 1 if $\mathcal{P}_{si} \in \mathcal{B}_i^k$ and 0, otherwise; where $\mathcal{B}_i^k$ is the set of all partial paths to v_i such that $\mathcal{P}_{si}$ satisfies that $c^k(\mathcal{P}_{si}) + \underline{c}^k(v_i) \leq b$ in scenario $k \in \mathcal{S}$. With this indicator function, we can calculate an upper bound $\overline{y}(\mathcal{P}_{si})$ for any partial path $\mathcal{P}_{si}$ as:

$$\overline{y}(\mathcal{P}_{si}) = \sum_{k \in \mathcal{S}} 1_{\mathcal{B}_i^k}(\mathcal{P}_{si}), \tag{6}$$

where,

$$1_{\mathcal{B}_i^k}(\mathcal{P}_{si}) = \begin{cases} 1, & \text{if } \mathcal{P}_{si} \in \mathcal{B}_i^k \\ 0, & \text{if } \mathcal{P}_{si} \notin \mathcal{B}_i^k \end{cases}, \tag{7}$$

$$\mathcal{B}_i^k = \{\mathcal{P}_{si} | c^k(\mathcal{P}_{si}) + \underline{c}^k(v_i) \leq b\}. \tag{8}$$

Since we are looking for a path that maximizes (1), we can prune any partial path if $\overline{y}(\mathcal{P}_{si}) < \underline{y}$ because completing $\mathcal{P}_{si}$ with the best possible path to the end in every scenario will not improve the lower bound, that is, the number of scenarios not exceeding the cost of b. Additionally, if we find a path such that $\underline{y} = |S|$, there is no need to keep exploring the network because there is proof that the algorithm has found an optimal solution because $\underline{y}$ is bounded by $|S|$. Algorithm 5 shows b-robustness verification.

Algorithm 5 Pruning by b-robustness: `check_b_Robustness`$(v_i, \mathcal{P}_{si}, \mathbf{c}(\mathcal{P}_{si}))$.

Require: v_i, current node; $\mathcal{P}_{si}$, partial path; $\mathbf{c}(\mathcal{P}_{si})$, cumulative costs for all scenarios.
Ensure: `boolean`
1: prune ← `false`
2: $\overline{y}(\mathcal{P}_{si}) \leftarrow \sum_{k \in S} 1_{\mathcal{B}_i^k}(\mathcal{P}_{si})$
3: **if** $\overline{y}(\mathcal{P}_{si}) < \underline{y}$ **then**
4: prune ← `true`
5: **end if**
6: **return** prune

3.2.3 | Pruning by labels

We use a set of labels to prove dominance relations over partial paths. For each node v_i, we store a fixed number of labels Q, each of them related to a partial solution that has already visited node v_i. Labels at node v_i are tuples that store the cost for each scenario of partial paths, and are denoted by $\mathcal{L}(v_i) = \{(c_{il}^1, \ldots, c_{il}^r) | l = 1, \ldots, Q\}$, where c_{il}^k is the cumulative cost of a partial path to node v_i in scenario $k \in \mathcal{S}$ stored in the l-th label. We test dominance over a partial path $\mathcal{P}_{si}$ using the cumulative costs for each scenario $c^k(\mathcal{P}_{si})$, that is, for an incoming partial path $\mathcal{P}_{si}$, the algorithm checks if it is dominated (or not) by any label in $\mathcal{L}(v_i)$. A partial path $\mathcal{P}_{si}$ is dominated (or at best is an alternative path) if a label $(c_{il}^1, \ldots, c_{il}^r) \in \mathcal{L}(v_i)$ satisfies that $c_{il}^k \leq c^k(\mathcal{P}_{si})$

for all $k \in \mathcal{S}$. When a partial path is not pruned by any of the strategies, the algorithm stores the partial path as a label in an empty slot of $\mathcal{L}(v_i)$ following a nondecreasing order according to the cost of the first scenario. If there are no empty slots, it overwrites any (randomly selected) label. Algorithm 6 shows the dominance verification. Theorem 3.1 supports the correctness of the pruning strategy.

Algorithm 6 Pruning by dominance: `checkLabels`($v_i, \mathbf{c}(\mathcal{P}_{si})$).

Require: v_i, current node; $\mathbf{c}(\mathcal{P}_{si})$, cumulative costs for all scenarios.
Ensure: `boolean`
1: prune ← `false`
2: **for** $(c^1_{il}, \dots, c^r_{il}) \in \mathcal{L}(v_i)$ **do**
3: **if** $(c^1_{il}, \dots, c^r_{il}) \leq \mathbf{c}(\mathcal{P}_{si})$ **then**
4: prune ← `true`
5: **end if**
6: **end for**
7: **return** prune

Theorem 3.1. *Let $\mathcal{P}_{si}$ be a partial path from node v_s to node v_i. If there exists a partial path $\mathcal{P}'_{si}$ from node v_s to node v_i such that for all scenario $k \in \mathcal{S}$, $c^k(\mathcal{P}'_{si}) \leq c^k(\mathcal{P}_{si})$, then $\mathcal{P}_{si}$ can be pruned.*

Proof. Let $\mathcal{P} = \mathcal{P}_{si} \cup \mathcal{P}_{ie}$ and $\mathcal{P}' = \mathcal{P}'_{si} \cup \mathcal{P}_{ie}$ be two complete paths sharing the same ending path $\mathcal{P}_{ie}$ from node v_i to node v_e. For all $k \in \mathcal{S}$, $c^k(\mathcal{P}) = c^k(\mathcal{P}_{si}) + c^k(\mathcal{P}_{ie})$ and $c^k(\mathcal{P}') = c^k(\mathcal{P}'_{si}) + c^k(\mathcal{P}_{ie})$. Given that $c^k(\mathcal{P}'_{si}) \leq c^k(\mathcal{P}_{si})$ for all $k \in \mathcal{S}$, then $c^k(\mathcal{P}') \leq c^k(\mathcal{P})$ for all $k \in \mathcal{S}$. From Equation (6), the objective function value for both paths can be computed as $y(\mathcal{P}')$ and $y(\mathcal{P})$. Since $c^k(\mathcal{P}') \leq c^k(\mathcal{P})$ for all $k \in \mathcal{S}$, it is always true that $y(\mathcal{P}') \geq y(\mathcal{P})$ for any common ending path $\mathcal{P}_{ie}$, which means that $\mathcal{P}_{si}$ can be safely pruned. ∎

4 | COMPUTATIONAL EXPERIMENTS

We tested our algorithm over six networks derived from the ones proposed by Beasley and Christofides [2] for the resource constrained shortest path problem and two networks derived from the biobjective shortest path problem [20]. The first six networks range from 100 to 500 nodes and from 990 to 4868 arcs, while the last two networks have up to 53 658 nodes and 192 094 arcs. For the smaller networks, we set the number of scenarios to 10, 100, 1000, and 10 000; whereas for the larger networks, the number of scenarios takes the values of 10, 50, 100, 200, 500, and 1000. Although a larger number of scenarios might represent better the uncertainty of the costs, we set the maximum number of scenarios to 1000 in the larger networks to balance the computational requirements of storing the costs for all arcs. The arc costs were randomly generated following a Gamma distribution $\Gamma(\alpha_{ij},\theta_{ij})$; the shape parameter α_{ij} was randomly drawn from the set $\alpha_{ij} \in \{1,2,3\}$; and the scale parameter was set by $\theta_{ij} = \alpha_{ij}/\mu_{ij}$, where μ_{ij} is the nominal value of the arc generated by $\mu_{ij} \sim U(1000, 3000)$. For each combination of network and number of scenarios, we run six different setups varying the value of the parameters w and b in a similar fashion to Gabrel et al. [8]. The purpose of this sensitivity analysis is to stress the algorithm under different tightness levels for constraint (2) of the *bw*-RSP. The values for w and b were computed as follows:

$$w = \begin{cases} z^{worst} \\ 0.5z^{worst} + 0.5z^{worst}_{EV} \\ z^{worst}_{EV} \end{cases},$$

$$b = \begin{cases} z^{best} + 0.5(w - z^{best}) \\ z^{best} + 0.8(w - z^{best}) \end{cases},$$

where z^{worst} is the cost of the ARSP (i.e., $z^{worst} = \min \max_{k \in \mathcal{S}} \sum_{(i,j) \in \mathcal{A}} c^k_{ij} x_{ij}$), and is the tightest value that w can take. On the other hand, z^{worst}_{EV} is the cost of the worst scenario of the shortest path using the nominal values μ_{ij} which is a loose bound for w. To set values for parameter b, we used a fixed value of w and z^{best}, which is the least cost of the shortest path among all scenarios. Note that b has to be bounded by w, and even if w takes a very tight value, different values of b might lead to different (sometimes alternative) solutions. To obtain z^{worst} we executed a modified version of the algorithm that accounts for this robustness criterion.

We coded our algorithm in Java and compiled it using Eclipse SDK version 4.2.2. The computational experiments for Sections 4.1, 4.2, 4.3.1, and 4.3.3 were performed on a computer with an Intel Xeon E5-2695 @2.4Ghz (10 cores); and for

TABLE 1 Computational results for the instances derived from Beasley and Christofides [2]

				IP			Pulse			
Network	**Nodes**	**Arcs**	**Scenarios**	**Solved**	**Avg. time (s)**	**Max. time (s)**	**Solved**	**Avg. time (s)**	**Max. time (s)**	**Speedup**
rcsp5	100	990	10	6/6	0.099	0.125	6/6	0.005	0.005	19
			100	6/6	0.666	0.687	6/6	0.042	0.042	16
			1000	6/6	11.484	11.576	6/6	0.178	0.178	65
			10 000	6/6	246.843	258.838	6/6	2.080	2.081	119
rcsp7	100	999	10	6/6	0.197	0.374	6/6	0.005	0.005	42
			100	6/6	2.707	8.783	6/6	0.021	0.023	126
			1000	4/6	237.542	>600	6/6	0.183	0.191	>1298
			10 000	3/6	482.828	>600	6/6	1.894	1.959	>255
rcsp13	200	2080	10	6/6	0.255	0.297	6/6	0.003	0.003	97
			100	6/6	6.045	6.818	6/6	0.018	0.018	336
			1000	6/6	151.257	253.782	6/6	0.189	0.190	800
			10 000	0/6	600.000	>600	6/6	1.864	1.870	>322
rcsp15	200	1960	10	6/6	0.858	1.248	6/6	0.002	0.002	366
			100	6/6	12.878	35.693	6/6	0.021	0.027	613
			1000	3/6	402.022	>600	6/6	0.199	0.237	>2017
			10 000	0/6	600.000	>600	6/6	2.097	2.185	>286
rcsp21	500	4847	10	6/6	2.714	4.789	6/6	0.003	0.003	898
			100	6/6	19.999	32.479	6/6	0.029	0.030	689
			1000	4/6	450.240	>600	6/6	0.264	0.273	>1708
			10 000	0/6	600.000	>600	6/6	3.453	3.660	>174
rcsp23	500	4868	10	6/6	1.485	1.544	6/6	0.003	0.003	469
			100	4/6	242.439	>600	6/6	0.035	0.044	>6958
			1000	2/6	495.769	>600	6/6	0.305	0.391	>1627
			10 000	0/6	600.000	>600	6/6	3.582	4.405	>168
				104/144			144/144			

Abbreviation: IP, integer program.

Section 4.3.2, we used an Intel Xeon E5-2673 v4 @2.3Ghz (8 virtual processors). In both machines, we allocated 32GB of RAM to the memory heap of the Java virtual machine. The initialization procedure is executed in parallel cores since all one-to-all single objective shortest path problems are independent from each other.

4.1 | Performance and scalability against an IP-based approach

This experiment seeks to compare our algorithm against the IP formulation proposed by Gabrel et al. [8], which is to the best of our knowledge, the only method available that solves the *bw*-RSP. In this experiment we used the six networks derived from Beasley and Christofides [2], and fixed the number of scenarios and values for b and w as described above, for a grand total of 144 instances. We implemented the IP-based approach for the *bw*-RSP proposed by Gabrel et al. [8] in Gurobi 6.0.2 using the Java interface and allocated 32GB of RAM. For both approaches, we set a time limit of 600 seconds to solve the problem.

Table 1 compares our algorithm against the IP approach. Columns 1-4 present the name of the network, the number of nodes, the number of arcs, and the number of scenarios. Columns 5-7 present the performance metrics for the IP; where column 5 shows the number of instances solved to optimality out of the six setups, column 6 shows the average computational time, and column 7 exhibits the maximum computational time among the six setups. Columns 8-10 present the same metrics for the pulse algorithm. Finally, column 11 shows the speedup measured as the ratio of the average time of the IP approach to the average time of the pulse algorithm.

The proposed pulse-based approach solved all instances faster than the IP approach. Note that the IP solves only 104 out of 144 instances (72%) within the time limit, yet taking longer computational times. The speedups of the average central processing unit (CPU) times range from 16 to 6958 times faster, depending on the network and number of scenarios. As expected, regardless of the network size, the computational time required to solve the problem increases as the number of scenarios grows for both approaches. Nevertheless, the pulse-based algorithm scales better in CPU time than the IP when it comes to the network size and the number of scenarios.

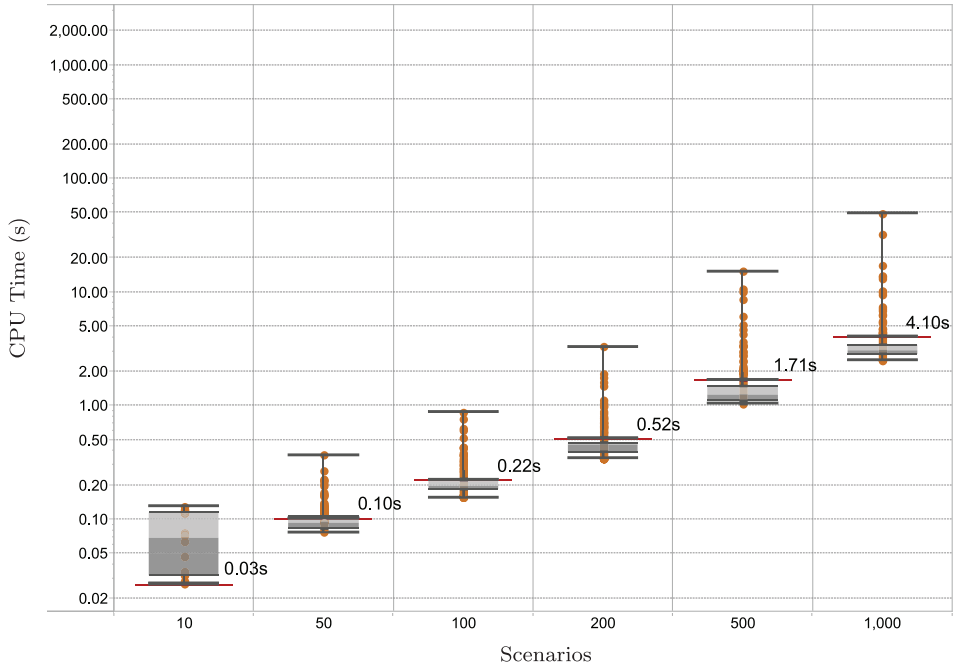

FIGURE 1 Computational results for the road network of Washington D.C. (DC instances) [Colour figure can be viewed at wileyonlinelibrary.com]

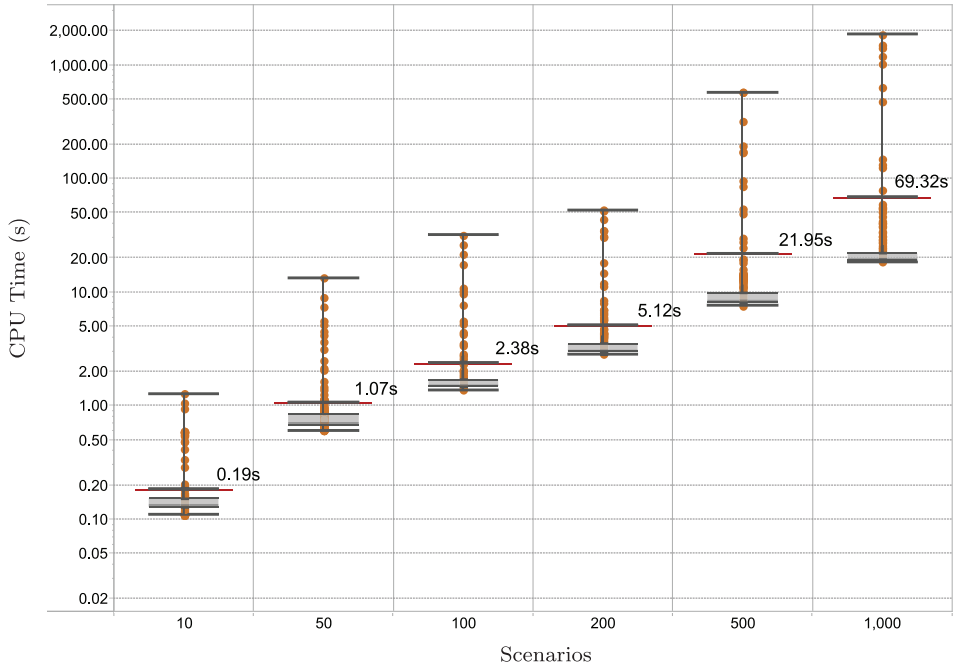

FIGURE 2 Computational results for the road network of Rhode Island (RI instances) [Colour figure can be viewed at wileyonlinelibrary.com]

4.2 | Scalability of the pulse-based approach on larger networks

The second set of experiments aims to stretch the limits of the pulse-based algorithm on two large-scale networks derived from the ones proposed by Raith and Ehrgott [20]. The first network corresponds to the real road network of Washington D.C. (henceforth labeled DC) which is comprised of 9559 nodes and 39 377 arcs. The second network corresponds to Rhode Island (henceforth labeled RI) which is comprised of 53 658 nodes and 192 084 arcs. The number of scenarios that we tested were $|S| \in \{10, 50, 100, 200, 500, 1000\}$, and additional to the six setups that vary parameters b and w, we generated 30 random origin-destination (OD) pairs for a total of 2160 instances.

Figures 1 and 2 show the computational results for all DC and RI instances. In the y-axis, the computational time is shown in logarithmic scale, while the x-axis shows the number of scenarios. The boxplots show the minimum, maximum, median, and quartiles of the computational time; and the red line indicates the overall average.

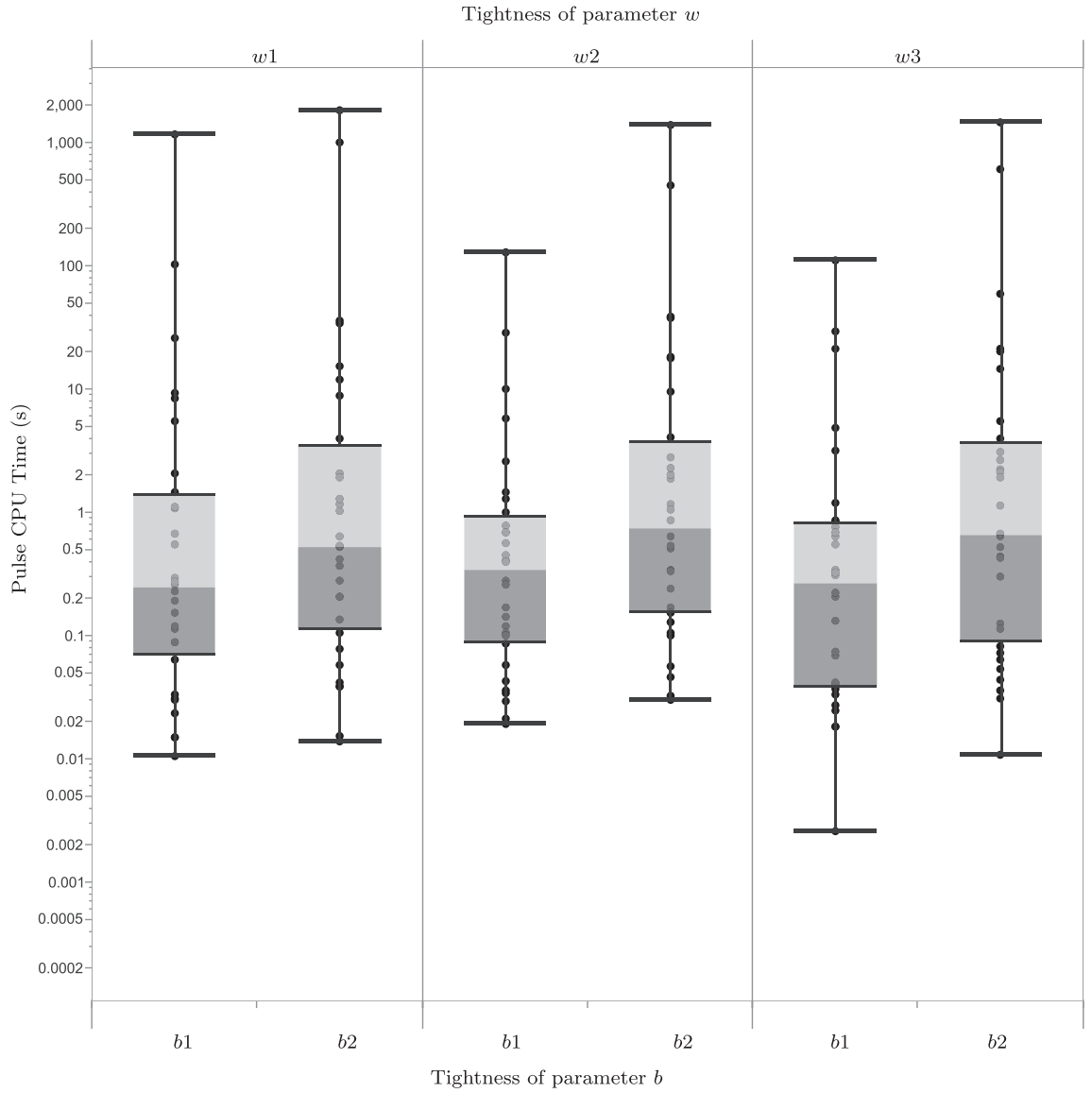

FIGURE 3 Sensitivity analysis of parameters b and w on the RI instances with 1000 scenarios. Tightness parameters presented from tight (left) to loose (right) values

For the DC instances, the average computational times range from 0.03 to 4.10 seconds depending on the number of scenarios; while the hardest instances with 1000 scenarios were solved by the pulse algorithm in less than 50 seconds. It is worth noting that instances with 100 scenarios where solved within 1 second, and for the bulk of cases the solution obtained with a larger number of scenarios did not change. This is the case for DC instances, where just 100 scenarios are enough to capture the cost uncertainty. For the RI instances, average times range from 0.19 to 69.32 seconds, while the hardest instance was solved in less than 2000 seconds. For both networks it is expected that the computational time grows due to the number of shortest path problems to be solved in the initialization procedure and the complexity of the problem as it scales in size.

4.3 | Introspective assessment of the pulse algorithm

We conducted three analyses to better assess the performance of the pulse algorithm. First, we conducted a sensitivity analysis on the values of the parameters b and w to identify whether the values of these parameters affect the performance of the algorithm. Second, we conducted a sensitivity analysis on the number of labels per node to measure the impact on the performance and the variability due to the nondeterministic behavior caused by the random overwriting of labels when the slots are full. Third, we conducted an analysis on the relative effectiveness of the pruning strategies. For these analyses we used the 30 instances of the RI network with 1000 scenarios, as these problems are the hardest ones.

TABLE 2 Pulse variability of the CPU time measured by the coefficient of variation CV for $\gamma = 1$

	Tightness parameter combination (w, b)						
OD pair	$(w1, b1)$	$(w1, b2)$	$(w2, b1)$	$(w2, b2)$	$(w3, b1)$	$(w3, b2)$	**Average**
1	0.0694	0.0364	0.0227	0.1042	0.0457	0.0905	0.0615
2	0.0286	0.0234	0.0136	0.0170	0.0133	0.0101	0.0177
3	0.0480	0.0737	0.0304	0.0678	0.0268	0.0751	0.0536
4	0.2017	0.0469	0.0168	0.0363	0.0329	0.0461	0.0635
5	0.0099	0.0323	0.0183	0.0216	0.0272	0.0178	0.0212
6	0.0269	0.0169	0.0217	0.0139	0.0531	0.0152	0.0246
7	0.0379	0.2549	0.0445	0.0557	0.0425	0.0550	0.0818
8	0.0180	0.0177	0.0066	0.0116	0.0088	0.0134	0.0127
9	0.0280	0.0049	0.0051	0.0112	0.0136	0.0104	0.0122
10	0.0153	0.0077	0.0089	0.0068	0.0095	0.0053	0.0089
11	0.0381	0.0308	0.0378	0.0239	0.0211	0.0276	0.0299
12	0.0484	0.0371	0.0340	0.0262	0.0299	0.0333	0.0348
13	0.0578	0.0563	0.0453	0.0643	0.0303	0.0546	0.0514
14	0.0999	0.0343	0.0547	0.0546	0.0458	0.0519	0.0569
15	0.0231	0.0215	0.0259	0.0140	0.0335	0.0174	0.0226
16	0.0414	0.0415	0.0329	0.0208	0.0663	0.0247	0.0379
17	0.0739	0.1254	0.0461	0.0338	0.0598	0.0503	0.0649
18	0.0505	0.0547	0.0281	0.0696	0.0590	0.0470	0.0515
19	0.0589	0.0360	0.0589	0.0494	0.0561	0.0482	0.0513
20	0.0719	0.0308	0.0603	0.0525	0.0443	0.0610	0.0535
21	0.0313	0.0247	0.0359	0.0331	0.0499	0.0280	0.0338
22	0.0591	0.0719	0.0470	0.0183	0.0342	0.0404	0.0452
23	0.0770	0.0684	0.2455	0.0452	0.0403	0.0915	0.0947
24	0.0233	0.0129	0.0268	0.0270	0.0169	0.0234	0.0217
25	0.0516	0.0560	0.0802	0.0720	0.0443	0.0319	0.0560
26	0.0483	0.0220	0.0539	0.0314	0.0532	0.0189	0.0380
27	0.0167	0.0232	0.0289	0.0651	0.0499	0.0257	0.0349
28	0.0523	0.0556	0.0658	0.0458	0.0558	0.0492	0.0541
29	0.0387	0.0349	0.0471	0.0264	0.0511	0.0353	0.0389
30	0.0809	0.0627	0.0475	0.0635	0.1421	0.0570	0.0756
Average	0.0509	0.0472	0.0430	0.0394	0.0419	0.0385	0.0435

4.3.1 | Sensitivity on b and w

Figure 3 presents the computational time of the pulse algorithm without counting the initialization procedure. Each boxplot is associated to a particular configuration of the value of b and w, where $b1 = z^{best} + 0.5(w - z^{best})$, $b2 = z^{best} + 0.8(w - z^{best})$, $w1 = z^{worst}$, $w2 = 0.5z^{worst} + 0.5z_{EV}^{worst}$, and $w3 = z_{EV}^{worst}$.

Among the three values of w and the two values of b, the average time and standard deviation (SD) varies widely, however, the median values are more alike. For the six setups, the median time falls within 0.1 and 1 second, and 75% of the instances are solved in less than 10 seconds. It is worth noting, that as the b parameter becomes loose for a given w, the instances tend to become slightly harder. Summarizing, the values of the parameters do affect the computational time, but for the large part of the instances the computational time behavior is very similar.

4.3.2 | Sensitivity on the number of labels Q

For this experiment, we parameterized the number of labels per node $Q = \gamma \cdot |S|$, where S is the set of scenarios and $\gamma \in \{0.5,1,1.5\}$. The base value for γ is 1, as this factor provides a number of labels with a stable behavior across all instances. We conducted 10 independent replications for each of the 30 OD pairs of the RI instances, with the six parameter tightness combinations described in Section 4.3.1 (namely, $b \in \{b1,b2\}$ and $w \in \{w1,w2,w3\}$), and the three values of γ.

First, we address the variability of the computational time due to the nondeterministic behavior caused by the random overwriting of labels. We fixed γ in 1 (i.e., $Q = 1000$), and for a given combination of b and w, we calculate the coefficient of variation $CV(\mathbf{t}) = \frac{\sigma(\mathbf{t})}{\mu(\mathbf{t})}$, where $\sigma(\mathbf{t})$ and $\mu(\mathbf{t})$ are the SD and mean of the CPU times $\mathbf{t}$ comprised of the 10 replications of the parameters' combination, respectively. Values of the coefficient of variation larger than 1 indicate a sample with a large variability. We computed the average coefficient of variation for each OD pair by calculating the mean over the six tightness

TABLE 3 Impact on the increment of the number of labels $\gamma = 1.5$ ($Q = 1500$) measured by the slowdown with respect to the base case

	Tightness parameter combination (w, b)						
OD pair	**($w1$, $b1$)**	**($w1$, $b2$)**	**($w2$, $b1$)**	**($w2$, $b2$)**	**($w3$, $b1$)**	**($w3$, $b2$)**	**Average**
1	0.99	0.98	1.00	0.94	1.03	1.05	1.00
2	1.06	1.10	1.03	1.15	1.03	1.16	1.09
3	1.02	0.98	1.03	0.99	1.00	0.97	1.00
4	0.96	0.99	1.05	1.01	0.99	1.03	1.00
5	0.99	1.01	1.01	1.01	1.00	1.03	1.01
6	0.99	1.01	1.00	1.01	0.97	1.03	1.00
7	0.97	0.98	1.01	0.99	0.99	1.03	0.99
8	1.05	1.06	1.03	1.10	1.05	1.12	1.07
9	1.35	1.41	1.20	1.37	1.21	1.34	1.31
10	1.14	1.23	1.12	1.23	1.09	1.24	1.17
11	1.00	0.99	1.00	0.98	1.01	0.98	0.99
12	0.91	0.95	0.96	1.02	1.02	1.00	0.98
13	1.04	1.01	1.03	1.01	1.06	0.94	1.02
14	0.87	1.00	1.11	1.04	1.19	0.98	1.03
15	1.02	1.03	1.02	1.07	1.01	1.04	1.03
16	0.98	0.97	1.04	0.98	0.97	0.99	0.99
17	1.06	1.03	0.91	0.95	1.11	1.10	1.03
18	0.97	0.97	0.99	0.96	1.04	1.02	0.99
19	1.13	0.97	1.01	0.98	0.97	0.98	1.01
20	0.99	1.03	0.98	1.03	0.97	0.99	1.00
21	1.02	1.02	1.00	1.01	0.98	1.01	1.01
22	0.99	0.97	0.98	0.94	1.07	1.13	1.02
23	0.91	0.98	0.83	0.94	1.07	0.99	0.95
24	1.03	1.11	1.03	1.07	1.00	1.09	1.06
25	0.96	1.06	0.99	0.92	0.95	1.08	0.99
26	1.00	1.00	0.98	0.99	0.99	1.01	1.00
27	1.09	1.06	0.98	1.03	0.99	1.05	1.03
28	0.96	0.98	1.05	0.96	0.97	0.96	0.98
29	1.00	1.00	0.98	1.00	0.99	0.97	0.99
30	1.07	0.98	0.97	1.00	0.90	1.12	1.01
Average	1.02	1.03	1.01	1.02	1.02	1.05	1.02

combinations. Also, we computed for each tightness combination, the average coefficient of variation over all pairs. Table 2 shows the lack of variability in terms of the coefficient of variation for the pulse algorithm, with values of *CV* consistently less than one for each parameter combination and across OD pairs; and with an overall average *CV* of 0.0435. It is worth mentioning that in the experiment we only timed the pulse component to isolate the random effect from the deterministic initialization.

Second, we address the choice of the number of labels Q and its impact on the computational performance. Tables 3 and 4 show the impact on the computational time when increasing and decreasing the number of labels with respect to the base case $\gamma = 1$ ($Q = 1000$), respectively. The metric we use is the average speedup with respect to the base case, that is, the ratio between the average computational times using the base case as the denominator. Values larger (smaller) than 1 denote a slowdown (speedup) with respect to the base case. We computed the average speedup for each OD pair by calculating the mean over the six tightness combinations. Also, we computed for each tightness combination, the average speedup over all pairs. Table 3 shows a slight slowdown that ranges from 0% to 4% for the tightness combinations across all OD pairs; and an overall average slowdown of 2%, which is the computational price for using more labels ($Q = 1500$) than the base case. On the other hand, Table 4 shows a slight speedup from 0% to 3% for the tightness combinations across all OD pairs; and an overall average speedup of 3% by using less labels ($Q = 500$) than the base case.

To gain more insight, we now focus our analysis on the effect of the choice of the number of labels and tightness parameters on a hard instance (OD pair # 10) that took more than 10 seconds on average to solve (only four OD pairs out of 30 meet this criterion) across the six combinations of b and w. Figure 4 shows that regardless of the tightness settings, as we use more labels the CPU time degrades. Note that given a parameter setting (b, w, and Q), the CPU time has a relatively low variability. This instance also confirms what we have observed before, that is, as the parameter b becomes loose given a value of w, the problem becomes harder.

TABLE 4 Impact on the decrement of the number of labels $\gamma = 0.5$ ($Q = 500$) measured by the speedup with respect to the base case

	Tightness parameter combination (w, b)						
OD pair	$(w1, b1)$	$(w1, b2)$	$(w2, b1)$	$(w2, b2)$	$(w3, b1)$	$(w3, b2)$	**Average**
1	0.93	1.03	1.03	0.93	0.99	1.01	0.99
2	0.91	0.82	0.96	0.76	0.94	0.75	0.86
3	1.00	1.04	1.06	1.00	1.02	0.99	1.02
4	0.90	1.17	1.00	0.99	1.05	1.03	1.02
5	0.98	0.99	0.96	0.94	0.98	0.91	0.96
6	1.01	0.95	1.00	0.96	0.97	0.96	0.98
7	1.00	0.98	1.03	1.36	1.00	1.15	1.09
8	0.88	0.86	0.92	0.83	0.88	0.81	0.87
9	0.67	0.65	0.74	0.63	0.72	0.65	0.68
10	0.81	0.75	0.84	0.76	0.84	0.75	0.79
11	1.00	1.00	1.02	1.00	1.01	0.99	1.00
12	0.94	0.99	1.02	1.00	1.01	1.01	0.99
13	0.96	1.00	1.02	1.02	1.05	0.97	1.00
14	0.94	1.08	1.09	1.06	0.98	0.98	1.02
15	1.00	0.93	1.00	0.92	1.04	0.90	0.96
16	1.00	1.02	0.99	0.98	1.01	1.00	1.00
17	1.01	0.95	0.96	1.07	1.11	1.12	1.04
18	0.99	1.02	0.98	0.96	1.06	1.01	1.00
19	0.99	0.94	1.03	1.00	0.94	0.98	0.98
20	1.01	1.04	0.97	1.00	0.97	0.99	0.99
21	0.99	0.98	1.03	0.94	0.99	0.94	0.98
22	1.02	0.97	0.99	0.95	0.99	1.11	1.00
23	1.00	0.99	0.87	0.93	1.10	0.97	0.98
24	0.91	0.84	0.97	0.89	0.96	0.87	0.91
25	1.00	1.07	1.01	0.98	1.05	1.07	1.03
26	1.01	1.00	1.02	1.00	1.01	1.01	1.01
27	0.88	0.87	1.02	0.86	1.01	0.94	0.93
28	1.01	1.01	1.01	0.97	0.93	1.00	0.99
29	1.04	0.99	1.01	0.98	1.04	1.00	1.01
30	1.10	1.05	1.01	1.00	0.96	0.99	1.02
Average	0.96	0.96	0.99	0.96	0.99	0.96	0.97

In summary, the proposed algorithm shows a stable behavior in terms of a low variability of the computing time, regardless of the effect introduced by the random label replacement. By using few labels, both the lexicographic insertion and the dominance test are fast. On the other hand, a few number of labels might work well for most instances, but not for the hardest ones. This might cause the recursion stack to grow beyond the allocated memory. Although using a larger number of labels might become slow, it is often a safe trade-off for solving hard instances. Nevertheless, on large-scale networks a disproportionate number of labels might also cause the algorithm to exceed the allocated memory. It is important to calibrate the behavior of the algorithm with a set of instances with a topology close to the problem at hand.

4.3.3 | Relative effectiveness of the pruning strategies

For the third experiment, Figure 5 presents how frequent the pruning strategies are used in this assessment. In general, the *b*-robustness pruning strategy is the most used strategy. On average, the algorithm relies on this strategy to prune 63% of the paths, followed by the *w*-robustness pruning strategy with 25%, and labels pruning strategy with almost 12%. Nonetheless, this analysis measures the relative effectiveness of strategies since pruning a path at an early stage might be more effective than a very frequent pruning strategy at a latter stage.

5 | CONCLUSIONS

In this work we extended the pulse framework for the RSP problem where the uncertainty of the costs is addressed with a scenario approach in which a finite set of scenarios determine different realizations for the cost of each arc. To account for

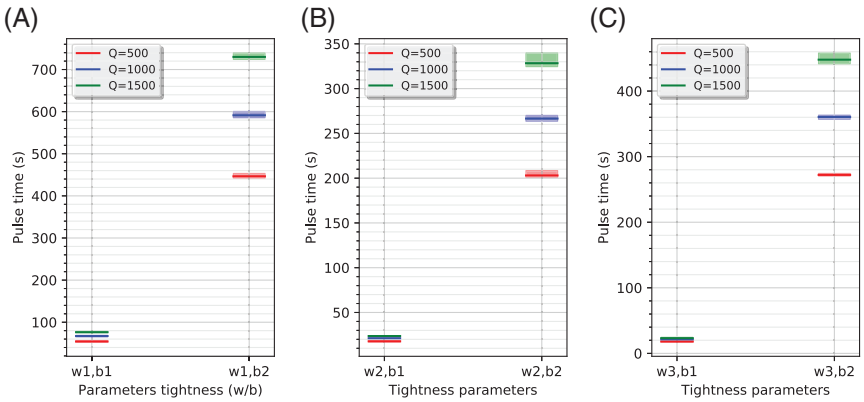

FIGURE 4 Sensitivity on w, b, and Q for the hard OD-pair # 10: A, tight ($w = w1$); B, half-tight ($w = w2$); and C, loose ($w = w3$) [Colour figure can be viewed at wileyonlinelibrary.com]

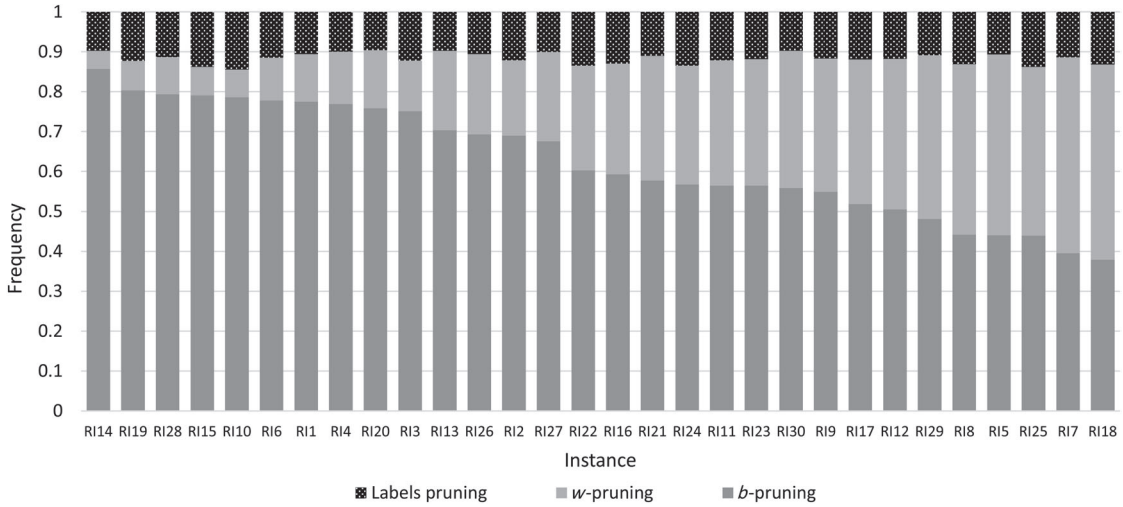

FIGURE 5 Relative effectiveness of the pruning strategies

a robust path, we used the bw-robustness criterion as shown by Gabrel et al. [8]. With this criterion the solution depends on two parameters that take into account a target value b and an upper bound for the cost of a path w. To tackle this particular variant, we extended the pulse framework and pruning strategies presented by [13]. We modeled the upper bound of the path cost as multiple resource constraints in the algorithm, and prune suboptimal paths by calculating upper bounds for the objective function at hand.

We compared the proposed pulse-based algorithm against the integer program presented in Gabrel et al. [8]. The proposed algorithm outperformed the IP-based approach with speedups of up to 6000 and solved mid-size instances with 10 000 scenarios in less than 5 seconds. In terms of scalability, our algorithm was able to solve all instances (38.5% more than the IP-based approach) within the time limit, especially under a large number of scenarios. On the large-scale networks, the proposed algorithm solved instances with up to 1000 scenarios in at most 2000 seconds for the largest network, yet achieving average times below 70 seconds.

We performed an assessment of the proposed algorithm to analyze the impact of the input parameters on the computational time and to evaluate the relative effectiveness of the strategies used in the algorithm. Although there are variations in the computational time among different values of the parameters, most of the hardest instances are solved under 10 seconds. In general, tighter instances in terms of w and b tend to be slightly easier than loose instances. The number of labels Q requires an a-priori calibration. Small values tend to work well for most instances, but might not work for the hardest ones. On the other hand, larger values tend to make the algorithm slower, but it often presents a safe compromise. In terms of the relative effectiveness of the pruning strategies, the bounds-based strategy was the most frequently used while the least used was the label-based strategy.

Research currently underway focuses on adapting the algorithm to other robustness criteria. We are also dealing with shortest path variants in which the uncertainty is not defined by a set of scenarios, but a probability distribution. For this stochastic variant, key challenges arise in the convolution of random variables and the bounds for the pruning strategies.

ACKNOWLEDGMENTS

We would like to thank Leonardo Lozano at the Lindner College of Business of the University of Cincinnati for his insightful comments on the article. Also, we thank the members of the Center for Optimization and Applied Probability (COPA) for the in-depth discussion and feedback that ultimately enriched the article.

ORCID


Andrés L. Medaglia https://orcid.org/0000-0003-1529-0322

REFERENCES

[1] H. Aissi, C. Bazgan, and D. Vanderpooten, *Min-max and min-max regret versions of combinatorial optimization problems: A survey*, European J. Oper. Res. **197** (2009), 427–438.
[2] J.E. Beasley and N. Christofides, *An algorithm for the resource constrained shortest path problem*, Networks **19** (1989), 379–394.
[3] M.A. Bolívar, L. Lozano, and A.L. Medaglia, *Acceleration strategies for the weight constrained shortest path problem with replenishment*, Optim. Lett. **8** (2014), 2155–2172.
[4] A. Chassein, T. Dokka, and M. Goerigk, *Algorithms and Uncertainty Sets for Data-Driven Robust Shortest Path Problems*. ArXiv e-prints 2018 February.
[5] I. Dumitrescu and N. Boland, *Improved preprocessing, labeling and scaling algorithms for the weight–constrainted shortest path problem*, Networks **42** (2003), 135–153.
[6] D. Duque, L. Lozano, and A.L. Medaglia, *An exact method for the biobjective shortest path problem for large-scale road networks*, European J. Oper. Res. **242** (2015), 788–797.
[7] D. Duque, L. Lozano, and A.L. Medaglia, *Solving the orienteering problem with time windows via the pulse framework*, Comput. Oper. Res. **54** (2015), 168–176.
[8] V. Gabrel, C. Murat, and L. Wu, *New models for the robust shortest path problem: Complexity, resolution and generalization*, Ann. Oper. Res. **207** (2013), 97–120.
[9] M. Goerigk, H.W. Hamacher, and A. Kinscherff, *Ranking robustness and its application to evacuation planning*, European J. Oper. Res. **17** (2016), 38–31.
[10] T. Hasuike, *Robust shortest path problem based on a confidence interval in fuzzy bicriteria decision making*, Inform. Sci. **221** (2013), 520–533.
[11] A. Kasperski and P. Zieliński, "*Robust discrete optimization under discrete and interval uncertainty*," *A Survey Cham*, M. Doumpos, C. Zopounidis, and E. Grigoroudis (eds), Springer International Publishing, Switzerland, 2016, pp. 113–143.
[12] L. Lozano, D. Duque, and A.L. Medaglia, *An exact algorithm for the elementary shortest path problem with resource constraints*, Transp. Sci. **50** (2016), 348–357.
[13] L. Lozano and A.L. Medaglia, *On an exact method for the constrained shortest path problem*, Comput. Oper. Res. **40** (2013), 378–384.
[14] E. Machuca, L. Mandow, J.L. Pérez de la Cruz, and A. Ruiz-Sepulveda, *A comparison of heuristic best-first algorithms for bicriterion shortest path problems*, European J. Oper. Res. **217** (2012), 44–53.
[15] R. Montemanni and L.M. Gambardella, *An exact algorithm for the robust shortest path problem with interval data*, Comput. Oper. Res. **31** (2004), 1667–1680.
[16] R. Montemanni and L.M. Gambardella, *The robust shortest path problem with interval data via Benders decomposition*, 4OR **3** (2005), 315–328.
[17] M. Müller-Hannemann and M. Schnee, "*Finding all attractive train connections by multi-criteria Pareto search*," *Algorithmic Methods for Railway Optimization, vol. 4359 of Lecture Notes in Computer Science*, F. Geraets, L. Kroon, A. Schoebel, D. Wagner, and C. Zaroliagis (eds), Springer, Berlin, Heidelberg, 2007, pp. 246–263.
[18] I. Murthy and S. Her, *Solving min-max shortest-path problems on a network*, Naval Res. Logist. **39** (1992), 669–683.
[19] M.M.B. Pascoal and M. Resende, *The minmax regret robust shortest path problem in a finite multi-scenario model*, Appl. Math. Comput. **241** (2014), 88–111.
[20] A. Raith and M. Ehrgott, *A comparison of solution strategies for biobjective shortest path problems*, Comput. Oper. Res. **36** (2009), 1299–1331.
[21] M.I. Restrepo, L. Lozano, and A.L. Medaglia, *Constrained network-based column generation for the multi-activity shift scheduling problem*, Int. J. Prod. Econ. **140** (2012), 466–472.
[22] B. Roy, *Robustness in operational research and decision aiding: A multi-faceted issue*, European J. Oper. Res. **200** (2010), 629–638.
[23] O.J. Smith, N. Boland, and H. Waterer, *Solving shortest path problems with a weight constraint and replenishment arcs*, Comput. Oper. Res. **39** (2012), 964–984.
[24] C.T. Tung and K.L. Chew, *A multicriteria Pareto-optimal path algorithm*, European J. Oper. Res. **62** (1992), 203–209.
[25] G. Yu and J. Yang, *On the robust shortest path problem*, Comput. Oper. Res. **25** (1998), 457–568.

How to cite this article: Duque D, Medaglia AL. An exact method for a class of robust shortest path problems with scenarios. *Networks.* 2019;1–14. https://doi.org/10.1002/net.21909