This document is the accepted version of the research paper:

David García-Heredia, Elisenda Molina, Manuel Laguna, Antonio Alonso-Ayuso (2021). A Solution Method for the Shared Resource-Constrained Multi-Shortest Path Problem Expert Systems with Applications 182, 115193. Doi: 10.1016/j.eswa.2021.115193

Dataset and codes used in computational experiments are available at https://github.com/DavidGarHeredia/ATFM-datasets

A Solution Method for the Shared Resource-Constrained Multi-Shortest Path Problem*

David García-Heredia[a,*], Elisenda Molina[a], Manuel Laguna[b], Antonio Alonso-Ayuso[c]

[a] *Departamento de Estadística, Universidad Carlos III de Madrid, Getafe (Madrid), Spain*
[b] *Leeds School of Business, University of Colorado Boulder, USA*
[c] *Área de Estadística e Investigación Operativa, Universidad Rey Juan Carlos, Móstoles (Madrid), Spain*

Abstract

We tackle the problem of finding, for each network within a collection, the shortest path between two given nodes, while not exceeding the limits of a set of shared resources. We present an integer programming (IP) formulation of this problem and propose a parallelizable matheuristic consisting of three phases: 1) generation of feasible solutions, 2) combination of solutions, and 3) solution improvement. We show that the shortest paths found with our procedure correspond to the solution of some type of scheduling problems such as the Air Traffic Flow Management (ATFM) problem. Our computational results include finding optimal solutions to small and medium-size ATFM instances by applying Gurobi to the IP formulation. We use those solutions to assess the quality of the output produced by our proposed matheuristic. For the largest instances, which correspond to actual flight plans in ATFM, exact methods fail and we assess the quality of our solutions by means of Lagrangian bounds. Computational results suggest that the proposed procedure is an effective approach to the family of shortest path problems that we discuss here.

Keywords: Matheuristics, Shortest Path Problem, Shortest Path with Constrained Resources, Air Traffic Flow Management.

2010 MSC: 00-01, 99-00

1. Introduction

*This research was partially funded by projects MTM2015-63710-P and RTI2018-094269-B-I00 (A. Alonso-Ayuso and D. García-Heredia) from the Government of Spain

*Corresponding author.

Email addresses: dgheredi@est-econ.uc3m.es (David García-Heredia), emolina@est-econ.uc3m.es (Elisenda Molina), laguna@colorado.edu (Manuel Laguna), antonio.alonso@urjc.es (Antonio Alonso-Ayuso)

The Shortest Path Problem (SPP) is very well known in the Operations Research literature. The goal is to find the minimum-cost path connecting an origin node and a destination node in a network. The variety of applications is what makes this problem relevant. Applications range from computing the best route in GPS navigators and obtaining minimum risk routes for hazmat shipments (Carotenuto, Giordani & Ricciardelli, 2007; Holeczek, 2019) to routing protocols in IP or Wireless networks (Pióro, Szentesi, Harmatos, Jüttner, Gajowniczek & Kozdrowski, 2002; Yan, Zhou & Ding, 2016). Moreover, it often appears as a subproblem of other optimization problems, e.g., in Air Traffic Flow Management (García-Heredia, Alonso-Ayuso & Molina, 2019).

SPP is commonly solved with algorithms that have a polynomial-time complexity, such as Dijkstra's (Dijkstra, 1959) and Bellman-Ford (Bellman, 1958). In directed acyclic networks, SPP has a solution complexity that is linear with respect to the number of arcs (Bellman, 1957).

Some extensions of SPP include the k-shortest path problem (Yen, 1971; Eppstein, 1998), where the goal is to find the k paths of minimum cost between two nodes (e.g., for hazmat shipments), the multi-objective shortest path problem (Aneja & Nair, 1979; Guerriero & Musmanno, 2001; Raith & Ehrgott, 2009; Hrnčíř, Žilecký, Song & Jakob, 2016), where the goal is to find efficient (non-dominated) solutions according to more than one criterion (e.g., to balance time and fuel consumption), and the dynamic shortest path problem (Chabini, 1998; Ahuja, Orlin, Pallottino & Scutella, 2003; Thomas & White, 2007; Sever, Zhao, Dellaert, Demir, Van Woensel & De Kok, 2018), where properties such as the cost of each arc may change over time (e.g., shipping costs may vary throughout a day).

One of the most popular extensions of SPP is the Resource Constrained Shortest Path Problem (RCSPP), where the consumption of various resources is associated with each arc of the network. The objective is to find the Shortest Path (SP) connecting two nodes, while ensuring that available resource quantities are not exceeded (Handler & Zang, 1980; Beasley & Christofides, 1989). In addition to the straightforward application of finding the shortest path subject to time or fuel limits, RCSPP with one resource has been embedded as a pricing subproblem in column-generation procedures for vehicle routing problems (see Chabrier (2006) or Montoya, Guéret, Mendoza & Villegas (2016)).

RCSPP belongs to the $\mathcal{NP}-$complete class (Ahuja et al. (1993, Appx. B)) and it is usually solved by means of specialized algorithms (see, for instance, Dumitrescu & Boland (2003), Garcia

(2009), Lozano & Medaglia (2013) or Horváth & Kis (2016)). The basic idea behind these algorithms is to identify and prune dominated and infeasible paths in order to reduce the size of the problem to be able to solve it via dynamic programming or similar approaches.

In this work we address an extension of RCSPP that, as far as we know, has not appeared in the literature: the Shared Resource-Constrained Multi-Shortest Path Problem (SRMSPP). The problem consists of finding, for each network within a collection, a path between a given source (or origin) node and a given sink (or destination) node that minimizes the total cost (i.e., the sum of the costs in the arcs that are in the optimal path), while not exceeding a limit in the usage of a set of common resources shared by all networks. This extension is motivated by previous research that some of the authors carried out in Air Traffic Flow Management (ATFM) (García-Heredia et al., 2019).

ATFM problem (see a detailed description in (García-Heredia et al., 2019)) consists of determining the optimal routes for a set of flights. Each route includes the departure and arrival time, regions of the airspace (air sectors) to cross, and the time spent passing through each region (aircraft speed). The solution space is restricted by capacity values that limit the number of aircraft that can be in an air sector at the same time. Likewise, airports are limited in the number of departure and landing operations that they can handle. The collection of possible time-space routes for the flights assigned to an aircraft can be represented as a time-expanded network. A path represents an aircraft route. Without capacity constraints, the optimal route for each aircraft is given by the solution of SPP in its associated time-expanded network. However, as these capacity restrictions do exist, the problem to be solved becomes SRMSPP.

Our interest is to study SRMSPP by first formulating the problem as a mathematical programming model and then developing a heuristic solution approach. We also discuss how the problem can be applied to settings other than ATFM.

The motivation for the development of a heuristic solution method for SRMSPP is that exact mathematical programming solvers are not able to solve instances of the size found in practice and that, as we discuss in Section 2, it is not possible to simply apply the solution methods available in the literature that have been developed for the resource constrained shortest path problem. Our work falls within the area known as matheuristic optimization, which combines mathematical programming and heuristic search (Fischetti & Fischetti, 2018). Our heuristic approach uses some

metaheuristic principles but it cannot be classified as one of the well-known methodologies, such as tabu search (Glover & Laguna, 1998) or scatter search (Laguna & Martí, 2012).

Applications of SRMSPP include the aforementioned ATFM problem, the train rescheduling problem (Törnquist, 2006, 2012; Josyula, Törnquist Krasemann & Lundberg, 2018) or planning garbage collection (Jin, Qin, Zhang, Zhou & Wang, 2020). The problem is also linked to project scheduling (see Kolisch & Padman (2001), Kolisch & Hartmann (2006), Hartmann & Briskorn (2010), or Zheng & Wang (2015)) where all the activities are serial. Problems of this type include extensions such as several execution modes (Kuster, Jannach & Friedrich, 2009; Kellenbrink & Helber, 2015), time lag between activities (Klein & Scholl, 1999; Klein, 2000; Chassiakos & Sakellaropoulos, 2005), forbidden periods (Drexl, Nissen, Patterson & Salewski, 2000), multiple projects (Confessore, Giordani & 2007; Gonçalves, Mendes & Resende, 2008; Krüger & Scholl, 2009), project selection (Chen & Askin, 2009), flexible objective functions (Achuthan & Hardjawidjaja, 2001), and alternative sequences of activities to complete a project.

The main contributions of this work are: 1) The introduction of a shortest path formulation that can be employed to model some scheduling problems (Section 2), 2) A matheuristic search to solve the problem that is designed to take advantage of the modern computer architecture with multiple cores (Section 3), and 3) Testing and analysis of the proposed procedure to assess the contribution of its key elements (Section 4).

2. Problem Description and Mathematical Formulation

Before formally formulating SRMSPP, we introduce mathematical formulations for SPP and RCSPP. The notation employed in this section can be found in Table 1.

Table 1: Notation for the problem formulation.

Sets	
$\mathcal{N} = \{1, \ldots, \lvert\mathcal{N}\rvert\}$	set of network indices.
$\mathcal{V}_n$	set of vertices (or nodes) of the n-th network, $n \in \mathcal{N}$.
$\mathcal{A}_n$	set of arcs of the n-th network, $n \in \mathcal{N}$.
$\mathcal{C}_n$	set costs of the n-th network.
$\mathcal{G}_n = (\mathcal{V}_n, \mathcal{A}_n, \mathcal{C}_n)$	n-th network, $n \in \mathcal{N}$.
$\mathcal{G}_\mathcal{N} = \{\mathcal{G}_n\}_{n \in \mathcal{N}}$	set of networks.
$\Lambda_n^+(i), \Lambda_n^-(i)$	set of incoming and outgoings arcs, respectively, of node $i \in \mathcal{V}_n$, $n \in \mathcal{N}$.
$\mathcal{R}$	set of constrained resources.
Indices	
n	network index, $n \in \mathcal{N}$.
i, j	two nodes of a given network, $i, j \in \mathcal{V}_n$, $n \in \mathcal{N}$.
a	arc of a given network, $a \in \mathcal{A}_n$, $n \in \mathcal{N}$.
r	resource of the constrained set, $r \in \mathcal{R}$.
Parameters	
c_a	cost of using arc a, $\forall a = (i, j) \in \mathcal{A}_n$, $n \in \mathcal{N}$.
o_n, d_n	source and sink nodes, respectively, of the n-th network, $o_n, d_n \in \mathcal{V}_n$, $n \in \mathcal{N}$.
w_a^r	consumption of resource $r \in \mathcal{R}$ by arc $a \in \mathcal{A}_n$, $n \in \mathcal{N}$.
W^r	availability of resource $r \in \mathcal{R}$.
Variables	
x_a	binary variable equal to 1 if arc $a \in \mathcal{A}_n$, $n \in \mathcal{N}$ is part of the solution, and 0 otherwise.

2.1. Shortest Path Problem and Resource Constrained Shortest Path Problem

Since SRMSPP is a generalization of SPP and RCSPP, we first present mathematical programming formulations of these two problems. This is a formulation of SPP[1].

$$\min \sum_{a \in \mathcal{A}} c_a x_a, \tag{1}$$

[1]Note that as both problems, SPP and RCSPP, deal only with one network, the subscript n employed when defining notation in Table 1 is not needed.

subject to:

$$\sum_{a\in\Lambda^-(i)} x_a - \sum_{a\in\Lambda^+(i)} x_a = \begin{cases} 1, & \text{if } i = o, \\ -1, & \text{if } i = d, \\ 0, & \text{otherwise.} \end{cases} \quad \forall i \in \mathcal{V} \tag{2}$$

$$x_a \in \{0,1\}, \quad \forall a \in \mathcal{A}. \tag{3}$$

In this formulation, (1) establishes the objective of minimizing the cost of the arcs selected, while (2) are the classical flow conservation constraints which guarantee that the arcs in the solution form a path.

An IP formulation of RCSPP is obtained by adding to model (1)-(3) the following capacity (knapsack) constraints:

$$\sum_{a\in\mathcal{A}} w_a^r x_a \leq W^r, \quad \forall r \in \mathcal{R}. \tag{4}$$

2.2. Mathematical formulation of the Shared Resource-Constrained Multi-Shortest Path Problem

The goal of SRMSPP is to find, for each network in $\mathcal{G}_\mathcal{N}$, a path between a given source node o_n and a given sink node d_n, $o_n, d_n \in \mathcal{V}_n$, $n \in \mathcal{N}$, that minimizes the overall cost while not exceeding the capacity of a set of resources shared by all networks. The problem can be formulated as an integer program as follows:

$$\min \sum_{n\in\mathcal{N}} \sum_{a\in\mathcal{A}_n} c_a x_a, \tag{5}$$

subject to:

$$\sum_{a\in\Lambda_n^-(i)} x_a - \sum_{a\in\Lambda_n^+(i)} x_a = \begin{cases} 1, & \text{if } i = o_n, \\ -1, & \text{if } i = d_n, \\ 0, & \text{otherwise.} \end{cases} \quad \forall i \in \mathcal{V}_n, n \in \mathcal{N} \tag{6}$$

$$\sum_{n\in\mathcal{N}} \sum_{a\in\mathcal{A}_n} w_a^r x_a \leq W^r, \quad \forall r \in \mathcal{R}, \tag{7}$$

$$x_a \in \{0,1\}, \quad \forall a \in \mathcal{A}_n, n \in \mathcal{N}. \tag{8}$$

The objective function (5) minimizes the cost of the arcs in the solution. Equations (6) are the flow conservation constraints for each network, which force the arcs of each network to form a path from origin to destination. Constraints (7) enforce the resource limits. The model finishes with the integrality constraints (8).

2.3. The Shared Resource-Constrained Multi-Shortest Path Problem

We now highlight the main features of the shared resource-constrained multi-shortest path problem, while pointing out the differences with the resource constrained shortest path problem.

First of all notice that in contrast to RCSPP, where the arcs in a single path compete for a set of available resources, in SRMSPP, the arcs in paths from multiple networks must share the constrained set of resources (see constraints (7)).

In second place, the structure of the solutions is also different. In RCSPP, a solution (i.e., a single path) is infeasible when, for one or more resources, it consumes more than what is available. That is, the feasibility of a path depends only on the arcs in the path and it can be detected without any additional information (local conditions). By contrast, while the path for each single network in a solution to SRMSPP may be feasible (local conditions), the entire solution may not be feasible when considering the aggregated resource consumption of all paths (global conditions). In other words, instead of infeasible paths, SRMSPP deals with infeasible combinations of paths. This is why the methods for RCSPP, which are largely based on identifying and pruning dominated and infeasible paths, instead of combinations of paths, are not applicable to the problem addressed here.

To illustrate this point, consider ATFM problem described in the introduction. Under typical circumstances, all elements forming the set of shared resources (sectors and airports) have enough capacity to handle at least one aircraft. Therefore, no route by itself is ever infeasible (local conditions). This means that a route cannot be eliminated when considered in isolation. The feasibility of a route depends on other routes in a chosen set. Therefore, there are certain route combinations that are feasible and others that are infeasible.

SRMSPP formulation allows each arc, when used, to consume a specified amount of all of the available resources, i.e., $w_a^r > 0$,$\forall a \in \{\mathcal{A}_n\}_{n\in\mathcal{N}}, r \in \mathcal{R}$. However, in real applications (such as train rescheduling) arcs require only a subset of the resources. That is, for some resources $r \in \mathcal{R}$, $w_a^r = 0$.

As we mentioned in the introduction, SRMSPP allows the modeling of project scheduling problems for which the activities are serial and that simultaneously consider some extensions. We illustrate these extensions with ATFM problem. An aircraft in ATFM can be viewed as a project. Therefore, the multiple project extension corresponds to the problem with multiple aircraft. For each aircraft, the sequence of serial activities corresponds to the airports and sectors that the aircraft encounters in a flight. The route schedule is known as the flight plan. A flight plan specifies the departure time (which is restricted to a time window), the sectors to cross (different possibilities exist to reach a destination, which is equivalent to alternative sequences of activities to complete the project), the speed of the aircraft when traversing each air sector (execution modes), and the landing time (completion time of the project). Furthermore, each aircraft must cross each sector within a given time interval (forbidden time periods, in project scheduling). There is a setup time between flights that can be equated to the lag time extension in project scheduling. Several objective functions can be considered, such as minimizing the cost of the routes or minimizing late landings. To solve this type of scheduling projects as a SRMSPP, the graph of serial activities of each project has to be transformed into a time-expanded network, as illustrated in García-Heredia et al. (2019) for ATFM.

3. Solution Method for the Shared Resource-Constrained Multi-Shortest Path Problem

In this section, we propose a three-phase matheuristic algorithm to solve SRMSPP. We first give an overview of the procedure and then provide a detailed description of each phase. The notation for the algorithm can be found in Table 2 and 3.

In the first phase, a pool $\boldsymbol{\mathcal{X}} = \{\mathcal{X}^1, \ldots, \mathcal{X}^S\}$ of solutions is generated. A solution s in the pool is represented as $\mathcal{X}^s = \{\mathcal{X}_n^s\}_{n\in\mathcal{N}}$, where $\mathcal{X}_n^s \subseteq \mathcal{A}_n$ is a set of arcs defining a path from o_n to d_n in network $\mathcal{G}_n$. This phase attempts to generate solutions $\mathcal{X}^s \in \boldsymbol{\mathcal{X}}$ that are feasible for the original problem (5)-(8). However, as we discuss below, the resulting pool may include some solutions that violate one or more capacity constraints.

In the second phase, the solutions in $\boldsymbol{\mathcal{X}}$ are combined to obtain new (and perhaps better) feasible solutions, denoted by $\mathcal{X}^{\text{II}}$. This is done by solving the original IP model (5)-(8) only with the arcs in $\boldsymbol{\mathcal{X}}$. By construction, $\mathcal{X}^{\text{II}}$ cannot be worse than the best feasible solution in $\boldsymbol{\mathcal{X}}$.

The third phase applies a local search to $\mathcal{X}^{\text{II}}$. The local search attempts to close the gap between the cost of each path in $\mathcal{X}^{\text{II}}$ and its corresponding lower bound. The lower bound for each network can be found by solving the corresponding RCSPP in that network. For instances of SRMSPP for which all paths are feasible for an individual network, the solution of RCSPP is equivalent to the solution of SPP. Therefore, the lower bound for an individual network can be found by solving SPP instead of RCSPP. The goal of the local search is to find improved solutions, which tend to be those for which the gaps are balanced across all networks. Solution $\mathcal{X}^{\text{III}}$ denotes the outcome of the local search.

Algorithm 1 shows a pseudo-code of the proposed procedure.

Algorithm 1 Matheuristic

function MATHEURISTIC
    $\boldsymbol{\mathcal{X}} \leftarrow$ `GeneratePool`$(\mathcal{G}_{\mathcal{N}}, \mathcal{R}, maxIter, minIterIP, maxNets, penalty, \alpha, \beta, S)$;
    $\mathcal{X}^{\text{II}} \leftarrow$ `SolutionCombination`$(\mathcal{G}_{\mathcal{N}}, \mathcal{R}, \boldsymbol{\mathcal{X}})$;
    $\mathcal{X}^{\text{III}} \leftarrow$ `LocalSearch`$(\mathcal{G}_{\mathcal{N}}, \mathcal{R}, \mathcal{X}^{\text{II}}, \delta, \gamma)$;
    return $\mathcal{X}^{\text{III}}$;
end function

Table 2: Parameters for the algorithm.

Parameter	
$maxIter$	maximum number of failed attempts to generate a feasible solution.
$minIterIP$	minimum number of failed attempts between calls to the IP solver. The IP solver is used, as shown later, to help the heuristic find feasible solutions.
$maxNets$	maximum number of networks with penalized arcs in the previous iteration to consider using the IP solver.
$penalty$	value to penalize the usage of arcs contributing to solution infeasibility.
α	probability of penalizing a set of arcs that has been identified as contributing to the solution infeasibility.
β	percentage of networks, whose arcs have not been penalized in the previous iteration, that are fixed to their current paths when calling the IP solver.
S	number of solutions to generate in the first phase of the algorithm.
Δ_n	difference between the solution cost after the second phase and its lower bound, $n \in \mathcal{N}$.
δ	number of networks for which the algorithm tries to improve their solution in the third phase. The networks with larger Δ_n values are the ones to be improved.
γ	number of networks that the algorithm will use to trade resources with the δ networks above.

Table 3: Sets for the algorithm.

Set	
$\mathcal{X}_n^s \subseteq \mathcal{A}_n$	set of arcs defining a path from o_n to d_n in the n-th network for the s-th solution generated, $n \in \mathcal{N}$, $s \in \{1, \ldots, S\}$.
$\mathcal{X}^s = \{\mathcal{X}_n^s\}_{n \in \mathcal{N}}$	s-th solution generated, $s \in \{1, \ldots, S\}$.
$\boldsymbol{\mathcal{X}} = \{\mathcal{X}^1, \ldots, \mathcal{X}^S\}$	set of solutions generated in the first phase of the algorithm. For analogy with other metaheuristics such as Scatter Search, we will refer to $\boldsymbol{\mathcal{X}}$ as pool instead of set.
$\mathcal{X}^{II}$	solution generated in the second phase of the algorithm.
$\mathcal{X}^{III}$	solution generated in the third and last phase of the algorithm.
$\mathcal{X}^{LB}$	lower-bound solution to the problem.
$\mathcal{G}_n^* = (\mathcal{V}_n, \mathcal{A}_n, \mathcal{C}_n^*)$	n-th network of the problem with the set of costs different (due to the penalization process of the algorithm) than the original network $\mathcal{G}_n$, $n \in \mathcal{N}$.
$\mathcal{G}_{\mathcal{N}}^* = \{\mathcal{G}_n^*\}_{n \in \mathcal{N}}$	set of networks with modified costs in the problem. $\mathcal{G}_n^* = \mathcal{G}_n$ for those networks whose costs have not been modified.
$\mathcal{A}_n^r \subseteq \mathcal{A}_n$	subset of arcs in network $\mathcal{G}_n$, $n \in \mathcal{N}$ that use resource $r \in \mathcal{R}$. Note that an arc may belong to more than one $\mathcal{A}_n^r$.
$\mathcal{N}^* \subseteq \mathcal{N}$	network indices with at least one penalized arc in the previous iteration.
$\mathcal{G}_{\mathcal{N}}^{\beta}$, $\mathcal{N}^{\beta}$	networks and the corresponding indices as described for parameter β above.
$\mathcal{G}_{\mathcal{N}}^{\delta}$, $\mathcal{N}^{\delta}$	networks and the corresponding indices as described for parameter δ above.
$\mathcal{G}_{\mathcal{N}}^{\gamma}$, $\mathcal{N}^{\gamma}$	networks and the corresponding indices as described for parameter γ above.

3.1. Phase I: The `GeneratePool` *Function*

The goal of the first phase of our procedure is to generate a pool of feasible solutions $\boldsymbol{\mathcal{X}}$. The procedure (shown in Algorithm 2) starts with the solution of SPP[2] (i.e., model (1)–(3)) for each network (line 2). The total cost associated with the collection of all the shortest paths, $\mathcal{X}^{\text{LB}}$, is a lower bound for the original problem. If this collection of shortest paths meets all the capacity constraints, then $\mathcal{X}^{\text{LB}}$ is an optimal solution to the original problem and the procedure terminates (lines 3–5). Otherwise (i.e., at least one capacity constraint has been violated), a for-loop to

[2]We tackle SRMSPP instances for which the W^r values are such that no individual shortest path violates the capacity constraints, making RCSPP for each network equivalent to solving SPP. Since the networks in our computational testing are acyclic, we solve SPP using the Bellman principle of optimality (Bellman, 1957).

generate S feasible solutions is executed (lines 8–11). At each iteration of the loop, the `FeasibleSol` function attempts to generate a feasible solution using $\mathcal{X}^{\text{LB}}$ as a starting seed. This for-loop is amenable to parallel execution in the presence of multiple cores because the calls to `FeasibleSol` are independent. `FeasibleSol` is a non-deterministic iterative procedure that attempts to create a feasible solution $\mathcal{X}^s$ from a starting solution $\mathcal{X}^{\text{LB}}$ that does not meet the capacity constraints in the original model (5)-(8). Due to its non-deterministic nature, it is expected that `FeasibleSol` will generate a different solution every time is called.

Algorithm 2 Generating pool $\boldsymbol{\mathcal{X}}$

1: **function** GeneratePool($\mathcal{G}_{\mathcal{N}}, \mathcal{R}, maxIter, minIterIP, maxNets, penalty, \alpha, \beta, S$)
2: $\quad\mathcal{X}^{\text{LB}} \leftarrow$ `ShortestPath`($\mathcal{G}_{\mathcal{N}}$);
3: $\quad feasible \leftarrow$ `CheckFeasibility`($\mathcal{R}, \mathcal{X}^{\text{LB}}$);
4: $\quad$**if** $feasible ==$ `true` **then**
5: $\quad\quad\boldsymbol{\mathcal{X}} \leftarrow \{\mathcal{X}^{\text{LB}}\}$;
6: $\quad$**else**
7: $\quad\quad\boldsymbol{\mathcal{X}} \leftarrow \emptyset$;
8: $\quad\quad$**for** $s = 1, \ldots, S$ **do**
9: $\quad\quad\quad\mathcal{X}^s \leftarrow$ `FeasibleSol`($\mathcal{G}_{\mathcal{N}}, \mathcal{R}, \mathcal{X}^{\text{LB}}, maxIter, minIterIP, maxNets, penalty, \alpha, \beta$);
10: $\quad\quad\quad\boldsymbol{\mathcal{X}} \leftarrow \boldsymbol{\mathcal{X}} \cup \{\mathcal{X}^s\}$;
11: $\quad\quad$**end for**
12: $\quad$**end if**
13: $\quad$**return** $\boldsymbol{\mathcal{X}}$;
14: **end function**

Algorithm 3 shows the steps associated with the function that attempts to produce a feasible solution from the collection of shortest paths $\mathcal{X}^{\text{LB}}$. The procedure identifies, for the current solution, the arcs that are contributing to the infeasibility of the solution and adds a penalty to the cost of a random subset of these arcs. Then, SPP is solved for each penalized network (denoted as $\mathcal{G}_n^*$), producing shortest paths that exclude most or all of the penalized arcs. These steps are repeated in search for a feasible solution. If this fails, a final attempt to find a feasible solution is made by way of solving a reduced version of the original integer programming model. In this reduced version, some of the arcs are fixed and only a subset of the arcs is included as decision variables in the model. This step does not guarantee a feasible solution because the variable fixing may render the model infeasible. The search for a feasible solution ends once one is found or after a specified limit on the number of failed attempts. For reasons that will become clear below, adding

an infeasible solution to the pool is not an issue as long as the pool contains at least one feasible solution.

The `FeasibleSol` function takes as input the set of networks (i.e., $\mathcal{G}_\mathcal{N}$), the set of resources ($\mathcal{R}$), the collection of shortest paths associated with the lower bound (i.e., $\mathcal{X} = \mathcal{X}^{\text{LB}}$), the maximum number of failed attempts ($maxIter$), the number of failed attempts between calls to the IP solver for the reduced model ($minIterIP$), the maximum number of networks with penalized arcs in the previous iteration to consider using the IP solver ($maxNets$), the penalty value ($penalty$), the probability of penalizing a set of arcs that has been identified as contributing to the solution infeasibility (α), and the percentage of networks, whose arcs have not been penalized in the previous iteration, that are fixed to their current paths when solving the reduced IP model (β).

Lines 2–6 initialize the local elements of the `FeasibleSol` function. Let $\mathcal{A}_n^r \subseteq \mathcal{A}_n$ be the subset of arcs in network $\mathcal{G}_n$ that use resource r. We point out that an arc may belong to more than one $\mathcal{A}_n^r$. $\mathcal{A}_\mathcal{N}^\mathcal{R}$ contains the unpenalized subsets of arcs that use resource $r \in \mathcal{R}$. At the beginning, no arc subsets have been penalized and therefore all arcs subsets are included in $\mathcal{A}_\mathcal{N}^\mathcal{R}$. $\mathcal{G}_\mathcal{N}^* = \{(\mathcal{V}_n, \mathcal{A}_n, \mathcal{C}_n^*)\}_{n\in\mathcal{N}}$ consists of all the penalized networks and starts as a copy of $\mathcal{G}_\mathcal{N}$, indicating that at the beginning no arcs have been penalized (i.e., $\mathcal{C}_n^* = \mathcal{C}_n$). The $feasible$ Boolean variable keeps track of the feasibility of the solution obtained at the current iteration. The $nIter$ counter is the number of failed attempts to produce a feasible solution and $nIterIP$ counts the number of failed attempts since the last time the IP model was executed.

After the initialization, a while-loop (lines 7–27) that attempts to create a feasible solution out of the current $\mathcal{X}$ begins. In the first step of the loop, the `PenalizeArcs` function (see pseudo-code and detailed description in Appendix A) seeks to identify and penalize (using a non-deterministic procedure) subsets of arcs that are contributing to infeasibility in the current solution. For that, `PenalizeArcs` creates a list of resources for which their capacity is exceeded by the current solution. We will refer to this as the list of infeasible resources. Then, for each resource r^* in the list of infeasible resources, the procedure, with probability α, penalizes each unpenalized subset $\mathcal{A}_n^{r^*} \in \mathcal{A}_\mathcal{N}^\mathcal{R}$ if at least one arc in $\mathcal{A}_n^{r^*}$ is also in $\mathcal{X}$. The subset penalization consists of adding a $penalty$ value to the current cost of all arcs in $\mathcal{A}_n^{r^*}$. That is, the penalization process is cumulative.

`PenalizeArcs` returns the set of network indices with at least one penalized arc in the current iteration ($\mathcal{N}^* \subseteq \mathcal{N}$), the set of networks with penalized costs ($\mathcal{G}_\mathcal{N}^*$), and an updated $\mathcal{A}_\mathcal{N}^\mathcal{R}$ set in

Algorithm 3 Producing a feasible solution from $\mathcal{X}^{\text{LB}}$

function FEASIBLESOL($\mathcal{G}_{\mathcal{N}}, \mathcal{R}, \mathcal{X}, maxIter, minIterIP, maxNets, penalty, \alpha, \beta$)
  $\mathcal{A}^{\mathcal{R}}_{\mathcal{N}} \leftarrow \{\mathcal{A}^r_n \,|\, n \in \mathcal{N}, r \in \mathcal{R}\}$;
  $\mathcal{G}^*_{\mathcal{N}} \leftarrow \mathcal{G}_{\mathcal{N}}$;
  $feasible \leftarrow$ `false`;
  $nIter \leftarrow 0$;
  $nIterIP \leftarrow minIterIP$;
  while $feasible ==$ `false` & $nIter \leq maxIter$ **do**
    $\mathcal{N}^*$, $\mathcal{G}^*_{\mathcal{N}}$, $\mathcal{A}^{\mathcal{R}}_{\mathcal{N}} \leftarrow$ `PenalizeArcs`$(\mathcal{X}, \mathcal{G}^*_{\mathcal{N}}, \mathcal{R}, \mathcal{A}^{\mathcal{R}}_{\mathcal{N}}, penalty, \alpha)$;
    if $\mathcal{N}^* == \emptyset$ **then**
      $\mathcal{A}^{\mathcal{R}}_{\mathcal{N}} \leftarrow \{\mathcal{A}^r_n \,|\, n \in \mathcal{N}, r \in \mathcal{R}\}$;
    else
      if $|\mathcal{N}^*| \leq maxNets$ & $nIterIP \geq minIterIP$ **then**
        $\mathcal{X}^{trial} \leftarrow$ `SolveIPModel`$(\mathcal{R}, \mathcal{G}_{\mathcal{N}}, \mathcal{N}^*, \mathcal{X}, \beta)$;
        $feasible \leftarrow$ `CheckFeasibility`$(\mathcal{R}, \mathcal{X}^{trial})$;
        if $feasible == true$ **then**
          $\mathcal{X} \leftarrow \mathcal{X}^{trial}$;
        else
          $nIterIP \leftarrow 0$;
        end if
      else
        $\mathcal{X} \leftarrow$ `ShortestPath`$(\mathcal{G}^*_{\mathcal{N}})$;
        $feasible \leftarrow$ `CheckFeasibility`$(\mathcal{R}, \mathcal{X})$;
      end if
    end if
    $nIterIP \leftarrow nIterIP + 1$;
    $nIter \leftarrow nIter + 1$;
  end while
  return $\mathcal{X}$;
end function

which the penalized arc subsets have been removed. This is to avoid penalizing the same subsets $\mathcal{A}_s^r$ more than once and to foster diversity in the search. Note that, since an arc can consume more than one resource, removing a subset does not completely remove an arc. That is, an arc that has been penalized for consuming one resource may belong to an unpenalized subset of a different resource. Thus, an arc can be penalized multiple times, once per infeasible constraint to which it belongs. The algorithm allows penalizing an arc multiple times to further discourage the use of those arcs with the largest contribution to the infeasibility of the solution.

Since `PenalizeArcs` removes penalized arc subsets from $\mathcal{A}_{\mathcal{N}}^{\mathcal{R}}$ after each iteration, a point may be reached in which $\mathcal{A}_{\mathcal{N}}^{\mathcal{R}}$ is of a size that `PenalizeArcs` might return an empty $\mathcal{N}^*$ set. If this occurs, $\mathcal{A}_{\mathcal{N}}^{\mathcal{R}}$ is reset to include all arcs (line 10).

As long as $\mathcal{N}^*$ is not empty, an attempt to find a feasible solution is made. The attempt takes on two different forms. An exact method solves a reduced version of the integer programming model (lines 12–20) or new shortest paths are found for the networks in $\mathcal{G}^*$ (lines 21–22). The exact method is used when the number of penalized networks is small enough (i.e.: $|\mathcal{N}^*| \leq maxNets$) and the number of attempts to reach feasibility by recomputing the shortest paths reaches $minIterIP$. The exact method solves a reduced version of the original model (5)–(8) in which we fix a large percentage of the variables in the original problem. We start by selecting β% of the networks[3] in $\{\mathcal{G}_n\}_{n\in\mathcal{N}\setminus\mathcal{N}^*}$. The selection is made balancing solution quality (networks with the best objective function are selected) and diversity (networks are selected at random). We alternate the use of these two criteria. We denote the resulting subset of networks as $\mathcal{G}_{\mathcal{N}}^{\beta}$, indexed by $\mathcal{N}^{\beta}$. Then, variables in the set $\{x_a \,|\, a \in \{\mathcal{A}_n\}_{n\in\mathcal{N}^{\beta}}\}$ are fixed to 1 if $a \in \mathcal{X}$, and to 0 otherwise. This means that the paths for the networks in $\mathcal{G}_{\mathcal{N}}^{\beta}$ are fixed as dictated by the current solution. Therefore, the only variables in the reduced model are those associated with the arcs in $\{\mathcal{A}_n\}_{n\in\mathcal{N}\setminus\mathcal{N}^{\beta}}$. The reduced model uses the original cost values for the objective function calculation and a resource availability that is reduced by the resources requirements of the fixed variables.

After obtaining a new solution (by either of the methods described above), the procedure checks for feasibility (lines 14 and 22). If the solution is feasible, then the procedure ends and returns $\mathcal{X}$. If the solution is not feasible then the current solution changes only if the SPP method was used

[3]In our original algorithmic design, we fixed all paths in networks without penalized arcs, i.e., all networks in $\{\mathcal{G}_n\}_{n\in\mathcal{N}\setminus\mathcal{N}^*}$. However, this proved to be too restrictive, frequently making the reduced model infeasible. The β parameter allows us to include some additional networks in the formulation and increase the flexibility of the model.

to find it. The current solution is not changed when the IP model is not able to find a feasible solution. The process ends after $maxIter$ failed attempts and it returns the current infeasible solution. If the solution pool does not include any feasible solutions, then `SolutionCombination` might not return a feasible solution. The probability of observing this, however, decreases with the size of the solution pool. In fact, with the size that we used in our computational experiences, the procedure never encountered this situation.

Before describing additional elements of the proposed procedure, we would like to make a brief comment on the use of the IP model to solve the reduced problem. Our original design of the `FeasibleSol` function did not include this component. We added it after preliminary computational experiences showed that, for small $|\mathcal{N}^*|$ values, feasibility could be reached faster and with better solution quality by solving the reduced problem instead of continuing to penalize arcs and finding the revised shortest paths. In this design, the IP exact solver is meant to be invoked occasionally. That is, it is not meant to be the first option. Therefore, the values of the $maxNets$ and $minIterIP$ parameters must be chosen accordingly. The $maxNets$ parameter controls the size of the subproblem. Given that we are using an exact solver, we need to limit the size of the model that we are asking the solver to tackle. The $minIterIP$ parameter is a proxy for directly monitoring the changes in $\mathcal{N}^*$. Note that if the IP model fails to produce a feasible solution with a particular $\mathcal{N}^*$, it only makes sense to invoke it again after $\mathcal{N}^*$ has experienced some changes. Instead of keeping track of the changes in $\mathcal{N}^*$, we experimentally adjusted the value of $minIterIP$ to allow the arc penalization function to change the composition of $\mathcal{N}^*$.

3.2. Phase II: The `SolutionCombination` Function

As discussed at the beginning of Section 3, the `SolutionCombination` function combines the solutions in $\boldsymbol{\mathcal{X}}$ and produces a new solution $\mathcal{X}^{\text{II}}$, as long as at least one solution in the pool is feasible. The combination process consists of solving a reduced version of the original IP model (5)–(8), where the only variables are those associated with arcs in the pool of solutions. That is, the set of variables in the model is $\{x_a\}_{a\in\boldsymbol{\mathcal{X}}}$.

This form of combination of solutions has two key properties. First, the resulting solution $\mathcal{X}^{\text{II}}$ is at least as good as the best feasible in $\boldsymbol{\mathcal{X}}$. In our computational experiments, we observed that $\mathcal{X}^{\text{II}}$ was always better than any of the solutions in $\boldsymbol{\mathcal{X}}$. Second, $\mathcal{X}^{\text{II}}$ is guaranteed to be no worse than any solution found as a combination of the paths associated with the solutions in $\boldsymbol{\mathcal{X}}$. This

is due to the generation of $\mathcal{X}^{\text{II}}$ by a combination of arcs (instead of paths) that allows forks and joints to be produced at the nodes of the subnetwork induced by the arcs in $\boldsymbol{\mathcal{X}}$, leading to paths that are not in the pool of solutions. Our experiments with various designs led us to conclude that this combination method is superior to others in terms of the quality of the combined solution and the computational time to find it.

3.3. Phase III: The `LocalSearch` *Function*

In preliminary experimentation we observed that, for a given solution $\mathcal{X}^{\text{II}}$ resulting from the combination method, some networks used paths whose cost was much higher (relative to the known lower bound) than the cost associated with other networks. For each network $\mathcal{G}_n$, we calculate this difference as follows:

$$\Delta_n = \sum_{a \in \mathcal{A}_n} c_a (x_a^{\text{II}} - x_a^{\text{LB}}). \tag{9}$$

We concluded that the $\mathcal{X}^{\text{II}}$ solutions tend to be unbalanced, with a relatively small number of networks with much larger Δ_n values than others. We therefore developed the `LocalSearch` function (Algorithm 4) taking into account the structure of the $\mathcal{X}^{\text{II}}$ solutions. In particular, `LocalSearch` focuses on improving the paths in networks with relatively large Δ_n values. This is done at the possible expense of worsening the delta values of other networks. The search, in other words, is for a balanced solution. That is, one for which the collection of Δ_n values for all networks have less variance.

The local search uses two parameters, δ and γ to operate on $\mathcal{X}^{\text{II}}$. The value of δ is the number of networks being improved. The value of γ is number of networks that the algorithm will use to trade resources with those networks being improved.

The procedure starts by identifying the δ networks with the largest Δ_n values (lines 2–5). In Algorithm 4, $\Delta[\delta]$ is the δ-th element in the descending-ordered set Δ. Set $\mathcal{N}^{\delta}$ contains the indices of the networks for which the local search is trying to improve their Δ_n values. We denote the subset of networks indexed by $\mathcal{N}^{\delta}$ as $\mathcal{G}^{\delta}_{\mathcal{N}}$. The procedure then selects, from all the networks not in $\mathcal{G}^{\delta}_{\mathcal{N}}$, using a first-match rule, γ networks, each of them sharing at least one resource with one or more networks in $\mathcal{G}^{\delta}_{\mathcal{N}}$ (lines 6–14). We denote the resulting subset of networks as $\mathcal{G}^{\gamma}_{\mathcal{N}}$, indexed by $\mathcal{N}^{\gamma}$. The networks in $\mathcal{G}^{\gamma}_{\mathcal{N}}$ are used as "partners" for the networks in $\mathcal{G}^{\delta}_{\mathcal{N}}$ in order to trade off

Algorithm 4 Improving solution $\mathcal{X}^{\mathrm{II}}$

function LocalSearch($\mathcal{G}_{\mathcal{N}}, \mathcal{R}, \mathcal{X}^{\mathrm{II}}, \delta, \gamma$)
  $\Delta_n \leftarrow \sum_{a \in \mathcal{A}_n} c_a(x_a^{\mathrm{II}} - x_a^{\mathrm{LB}})\ \forall n \in \mathcal{N}$;
  $\Delta \leftarrow \{\Delta_n\}_{n \in \mathcal{N}}$;
  $\Delta \leftarrow$ `SortDescending`(Δ);
  $\mathcal{N}^{\delta} \leftarrow \{n \in \mathcal{N} \,|\, \Delta_n \geq \Delta[\delta]\}$;
  $\mathcal{N}^{\gamma} \leftarrow \emptyset$;
  for $n \in \mathcal{N} \setminus \mathcal{N}^{\delta}$ **do**
    if $\exists\, r \in \mathcal{R}, a \in \mathcal{A}_n, a' \in \{\mathcal{A}_{n'}\}_{n' \in \mathcal{N}^{\delta}} : w_a^r > 0\ \&\ w_{a'}^r > 0$ **then**
      $\mathcal{N}^{\gamma} \leftarrow \mathcal{N}^{\gamma} \cup \{n\}$;
    end if
    if $|\mathcal{N}^{\gamma}| == \gamma$ **then**
      break;
    end if
  end for
  $\mathcal{X}^{\mathrm{III}} \leftarrow$ `SolveIPModel`($\mathcal{R}, \mathcal{G}_{\mathcal{N}}, \mathcal{N}^{\delta} \cup \mathcal{N}^{\gamma}, \mathcal{X}^{\mathrm{II}}$);
  return $\mathcal{X}^{\mathrm{III}}$;
end function

the use of resources. The values of the parameters associated with the local search are such that $\delta \ll \gamma$.

We define $\mathcal{N}^{\mathrm{LS}} = \mathcal{N}^{\delta} \cup \mathcal{N}^{\gamma}$ and solve the original model (5)–(8) by fixing the paths in $\mathcal{N} \setminus \mathcal{N}^{\mathrm{LS}}$. That is, variables in the set $\{x_a \,|\, a \in \{\mathcal{A}_n\}_{n \in \mathcal{N} \setminus \mathcal{N}^{\mathrm{LS}}}\}$ are fixed to 1 if $a \in \mathcal{X}^{\mathrm{II}}$, and to 0 otherwise; and the remaining variables (i.e., $\{x_a \,|\, a \in \{\mathcal{A}_n\}_{n \in \mathcal{N}^{\mathrm{LS}}}\}$) are the only ones in the IP model. The solution of the IP model is denoted by $\mathcal{X}^{\mathrm{III}}$. This solution is guaranteed to be no worse than $\mathcal{X}^{\mathrm{II}}$. We have observed that the local search, as defined above, is often able to improve upon the solution constructed by the combination method, except in those cases when $\mathcal{X}^{\mathrm{II}}$ is near-optimal. We have experimented with a local search that focuses only on the reduced set of networks with the worst Δ_n values (i.e., the networks in $\mathcal{G}_{\mathcal{N}}^{\delta}$) and determined that this strategy provides very little room for improvement because most of the resources are committed to the paths that are fixed prior to solving the IP model.

4. Computational Experience

We test our procedure on a set of instances of an Air Traffic Flow Management (ATFM) problem, which represents a context where SRMSPP arises in practice.

In this approach, each aircraft (with one or multiple continued flights) corresponds to a network where the potential modifications to the flight plan (e.g., departure delays or speed changes) are the arcs (García-Heredia et al. (2019)). Air sectors and airports are the set of resources with limited capacity. For ATFM, SRMSPP has weights w_a^r equal to 1 in constraints (7) because the capacity W^r is given as number of aircraft.

4.1. Problem Instances

We use publicly available flight plans for our computational experience (Bureau of Transportation Statistics (a,b)). The data correspond to domestic flights in the US for the 16th of January, May, and September of 2019. The choice of the 16th is due to the high-volume of air traffic on that day for each of these months.

With the values in the data sets and customary procedures in ATFM literature (see Bertsimas, Lulli & Odoni (2011) or Agustín, Alonso-Ayuso, Escudero & Pizarro (2012)), we generated all the data required for the problem, including capacity limits.

In order to build a test set with various problem sizes, we created smaller versions of the original flight plans with 30% and 65% of the total number of aircraft. The dimensions of the instances in our test set are shown in Table 4. Note that each arc corresponds to a variable in the original IP model (5)-(8), each node to a flow constraint, and each resource to a capacity constraint. The large number of capacity constraints is due to the product of the number of capacity elements and periods in the planning horizon. Since most of the constraints in the problem define facets (flow constraints), a strong LP relaxation is expected when attempting to solve the problem using exact methods. Our computational experiments corroborate this important characteristic of the model, which others have pointed out as related to time-expanded network IP formulations (Boland & Savelsbergh, 2019).

For each case in the table, we generated twelve scenarios based on different capacity levels to test robustness of the proposed solution method. Thus, a total of 108 instances form our test set.

The twelve scenarios are grouped in three categories (easy, medium, and difficult). To generate them, we first created a base scenario of capacity values which is based on the minimum value

Table 4: Dimensions of the instances in our test set.

Flight plan	Size	#Flights	#Networks	#Arcs	#Nodes	#Resources
Jan	30%	5,596	1,329	3,116,931	1,482,732	136,887
May	30%	6,951	1,368	4,043,046	1,868,888	145,711
Sep	30%	6,610	1,368	3,6706,44	1,729,221	142,734
Jan	65%	11,788	2,879	7,195,904	3,336,779	173,249
May	65%	13,752	2,963	9,717,333	4,341,805	178,451
Sep	65%	13,530	2,964	8,597,363	3,922,250	185,519
Jan	100%	18,100	4,429	11,934,419	5,446,911	184,404
May	100%	20,634	4,558	14,475,487	6,457,815	204,724
Sep	100%	20,581	4,559	13,993,972	6,248,302	201,451

required for the original flight plan to be feasible.

Easy cases simulate the effect of bad weather moving across sectors and causing capacity reductions (Bertsimas et al., 2011). The reductions were set at 10%, 20%, 30% and 40% of the base capacity, leading to a total of four scenarios. Medium cases are based on the easy ones with an additional capacity reduction randomly applied to 50% of the elements and enforced during the entire planning horizon. The additional reduction of the capacity of each element is randomly chosen between 1% and 20%. For the difficult cases, the randomly generated capacity reduction is applied to all elements.

4.2. Parameter Setting

We perform all experiments on an HP Z230 Tower Workstation (processor i7-4770 3.40GHz) and 32 GB of RAM. Our MIP solver is Gurobi 9.0 (Gurobi Optimization (2020)). The algorithm was coded in C++, compiled using the GNU compiler 6.3, and run on a Debian 9 Operating System. The for-loop to generate the pool of solutions (Algorithm 2) was parallelized using OpenMP (OpenMP Architecture Review Board (2015)) with 8 threads. The code and data sets can be found in `https://github.com/DavidGarHeredia/SRC-MSPP`.

To take into account the random elements in the proposed procedure, we solved each problem instance five times. We represent the associated variability with violin plots, a well-known extension of the boxplot that shows the distribution of the data. In violin plots, the quantiles corresponding

to 25%, 50% (median), and 75% are depicted with a solid line, while a dark diamond shows the mean.

We used ParamILS, an automated system for parameter setting and algorithm configuration, (Hutter, Hoos & Stützle, 2007) to set the values for the parameters of our search procedure, except for the value of S, the size of the solution pool. (Below, we discuss how we set this value.) The parameter values that we used for our experimentation are shown in Table 5. The penalty value is about 4 times the maximum cost in the networks.

Table 5: Parameters values in experimentation.

$maxIter$	$minIterIP$	$maxNets$	$penalty$	α	β	δ	γ
100	20	$5\%\|\mathcal{N}\|$	6000	50%	95%	2%	30%

In addition, we set the MIP gap in Gurobi at various levels depending on the purpose for calling this optimizer: solving full model (0.5%), generating the pool of solutions (1%), combining the solutions (1%), and applying local search (0.5%).

To determine the pool size (parameter S), we apply the procedure to the instances of size 30% using four different pool size values (16, 32, 48, and 64). We compared the results obtained with those from Gurobi. We observed that larger pool sizes achieve smaller optimality gaps with lower variability. For instance, for a pool size of 64, the optimality gap is always less than 5%. For the easy cases, solution times exceeded those of Gurobi. The procedure outperformed Gurobi in the medium and difficult cases.

Extrapolating from the experiments with instances of size 30% and in order to balance solution quality and computational effort, we chose a pool of 80 solutions for instances of size 65% and a pool of 96 solutions for instances with the complete flight plans.

4.3. Integer Programming Results

We attempted to solve the IP model for the 108 instances in our test set with Gurobi. However, 44 of the runs terminated in an out-of-memory error and without an integer solution. These instances correspond to the full flight plans and for the 65% reduced flight plans for the May and September days under the difficult scenario. Table 6 summarizes the results of the Gurobi runs.

The first three columns of the table show the date of the flight plan, the percentage of flights

Table 6: Summary of Gurobi results.

Flight plan	Size	Difficulty	$\bar{t}$	t_Δ	$root_\%$	$\overline{Gap}$	Gap_Δ	$col_\%$	$row_\%$
Jan	30%	easy	126.84	1.12	75%	0.08%	[0, 0.34]%	17.81%	38.34%
Jan	30%	medium	238.34	1.87	0%	0.11%	[0.05, 0.15]%	17.94%	38.39%
Jan	30%	difficult	337.61	1.19	0%	0.34%	[0.27, 0.4]%	18.12%	38.62%
May	30%	easy	172.56	1.15	100%	0%	[0, 0]%	16.32%	35.77%
May	30%	medium	253.61	1.68	75%	0.02%	[0, 0.06]%	16.39%	35.80%
May	30%	difficult	722.91	2.33	0%	0.33%	[0.18, 0.44]%	16.50%	35.89%
Sep	30%	easy	155.68	1.14	100%	0%	[0, 0]%	17.38%	37.69%
Sep	30%	medium	163.59	1.16	100%	0%	[0, 0]%	17.40%	37.65%
Sep	30%	difficult	505.21	1.41	0%	0.44%	[0.35, 0.52]%	17.51%	37.70%
Jan	65%	easy	335.97	1.38	75%	0.04%	[0, 0.15]%	16.61%	36.13%
Jan	65%	medium	856.33	1.79	25%	0.14%	[0, 0.24]%	16.68%	36.18%
Jan	65%	difficult	2,117.18	1.44	0%	0.29%	[0.25, 0.39]%	16.75%	36.28%
May	65%	easy	523.56	1.72	75%	0.05%	[0, 0.21]%	14.87%	33.35%
May	65%	medium	1,271.99	1.89	25%	0.17%	[0, 0.37]%	14.91%	33.39%
Sep	65%	easy	374.24	1.05	100%	0%	[0, 0]%	15.73%	34.74%
Sep	65%	medium	862.39	1.30	0%	0.31%	[0.29, 0.38]%	15.78%	34.76%

chosen from the flight plan (Size), and the level of difficulty. This is followed by the average solution time in seconds ($\bar{t}$) and the $\frac{t_{\max}}{t_{\min}}$ ratio (t_Δ), where $t_{\max}$ and $t_{\min}$ are the worst and best solution times. t_Δ measures solution time variability, where values close to one indicate less sensitivity to the different weather scenarios. $root_\%$ denotes the percentage of instances solved at the root node of the Branch & Bound (B&B) tree. In all other instances, only an additional node had to be explored. $\overline{Gap}$ and Gap_Δ are the average and the range of the integrality gap, defined as $100\frac{z^{\text{IP}}-z^{\text{LP}}}{z^{\text{LP}}}$, where z^{LP} and z^{IP} are the objective function value of the LP relaxation and the integer solution, respectively. These values along with $root_\%$ measure the strength of the IP formulation. $col_\%$ and $row_\%$ represent the percentage of columns and rows that Gurobi deleted in the preprocessing phase.

The following observations relate to the values in Table 6:

1. Gurobi's solution times are within a range that is acceptable in practice for ATFM problem (35 minutes in the worst case).
2. The value of t_Δ is greater than or equal to 1.3 in 10 out of 16 cases, indicating that for a given difficulty level, weather scenarios have a significant impact (i.e., 30% difference) in the

solution time.

3. As customary in B&B, solution times exhibit nonlinear increases with the problem size.
4. $root_{\%}$, $\overline{Gap}$ and Gap_{Δ} show that the formulation has a strong LP relaxation, with 18 out of 64 instances solved at the root node and an integrality gap of less than 0.45% after one node.
5. The last two columns show that Gurobi achieved a significant reduction of the original size of the problem.

ATFM problems have a very specific cost structure, as described in (García-Heredia et al., 2019). To test the sensitivity of Gurobi to the cost structure in the IP formulation, we generated costs from a continuous uniform distribution $(0, 100)$ for the arcs in the networks in our problem test set. This new cost structure turned out to increase the difficulty of the problems in such a way that, out of the 108 instances, Gurobi was able to solve only 24. All of these instances belong to the sets of size 30%.

4.4. Performance Assessment

We used the smallest problem instances to find the best values for our search parameters as well as adjusting the size of the solution pool. We now assess the performance of the procedure with the larger problem instances. For the cases that we would also solve with Gurobi, we are able to calculate optimality gaps and speed-up values when applying our procedure to these problem instances. The speed up is computed as wall time required by our procedure divided by the wall time required by Gurobi. The results are summarized in Figure 1 (a) and (b), where violin plots show, for each difficulty level (X axis), the distribution of the optimalitiy gap and the speed-up (Y axis) with respect to Gurobi's solutions.

The first figure shows that the proposed heuristic generates high-quality solutions for these instances. For the easy cases, the heuristic solutions are, on average, less than 3% away from optimality. For the medium cases, in more than 75% of the times the optimality gap is below 9% and the average is less than 7.5%. For the difficult cases, the optimality gap is below 10%, with an average of 8.5%. In terms of computational effort (second figure), our speed-up calculation shows that the heuristic was faster (values above the dashed line) than the exact method in the medium and difficult cases, but not in the easy ones. It is interesting to point out that in the instances when the proposed heuristic is faster than the exact method, Gurobi is still solving the LP relaxation at the root node when the heuristic has already finished.

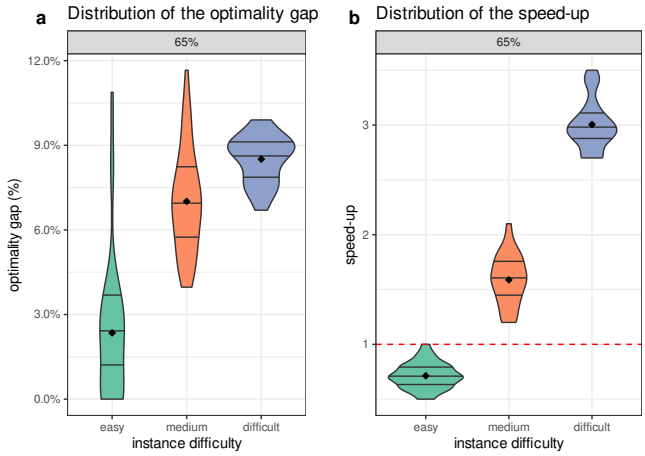

Figure 1: Results of the algorithm for instances of size 65% that exact methods could also solve.

A possible conclusion from these results is that Gurobi should be the preferred method to solve the easy problems, since it can confirm optimality before our procedure finishes. However, this result depends on the cost structure. Recall that when we used randomly generated costs, Gurobi was only able to solve 24 out of the 108 test problems. For those cases, the solutions achieved by our procedure are near-optimal (optimality gaps below 1%), and they are reached faster (1.5 to 8 times faster than Gurobi). This experiment shows that the performance of the exact method within Gurobi is sensitive to the cost structure. On the other hand, our heuristic-based approach exhibits a robust performance across cost structures.

For the cases that, due to memory limitations, Gurobi[4] was unable to even solve the LP relaxation of the problem, we obtained lower bounds dualizing the capacity constraints of the original problem (5)-(8) and solving the corresponding Lagrangian Relaxation. Since the non-dualized constraints (flow constraints) define facets for the problem, the optimum of the Lagrangian dual problem is guaranteed to be equal to the LP relaxation of problem (5)-(8) (Guignard (2003)). Based on the gaps reported in Table 6, the lower bounds resulting from the Lagrangian Relaxation are expected to be tight.

Figure 2 shows, for each instance (X-axis) and size group, the gap between all the solutions found throughout our experimentation and the Lagrangian bound (Y-axis). The dashed line is the mean gap of the heuristic solution with respect to the Lagrangian bound. The band around the

[4]We also attempted to find solutions for these instances with LocalSolver (LocalSolver (2020)), a general-purpose optimizer based on metaheuristic principles and methodologies, but this software also failed to handle them.

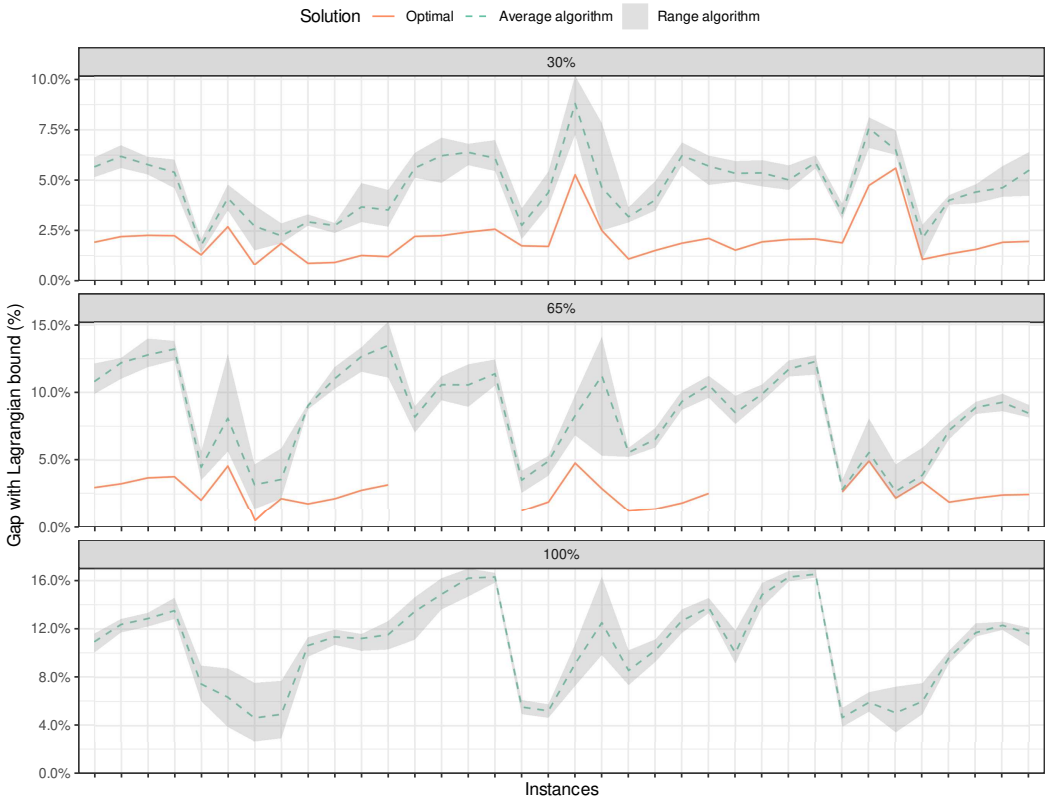

Figure 2: Deviation from Lagrangian lower bounds.

mean, represents the range of gaps obtained from the 5 runs of our procedure. Similarly, the solid line in the figure shows the gap of Gurobi's solutions (when available) with the Lagrangian bound.

The general observation from Figure 2 is that the deviation from the Lagrangian bounds increases with the size of the problem. This is true for both the heuristic and the optimal solutions. For the full flight plans, our procedure obtained, in the worst case, solutions with average gaps of 16%. However, in most cases the gap varied between 4% and 14.5%. Considering that these gaps are against a lower bound, these results are more than reasonable for the practical application that we studied. Note that some of the optimal solutions have gaps of up to 5% with respect to the lower bound. In terms of computational time, our procedure found the solutions to most of the full-size problems in less than 25 minutes. In the worst case, the procedure took 40 minutes. These computational times are within the valid range for ATFM.

4.5. Analysis of the Search Procedure

The preceding results show that we are able to produce high-quality solutions in reasonable computational times. Our procedure was able to find solutions to problems that a commercial solver could not handle. The purpose of the following analysis is to provide insights into the performance of our procedure.

In terms of computational effort, we observed that our parallelization scheme for generating the pool of solutions scaled linearly. That is, twice as much time is required to generate the same amount of solutions with half the threads. Furthermore, we observed that: 1) the solution pool generation represents the bottleneck, consuming from 52% to 79% of the solution time; 2) combining solutions consumes a negligible amount of time (2.5% in the worst case); and 3) the time required by local search is sensitive to the difficulty of the problem, ranging from 21% to 46% for the biggest and most difficult instances.

The good news of the solution pool generation being the procedure's bottleneck is that, as mentioned, the parallelization of this phase linearly scales with the number of threads. This means that when generating a pool of 96 solutions, if instead of 8 threads we had 48, the time in this phase would have been $\frac{1}{6}$ of the time obtained. In such a situation, the procedure could be set up to generate more solutions, which in turn could improve the quality of the results.

Figure 3 (a) and (b) shows the percentage of improvement (Y-axis) in solution quality achieved after each phase of the procedure, as a function of the problem size (X-axis) and level of difficulty. Figure 3 (a) reveals the large improvement achieved by the solution combination phase, particularly as the difficulty of the problem increases. The large solution-quality improvement combined with the modest amount of computational time make this phase one of the key elements for the success of the solution method.

When compared to the solution combination phase, the local search achieves a significantly smaller improvement in solution quality. This is mainly due to the reduced opportunities for improvement after solutions are combined. Nonetheless, the improvement achieved by the local search justifies its computational requirements and therefore its inclusion as part of the solution process.

We also studied how much our algorithm could reduce the optimality gaps reported in the previous subsection by simply using larger pool sizes. For that, we employed pools of 200 solutions

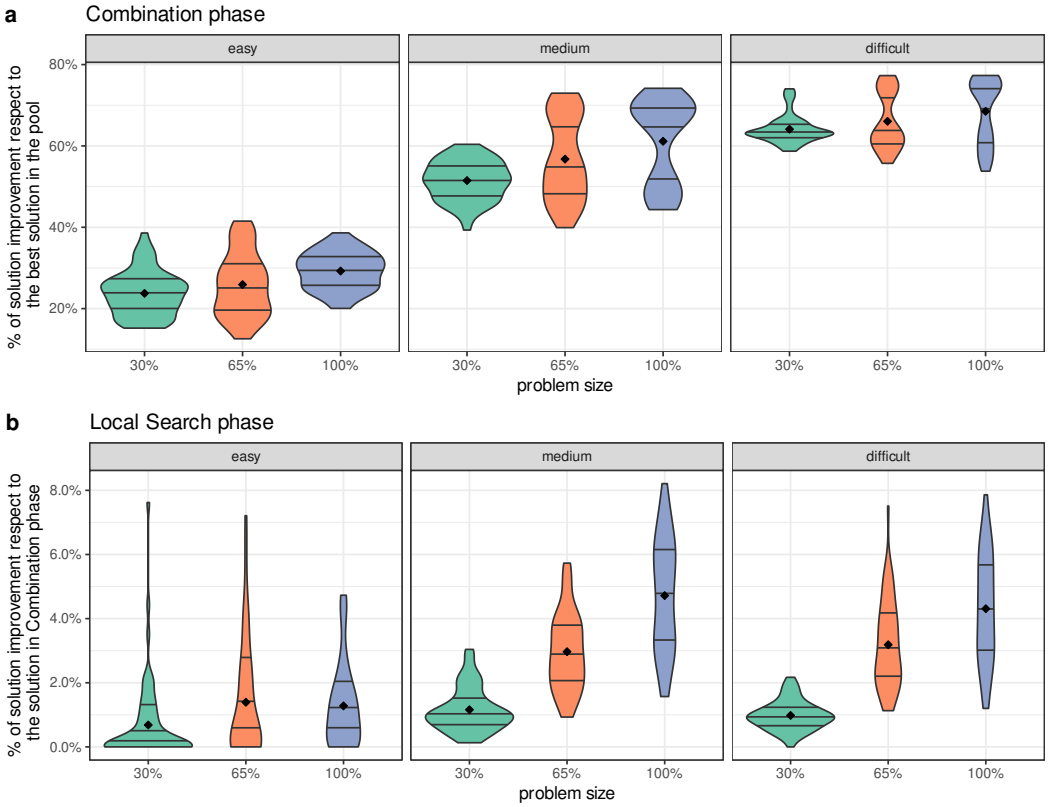

Figure 3: Solution improvement achieved at each phase of the proposed procedure.

for the instances of size 30%, and 300 for instances of size 65%. We verified that in the worst case the optimality gap was below 6.5%, being only larger than 2% in 16 of the 64 instances. Moreover, except for 7 instances, less than 30 minutes were required to complete the search. These times are valid for ATFM application, but larger than the times required by Gurobi. Thus, larger pool sizes result in near-optimal solutions, but large pool sizes are only recommended when more than 8 threads are available.

To conclude the analysis of the proposed procedure, we point out that we designed the algorithmic elements to take advantage of the structure of the problem. In particular, the generation of the solution pool exploits a very specific characteristic of our problem, which turns out to also be true in the context of the resource constrained shortest path problem. In general, the least expensive arcs are also those that consume the most limited resources at a higher rate. The key is to generate a set of diverse solutions that include the use of arcs with limited or nonexistent consumption of the most limited resources. Typically, these arcs are the most costly or else the

trivial solution in which all origins and destinations are connected by their shortest paths would be feasible and therefore optimal. The controlled random element included in the generation of solutions results in a diversity of arcs in the solution pool that is then exploited by the combination method.

5. Conclusions and Future Research

In this work we presented a new problem, the Shared Resource-Constrained Multi-Shortest Path Problem (SRMSPP), and a matheuristic algorithm for obtaining good feasible solutions for it. Applications of SRMSPP include the Air Traffic Flow Management (ATFM) problem, a problem that has been used to test the algorithm, or the train rescheduling problem. These problems can actually be considered multi-project scheduling problems where the activities of the projects are serial, and multiple extensions (e.g., different execution modes) are considered at the same time. Insights on how and why the algorithm works have been exposed along the computational experience. Among the conclusions drawn for the algorithm, we emphasize the following: 1) It is parallel-computing oriented, with a design that allows to efficiently take advantage of all the cores available in a computer; 2) It achieves good-quality solutions in short periods of time for large instances; 3) It can solve bigger and more difficult instances than exact methods; and 4) It is robust across changes problem size, difficulty, and cost structure.

Future research directions include: 1) Exploring a local search phase able to achieve larger improvements in shorter time; 2) Studying new algorithms with which to compare the one proposed in this paper (e.g., trying to adapt the ideas from Tian, Ren & Zhou (2016)), 3) Extending the Shortest Path Problem to the Generalized Minimum Cost Flow Problem to model more complex scheduling situations; and 4) Addressing the stochastic version of the problem. A stochastic approach to the problem would be particularly relevant in the context of ATFM.

References

Achuthan, N., & Hardjawidjaja, A. (2001). Project scheduling under time dependent costs–a branch and bound algorithm. *Annals of Operations Research*, *108*, 55–74.

Agustín, A., Alonso-Ayuso, A., Escudero, L. F., & Pizarro, C. (2012). On air traffic flow management with rerouting. part I: Deterministic case. *European Journal of Operational Research*, *219*, 156–166.

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows: theory, algorithms, and applications*. Upper Saddle River (New Jersey): Prentice Hall.

Ahuja, R. K., Orlin, J. B., Pallottino, S., & Scutella, M. G. (2003). Dynamic shortest paths minimizing travel times and costs. *Networks: An International Journal*, *41*, 197–205.

Aneja, Y. P., & Nair, K. P. (1979). Bicriteria transportation problem. *Management Science*, *25*, 73–78.

Beasley, J. E., & Christofides, N. (1989). An algorithm for the resource constrained shortest path problem. *Networks*, *19*, 379–394.

Bellman, R. (1957). *Dynamic Programming*. (1st ed.). Princeton, NJ, USA: Princeton University Press.

Bellman, R. (1958). On a routing problem. *Quarterly of applied mathematics*, *16*, 87–90.

Bertsimas, D., Lulli, G., & Odoni, A. (2011). An integer optimization approach to large-scale air traffic flow management. *Operations research*, *59*, 211–227.

Boland, N. L., & Savelsbergh, M. W. (2019). Perspectives on integer programming for time-dependent models. *Top*, *27*, 147–173.

Carotenuto, P., Giordani, S., & Ricciardelli, S. (2007). Finding minimum and equitable risk routes for hazmat shipments. *Computers & Operations Research*, *34*, 1304–1327.

Chabini, I. (1998). Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. *Transportation research record*, *1645*, 170–175.

Chabrier, A. (2006). Vehicle routing problem with elementary shortest path based column generation. *Computers & Operations Research*, *33*, 2972–2990.

Chassiakos, A. P., & Sakellaropoulos, S. P. (2005). Time-cost optimization of construction projects with generalized activity constraints. *Journal of Construction Engineering and Management*, *131*, 1115–1124.

Chen, J., & Askin, R. G. (2009). Project selection, scheduling and resource allocation with time dependent returns. *European Journal of Operational Research*, *193*, 23–34.

Confessore, G., Giordani, S., & Rismondo, S. (2007). A market-based multi-agent system model for decentralized multi-project scheduling. *Annals of Operations Research*, *150*, 115–135.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*, 269–271.

Drexl, A., Nissen, R., Patterson, J. H., & Salewski, F. (2000). Progen/πx–an instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions. *European Journal of Operational Research*, *125*, 59–72.

Dumitrescu, I., & Boland, N. (2003). Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks: An International Journal*, *42*, 135–153.

Eppstein, D. (1998). Finding the k shortest paths. *SIAM Journal on computing*, *28*, 652–673.

Fischetti, M., & Fischetti, M. (2018). Matheuristics. In R. Martí, P. M. Pardalos, & M. G. C. Resende (Eds.), *Handbook of Heuristics* (pp. 121–153). Cham: Springer International Publishing. URL: https://doi.org/10.1007/978-3-319-07124-4_14. doi:10.1007/978-3-319-07124-4_14.

Garcia, R. (2009). *Resource constrained shortest paths and extensions*. Ph.D. thesis Georgia Institute of Technology.

García-Heredia, D., Alonso-Ayuso, A., & Molina, E. (2019). A combinatorial model to optimize air traffic flow management problems. *Computers & Operations Research*, *112*, 104768.

Glover, F. W., & Laguna, M. (1998). *Tabu Search*. Springer Science & Business Media.

Gonçalves, J. F., Mendes, J. J., & Resende, M. G. (2008). A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research*, *189*, 1171–1190.

Guerriero, F., & Musmanno, R. (2001). Label correcting methods to solve multicriteria shortest path problems. *Journal of optimization theory and applications*, *111*, 589–613.

Guignard, M. (2003). Lagrangean relaxation. *Top*, *11*, 151–200.

Gurobi Optimization, L. (2020). Gurobi optimizer reference manual. URL: http://www.gurobi.com accessed on December 17, 2023.

Handler, G. Y., & Zang, I. (1980). A dual algorithm for the constrained shortest path problem. *Networks*, *10*, 293–309.

Hartmann, S., & Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research*, *207*, 1–14.

Holeczek, N. (2019). Hazardous materials truck transportation problems: A classification and state of the art literature review. *Transportation Research Part D: Transport and Environment*, *69*, 305 – 328. URL: http://www.sciencedirect.com/science/article/pii/S1361920918311362. doi:https://doi.org/10.1016/j.trd.2019.02.010.

Horváth, M., & Kis, T. (2016). Solving resource constrained shortest path problems with lp-based methods. *Computers & Operations Research*, *73*, 150–164.

Hrnčíř, J., Žileckỳ, P., Song, Q., & Jakob, M. (2016). Practical multicriteria urban bicycle routing. *IEEE Transactions on Intelligent Transportation Systems*, *18*, 493–504.

Hutter, F., Hoos, H. H., & Stützle, T. (2007). Automatic algorithm configuration based on local search. In *Aaai* (pp. 1152–1157). volume 7.

Jin, X., Qin, H., Zhang, Z., Zhou, M., & Wang, J. (2020). Planning of garbage collection service: An arc-routing problem with time-dependent penalty cost. *IEEE Transactions on Intelligent Transportation Systems*, .

Josyula, S. P., Törnquist Krasemann, J., & Lundberg, L. (2018). A parallel algorithm for train rescheduling. *Transportation Research Part C: Emerging Technologies*, *95*, 545 – 569. URL: http://www.sciencedirect.com/science/article/pii/S0968090X18309410. doi:https://doi.org/10.1016/j.trc.2018.07.003.

Kellenbrink, C., & Helber, S. (2015). Scheduling resource-constrained projects with a flexible project structure. *European Journal of Operational Research*, *246*, 379–391.

Klein, R. (2000). Project scheduling with time-varying resource constraints. *International Journal of Production Research*, *38*, 3937–3952.

Klein, R., & Scholl, A. (1999). Computing lower bounds by destructive improvement: An application to resource-

constrained project scheduling. *European Journal of Operational Research*, *112*, 322–346.

Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European journal of operational research*, *174*, 23–37.

Kolisch, R., & Padman, R. (2001). An integrated survey of deterministic project scheduling. *Omega*, *29*, 249–272.

Krüger, D., & Scholl, A. (2009). A heuristic solution framework for the resource constrained (multi-) project scheduling problem with sequence-dependent transfer times. *European Journal of Operational Research*, *197*, 492–508.

Kuster, J., Jannach, D., & Friedrich, G. (2009). Extending the rcpsp for modeling and solving disruption management problems. *Applied Intelligence*, *31*, 234.

Laguna, M., & Martí, R. (2012). *Scatter search: methodology and implementations in C* volume 24. Springer Science & Business Media.

LocalSolver (2020). Localsolver reference manual. URL: `https://www.localsolver.com/` accessed on December 17, 2023.

Lozano, L., & Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. *Computers & Operations Research*, *40*, 378–384.

Montoya, A., Guéret, C., Mendoza, J. E., & Villegas, J. G. (2016). A multi-space sampling heuristic for the green vehicle routing problem. *Transportation Research Part C: Emerging Technologies*, *70*, 113–128.

OpenMP Architecture Review Board (2015). OpenMP application program interface version 4.5. URL: `https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf` accessed on December 17, 2023.

Pióro, M., Szentesi, Á., Harmatos, J., Jüttner, A., Gajowniczek, P., & Kozdrowski, S. (2002). On open shortest path first related network optimisation problems. *Performance evaluation*, *48*, 201–223.

Raith, A., & Ehrgott, M. (2009). A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research*, *36*, 1299–1331.

Sever, D., Zhao, L., Dellaert, N., Demir, E., Van Woensel, T., & De Kok, T. (2018). The dynamic shortest path problem with time-dependent stochastic disruptions. *Transportation Research Part C: Emerging Technologies*, *92*, 42 – 57. URL: `http://www.sciencedirect.com/science/article/pii/S0968090X1830531X`. doi:`https://doi.org/10.1016/j.trc.2018.04.018`.

Thomas, B. W., & White, C. C. (2007). The dynamic shortest path problem with anticipation. *European Journal of Operational Research*, *176*, 836 – 854. URL: `http://www.sciencedirect.com/science/article/pii/S0377221705007228`. doi:`https://doi.org/10.1016/j.ejor.2005.09.019`.

Tian, G., Ren, Y., & Zhou, M. (2016). Dual-objective scheduling of rescue vehicles to distinguish forest fires via differential evolution and particle swarm optimization combined algorithm. *IEEE Transactions on intelligent transportation systems*, *17*, 3009–3021.

Törnquist, J. (2006). Computer-based decision support for railway traffic scheduling and dispatching: A review of models and algorithms. In L. G. Kroon, & R. H. Möhring (Eds.), *5th Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'05)*. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik volume 2 of *OpenAccess Series in Informatics (OASIcs)*. URL: `http://drops.dagstuhl.de/opus/volltexte/2006/659`. doi:`10.4230/OASIcs.ATMOS.2005.659`.

Törnquist, J. (2012). Design of an effective algorithm for fast response to the re-scheduling of railway traffic during disturbances. *Transportation Research Part C: Emerging Technologies*, *20*, 62 – 78. URL: http://www.sciencedirect.com/science/article/pii/S0968090X10001671. doi:https://doi.org/10.1016/j.trc.2010.12.004. Special issue on Optimization in Public Transport+ISTT2011.

Bureau of Transportation Statistics, U. D. o. T. (a). Airline on-time performance data. URL: https://www.transtats.bts.gov/tables.asp?db_id=120&DB_Name= accessed on December 17, 2023.

Bureau of Transportation Statistics, U. D. o. T. (b). Aviation support tables. URL: https://www.transtats.bts.gov/tables.asp?DB_ID=595&DB_Name=&DB_Short_Name= accessed on December 17, 2023.

Yan, J., Zhou, M., & Ding, Z. (2016). Recent advances in energy-efficient routing protocols for wireless sensor networks: A review. *IEEE Access*, *4*, 5673–5686.

Yen, J. Y. (1971). Finding the k shortest loopless paths in a network. *management Science*, *17*, 712–716.

Zheng, X.-l., & Wang, L. (2015). A multi-agent optimization algorithm for resource constrained project scheduling problem. *Expert Systems with Applications*, *42*, 6039–6049.

Appendix A. Pseudo-code for function `PenalizeArcs`

In the first two lines, $\mathcal{N}^*$ saves the networks' indices whose arcs are penalized, and $\mathcal{R}^*$ is the set of resources whose capacity is exceeded by the current solution.

Then, the algorithm loops through the elements in $\mathcal{R}^*$ and $\mathcal{N}$. In each loop $r \in \mathcal{R}^*$, the local variable *total* (line 5) represents the amount by which resource r is exceeded.

$\mathcal{A}_n^r$ is a candidate to be penalized if: 1) It has not been penalized in previous iterations ($\mathcal{A}_n^r \in \mathcal{A}_{\mathcal{N}}^{\mathcal{R}}$), and 2) At least one of its arcs is in the current solution ($\mathcal{A}_n^r \cap \mathcal{X} \neq \emptyset$). If both conditions hold, then subset $\mathcal{A}_n^r$ is penalized with probability α.

When penalizing, the cost of the arcs in $\mathcal{A}_n^r$ is increased by *penalty* (line 9), subset $\mathcal{A}_n^r$ is removed from $\mathcal{A}_{\mathcal{N}}^{\mathcal{R}}$, set $\mathcal{N}^*$ is updated, and *total* is reduced by the amount the penalized subset contributes to infeasibility. If *total* becomes less or equal to zero, then the penalization of the resource stops. This early stopping rule is based on the idea that, if the penalized arcs were not used any longer, then the constraint would not be violated, and penalizing more would be counterproductive (it would discourage other networks from using their best current path).

Note that some of the arcs in a penalized subset $\mathcal{A}_n^r$ might not be in the current solution $\mathcal{X}$. The reason for penalizing these inactive arcs too is to decrease their attractiveness in subsequent iterations, since their addition to the solution may cause a resource that has been made feasible to become infeasible again.

Algorithm 5 Penalizing arcs contributing to infeasibility

function PENALIZEARCS($\mathcal{X}$, $\mathcal{G}^*_{\mathcal{N}}$, $\mathcal{R}$, $\mathcal{A}^{\mathcal{R}}_{\mathcal{N}}$, *penalty*, α)
  $\mathcal{N}^* \leftarrow \emptyset$;
  $\mathcal{R}^* \leftarrow \{r \in \mathcal{R} \mid \sum_{a \in \mathcal{X}} w^r_a - W^r > 0\}$;
  for $r \in \mathcal{R}^*$ **do**
    $total \leftarrow \sum_{a \in \mathcal{X}} w^r_a - W^r$;
    for $n \in \mathcal{N}$ **do**
      if $\mathcal{A}^r_n \in \mathcal{A}^{\mathcal{R}}_{\mathcal{N}}$ & $\mathcal{A}^r_n \cap \mathcal{X} \neq \emptyset$ **then**
        if `Rand()` $< \alpha$ **then**
          $c^*_a \leftarrow c^*_a + penalty,\ \forall\ a \in \mathcal{A}^r_n, c^*_a \in \mathcal{C}^*_n$;
          $\mathcal{A}^{\mathcal{R}}_{\mathcal{N}} \leftarrow \mathcal{A}^{\mathcal{R}}_{\mathcal{N}} \setminus \{\mathcal{A}^r_n\}$;
          $\mathcal{N}^* \leftarrow \mathcal{N}^* \cup \{n\}$;
          $total \leftarrow total - \sum_{a \in \mathcal{A}^r_n \cap \mathcal{X}} w^r_a$;
          if $total \leq 0$ **then**
            break;
          end if
        end if
      end if
    end for
  end for
  return $\mathcal{N}^*$, $\mathcal{G}^*_{\mathcal{N}}$, $\mathcal{A}^{\mathcal{R}}_{\mathcal{N}}$;
end function

Note also that penalizing by subsets is valid as far as not every arc of every network consumes all the resources. In cases where all w_a^r values are greater than zero, then penalizing subsets would not be recommended. This is because all the arcs in a network would belong to the same subset and, therefore, they would be penalized at the same time, making no difference between penalizing or not. For those cases, the penalization must be done only for the arcs that belong to the current solution, not for the subsets with at least one arc in the current solution.

After termination, the procedure returns sets $\mathcal{N}^*$, $\mathcal{G}^*_{\mathcal{N}}$, $\mathcal{A}^{\mathcal{R}}_{\mathcal{N}}$.