Noname manuscript No.
(will be inserted by the editor)

A Distributed Method for Optimal Capacity Reservation

Nicholas Moehle · Xinyue Shen ·
Zhi-Quan Luo · Stephen Boyd

the date of receipt and acceptance should be inserted later

Abstract We consider the problem of reserving link capacity in a network in such a way that any of a given set of flow scenarios can be supported. In the optimal capacity reservation problem, we choose the reserved link capacities to minimize the reservation cost. This problem reduces to a large linear program, with the number of variables and constraints on the order of the number of links times the number of scenarios. We develop a scalable, distributed algorithm for the problem that alternates between solving (in parallel) one flow problem per scenario, and coordination steps, which connect the individual flows and the reservation capacities.

1 Introduction

We address the *capacity reservation problem*, the problem of reserving link capacity in a network in order to support multiple possible flow patterns. We are given a description of the network, and a collection of traffic demand scenarios, which specify the amount of flow to be routed from some sources

Communicated by Paul I. Barton

Nicholas Moehle
Mechanical Engineering Department
Stanford University
moehle@stanford.edu

Xinyue Shen
Electronic Engineering Department
Tsinghua University

Zhi-Quan Luo
Electrical and Computer Engineering
University of Minnesota

Stephen Boyd
Electrical Engineering Department
Stanford University

to some sinks. We must reserve enough link capacity so that for any of these scenarios, there exist flows that route the traffic demand while using no more than the reserved capacity on each link. Subject to this requirement, we seek to minimize a linear reservation cost. This problem is also referred to as the *network design problem*.

The capacity reservation problem can be expressed as a linear program (LP), a fact first recognized by Gomory and Hu in 1962 [1]. For moderate problem sizes, it can be solved using generic LP solvers. However, problems with a large number of scenarios (and especially those that do not fit in a single computer's memory) are beyond the reach of such solvers.

We present an iterative method for solving the capacity reservation problem based on the alternating direction method of multipliers (ADMM). Each iteration of the algorithm involves solving, in parallel, a single minimum-cost flow problem for each scenario. This means that our method can easily exploit parallel computing architectures, and can scale to enormous problem instances that do not fit in the memory of a single computer. Unlike general ADMM, our method maintains a feasible point at each iteration (and therefore an upper bound on the problem value). We can also compute a lower bound at modest cost, which can be used to bound the suboptimality of the current iterate, allowing for non-heuristic termination of the algorithm.

Previous work. The optimal reservation problem has a long history, and was first proposed by Gomory and Hu in 1962 [1], who consider the case of a finite source set, and a single commodity. In their extension to the multicommodity case, they propose a decomposition method based on duality [2]. They also note that the structure of the capacity reservation problem makes decomposition techniques attractive. Labbé et al. [3] propose two decomposition techniques, one of which is based on Lagrangian duality and foreshadows the method we propose here. Other decomposition approaches are based on cutting plane methods, which can be interpreted as using only a small subset of the source set, and adding additional source vectors as necessary. Examples of this approach include the work of Minoux et al. [4] and Petrou et al. [5].

Several other source set descriptions are given in the literature, but are typically intractable. For example, when the source set is given by polyhedron (described by a set of inequalities), the capacity reservation problem is known to be NP-hard [6]. However, for this and other cases, such as ellipsoidal and conic source sets, several effective heuristics exist. One tractable approach is to restrict the flows to be an affine function of the demands. This approach was first proposed by Ben-Ameur and Kerivin [7,8], and is called *static routing*, *oblivious routing*, *dynamic routing*, or *affine routing*, depending on the form of the affine function used. For a summary, see [9]. Such a heuristic can be viewed as an application of affinely adjustable robust optimization, has been studied extensively in robust optimization; see [10].

In the case of a finite source set, the capacity reservation problem can be reduced to a convex optimization problem (indeed, a linear program). These problems are tractable and mature software exists that can be used to solve

them [11]. Our approach to the capacity reservation problem is based on a particular algorithm for solving convex optimization problems, called the alternating direction method of multipliers (ADMM) [12,13], which is well-suited for solving large convex problems on distributed computers.

Outline. We start by describing the capacity reservation problem in §2, and a simple heuristic for solving it. In §3, we provide a duality-based method for obtaining lower bounds on the problem value. In §4, we illustrate the heuristic, as well as the lower bounds, on two illustrative examples. In §5, we give an ADMM-based distributed algorithm for solving the capacity reservation problem. We conclude with a numerical example in §6.

2 Capacity Reservation Problem

We consider a single-commodity flow on a network. The network is modeled as a directed connected graph with n nodes and m edges, described by its incidence matrix $A \in \mathbb{R}^{n\times m}$,

$$A_{ij} = \begin{cases} 1, \text{ if edge } j \text{ points to node } i \\ -1, \text{ if edge } j \text{ points from node } i \\ 0 \ \text{ otherwise.} \end{cases}$$

We let $f \in \mathbb{R}^m$ denote the vector of edge flows, which we assume are nonnegative and limited by a given edge capacity, *i.e.*, $0 \le f \le c$, where the inequalities are interpreted elementwise, and $c \in \mathbb{R}^m$ is the vector of edge capacities. We let $s \in \mathbb{R}^n$ denote the vector of sources, *i.e.*, flows that enter (or leave, when negative) the node. Flow conservation at each node is expressed as $Af + s = 0$. (This implies that $\mathbf{1}^T s = 0$.) We say that f is a *feasible flow* for the source s if there exists f that satisfies $Af + s = 0$ and $0 \le f \le c$. For a given source vector s, finding a feasible flow (or determining that none exists) reduces to solving a linear program (LP).

We consider a set of source vectors $\mathcal{S} \subset \mathbb{R}^n$, which we call the *source set*. A *flow policy* is a function $\mathcal{F} : \mathcal{S} \to \mathbb{R}^m$. We say that a flow policy is feasible if for each $s \in \mathcal{S}$, $\mathcal{F}(s)$ is a feasible flow for s. Roughly speaking, a flow policy gives us a feasible flow for each possible source. Our focus is on the selection of a flow policy, given the network (*i.e.*, A and c) and a description of the source set $\mathcal{S}$.

A *reservation* (or reserved capacity) is a vector $r \in \mathbb{R}^m$ that satisfies $0 \le r \le c$. We say that the reservation r *supports* a flow policy $\mathcal{F}$ if for every source vector $s \in \mathcal{S}$, we have $\mathcal{F}(s) \le r$. Roughly speaking, we reserve a capacity r_j on edge j; then, for any possible source s, we can find a feasible flow f that satisfies $f_j \le r_j$, *i.e.*, it does not use more than the reserved capacity on each edge. The cost of the reservation r is $p^T r$, where $p \in \mathbb{R}^m$, $p \ge 0$ is the vector of edge reservation prices. The *optimal capacity reservation* (CR) problem is to find a flow policy $\mathcal{F}$ and a reservation r that supports it, while minimizing the reservation cost.

The CR problem can be written as

$$\begin{array}{ll} \text{minimize} & p^T r \\ \text{subject to} & A\mathcal{F}(s) + s = 0, \quad 0 \le \mathcal{F}(s) \le r, \quad \forall s \in \mathcal{S} \\ & r \le c. \end{array} \tag{1}$$

The variables are the reservation $r \in \mathbb{R}^m$ and the flow policy $\mathcal{F} : \mathcal{S} \to \mathbb{R}^m$. The data are A, $\mathcal{S}$, c, and p. This problem is convex, but when $\mathcal{S}$ is infinite, it is infinite dimensional (since the variables include a function on an infinite set), and contains a semi-infinite constraint, *i.e.*, linear inequalities indexed by an infinite set. We let $J^\star$ denote the optimal value of the CR problem.

We will focus on the special case when $\mathcal{S}$ is finite, $\mathcal{S} = \{s^{(1)}, \ldots, s^{(K)}\}$. We can think of $s^{(k)}$ as the source vector in the kth *scenario*. The CR problem (1) is tractable in this case, indeed, an LP:

$$\begin{array}{ll} \text{minimize} & p^T r \\ \text{subject to} & Af^{(k)} + s^{(k)} = 0, \quad 0 \le f^{(k)} \le r, \quad k = 1, \ldots, K, \\ & r \le c. \end{array} \tag{2}$$

The variables are the reservations $r \in \mathbb{R}^m$ and the scenario flow vectors $f^{(k)} \in \mathbb{R}^m$ for each scenario k. This LP has $m(K+1)$ scalar variables, nK linear equality constraints, and $m(2K+1)$ linear inequality constraints. In the remainder of the paper, we assume the problem data are such that this problem is feasible, which occurs if and only if for each scenario there is a feasible flow. (This can be determined by solving K independent LPs.)

We note that a solution to (2) is also a solution for the problem (1) with the (infinite) source set $\mathcal{S} = \mathbf{Conv}\{s^{(1)}, \ldots, s^{(K)}\}$, where $\mathbf{Conv}$ denotes the convex hull. In other words, a reservation that is optimal for (2) is also optimal for the CR problem in which the finite source set $\mathcal{S}$ is replaced with its convex hull. To see this, we note that the optimal value of (2) is a lower bound on the optimal value of (1) with $\mathcal{S}$ the convex hull, since the feasible set of the former includes the feasible set of the latter. But we can extend a solution of the LP (2) to a feasible point for the CR problem (1) with the same objective, which therefore is optimal. To create such as extension, we define $\mathcal{F}(s)$ for any $s \in \mathcal{S}$. We define $\theta(s)$ as the unique least-Euclidean-norm vector with $\theta(s) \ge 0$, $\mathbf{1}^T\theta(s) = 1$ (with $\mathbf{1}$ the vector with all entries one) that satisfies $s = \sum_k \theta_k s^{(k)}$. (These are barycentric coordinates.) We then define $\mathcal{F}(s) = \sum_{k=1}^K \theta_k(s) f^{(k)}$.

We note for future use another form of the CR problem (2). We eliminate r using $r = \max_k f^{(k)}$, where the max is understood to apply elementwise. Then we obtain the problem

$$\begin{array}{ll} \text{minimize} & p^T \max_k f^{(k)} \\ \text{subject to} & Af^{(k)} + s^{(k)} = 0, \quad 0 \le f^{(k)} \le c, \quad k = 1, \ldots, K, \end{array} \tag{3}$$

where the variables are $f^{(k)}$, $k = 1, \ldots, K$. This is a convex optimization problem.

We will also express the CR problem using matrices, as

$$\begin{array}{ll} \text{minimize} & p^T \max(F) \\ \text{subject to} & AF + S = 0, \\ & 0 \leq F \leq C \end{array} \tag{4}$$

with variable $F \in \mathbb{R}^{m \times K}$. The columns of F are the flow vectors $f^{(1)}, \ldots, f^{(K)}$. The columns of S are the source vectors $s^{(1)}, \ldots, s^{(K)}$, and the columns of C are all the capacity vector c, *i.e.*, $C = c\mathbf{1}^T$, where $\mathbf{1}$ is the vector with all entries one. The inequalities above are interpreted elementwise, and the max of a matrix is taken over the rows, *i.e.*, it is the vector containing the largest element in each row.

Heuristic flow policy. We mention a simple heuristic for the reservation problem that foreshadows our method. We choose each scenario flow vector $f^{(k)}$ as a solution to the capacitated minimum-cost flow problem

$$\begin{array}{ll} \text{minimize} & p^T f \\ \text{subject to} & Af + s^{(k)} = 0, \quad 0 \leq f \leq c, \end{array} \tag{5}$$

with variable $f \in \mathbb{R}^m$. These flow vectors are evidently feasible for (3), and therefore the associated objective value J^{heur} is an upper bound on the optimal value of the problem, *i.e.*,

$$J^\star \leq J^{\text{heur}} = p^T \max_k f^{(k)}.$$

Note that finding the flow vectors using this heuristic involves solving K smaller LPs independently, instead of the one large LP (2). This heuristic flow policy greedily minimizes the cost for each source separately; but it does not coordinate the flows for the different sources to reduce the reservation cost.

We will now show that we also have

$$J^{\text{heur}}/K \leq J^\star,$$

i.e., J^{heur}/K is a lower bound on $J^\star$. This shows that the heuristic policy is K*-suboptimal*, that is, it achieves an objective value within a factor of K of the optimal objective value. We first observe that for any nonnegative vectors $f^{(k)}$, we have

$$(1/K) \sum_{k=1}^{K} p^T f^{(k)} = p^T \left((1/K) \sum_{k=1}^{K} f^{(k)} \right) \leq p^T \max_k f^{(k)}, \tag{6}$$

which follows from the fact that the maximum of a set of K numbers is at least as big as their mean. Now minimize the lefthand and righthand sides over the feasible set of (2). The the righthand side becomes $J^\star$, and the lefthand side becomes J^{heur}.

In §4.2 we give an example showing that this bound is tight, *i.e.*, the heuristic flow policy can indeed produce an objective value (nearly) K times the optimal value.

3 Lower Bounds and Optimality Conditions

The heuristic flow policy described above provides an upper and lower bound on $J^\star$. These bounds are computationally appealing because they involve solving K small LPs independently. Here we extend the basic heuristic method to one in which the different flows are computed using different prices. This gives a parametrized family of lower and upper bounds on $J^\star$. Using ideas based on linear programming duality, we can show that this family is tight, *i.e.*, there is a choice of prices for each scenario that yields lower and upper bound $J^\star$. This duality idea will form the basis for our distributed solution to (2), described in §5.

Scenario prices. We consider nonnegative *scenario pricing vectors* $\pi^{(k)} \in \mathbb{R}^m$, for $k = 1, \ldots, K$, satisfying $\sum_{k=1}^K \pi^{(k)} = p$.

Then we have

$$p^T \max_k f^{(k)} \geq \sum_{k=1}^K \pi^{(k)T} f^{(k)}. \tag{7}$$

It follows that the optimal value of the LP

$$\begin{array}{ll} \text{minimize} & \sum_{k=1}^K \pi^{(k)T} f^{(k)} \\ \text{subject to} & Af^{(k)} + s^{(k)} = 0, \quad 0 \leq f^{(k)} \leq c, \quad k = 1, \ldots, K \end{array} \tag{8}$$

is a lower bound on $J^\star$. This problem is separable; it can be solved by solving K small LPs (capacited flow problems) independently. In other words, to obtain the bound, we first decompose the reservation prices on each edge into scenario prices. Then, for each scenario k, the flow $f^{(k)}$ is chosen (independently of the other scenarios) to be a minimum-cost flow according to price vector $\pi^{(k)}$. The scenario price vectors can be interpreted as a decomposition of the reservation price vector, *i.e.*, the price of reservation along each edge can be decomposed into a collection of flow prices for that edge, with one for each scenario. Finally, we note that this lower bounding property can also be derived as a special case of linear programming duality.

Note that the inequality (6) is a special case of (7), obtained with scenario prices $\pi^{(k)} = (1/K)p$. In fact, with these scenario prices, the heuristic subproblem (5) is also just a special case of (8), *i.e.*, the family of lower bounds obtained by solving (8) with different scenario pricing vectors includes the single lower bound obtained using the heuristic method.

Given a set of scenario prices, the flows found as above are feasible for the CR problem. Therefore $p^T \max_k f^{(k)}$ is an upper bound on $J^\star$.

Tightness of the bound and optimality conditions. There exist some scenario prices for which the optimal value $J^\star$ of the CR problem (2) and its relaxation (8) are the same. This is a consequence of linear programming duality; a (partial) dual of (2) is the problem of determining the scenario pricing vectors that maximize the optimal value of (8). In addition, given such scenario prices, any optimal flow policy for (2) is also optimal for (8) with these scenario prices.

We can formalize this notion as follows. A flow policy $f^{(k)}$ is optimal if and only if there exists a collection of scenario prices $\pi^{(k)}$ such that the following conditions hold.

1. The vectors $\pi^{(k)}$ must be valid scenario prices, *i.e.*,

$$\sum_{k=1}^{K} \pi_j^{(k)} = p_j, \quad \pi^{(k)} \geq 0, \quad j = 1, \ldots, m. \tag{9}$$

2. The inequality (7) must be tight:

$$p^T \max_k f^{(k)} = \sum_{k=1}^{K} \pi^{(k)T} f^{(k)}.$$

3. The flow vectors $f^{(k)}$, for $k = 1, \ldots, K$, must be optimal for (5), with scenario prices $\pi^{(k)}$.

Condition 2 has the interpretation that the optimal reservation cost can be written as a sum of K cost terms corresponding to the scenarios, *i.e.*, the reservation cost can be allocated to the K scenarios. This can be made the basis for a payment scheme, in which scenario k is charged with a fraction $\pi^{(k)T} f^{(k)}$ of the total reservation cost. Note that because the scenario prices that show optimality of a collection of scenario flow vectors are not necessarily unique, these cost allocations may not be unique.

Condition 2 also implies the complimentarity condition

$$\pi_j^{(k)T} (f^{(k)} - r)_j = 0$$

for all k and j. In other words, a positive scenario price on an edge implies that the corresponding scenario edge flow is equal to the optimal reservation, *i.e.*, this scenario edge flow must contribute to the reservation cost. Similarly, if a scenario edge flow is not equal to that edge's reservation, then the scenario price must be zero for that edge.

Condition 3 means that, given the scenario prices, the flow vector for each scenario is a minimum-cost flow using the scenario price vector associated with that scenario. We can interpret the price vectors as weighting the contribution of a scenario edge flow to the total reservation cost. In the case that $\pi_j^{(k)} = 0$, then the flow over edge j under scenario k does not contribute to the total reservation cost. If $\pi_j^{(k)} > 0$, then the flow over edge j under scenario k contributes to the total reservation cost, and from before, we have $f_j^{(k)} = r_j$. Additionally, if $\pi^{(k)} = 0$ for some k, then scenario k is irrelevant *i.e.*, removing it has no effect on the reservation cost. If instead $\pi^{(k)} = p$, we conclude that scenario k is the only relevant scenario.

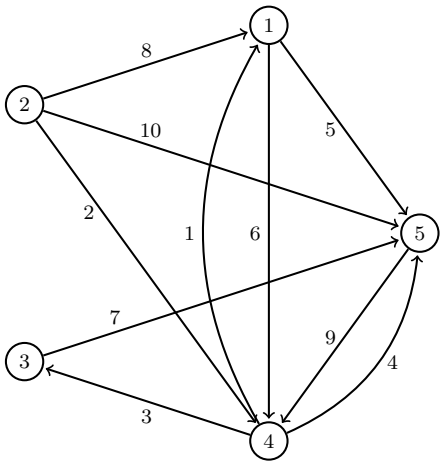

Fig. 1 The graph of the illustrative example.

4 Examples

4.1 Illustrative Example

Here we consider a simple example with $n = 5$ nodes, $m = 10$ edges, and $K = 8$ scenarios. The graph was generated randomly, from a uniform distribution on all connected graphs with 5 nodes and 10 edges, and is shown in figure 1. We use price vector $p = \mathbf{1}$ and capacity vector $c = \mathbf{1}$. The 8 scenario source vectors were chosen according to $s^{(k)} = -Az^{(k)}$, where each element of $z^{(k)}$ was randomly chosen from $\{0, 1/3, 2/3, 1\}$. This guarantees that there is a feasible flow for each scenario.

The optimal reservation cost is 6.0, and the objective of the heuristic policy is 7.6. (The lower bound from the heuristic policy is 2.3.) The optimal and heuristic flow policies are shown in figure (2). The upper plot shows the optimal policy, and the lower plot shows the heuristic policy. For each plot, the bars show the flow policy; the 10 groups are the edges, and the 8 bars are the edge flows under each scenario. The line above each group of bars is the reservation for that edge.

Optimal scenario prices are given in table 4.1. First, we confirm that the row sums are indeed p. The fact that only some scenario prices are positive for each edge indicates that no scenario is relevant for every edge, *i.e.*, that there is no single scenario that is being planned for. However, for every scenario, the scenario prices are positive for at least one edge, which means that every scenario is potentially relevant to the optimal reservation.

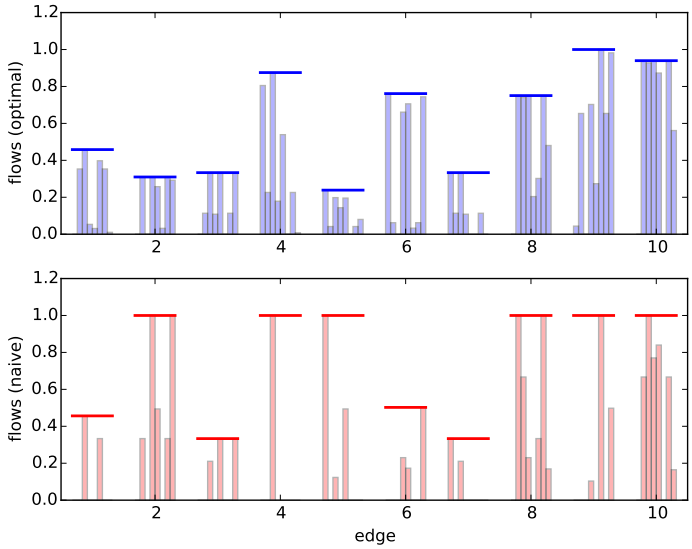

Fig. 2 The optimal and naive flow policies.

Table 1 The scenario prices $\pi^{(1)}, \ldots, \pi^{(8)}$. Column k in the table is the scenario price vector $\pi^{(k)}$. Blank spaces correspond to 0 entries.

		Scenario							
		1	2	3	4	5	6	7	8
Edge	1			1.0					
	2		0.33		0.33			0.33	
	3			0.38		0.28			0.33
	4			1.0					
	5	1.0							
	6	1.0							
	7	0.38		0.62					
	8		0.33		0.33			0.33	
	9						1.0		
	10		0.33		0.33			0.33	

4.2 K-Suboptimal Example

In §2, we showed that the heuristic method produces an objective value not more that K times the optimal value. The previous example showed that the heuristic approach is not, in fact, optimal. Here we give an example that shows that this bound is in fact tight, *i.e.*, for every positive integer K, there is an instance of (2) for which the optimal flow policy outperforms a heuristic solution factor of (nearly) K.

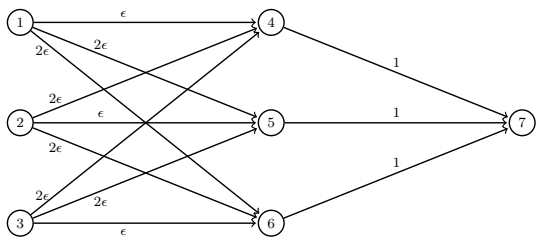

Fig. 3 Graph example for $a = 3$. The labels give the reservation price for that edge.

We consider a graph with $n = 2a+1$ nodes, for some positive integer a, with the nodes arranged in three layers, with a, a, and 1 nodes, respectively. (We show the graph for $a = 3$ in figure 3.) There is an edge connecting every node in the first layer with every node in the second layer, *i.e.*, for every $i = 1, \ldots, a$ and every $j = a+1, \ldots, 2a$, the edge (i, j) is in the graph. The price in this edge is ϵ if $i = j$, and 2ϵ if $i \neq j$. Similarly, every node in the second layer connects to the last node, *i.e.*, for every $i = a+1, \ldots, 2a$, the edge $(i, 2a+1)$ is in the graph. The price for this edge is 1.

We consider $K = a$ scenarios. In each scenario, one unit of flow in injected into a single node in the first layer, and must be routed to the last node. In other words, the source vector $s^{(k)}$ is all zeros, except for the kth element, which is 1, and the last element, which is -1.

A feasible (but not necessarily optimal) point for the CR problem (2) is to route all flow equally through node $a+1$, the first node in the second layer. We must then reserve one unit of capacity for edge $(a+1, 2a+1)$, and one unit of capacity from each node in the first layer to node $a+1$, *i.e.*, for edges $(i, a+1)$, for $i = 1, \ldots, a$. The total reservation cost is $(2a-1)\epsilon + 1 = (2K-1)\epsilon + 1$.

We now consider the (unique) heuristic solution. The solution to (5) under scenario k is to route all flow from node k to node $a+k$, and finally to node $2a+1$, Because no edges support flow under any two different scenarios, the heuristic therefore achieves a reservation cost of $K(\epsilon + 1)$.

As $\epsilon \to 0$, the ratio of the costs of the feasible reservation for (2) and the heuristic reservation approaches K. Because the feasible reservation provides an upper bound on the optimal reservation cost, the ratio between the optimal reservation and the heuristic reservation also approaches K.

5 Distributed Solution Method

In this section we provide a parallelizable algorithm, based on the alternating direction method of multipliers (ADMM), to solve (2). (For details on ADMM, see [13].) We first express the problem (4) in the consensus form

$$\begin{array}{ll} \text{minimize} & p^T \max(\tilde{F}) + g(F) \\ \text{subject to} & F = \tilde{F} \end{array} \tag{10}$$

with variables $F \in \mathbb{R}^{m\times K}$ and $\tilde{F} \in \mathbb{R}^{m\times K}$. The function g is the indicator function for feasible flows,

$$g(F) = \begin{cases} 0, & \text{if } AF + S = 0 \text{ and } 0 \le F \le C \\ \infty & \text{otherwise.} \end{cases}$$

In the consensus problem (10) we have replicated the flow matrix variable, and added a consensus constraint, *i.e.*, the requirement that the two variables representing flow policy must agree.

The augmented Lagrangian for problem (10) is

$$L(F, \tilde{F}, \Pi) = p^T \max(\tilde{F}) + g(F) + \mathbf{Tr}\, \Pi^T (F - \tilde{F}) + (\rho/2)\|F - \tilde{F}\|_F^2,$$

where $\|\cdot\|_F$ is the Frobenius norm, and $\rho > 0$ is a parameter. Here $\Pi \in \mathbb{R}^{m\times K}$ is the dual variable.

ADMM. The ADMM algorithm (with over-relaxation) for (10) is given below. First we initialize the iterates $\tilde{F}(0)$ and $\Pi(0)$. Then we carry out the following steps:

$$\begin{aligned} F(l+1) &= \operatorname*{argmin}_{F} L\big(F, \tilde{F}(l), \Pi(l)\big) \\ \tilde{F}(l+1) &= \operatorname*{argmin}_{\tilde{F}} L\big(\alpha F(l+1) + (1-\alpha)\tilde{F}(l), \tilde{F}, \Pi(l)\big) \\ \Pi(l+1) &= \Pi(l) + \rho\big(\alpha F(l+1) + (1-\alpha)\tilde{F}(l) - \tilde{F}(l+1)\big), \end{aligned}$$

where α is an algorithm parameter in $(0, 2)$; the argument l is the iteration number. For reasons that will become clear, we call these three steps the *flow policy update*, the *reservation update*, and the *price update*, respectively.

Convergence. Given our assumption that (2) is feasible, we have $F(l) \to F^\star$, $\tilde{F}(l) \to F^\star$, and $\Pi(l) \to \Pi^\star$, where $(F^\star, \Pi^\star)$ is a primal-dual solution to (4). This follows from standard results on ADMM [13].

Flow policy update. Minimizing the Lagrangian over F requires minimizing a quadratic function over the feasible set of (4). This can be interpreted as the solution of a minimum-cost flow problem for each of the K scenarios. More specifically, column k of the matrix $F(l+1)$ is the unique solution to the quadratic minimum-cost flow problem:

$$\begin{array}{ll} \text{minimize} & \pi^{(k)T} f + (\rho/2)\|f - \tilde{f}^{(k)}\|^2 \\ \text{subject to} & Af + s^{(k)} = 0 \\ & 0 \le f \le c. \end{array} \tag{11}$$

Here $\tilde{f}^{(k)}$ and $\pi^{(k)}$ are the kth columns of the matrices $\tilde{F}(l)$, and $\Pi(l)$, respectively. These K flow problems can be solved in parallel to update the K columns of the iterate $F(l+1)$. Note that the objective can be interpreted as a sum of the edge costs, using the estimate of the scenario prices, plus a quadratic regularization term that penalizes deviation of the flow vector from the previous flow values $\tilde{f}^{(k)}$.

Reservation update. Minimizing the Lagrangian over $\tilde{F}$ requires minimizing the sum of a quadratic function of $\tilde{F}$ plus a positive weighted sum of its row-wise maxima. This step decouples into m parallel steps. More specifically, The jth row of $\tilde{F}(l+1)$ is the unique solution to

$$\text{minimize } p_j \max\left(\tilde{f}\right) - \pi_j^T \tilde{f} + (\rho/2)\left\|\tilde{f} - \left(\alpha f_j + (1-\alpha)\tilde{f}_j\right)\right\|^2, \tag{12}$$

where $\tilde{f}$ is the variable, and f_j, $\tilde{f}_j$, and π_j are the jth rows of $F(l+1)$, $\tilde{F}(l)$, and $\Pi(l)$, respectively. (The scalar p_j is the jth element of the vector p.) We interpret the max of a vector to be the largest element of that vector. This step can be interpreted as an implicit update of the reservation vector. The first two terms are the difference between the two sides of (7). The problem can therefore be interpreted as finding flows for each edge that minimize the looseness of this bound while not deviating much from a combination of the previous flow values f_j and $\tilde{f}_j$. (Recall that at a primal-dual solution to (2), this inequality becomes tight.) There is a simple solution for each of these subproblems, given in appendix A; the computational complexity for each subproblem scales like $K \log K$.

Price update. In the third step, the dual variable $\Pi(l+1)$ is updated.

Iterate properties. Note that the iterates $F(l)$, for $l = 1, 2 \ldots$, are feasible for (4), *i.e.*, the columns of $F(l)$ are always feasible flow vectors. This is a result of the simple fact that each column is the solution of the quadratic minimum-cost flow problem (11). This means that $U(l) = p^T \max F(l)$ is an upper bound on the optimal problem value. Furthermore, because $F(l)$ converges to a solution of (4), this upper bound $U(l)$ converges to $J^\star$ (though not necessarily monotonically).

It can be shown that $\Pi(l) \geq 0$ and $\Pi(l)\mathbf{1} = p$ for $l = 1, 2, \ldots$, *i.e.*, the columns of $\Pi(l)$ are feasible scenario price vectors satisfying optimality condition 1 of §3. This is proven in Appendix B. We can use this fact to obtain a lower bound on the problem value, by computing the optimal value of (8), where the scenario pricing vectors $\pi^{(k)}$ are given by the columns of $\Pi(l)$, which is a lower bound on the optimal value of (4). We call this bound $L(l)$. Because $\Pi(l)$ converges to an optimal dual variable of (10), $L(l)$ converges to $J^\star$ (though it need not converge monotonically).

Additionally, the optimality condition 2 is satisfied by iterates $\tilde{F}(l+1)$ and $\Pi(l+1)$, for $l = 1, 2, \ldots$, if we take $f^{(k)}$ and $\pi^{(k)}$ to be the columns $\tilde{F}(l+1)$ and $\Pi(l+1)$, respectively. This is shown in Appendix B.

Stopping criterion. Several reasonable stopping criteria are known for ADMM. One standard stopping criterion is that that the norms of the primal residual $\|F(l) - \tilde{F}(l)\|$ and the dual residual $\|\Pi(l+1) - \tilde{\Pi}(l)\|$ are small. (For a discussion, see [13, §3.3.1].)

In our case, we can use the upper and lower bounds $U(l)$ and $L(l)$ to bound the suboptimality of the current feasible point $F(l)$. More specifically, we stop when

$$U(l) - L(l) \leq \epsilon^{\mathrm{rel}} L(l) \tag{13}$$

which guarantees a relative error not exceeding ϵ^{rel}. Note that computing $L(l)$ requires solving K small LPs, which has roughly the same computational cost as one ADMM iteration; it may therefore be beneficial to check this stopping criterion only occasionally, instead of after every iteration.

Choice of parameter ρ. Although our algorithm converges for any positive ρ and any $\alpha \in (0, 2)$, the values of these parameters strongly impact the number of iterations required. In many numerical experiments, we have found that choosing $\alpha = 1.8$ works well. The choice of the parameter ρ is more critical. Our numerical experiments suggest choosing ρ as

$$\rho = \mu \frac{\mathbf{1}^T p}{\max(\mathbf{1}^T F^{\mathrm{heur}})}, \tag{14}$$

where F^{heur} is a solution to the heuristic problem described in §2, and μ is a positive constant. With this choice of ρ, the primal iterates $F(l)$ and $\tilde{F}(l)$ are invariant under positive scaling of p, and scale with S and c. (Similarly, the dual iterates $\Pi(l)$ scale with p, but are invariant to scaling of S and c.) Thus the choice (14) renders the algorithm invariant to scaling of p, and also to scaling of S and c. Numerical experiments suggest that values of μ in the range between 0.01 and 0.1 work well in practice. For details, see appendix C

Initialization. Convergence of $F(l)$, $\tilde{F}(l)$, and $\Pi(l)$ to optimal values is guaranteed for any initial choice of $\tilde{F}(0)$ and $\Pi(0)$. However we propose initializing $\tilde{F}(0)$ as a solution to the heuristic method described in §2, and $\Pi(0)$ as $(1/K)p\mathbf{1}^T$. (In this case, we have $F(1) = \tilde{F}(0)$, which means the initial feasible point achieves an objective within a factor of K of optimal.)

6 Numerical Example

We consider some numerical examples with varying problem sizes. For each problem size, *i.e.*, for each specific choice of n, m, and K, the graph is randomly chosen from a uniform distribution on all graphs with n nodes and m edges. The elements of the price vector p are uniformly randomly distributed over the interval $[0, 1]$, and $c = \mathbf{1}$. The elements of the source vectors for scenario k are generated according to $s^{(k)} = -Az^{(k)}$, where the elements of $z^{(k)}$ are uniformly randomly distributed over the interval $[0, 1]$.

6.1 Detailed Results for $n = 2000$, $m = 4000$, $K = 1000$

Problem instance. We first generate an example with $n = 2000$ nodes, $m = 5000$ edges, and $K = 1000$ scenarios, for which the total number of (flow) variables is 5 million. The optimal value is 18053, and the objective obtained by the heuristic flow policy is 24042. The initial lower bound is 5635, so the initial relative gap is $(U(0) - L(0))/L(0) = 3.26$.

Convergence. We use $\alpha = 1.8$, with ρ chosen according to (14) with $\mu = 0.05$. We terminate when (13) is satisfied with $\epsilon^{\mathrm{rel}} = 0.01$, *i.e.*, when we can guarantee that the flow policy is no more than 1% suboptimal (in the sense defined below). We check the stopping criterion at every iteration; however, we only update the lower bound (*i.e.*, evaluate $L(l)$) when l is a multiple of 10.

The algorithm terminates in 95 iterations. The convergence is shown in figure 4. The upper plot shows the convergence of the upper and lower bounds to the optimal problem value, and the lower plot shows the relative gap $(U(l) - L(l))/L(l)$ and the relative suboptimality $(U(l) - J^\star)/J^\star$ (computed after the algorithm has converged to very high accuracy). (Note that both the upper and the lower bounds were updated by the best values to date. *i.e.*, they were non-increasing and non-decreasing, respectively. The lower bound was only updated once every 10 iterations.) Our algorithm guarantees that we terminate with a policy no more than 1% suboptimal; in fact, the final policy is around 0.4% suboptimal.

Timing results. We implemented our method in Python, using the multiprocessing package. The flow update subproblems were solved using the commercial interior-point solver Mosek [14], interfaced using CVXPY, a Python-based modeling language for convex optimization [15]. The reservation update subproblems were solved using the simple method given in Appendix A. (The simulation was also run using the open source solver ECOS [16], which however is single-threaded, and unable to solve the full problem directly. Mosek is able to solve the full problem directly, which allows us to compare the solve time with our distributed method.)

We used a Linux computer with 4 Intel Xeon E5-4620 CPUs, with 8 cores per CPU and 2 virtual cores per physical core. The flow update and reservation update subproblems were solved in parallel with 64 hyperthreads. Mosek is also configured to use 64 hyperthreads.

The compute times are listed in table 2. The first line gives the average time to solve a single flow update subproblem (11). The second line gives the total time taken to solve the problem using our distributed method (which took 95 iterations). The third line gives the time taken to solve the full CR problem directly. For this simulation, we terminated Mosek when the relative surrogate duality gap in the interior-point method reached 0.01. (Although this is a similar stopping criterion to the one used for our method, it is not

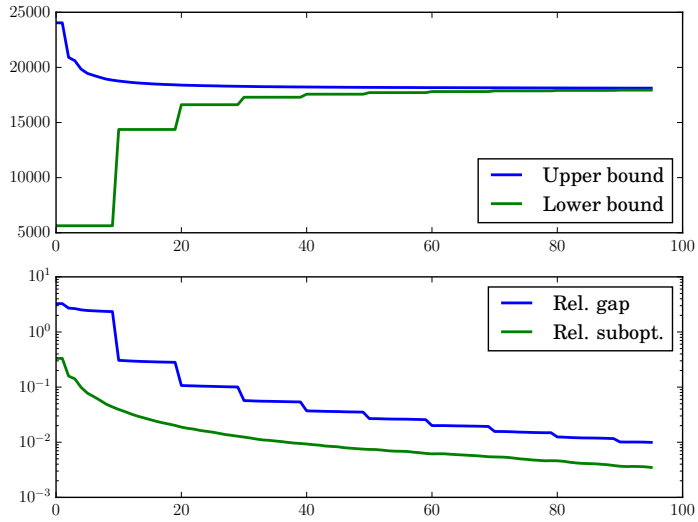

Fig. 4 *Top*: Upper and lower bounds $U(l)$ and $L(l)$ on the optimal value. *Bottom*: Estimated relative gap $(U(l) - L(l))/L(l)$ and relative suboptimality $(U(l) - J^\star)/J^\star$.

Table 2 The timing results for the numerical example.

method	solve time (s)
flow update	1.2
distributed method	6268
full CR problem	45953

equivalent to, nor a sufficient condition for, having a relative gap less than 0.02.)

For this example, our distributed method is about 7 times faster than Mosek. This timing advantage will of course scale with the number of cores or threads, since each iteration of our method is trivially parallelizable, whereas the sparse matrix factorization that dominates each iteration of an interior-point method is not. We expect that by scaling the problem larger, this gap in solve time will widen considerably.

6.2 Timing Experiments

Here we describe several experiments done to show the scalability of our algorithm, and compare it to Mosek. We note that the problem size here was limited by the memory of a single machine. However, by distributing the computation over several machines, or by using disk storage, our solution method could be scaled up further.

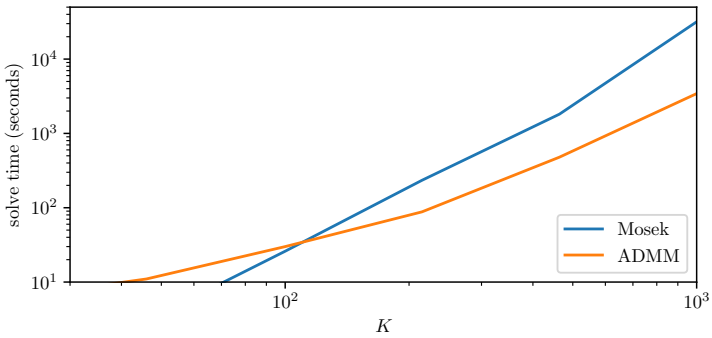

Fig. 5 The solve time as a function of graph size.

Varying problem size. To capture how our method scales with problem size, we generated several problem instances, with the number of scenarios K varying between 10 and 1000. the number of nodes was $2K$, and The number of edges was $5K$. The graph, as well as the scenario flows, were generated as described above (as in the previous numerical example). The parameters for both our method and Mosek are are also identical to the previous numerical example.

The solution time of our method, compared with that of Mosek, is shown in figure 5. We see that for small problem sizes, Mosek outperforms our method, but as the problem dimensions increase, our method begins to outperform Mosek. Overall, however, the methods performs similarly.

Varying number of scenarios. In order to show how our method scales with the number of scenarios, we generated several problem instances in which the size was fixed as $n = 2000$ and $m = 5000$, number of scenarios varied from $K = 10$ to $K = 600$. The graph, as well as the scenario flows, were generated as described above. The parameters for both our method and Mosek are are also identical to the first numerical example. The method for generating the graph was generated as described above, and the parameters for both our method and Mosek are are also identical to our first example.

The solution time of our method, compared with that of Mosek, is shown in figure 6. Mirroring our earlier results for a small number of scenarios, Mosek outperforms our method, but as the graph dimensions increase, our method begins to outperform Mosek.

7 Extensions

Capacitated nodes. In our formulation, only the links have capacity limits. However, in many practical cases, nodes may also have capacity limits, with a corresponding reservation cost.

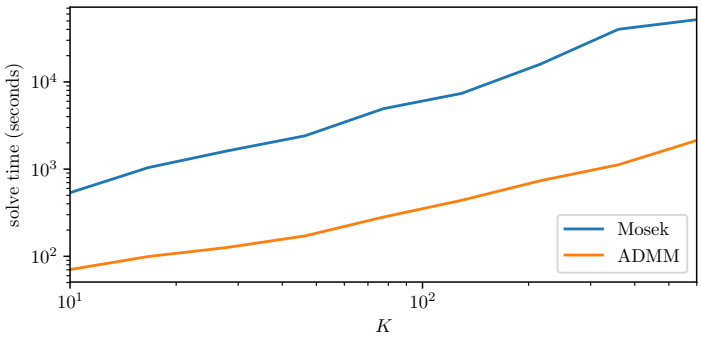

Fig. 6 The solve time as a function of number of scenarios.

Given a problem with a capacitated node, we show how to formulate an equivalent problem, with only capacitated edges. Consider a capacitated node i with capacity a. We can define new graph, with node i replaced by two nodes, i_1 and i_2; all edges previously entering i now enter i_1, and all edges previously leaving i now leave i_2. We then connect i_1 to i_2 with a single new edge, with capacity a and edge reservation price 0.

Multicommodity network. We can extend (2) to handle T different types of commodities. The multicommodity capacity reservation problem is

$$\begin{array}{lll} \text{minimize} & p^T r & \\ \text{subject to} & Af^{(k,t)} + s^{(k,t)} = 0, \quad 0 \le f^{(k,t)} & k = 1, \ldots, K, \quad t = 1, \ldots, T \\ & \sum_{t=1}^{T} f^{(k,t)} \le r, & k = 1, \ldots, K, \\ & r \le c. & \end{array} \tag{15}$$

The variables are the reservations $r \in \mathbb{R}^m$ and $f^{(k,t)} \in \mathbb{R}^m$ for $k = 1, \ldots, K$ and $t = 1, \ldots, T$. The vector is $s^{(k,t)}$ is the source vector associated with type t under scenario k. We note that the scenario prices of §3 extend immediately to the multicommodity case.

8 Conclusion

In this paper, We addressed the *capacity reservation problem*, by expressing it as a structured linear program (LP), and developing a scalable ADMM-based iterative method for solving it. Possible future work includes extensions to handle nonlinear reservation costs, or could allow flow costs in addition to reservation costs. Another extension is to replace our set-based uncertainty set with a probabilitic uncertainty description, and to replace the flow constraint $\mathcal{F}(s) \le r$ with a corresponding probabilistic constraint.

References

1. Gomory, R.E., Hu, T.C.: An application of generalized linear programming to network flows. Journal of the Society for Industrial and Applied Mathematics **10**(2), 260–283 (1962)
2. Gomory, R.E., Hu, T.C.: Synthesis of a communication network. Journal of the Society for Industrial and Applied Mathematics **12**(2), 348–369 (1964)
3. Labbé, M., Séguin, R., Soriano, P., Wynants, C.: Network synthesis with non-simultaneous multicommodity flow requirements: bounds and heuristics. Tech. rep., Institut de Statistique et de Recherche Opérationnelle, Université Libre de Bruxelles (1999)
4. Minoux, M.: Optimum synthesis of a network with non-simultaneous multicommodity flow requirements. North-Holland Mathematics Studies **59**, 269–277 (1981)
5. Petrou, G., Lemaréchal, C., Ouorou, A.: An approach to robust network design in telecommunications. RAIRO-Operations Research **41**(4), 411–426 (2007)
6. Chekuri, C., Shepherd, F.B., Oriolo, G., Scutellá, M.G.: Hardness of robust network design. Networks **50**(1), 50–54 (2007)
7. Ben-Ameur, W., Kerivin, H.: New economical virtual private networks. Communications of the ACM **46**(6), 69–73 (2003)
8. Ben-Ameur, W., Kerivin, H.: Routing of uncertain traffic demands. Optimization and Engineering **6**(3), 283–313 (2005)
9. Poss, M., Raack, C.: Affine recourse for the robust network design problem: between static and dynamic routing. In: Network Optimization, pp. 150–155. Springer (2011)
10. Ben-Tal, A., Ghaoui, L.E., Nemirovski, A.: Robust optimization. Princeton University Press (2009)
11. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004)
12. Parikh, N., Boyd, S.: Proximal algorithms. Foundations and Trends in Optimization **1**(3), 123–231 (2014)
13. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends in Machine Learning **3**(1), 1–122 (2011)
14. ApS, M.: MOSEK Optimizer API for Python 8.0.0.64 (2017). URL http://docs.mosek.com/8.0/pythonapi/index.html
15. Diamond, S., Boyd, S.: CVXPY: A Python-embedded modeling language for convex optimization. Journal of Machine Learning Research **17**(83), 1–5 (2016)
16. Domahidi, A., Chu, E., Boyd, S.: ECOS: An SOCP solver for embedded systems. In: Proceedings of the 12th European Control Conference, pp. 3071–3076. IEEE (2013)

Appendices

A Solution of the Reservation Update Subproblem

Here we give a simple method to carry out the reservation update problem (12). This section is mostly taken from [12, §6.4.1].

We can express (12) as

$$\begin{array}{ll} \text{minimize} & \beta t + (1/2)\|x-u\|^2 \\ \text{subject to} & x_i \le t, \quad i = 1, \dots K, \end{array} \tag{16}$$

Here there variables are $x \in \mathbb{R}^K$ and $t \in \mathbb{R}$. (The variable x corresponds to f_j in (12), and the parameters u and β correspond to $\alpha f_j + (1-\alpha\tilde{f}_j) + \pi_j/\rho$ and p_j/ρ, respectively.)

The optimality conditions are

$$x_i^\star \le t^\star, \quad \mu_i^\star \ge 0, \quad \mu_i^\star(x_i^\star - t^\star) = 0, \quad (x_i^\star - u_i) + \mu_i^\star = 0, \quad \mathbf{1}^T\mu^\star = \beta,$$

where $x^\star$ and $t^\star$ are the optimal primal variables, and $\mu^\star$ is the optimal dual variable. If $x_i < t$, the third condition implies that $\mu_i^\star = 0$, and if $x_i^\star = t^\star$, the fourth condition implies that $\mu_i^\star = u_i - t^\star$. Because $\mu_i^\star \ge 0$, this implies $\mu_i^\star = (u_i - t^\star)_+$. Substituting for $\mu_i^\star$ in the fifth condition gives

$$\sum_{i=1}^n (u_i - t^\star)_+ = \beta,$$

We can solve this equation for $t^\star$ by first finding an index k such that

$$u_{[k+1]} - u_{[k]} \le (u_{[k]} - \beta)/k \le u_{[k]} - u_{[k-1]}$$

where $u_{[i]}$ is the sum of the i largest elements of u. This can be done efficiently by first sorting the elements of u, then computing the cumulative sums in the descending order until the left inequality in the above holds. Note that in this way there is no need to check the second inequality, as it will always hold. (The number of computational operations required to sort u is $K \log K$, making this the most computationally expensive step of the solution.) With this index k computed, we have

$$t^\star = (u_{[k]} - \beta)/k.$$

We then recover $x^\star$ as $x_i^\star = \min\{t^\star, u_i\}$.

B Iterate Properties

Here we prove that, for any $l = 1, 2, \dots$, if $\pi^{(k)}$ are the columns of $\Pi(l)$, optimality condition 1 is satisfied, *i.e.*, $\Pi(l)\mathbf{1} = p$ and $\Pi(l) \ge 0$. Additionally, if $f^{(k)}$ are taken to be the columns of $F(l)$, then condition 2 is also satisfied, *i.e.*,

$$\sum_{j=1}^m \sum_{k=1}^K \Pi(l)_{jk}\tilde{F}(l)_{jk} = p^T \max \tilde{F}(l).$$

To see this, we first note that, by defining $h : \mathbb{R}^{m\times K} \to \mathbb{R}$ such that $h(F) = p^T \max F$, then the subdifferential of h at F is

$$\partial h(F) = \{\Pi \in \mathbb{R}^{m\times K} \mid \Pi\mathbf{1} = p,\ \Pi \ge 0,\ \Pi_{jk} > 0 \implies (\max F)_j = F_{jk}\},$$

i.e., it is the set of matrices whose columns are scenario prices and whose nonzero elements coincide with the elements of F that are equal to the row-wise maximum value.

Now, from the reservation update equation, we have

$$\tilde{F}(l+1) = \operatorname*{argmin}_{\tilde{F}} p^T \max(\tilde{F}) + (\rho/2)\|\alpha F(l+1) + (1-\alpha)\tilde{F}(l) + (1/\rho)\Pi(l) - \tilde{F}\|_F^2.$$

The optimality conditions are

$$\Pi(l) + \rho\big(- \tilde{F}(l+1) + \alpha F(l+1) + (1-\alpha)\tilde{F}(l)\big) \in \partial h\big(\tilde{F}(l+1)\big).$$

Using the price update equation, this is

$$\Pi(l+1) \in \partial h\big(\tilde{F}(l+1)\big).$$

This shows that the columns of $\Pi(l+1)$ are indeed valid scenario prices. Additionally, we have (dropping the indices $(l+1)$ for $\Pi(l+1)$ and $\tilde{F}(l+1)$)

$$\sum_{j=1}^{m}\sum_{k=1}^{K} \Pi_{jk}\tilde{F}_{jk} = \sum_{j=1}^{m}(\max \tilde{F})_j \sum_{k=1}^{K} \Pi_{jk} = \sum_{j=1}^{m}(\max \tilde{F})_j p_j = p^T \max \tilde{F}.$$

The first step follows from the fact that the only terms for which $\Pi(l+1)$ are positive, and thus the only terms that contribute to the sum, are those for which the corresponding element in $\tilde{F}(l+1)$ is equal to the maximum in that row. The second step follows from the fact that $\Pi(l+1)\mathbf{1} = p$.

C Choice of Parameter μ

Here we provide some supplemental details regarding the selection of the ADMM algorithm parameter ρ. (See equation (14).) A suitable range of values for the parameter μ was found on the basis of numerical experiments. In particular, we varied n, m, and K, and generated graphs and scenario flows according to the method of §6. We then studied the number of iterations required for convergence to a tolerance of 0.01. (Problem instances taking longer than 200 iterations were terminated.) Some representative examples are shown in figure 7. We see that the best choice of μ is reasonably consistent across problem dimensions. An appropriate range of μ is around 0.05 to 0.1.

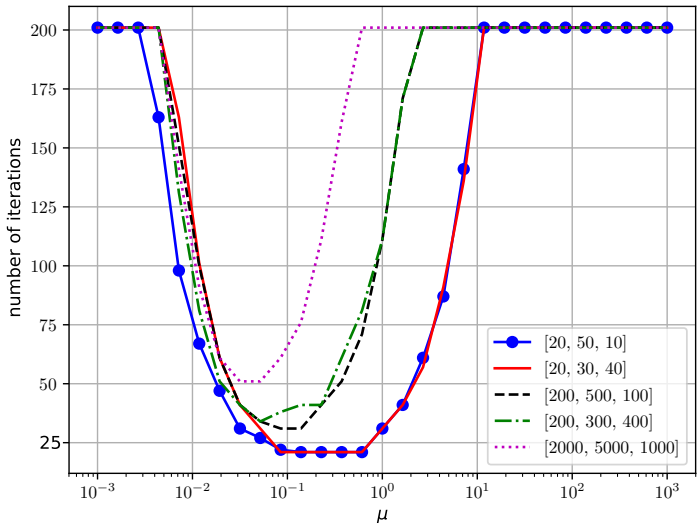

Fig. 7 Number of iterations for convergence, as a function of algorithm parameter μ. The legend shows the triple (n, m, K)