



**CIRRELT-2020-25**

# **An Integrated Learning and Progressive Hedging Mathuristic for Stochastic Network Design Problem**

**Fatemeh Sarayloo  
Teodor Gabriel Crainic  
Walter Rei**

**July 2020**

**Bureau de Montréal**  
Université de Montréal  
C.P. 6128, succ. Centre-Ville  
Montréal (Québec) H3C 3J7  
Tél : 1 514 343-7575  
Télécopie : 1 514 343-7121

**Bureau de Québec**  
Université Laval  
2325, rue de la Terrasse  
Pavillon Palasis-Prince, local 2415  
Québec (Québec) G1V 0A6  
Tél : 1 418 656-2073  
Télécopie : 1 418 656-2624



# An Integrated Learning and Progressive Hedging Matheuristic for Stochastic Network Design Problem

Fatemeh Sarayloo<sup>1,2</sup>, Teodor Gabriel Crainic<sup>1,3,\*</sup>, Walter Rei<sup>1,3</sup>

<sup>1</sup> Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

<sup>2</sup> Department of Computer Science and Operations Research, Université de Montréal, P.O. Box 6128, Station Centre-Ville, Montréal, Canada H3C 3J7

<sup>3</sup> Department of Management and Technology, Université du Québec à Montréal, P.O. Box 8888, Station Centre-Ville, Montréal, Canada H3C 3P8

**Abstract.** In this paper we address the Multicommodity Capacitated Fixed-charge Network Design problem with uncertain demands, which we formulate as a two-stage stochastic program. We rely on the progressive hedging (PH) algorithm of Rockafellar and Wets where the subproblems are defined using scenario groups. To solve the problem, we propose an efficient matheuristic approach which we refer to as the integrated learning and progressive hedging (ILPH). The proposed method takes advantage of a specialized learning-based matheuristic that is able to quickly produce high-quality solutions to multi-scenario subproblems. Furthermore, we propose a novel reference point definition, at each aggregation step of the PH algorithm, which leverages subproblem information regarding promising design variables. Extensive computational experiments illustrate that the proposed approach should be the method of choice when high-quality solutions to large instances of stochastic network problems need to be found quickly.

**Keywords:** Stochastic capacitated network design, uncertain multicommodity demand, two-stage formulation, progressive hedging matheuristic, scenario grouping, learning, reduced cost-based guidance

**Acknowledgements.** While working on this project, T.G. Crainic was Adjunct Professor with the Department of Computer Science and Operations Research, Université de Montréal. Partial funding for this project has been provided by the Natural Sciences and Engineering Council of Canada (NSERC), through its Discovery Grant program, and by the Fonds de recherche du Québec – Nature et technologies (FRQNT) through its strategic center infrastructure grant program.

Results and views expressed in this publication are the sole responsibility of the authors and do not necessarily reflect those of CIRRELT.

Les résultats et opinions contenus dans cette publication ne reflètent pas nécessairement la position du CIRRELT et n'engagent pas sa responsabilité.

---

\* Corresponding author: teodorgabriel.crainic@cirrelt.net

# 1 Introduction

*Multicommodity Capacitated Fixed-charge Network Design (MCND)* models are used to address many important planning problems in a variety of applications, including transportation, logistics, and telecommunications (e.g., Magnanti and Wong, 1984; Crainic, 2000). MCND models formulate the requirement to design a network, i.e., select among a set of potential arcs a subset of arcs, and associated capacity, to satisfy, at minimum total cost, the demand for transportation of a set of commodities between their respective origin-destination node pairs. Selecting an arc incurs a so-called *fixed cost*, while a unit transportation cost is associated to each commodity and arc. The goal is thus to find an optimal design that minimizes the sum of the fixed and transportation costs.

Uncertainty often characterizes the problems addressed. Demand, the volume of each commodity in particular, generally induces a certain level of uncertainty in network design problems. Notably, strategic and tactical planning for complex systems such as the ones mentioned above, yield plans defining the structure of the system or the service, respectively, which are then used repeatedly over quite long periods of time. Some form of forecast then needs to be built to represent the future demand and its variation when the plan is built and the network is designed. Yet, it is well known that solving a deterministic model using single-point estimates of the uncertain parameters, may yield arbitrarily bad solutions (Wallace, 2000; Higle and Wallace, 2003). Models explicitly accounting for uncertainty are then recommended. Stochastic programming and, in particular, two-stage formulations of the MCND with stochastic demand, *SMCND* in the following, has become the methodology of choice to account for demand uncertainty when building such plans. This *a-priori optimization* approach (Birge and Louveaux, 2011) mimics the planning decision process, the first stage design decisions being made prior to the realization of demand, while the second stage decides on the best utilization of the first-stage design (possibly, adjusted marginally) to satisfy the revealed demand. The goal is thus to design a network that remains feasible and cost effective when different demand realizations are encountered.

Demand uncertainty is very often represented through a finite set of discrete scenarios, suitably generated to collectively approximate the uncertainty present in the problem setting, and their associated probabilities. For network design, the design arcs are selected in the first stage, while flow-distribution decisions for each scenario belong to the second stage, resulting in a (deterministic) mixed integer formulation with an objective function minimizing the sum of the fixed costs of the selected design arcs and the expectation of the flow distribution computed over the scenarios considered. This formulation is quite difficult to address using exact methods as, on the one hand, the MCND is a difficult combinatorial problem even when uncertainty is not considered and, on the other hand, using scenarios to model uncertainty yields a large-scale model. Heuristic approaches are thus attractive to produce good-quality solutions within reasonable computing efforts.

The *progressive hedging (PH)* algorithm (Rockafellar and Wets, 1991), first introduced as an exact solution method for continuous stochastic programs, has been successfully adapted as a meta-heuristic for combinatorial optimization problems (e.g., Haugen et al., 2001; Crainic et al., 2011, 2014). PH implements a scenario-decomposition strategy through augmented Lagrangian relaxation of the constraints ensuring a unique design for all scenarios included in the SMCND formulation. This yields a deterministic MCND subproblem for each scenario and the PH iteratively performs three steps: 1) Solve each subproblem producing a scenario-specific design; 2) Compute a reference point, typically, the weighted average of the scenario-specific designs, indicating the level of consensus among the subproblems; 3) The arc fixed costs of the subproblems are modified to promote solution consensus with respect to the current reference point. The procedure stops when a pre-defined level of consensus is reached.

There are a number of challenges to applying the PH idea to the SMCND. First, the size of the problem and the number of iterations needed to reach consensus grow with the number of scenarios, while addressing each subproblem means tackling a difficult MCND problem. As shown by Crainic et al. (2014), the number of iterations can be reduced by grouping scenarios to build multi-scenario subproblems. This has the benefit of forcing a level of local consensus, but makes the subproblems harder to address. Second, the traditional reference-point definition is based exclusively on the subproblem designs available at each iteration, ignoring all possible information on the evolution of the subproblem designs.

We propose the *Integrated Learning and Progressive Hedging (ILPH)* solution method, a new PH-based matheuristic, that addresses these challenges. ILPH integrates a generalized *Learn&Optimize* matheuristic (Sarayloo et al., 2018a), which addresses efficiently multi-scenario network design problems, producing rapidly high-quality solutions. Moreover, the Learn&Optimize procedure provides ILPH with learning capabilities regarding promising design variables gathered during the search paths taken to find high-value solutions to each multi-scenario subproblem. This information is then directly integrated within the PH framework through the computation of a global design and the definition of a new reference point.

Extensive computational experiments show the efficiency of these methodological contributions, the proposed ILPH matheuristic producing a more efficient overall consensus search, both in terms of overall solution quality and total computational effort, compared to the literature. They thus indicate that ILPH should be the method of choice when high-quality solutions to large instances of stochastic network problems need to be found quickly.

The rest of paper is organized as follow. We recall the two-stage formulation for the stochastic network design problem in Section 2. We briefly review some relevant literature in Section 3. We then introduce the main ideas and a detailed description of

our solution methodology in Section 4. Finally, we present and analyze the experimental results in Section 5 and provide concluding remarks in Section 6.

## 2 SMCND Formulation

We briefly recall the two-stage stochastic formulation of the MCND with uncertain demand, where the first-stage decisions concern the network configuration selecting the design arcs, while the commodity flows are determined at the second stage, given the first-stage design and the realized random demands. Uncertainty is modeled through a set of scenarios. The formulation, detailed in Crainic et al. (2011), minimizes the sum of the total cost of designing the network and the expected cost of servicing the possible demand expressed through the considered scenarios. The following notation is used:

- $\mathcal{N}$ : Set of nodes  $i \in |\mathcal{N}|$ .
- $\mathcal{A}$ : Set of potential arcs  $(i, j) \in \mathcal{A}$ .
- $f_{ij}$ : Fixed cost of arc  $(i, j) \in \mathcal{A}$  if included in the final design.
- $u_{ij}$ : Capacity on arc  $(i, j) \in \mathcal{A}$ .
- $\mathcal{K}$ : Set of commodities  $k \in \mathcal{K}$  with origin and destination nodes  $o(k), s(k)$ .
- $c_{ij}^k$ : Unit routing cost for commodity  $k \in \mathcal{K}$  on arc  $(i, j) \in \mathcal{A}$ .
- $\mathcal{S}$ : Set of scenarios  $s \in \mathcal{S}$  with probabilities  $p^1, \dots, p^{|\mathcal{S}|}$ .
- $d_i^{ks}$ : Volume of commodity  $k \in \mathcal{K}$  at node  $i \in \mathcal{A}$  in scenario  $s \in \mathcal{S}$ .
- $y_{ij}$ : *Binary design variable*, equal to 1 if arc  $(i, j) \in \mathcal{A}$  is included in the network at the first stage.
- $x_{ij}^{ks}$ : *Continuous flow variable* representing the amount of commodity  $k$  on arc  $(i, j) \in \mathcal{A}$  in scenario  $s \in \mathcal{S}$ .

The SMCND formulation becomes

$$\text{Minimize } \sum_{(i,j) \in \mathcal{A}} f_{ij} y_{ij} + \sum_{s \in \mathcal{S}} p^s \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} c_{ij}^k x_{ij}^{ks} \quad (1)$$

$$\text{Subject to } \sum_{j \in \mathcal{N}^+(i)} x_{ij}^{ks} - \sum_{j \in \mathcal{N}^-(i)} x_{ji}^{ks} = d_i^{ks}, \quad \forall i \in \mathcal{N}, \forall k \in \mathcal{K}, \forall s \in \mathcal{S}, \quad (2)$$

$$\sum_{k \in \mathcal{K}} x_{ij}^{ks} \leq u_{ij} y_{ij}, \quad \forall (i, j) \in \mathcal{A}, \forall s \in \mathcal{S}, \quad (3)$$

$$y_{ij} \in \{0, 1\}, \quad \forall (i, j) \in \mathcal{A}, \quad (4)$$

$$x_{ij}^{ks} \geq 0, \quad \forall (i, j) \in \mathcal{A}, \forall k \in \mathcal{K}, \forall s \in \mathcal{S}. \quad (5)$$

The objective function (1) minimizes the total system cost, i.e., the sum of the fixed cost of the selected design arcs and the expectation of the routing costs taken over all the demand scenarios. Constraints (2) represent the flow conservation equations in each scenario, requiring that the volume  $d^{ks}$  of demand  $k$  in scenario  $s$  be moved from its origin ( $d_i^{ks} = d^{ks}$ ) to its destination ( $d_z^{ks} = -d^{ks}$ ;  $d_z^{ks} = 0$  at the other nodes). Constraints (3) enforce the linking relation between design and utilization, restricting flows to the capacity of selected arcs only. Notice that the definition of a single design decision variable  $y$  together with constraints (3) guarantee the respect of the non-anticipativity conditions requiring a single design network for all considered demand realization (design decisions must not be tailored to the observed demands of particular scenarios). Constraints (4) - (5) impose integrality and non-negativity restrictions on decision variables.

The SMCND model (1)-(5) is a large-scale mixed integer program with a block-diagonal structure, each block being defined by constraints (2) - (3). We present the solution method we propose following a brief literature review.

### 3 Literature review

When surveying the literature, one finds a limited number of methodologies to address stochastic network design problems. As mentioned earlier, when a finite set of scenarios is used to estimate the stochastic parameters (see Dupačová et al. (2000) and King and Wallace (2012) for an overview on scenario generation methods), a stochastic program can be formulated as an equivalent (multi-scenario) deterministic model. Considering the large scale of the resulting model, taking advantage of the structure through decomposition-based approaches is especially beneficial and is the focus of much of the algorithmic work in this area. The goal of decomposition-based approaches is to divide the complex problem into subproblems to be able to solve them more efficiently. In the case of stochastic programs, such decomposition strategies can be categorized into two types. The first type decomposes the problem via decisional stages while the second type decomposes by scenarios. The former category (referred to as the L-shape method introduced in Van Slyke and Wets (1969)) is a cutting-plane method that is based on the application of Benders decomposition to the equivalent (multi-scenario) deterministic model. Detailed

reviews on this type of decomposition approach for stochastic MCND may be found in Crainic et al. (2016) and Rahmaniani et al. (2017, 2018).

In the second category of decomposition strategies, referred to as scenario decomposition, the original problem is decomposed by scenarios by applying an augmented Lagrange relaxation strategy to the non-anticipativity constraints (i.e., the constraints ensuring that a single design is used under all considered scenarios). Once the problem is decomposed, each scenario becomes a deterministic problem (i.e., a single-scenario subproblem (SSP) defined for each scenario). The resulting scenario subproblems can then be used in various ways when solving stochastic network design problems. First, they enable a general lower bound to be obtained for the stochastic model by solving the Lagrange dual, as proposed in Schütz et al. (2009). In turn, such lower bounds can be applied to design exact solution methods for stochastic network design models, e.g., Escudero et al. (2012) and Alonso-Ayuso et al. (2003). Furthermore, considering that each subproblem takes the form of scenario-specific deterministic model, this decomposition approach also enables heuristics to be defined that leverage existing solution methods for deterministic network design models, e.g., the progressive hedging-based metaheuristics developed by Crainic et al. (2011). However, in the follow up study of Crainic et al. (2014), it was shown that defining the subproblems using multiple scenarios (i.e., multi-scenario subproblems (MSSP)) clearly improved the computational performance of this type of progressive hedging-based solution method.

When applying progressive hedging-based methods, one thus may leverage efficient metaheuristics that are available for deterministic network design models (in the case of single scenario-specific subproblems), or, for SMCND models to address the subproblems (in the case of MSSP).

Although the literature on efficient metaheuristic methods proposed for deterministic MCND problems is very rich (e.g., Crainic et al. (2000); Ghamlouche et al. (2003, 2004); Hewitt et al. (2010)), there are only limited contributions on efficient heuristic methods for solving the SMCND problem. For example, Sarayloo et al. (2018a) proposed a learning based metaheuristic approach where the main novelty is a learning heuristic able to effectively identify structures of good-quality solutions when the scenarios and their influences on design decisions are gradually considered. The proposed metaheuristic produces information on promising design variables related to the considered scenarios. As previously stated, we believe that this can be efficiently used in the PH algorithm to gather more refined and useful information regarding the solutions to the subproblems that will improve the search for a good overall solution to the SMCND.

Exploiting common solution structures that exist between deterministic and stochastic solutions is another feature that may be employed in the solution methods based on scenario decomposition. Due to the high complexity and difficulty of stochastic network design problems, a number of attempts in the literature on the stochastic network design

problems have been devoted to investigate how the solution to the deterministic model relates to the stochastic counterpart. It has been shown that, despite the fact that often times solutions to deterministic models behave badly in stochastic settings (Wallace, 2000; Hingle and Wallace, 2003), there are situations where the deterministic solutions share some properties with the corresponding stochastic solutions (Lium et al., 2009; Thapalia et al., 2011, 2012a,b; Crainic et al., 2018). In these studies, the authors show that the deterministic solutions carry useful information (i.e., some structural patterns) which can be extracted to simplify the stochastic case. Following this insight, Sarayloo et al. (2018b) proposed a number of strategies to extract reduced cost information from good quality solutions to be used as a guide to fix the variables in the SMCND problems.

Revisiting the PH algorithm, as previously stated, the algorithm enables available solution methods for the deterministic MCND to be directly applied to solve the stochastic version of the problem. However, as clearly illustrated in (Crainic et al., 2014), applying the solution approach by having multi-scenario subproblems greatly improves the search for a high quality consensus network. Although, this improvement comes at the expense of having to iteratively solve a series of multi-scenario subproblems, which remains challenging. Therefore, in the next section, we propose an integrated progressive hedging solution method which is able to 1) quickly produce high quality solutions for multi-scenario subproblems and 2) extract more refined information from the subproblems and produce a novel reference point leading to a more efficient overall consensus search.

## 4 Solution methodology

In this section, we propose a matheuristic that combines and enhances the Lagrangian-based multi-scenario decomposition strategy of progressive hedging and the information extraction and learning capabilities of Learn&Optimize. We cast the SMCND model (1)-(5) in a form suitable for multi-scenario decomposition and provide the outline of the proposed ILPH method in Section 4.1. Sections 4.2 and 4.3 describe the methodological developments and strategies proposed for each step of ILPH. The former describes the Learn&Optimize matheuristic generalized to address multi-scenario subproblems and gather relevant information while doing so. We then explain, in Section 4.3, how we exploit that information to compute a new reference point and update the Lagrange multipliers and penalties modifying the fixed costs used in the subproblems and, thus, guiding the search toward consensus. We provide the detailed pseudocode of ILPH in Section 4.4.

## 4.1 Outline of the proposed ILPH

Let  $\{C_1, \dots, C_{|G|}\}$  be a partition of the scenario set  $S$ , where  $G$  is the set of group indices and each group  $C_g \subset S$ ,  $\forall g \in G$ . Let  $p^g = \sum_{s \in C_g} p^s$ . Then, in order to apply the PH idea, we make explicit the non-anticipativity restrictions. Define the first-stage variables  $y_{ij}^g$  for each scenario group  $C_g$ ,  $\forall g \in G$ . Let  $\bar{y}_{ij}$  be the *reference point* standing for the design decision that is feasible in all scenarios and is the goal of the formulation:

$$\text{Minimize} \quad \sum_{g \in G} p^g \left( \sum_{(i,j) \in \mathcal{A}} f_{ij} y_{ij}^g + \sum_{s \in C_g} \frac{p^s}{p^g} \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} c_{ij}^k x_{ij}^{ks} \right) \quad (6)$$

$$\text{Subject to} \quad \sum_{j \in \mathcal{N}^+(i)} x_{ij}^{ks} - \sum_{j \in \mathcal{N}^-(i)} x_{ji}^{ks} = d_i^{ks}, \quad \forall i \in \mathcal{N}, \quad \forall k \in \mathcal{K}, \quad \forall s \in C_g, \quad \forall g \in G, \quad (7)$$

$$\sum_{k \in \mathcal{K}} x_{ij}^{ks} \leq u_{ij} y_{ij}^g, \quad \forall (i, j) \in \mathcal{A}, \quad \forall s \in C_g, \quad \forall g \in G, \quad (8)$$

$$y_{ij}^g = \bar{y}_{ij}, \quad \forall (i, j) \in \mathcal{A}, \quad \forall g \in G, \quad (9)$$

$$y_{ij}^g \in \{0, 1\}, \quad \forall (i, j) \in \mathcal{A}, \quad \forall g \in G, \quad (10)$$

$$x_{ij}^{ks} \geq 0, \quad \forall (i, j) \in \mathcal{A}, \quad \forall k \in \mathcal{K}, \quad \forall s \in C_g, \quad \forall g \in G, \quad (11)$$

where the non-anticipativity constraints (9) guarantee that design decisions are not tailored for each particular scenario and aim to yield a single implementable design. The augmented Lagrangian-relaxation of these constraints then yields the objective function

$$\begin{aligned} \text{Minimize} \quad \sum_{g \in G} p^g \left( \sum_{(i,j) \in \mathcal{A}} f_{ij} y_{ij}^g + \sum_{s \in C_g} \frac{p^s}{p^g} \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} c_{ij}^k x_{ij}^{ks} \right. \\ \left. + \sum_{(i,j) \in \mathcal{A}} \lambda_{ij}^g (y_{ij}^g - \bar{y}_{ij}) + \frac{\rho}{2} (y_{ij}^g - \bar{y}_{ij})^2 \right) \quad (12) \end{aligned}$$

where the Lagrange multipliers  $\lambda_{ij}^g$ ,  $\forall (i, j) \in \mathcal{A}, \forall g \in G$ , are associated with the relaxed constraints (9) and  $\rho$  is a penalty ratio. Given the binary requirements for the design variables, after rearranging the terms, the function may be reformulated as

$$\begin{aligned} \text{Minimize} \quad \sum_{g \in G} p^g \left( \sum_{(i,j) \in \mathcal{A}} (f_{ij} + \lambda_{ij}^g - \rho \bar{y}_{ij} + \frac{\rho}{2}) y_{ij}^g + \sum_{s \in C_g} \frac{p^s}{p^g} \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} c_{ij}^k x_{ij}^{ks} \right) \\ - \sum_{(i,j) \in \mathcal{A}} \lambda_{ij}^g \bar{y}_{ij} + \sum_{(i,j) \in \mathcal{A}} \frac{\rho}{2} \bar{y}_{ij} \quad (13) \end{aligned}$$

Model (13), ((7)-(11)) decomposes along the scenario groups when the overall design is fixed to the reference point. One thus obtains a series of subproblems, each taking the

form of a SMCND problem formulated using the scenarios of the corresponding group  $g \in G$  with fixed costs defined as  $f_{ij} + \lambda_{ij}^g - \rho \bar{y}_{ij} + \frac{\rho}{2}$ ,  $\forall (i, j) \in \mathcal{A}$ . The subproblem  $SP_g$  associated to group  $g \in G$  can then be expressed as:

$$SP_g : \text{Minimize} \quad \sum_{(i,j) \in \mathcal{A}} \left( f_{ij} + \lambda_{ij}^g - \rho \bar{y}_{ij} + \frac{\rho}{2} \right) y_{ij}^g + \sum_{s \in C_g} \frac{p^s}{p^g} \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} c_{ij}^k x_{ij}^{ks} \quad (14)$$

$$\text{Subject to} \quad \sum_{j \in \mathcal{N}^+(i)} x_{ij}^{ks} - \sum_{j \in \mathcal{N}^-(i)} x_{ji}^{ks} = d_i^{ks}, \quad \forall i \in \mathcal{N}, \quad \forall k \in \mathcal{K}, \quad \forall s \in C_g, \quad (15)$$

$$\sum_{k \in \mathcal{K}} x_{ij}^{ks} \leq u_{ij} y_{ij}, \quad \forall (i, j) \in \mathcal{A}, \quad \forall s \in C_g, \quad (16)$$

$$y_{ij}^g \in \{0, 1\}, \quad \forall (i, j) \in \mathcal{A}, \quad (17)$$

$$x_{ij}^{ks} \geq 0, \quad \forall (i, j) \in \mathcal{A}, \quad \forall k \in \mathcal{K}, \quad \forall s \in C_g. \quad (18)$$

Algorithm 1 displays the outline of our proposed *ILPH* method. We generalize the *Learn&Optimize* matheuristic (Sarayloo et al., 2018a) and use it as an efficient method to address the multi-scenario subproblems at each iteration of the PH search (Line 4). We exploit the knowledge learned through the *Learn&Optimize* procedure in computing the new value of the reference point (Line 5) and in updating the penalties used to modify the fixed costs in the subproblems (Line 6). We thus integrate the new information learned while working on the subproblem formulations to the progressive-hedging search strategy, for increased efficiency. This integrated methodology is our main contribution in this paper. Given the fact that the PH algorithm applied to integer problems does generally not converge to a single solution, the proposed matheuristic proceeds to the second phase when a consensus solution is not obtained, once the stopping criterion is met. Phase II (Line 9) follows the general canvas of the literature (Crainic et al., 2011): 1) identify a set of design variables for which the level of consensus is sufficiently high; 2) Fix the values of these variables in the original stochastic model; 3) solve the resulting restricted formulation using a commercial MIP solver.

---

**Algorithm 1** The outline of the proposed ILPH

---

- 1: *Initialization*
  - 2: *Phase I - Multi-scenario PH Search*
  - 3: **while** Stopping criteria not met **do**
  - 4:     Solve heuristically the multi-scenario subproblems ▷ Section 4.2
  - 5:     Aggregation - Compute new reference point ▷ Section 4.3
  - 6:     Penalty update ▷ Section 4.3
  - 7: **end while**
  - 8: *Phase II - Fix and Solve*
  - 9: Fix the design variables for which consensus is obtained and solve the restricted problem
-

## 4.2 Addressing subproblems with the *Learn&Optimize* matheuristic

One of the most challenging parts of the PH algorithm is addressing the multi-scenario stochastic network design problem  $SP_g^\nu$ : (14)-(18) defining the subproblems at iteration  $\nu$ . As defined, the subproblems appear as large-scaled integer programs, which are hard to solve to optimality. This being said, as evidenced by Kall et al. (1994), we may not need to obtain optimal solutions to the subproblems. Heuristics may find satisfactory subproblem solutions that produce an efficient search process for the PH algorithm (Løkketangen and Woodruff, 1996).

We generalize the *Learn&Optimize* procedure, (Sarayloo et al., 2018a), to address the subproblems of the PH algorithm. Two important motivations made us to apply the *Learn&Optimize* in this PH-base matheuristic. First, it provides an efficient heuristic to address the subproblems. Second, we aim to exploit the knowledge learned through the *Learn&Optimize* procedure in the aggregation step of the PH algorithm to further improve its performance. Towards this end, we inject the local information regarding what network characteristics (i.e., arcs to either include or exclude) work “best” for each multi-scenario subproblem that is gathered by *Learn&Optimize* method into the overall PH search procedure.

The *Learn&Optimize* procedure iteratively executes a *learning step*, to learn and build statistics on solution characteristics. Sections 4.2.1 and 4.2.2 provide the full description of the generalized *Learn&Optimize* procedure.

### 4.2.1 Learning step

In the *Learn&Optimize* procedure, Sarayloo et al. (2018a), the authors introduce two main concepts, the *Artificial Demand Scenario (ADS)*, which is defined as a particular combination of two scenarios, and the *Artificial Recourse Problem (ARP)*, which is a network flow problem. The main idea behind the ARP is to send the commodity flows associated with the demand vector of ADS, on a given design, and identify the required *corrective actions*, taking the form of extra arcs, that are needed to satisfy ADS demands. One then learns by iteratively defining ADSs, solving the associated ARPs (which constitutes the core of our learning mechanism), and gradually building an image of design variables that potentially belong to good solutions for the stochastic formulation. The results of this learning mechanism are then used to guide the overall search towards higher quality solutions.

Considering the fact that our goal is to apply the heuristic procedure repeatedly to the subproblems of the PH algorithm which contain multiple scenarios, we generalize

the idea of ADS and propose an generalized *Learn&Optimize*. To do so, we introduce *Group-based Artificial Demand Scenario*, *Gb-ADS*, with the associated ARP.

A *Group-based Artificial Demand Scenario Gb-ADS*  $\delta^g$ ,  $g \in G$ , given the scenarios in group  $g$ ,  $\{s_1^g, \dots, s_{|g|}^g\}$ , is defined as

$$\delta^g = \begin{pmatrix} \delta_1^g(s_1^g, \dots, s_{|g|}^g) \\ \delta_2^g(s_1^g, \dots, s_{|g|}^g) \\ \delta_3^g(s_1^g, \dots, s_{|g|}^g) \\ \vdots \\ \delta_{|\mathcal{K}|}^g(s_1^g, \dots, s_{|g|}^g) \end{pmatrix}, \text{ such that}$$

$$\delta_k^g(s_1^g, \dots, s_{|g|}^g) = d_k(s_1^g) \vee d_k(s_2^g) \vee \dots \vee d_k(s_i^g), \dots, \vee d_k(s_{|g|}^g), \forall k \in \mathcal{K}.$$

Let  $\Delta^g$  be a set of Gb-ADSs,  $\delta^g$ , generated for group  $g$  containing the scenarios  $C_g = \{s_1^g, \dots, s_{|g|}^g\}$ . We use algorithm 2 to construct the set  $\Delta^g$  of cardinality  $N_{\Delta^g}$ , through the random selection of the demand values of the scenarios in  $C_g$ . In other words, we randomly select  $s_i \in C_g$ , copy its demand value associated to commodity  $k$ ,  $d_k(s_i)$ , and let  $\delta_k^g \leftarrow d_k(s_i)$  (Lines 2-4). The procedure stops when it builds  $N_{\Delta^g}$  Gb-ADSs (Line 6).

---

**Algorithm 2** Construct  $\Delta^g$

---

- 1: **repeat**
  - 2:     **for all**  $k \in \mathcal{K}$  **do**
  - 3:         Randomly choose  $s_i \in C_g$  and let  $\delta_k^g \leftarrow d_k(s_i)$
  - 4:     **end for**
  - 5:     Let  $\Delta^g \leftarrow \delta^g \cup \Delta^g$
  - 6: **until**  $|\Delta^g| = N_{\Delta^g}$
  - 7: *Return*  $\Delta^g$
- 

Following Sarayloo et al. (2018a), we aim to explore the solution characteristics associated to each Gb-ADS,  $\delta^g \in \Delta^g$ , to extract information regarding promising design variables. The exploration is performed by solving an *Artificial Recourse Problem*,  $ARP(\delta^g, \hat{y})$ , for each artificial demand scenario  $\delta^g \in \Delta^g$  considering a given design  $\hat{y}$ .

To define  $ARP_g(\delta^g, \hat{y})$ , we separate the set of arcs  $\mathcal{A}$  according to  $\hat{y}$ . Then,  $\mathcal{A} = \mathcal{A}^0 \cup \mathcal{A}^1$ , where  $\mathcal{A}^0 = \{(i, j) | (i, j) \in \mathcal{A}, \hat{y}_{ij} = 0\}$  and  $\mathcal{A}^1 = \{(i, j) | (i, j) \in \mathcal{A}, \hat{y}_{ij} = 1\}$  are the sets of closed and open arcs in  $\hat{y}$ , respectively. By considering that the fixed cost  $f_{ij}^\nu$  is updated at each iteration of the PH algorithm, we then define a modified arc variable

cost  $\bar{c}_{ij}$  by linearizing the fixed cost of the closed arcs

$$\bar{c}_{ij} = \begin{cases} c_{ij} + \frac{f_{ij}^\nu}{u_{ij}}, & \forall (i, j) \in \mathcal{A}^0, \\ c_{ij}, & \forall (i, j) \in \mathcal{A}^1, \end{cases} \quad (19)$$

and solve the  $ARP_g(\boldsymbol{\delta}^g, \hat{y})$ , which is defined as a multi-commodity network flow problem:

$$ARP_g(\boldsymbol{\delta}^g, \hat{y}) : \text{Minimize } \sum_{k \in \mathcal{K}} \sum_{(i, j) \in \mathcal{A}} \bar{c}_{ij} x_{ij}^k \quad (20)$$

$$\text{Subject to } \sum_{j \in \mathcal{N}^+(i)} x_{ij}^k - \sum_{j \in \mathcal{N}^-(i)} x_{ji}^k = \delta_k^{ig}, \quad \forall i \in \mathcal{N}, \forall k \in \mathcal{K}, \quad (21)$$

$$\sum_{k \in \mathcal{K}} x_{ij}^k \leq u_{ij}, \quad \forall (i, j) \in \mathcal{A}, \quad (22)$$

$$x_{ij}^k \geq 0, \quad \forall (i, j) \in \mathcal{A}, \forall k \in \mathcal{K}. \quad (23)$$

Solving  $ARP_g(\boldsymbol{\delta}^g, \hat{y})$  yields  $x_{ij}(\boldsymbol{\delta}^g) = \sum_{k \in \mathcal{K}} x_{ij}^k, \forall (i, j) \in \mathcal{A}$ , with  $\mathcal{A}_{\boldsymbol{\delta}^g} = \{(i, j) | x_{ij}(\boldsymbol{\delta}^g) > 0\}$ . We then define a corresponding design solution as  $y_{ij}^{\boldsymbol{\delta}^g} = 1$ , when  $x_{ij}(\boldsymbol{\delta}^g) > 0$ , and 0, otherwise. It is noteworthy that some of the arcs in  $\mathcal{A}^0$ , closed in  $\hat{y}$ , may be open in  $y_{ij}^{\boldsymbol{\delta}^g}$  to satisfy the demand vector  $\boldsymbol{\delta}^g$ . These modifications capture the interactions occurring in the integration of multiple scenarios within  $\boldsymbol{\delta}^g$ , yielding partial information regarding the design arcs required to address the uncertainty captured by the scenarios involved in group  $g$ . Repeating this procedure for different Gb-ADSs builds the knowledge we seek.

#### 4.2.2 Learn&Optimize procedure

The generalized *Learn&Optimize* procedure iteratively executes the *learning step* to identify a promising set of design variables. Then, it performs the *partial-optimization step* where the identified promising variables are fixed and the reduced-size formulation is solved exactly.

The generalized *Learn&Optimize* procedure is described in Algorithm 3. The sets of Gb-ADSs,  $\Delta^{g\nu}$ , are reconstructed at each iteration  $\nu$  of PH algorithm as described in Algorithm 2. Such an approach, i.e., reconstructing a new set of  $\Delta^{g\nu}$  at each iteration of ILPH, may allow us to obtain more diverse information that would not be available from using a single set of  $\Delta^{g\nu}$  in all iterations. We define the *frequency memory*,  $F_{ij}^{g\nu}$ , representing how often arc  $(i, j)$  has been used in the solutions of the different  $ARP_g(\boldsymbol{\delta}^g, \hat{y})$  and the normalized frequencies values,  $\mathbf{f}_{ij}^{g\nu}$ , which is computed as  $\mathbf{f}_{ij}^{g\nu} := F_{ij}^{g\nu} / \max\{F_{ij}^{g\nu} | (i, j) \in \mathcal{A}\}$  (Line 10). We keep the frequency memories built in the previous iterations, i.e.,  $F_{ij}^{g\nu} \leftarrow F_{ij}^{g\nu-1}$

in order to keep track of promising arcs from the beginning. More specifically, we are implementing a long-term memory strategy on the solution information that is learned at the subproblem level (i.e., the arc frequencies) as a means to both warm start the solution method and to further drive solution consensus at the subproblem level. We also define  $\mathcal{A}_{\Delta^{g\nu}}^\nu$ , the set of design arcs used in at least one  $ARP_g(\delta^g, \hat{y})$  in iteration  $\nu$ , and  $\mathcal{A}^{g\nu}$ , the set of promising design variables to be identified by the procedure.

The main loop (Lines 3 to 9) iterates over the Gb-ADSs in  $\Delta^{g\nu}$ , each being discarded once examined. The loop stops when the set of artificial demand scenarios becomes empty. The  $ARP_g(\delta^g, \hat{y})$  is solved for each  $\delta^g \in \Delta^{g\nu}$ , to distribute the demand of  $\delta^g$  (Line 5). The corresponding design vector is created (Line 7), while the set of used design arcs and the frequency memories are updated at Line 6. Once the artificial demand scenarios  $\delta^g \in \Delta^{g\nu}$  are treated, a reduced problem, obtained by fixing  $\mathcal{A}^{g\nu}$  as the most frequently used arcs (given a threshold  $\tau$ ), is solved using a MIP solver. This yields the design solution  $y_{ij}^{g\nu}, \forall (i, j) \in \mathcal{A}$ . Finally, the procedure returns the normalized frequency values  $\mathbf{f}_{ij}^{g\nu}, \forall (i, j) \in \mathcal{A}$ , as well as the design solution  $y_{ij}^{g\nu}, \forall (i, j) \in \mathcal{A}$ .

---

**Algorithm 3** The generalized *Learn&Optimize* procedure to address  $SP_g^\nu$

---

- 1: *Initialization:*  $F_{ij}^{g\nu} \leftarrow F_{ij}^{g\nu-1}, \forall (i, j) \in \mathcal{A}, \mathcal{A}^{g\nu} \leftarrow \emptyset$ ; construct  $\Delta^{g\nu}$ ;
  - 2: *Learning and memorizing:*
  - 3: **repeat**
  - 4:     *Randomly choose* a Gb-ADS  $\delta^g \in \Delta^{g\nu}$ ;
  - 5:     *Solve*  $ARP_g(\delta^g, \hat{y}^\nu)$  yielding  $x_{ij}^\nu(\delta^g), \forall (i, j) \in \mathcal{A}$ ;
  - 6:     *Identify*  $\mathcal{A}_{\delta^g}^\nu$  and *compute*  $y_{ij}^{\delta^g}, \forall (i, j) \in \mathcal{A}_{\delta^g}^\nu$ ;
  - 7:     *Update*  $\mathcal{A}_{\Delta^{g\nu}}^\nu := \mathcal{A}_{\Delta^{g\nu}}^\nu \cup \mathcal{A}_{\delta^g}^\nu$  and  $F_{ij}^{g\nu} := F_{ij}^{g\nu} + 1$ , for all  $(i, j) \in \mathcal{A}_{\delta^g}^\nu$ ;
  - 8:     *Remove*  $\delta^g$  from  $\Delta^{g\nu}$ ;
  - 9: **until**  $\Delta^{g\nu} = \emptyset$ ;
  - 10: *Normalize* frequencies  $\mathbf{f}_{ij}^{g\nu} := F_{ij}^{g\nu} / \max\{F_{ij}^{g\nu} | (i, j) \in \mathcal{A}\}, \forall (i, j) \in \mathcal{A}$ ;
  - 11: **for all**  $(i, j) \in \mathcal{A}$  **do**
  - 12:     **if**  $\mathbf{f}_{ij}^{g\nu} \geq \tau$  **then**  $\mathcal{A}^{g\nu} \leftarrow \mathcal{A}^{g\nu} \cup \{(i, j)\}$ ;
  - 13:     **end if**
  - 14: **end for**
  - 15: *Partial optimization:*
  - 16: *Solve*  $SP_g^\nu$ , by fixing variables belonging to  $\mathcal{A}^{g\nu}$  to open, yielding solution  $y_{ij}^{g\nu}, \forall (i, j) \in \mathcal{A}$
  - 17: *Return* the normalized frequencies  $\mathbf{f}_{ij}^{g\nu}, \forall (i, j) \in \mathcal{A}$  and solution  $y_{ij}^{g\nu}, \forall (i, j) \in \mathcal{A}$ .
- 

### 4.3 Aggregation and penalty updates

We exploit and inject the local information obtained by the *Learn&Optimize* procedure into the PH algorithm through the definition of a new reference point, Section 4.3.1, and

a new heuristic design. The latter, presented in Section 4.3.2, is created by exploiting the dual information associated with the solutions obtained for the subproblems and serves as the initial solution provided for the search processes of the subproblems at the following iteration.

### 4.3.1 A new reference point

The reference point is a major component of PH matheuristics as it measures the degree of consensus among subproblems and indicates the current trend for opening and closing arcs amongst the subproblem designs. As mentioned earlier, the reference point in previous PH algorithms for SMCND, e.g., (Crainic et al., 2011, 2014), is constructed as  $\bar{y}_{ij}^\nu \leftarrow \sum_{g \in G} p^g y_{ij}^{g\nu}, \forall (i, j) \in \mathcal{A}$ . The dual prices  $\lambda_{ij}^{g\nu}$  are updated, using the reference solution  $\bar{y}_{ij}^\nu$  and the parameter  $\rho$ , as  $\lambda_{ij}^{g\nu} \leftarrow \lambda_{ij}^{g\nu-1} + \rho(y_{ij}^{g\nu} - \bar{y}_{ij}^\nu)$ . This strategy considers a single design vector only obtained by each subproblem to produce the reference point. Considering single solutions only may result in missing alternative cost-efficient subproblem solutions for which consensus could be more easily reached. Our proposed reference point directly targets this issue by explicitly considering the information learned through the use of the *Learn&Optimize* method regarding what generally constitutes good subproblem solutions, i.e., arcs that often appear in high-quality solutions to the subproblems.

Recall that the *Learn&Optimize* procedure creates a history of promising design variables in each subproblem through frequency memories  $F_{ij}^{g\nu}$ , and  $\mathbf{f}_{ij}^{g\nu} \in [0, 1], \forall (i, j) \in \mathcal{A}$ . The information provided by  $\mathbf{f}_{ij}^{g\nu}$  allows the opportunity to better explore the trend amongst the design variables observed during the iterations of the *Learn&Optimize* matheuristic. We propose  $\tilde{y}_{ij}^\nu \leftarrow \sum_{g \in G} p^g \mathbf{f}_{ij}^{g\nu}, \forall (i, j) \in \mathcal{A}$ , where  $p^g = \sum_{s \in C_g} p^s$ , as a reference point in the aggregation step of the ILPH. The Lagrangian multipliers  $\lambda_{ij}^{g\nu}, \forall (i, j) \in \mathcal{A}$  are updated using the new reference point  $\tilde{y}_{ij}^\nu, \forall (i, j) \in \mathcal{A}$  as  $\lambda_{ij}^{g\nu} \leftarrow \lambda_{ij}^{g\nu-1} + \rho(\mathbf{f}_{ij}^{g\nu} - \tilde{y}_{ij}^\nu)$ .

### 4.3.2 A new design solution to iteratively promote consensus

The second idea is to exploit the dual information provided by the subproblem solutions to create an integer design solution at each iteration of the ILPH. This integer design solution  $\hat{y}_{ij}^\nu, \forall (i, j) \in \mathcal{A}$ , is constructed in the aggregation step of the ILPH algorithm and serves as an initial design solution for the generalized *Learn&Optimize* procedure in the subproblems at the following iteration. Exploiting the reduced cost information associated with multiple solutions is motivated by the observation made in Sarayloo et al. (2018b) suggesting that this strategy allows the identification of good-quality solutions in the context of stochastic network design problems.

Note that the proposed design solution does not necessarily define a feasible solution to the original stochastic problem. However, using the same integer initial solution in the *Learn&Optimize* procedure, in all subproblems, we define a soft strategy to promote consensus at the subproblem level.

To create  $\hat{y}_{ij}^\nu$ , we first compute  $\tilde{y}_{ij}^\nu \leftarrow \sum_{g \in G} p^g f_{ij}^{g\nu}$ ,  $\forall (i, j) \in \mathcal{A}$ , and partition the set of design variables into two disjoint subsets:

- $\hat{\mathcal{A}}_1 = \{(i, j) | \tilde{y}_{ij}^\nu \leq l_0 \text{ or } \tilde{y}_{ij}^\nu \geq u_1\}$ : the set of design variables for which a consensus has been almost obtained (given thresholds  $l_0$  and  $u_1$ ) among the groups, or in other words, (almost) all groups agree that these arcs have to be opened or closed.

The design variables in  $\hat{\mathcal{A}}_1$  are then set to the value 0 or 1 as follows

$$\hat{y}_{ij}^\nu = \begin{cases} 0, & \text{if } \tilde{y}_{ij}^\nu \leq l_0, \\ 1, & \text{if } \tilde{y}_{ij}^\nu \geq u_1. \end{cases} \quad (24)$$

- $\hat{\mathcal{A}}_2 = \{(i, j) | l_0 < \tilde{y}_{ij}^\nu < u_1\}$ : the set of the remaining design variables or those for which a consensus has not been obtained.

For the variables in  $\hat{\mathcal{A}}_2$ , the decision is based on reduced cost information. Let  $r_{ij}^g, \forall (i, j) \in \hat{\mathcal{A}}_2$ , be the reduced cost associated with  $y_{ij}^g, \forall (i, j) \in \hat{\mathcal{A}}_2$ ;  $G_{ij}^1 = \{g | y_{ij}^g = 1\}$  the set of groups where the associated design variables  $y_{ij}^g$  are equal to one, and  $\bar{r}_{ij}^{g\nu} = \sum_{g \in G_{ij}^1} p^g r_{ij}^{g\nu}, (i, j) \in \hat{\mathcal{A}}_2$  the average reduced cost over groups  $g \in G_{ij}^1$ . It should be noted that, given the fact that we are solving the restricted problem  $SP_g^\nu$  with the integrality requirements, we need to perform one additional step to obtain the reduced cost values. Once the restricted problem  $SP_g^\nu$  is solved and its optimal (integer) solution,  $y_{ij}^{g\nu}, \forall (i, j) \in \mathcal{A}$ , is obtained, we need to solve the LP relaxation of the problem  $SP_g^\nu$  while the design variables are fixed to the values of the obtained optimal solution  $y_{ij}^{g\nu}, \forall (i, j) \in \mathcal{A}$ . In this way, one can obtain the set of reduced cost values associated with design variables.

We represent the set of reduced costs by  $R = \{\bar{r}_{ij}^{g\nu} | (i, j) \in \hat{\mathcal{A}}_2\}$ . In order to identify good candidate design variables to be fixed to 1 (open), we choose the variables with the smallest reduced cost values. To do so, we sort the reduced cost values  $\bar{r}_{ij}^{g\nu} \in R$  in non-decreasing order. Let  $r_1^{max}$  and  $r_1^{min}$  be the maximum and minimum values in  $R$ . We then divide the difference  $r_1^{max} - r_1^{min}$  in  $N_1$  equally-sized classes and store the indexes of variables belonging to the classes 1 to  $p_1$  ( $1 \leq p_1 \leq N_1$ ) in  $R_1$ . We then set  $\hat{y}_{i,j}^\nu \leftarrow 1, \forall (i, j) \in R_1$  and  $\hat{y}_{i,j}^\nu \leftarrow 0, \forall (i, j) \notin R_1$ . Consequently, the solution  $\hat{y}_{ij}^\nu, \forall (i, j) \in \mathcal{A}$ , at each iteration  $\nu$ , is created by exploiting the reduced cost information associated with multiple solutions obtained by all considered groups.

## 4.4 The algorithm

Algorithm 4 sums up the entire procedure. The algorithm is initialized by constructing the list of scenario groups  $\bar{C} = \{C_1, \dots, C_{|G|}\}$ . The scenarios within each group may be chosen using different strategies proposed in Crainic et al. (2014). For each group  $g \in G$ , we solve heuristically subproblem  $SP_g^0$ : Minimize  $\sum_{(i,j) \in \mathcal{A}} (f_{ij} y_{ij}^g + \sum_{s \in C_g} \frac{p^s}{p^g} \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} c_{ij}^k x_{ij}^{ks})$  st. (15) – (18), using the generalized *Learn&Optimize* procedure described in Section 4.2. Once we address the subproblems  $g \in G$ , we perform the aggregation step to produce the reference point as well as the integer design solution as explained in Section 4.3.

During Phase I, the multi-scenario subproblems are solved approximately as explained in Section 4.2. At each aggregation step,  $\tilde{y}$  and  $\hat{y}$  are constructed and the Lagrangian multipliers are updated as described in Section 4.3. We follow the strategy used in Crainic et al. (2011, 2014) to construct a heuristic feasible network  $y^{M\nu}$  at each iteration  $\nu$ , by setting the design variables  $y_{ij}^{M\nu}$ ,  $\forall (i, j) \in \mathcal{A}$  to

$$y_{ij}^{M\nu} = \begin{cases} 1, & \text{if } y_{ij}^{g\nu} = 1, \text{ for any } g \in G, \\ 0, & \text{otherwise.} \end{cases} \quad (25)$$

The best network found, i.e.,  $y^{Best}$ , is updated based on the quality (total cost) of the feasible solution  $y^{M\nu}$  obtained at each iteration  $\nu$ . We use similar stopping criteria (Line 9) as those in Crainic et al. (2014), namely, a total of  $N_{Itr}$  iterations,  $N_{Imp}$  consecutive iterations without improving, the best known solution,  $t_{max}$  CPU time, or when there are fewer than  $\gamma$  ( $0 \leq \gamma \leq 1$ ) percent of the arcs for which a consensus has not been reached. When such a situation occurs, *Phase II* is initiated. We fix the design variables for which a consensus is obtained and solve the original problem to obtain the final design solution  $y^{Final}$ . At Line 27, we update  $y^{best}$ , when needed.

## 5 Experimental results

This section presents the results obtained from extensive computational experiments performed to assess the performance of the proposed algorithm. We used two collections of instances which are described in Section 5.1. To evaluate the performance of the proposed ILPH algorithm, we compared its results to those of alternative approaches that were tested on the same instances: were used:

- IBM-ILOG CPLEX 12.6.1 with its default settings (CPLEX in the following) on the associated MIP,

---

**Algorithm 4** The proposed integrated learning and progressive hedging matheuristic

---

```

1: Initialization
2: Let  $\nu \leftarrow 0$ ,  $\lambda_{ij}^{g\nu} \leftarrow 0, \forall (i, j) \in \mathcal{A}$ ,  $\rho^\nu \leftarrow \rho^0$ 
3: Construct the list of scenario groups  $\bar{C} = \{C_1, \dots, C_{|\bar{C}|}\}$ 
4: for each group  $g$  do
5:   Solve  $SP_g^0$  heuristically by performing Algorithm 2 (Section 4.2)
6: end for
7: Construct solutions  $y_{ij}^{M\nu}$ ,  $\tilde{y}_{ij}^\nu$ , and  $\hat{y}_{ij}^\nu$ 
8: Phase I: Seek consensus on the arcs  $(i, j)$  that should exist in the design
9: while stopping criteria not met do
10:  Iteration update:
11:    $\nu \leftarrow \nu + 1$ 
12:  Solving subproblems heuristically:
13:  for each group  $g$  do
14:    Solve  $SP_g^\nu$  heuristically by performing Algorithm 2, considering Lagrangian multi-
    pliers  $\lambda_{ij}^{g\nu-1}, \forall (i, j) \in \mathcal{A}$  and solution  $\hat{y}_{ij}^{\nu-1}, \forall (i, j) \in \mathcal{A}$ , to obtain  $\mathbf{f}_{ij}^{g\nu}$  and  $y_{ij}^{g\nu}, \forall (i, j) \in \mathcal{A}$ 
15:  end for
16:  Aggregation:
17:  Construct solution  $y_{ij}^{M\nu}, \forall (i, j) \in \mathcal{A}$  according to (25)
18:  Update the best feasible solution  $y^{Best} \leftarrow y^{M\nu}$ , if appropriate;
19:  Let  $\tilde{y}_{ij}^\nu \leftarrow \sum_{g \in G} p^g \mathbf{f}_{ij}^{g\nu}, \forall (i, j) \in \mathcal{A}$  where  $p^g = \sum_{s \in C_g} p^s$ 
20:  Update solution  $\hat{y}_{ij}^\nu, \forall (i, j) \in \mathcal{A}$  (Section 4.3.2)
21:  Penalty updates:
22:  Adjust penalty values  $\lambda_{ij}^{g\nu} \leftarrow \lambda_{ij}^{g\nu-1} + \rho(\mathbf{f}_{ij}^{g\nu} - \tilde{y}_{ij}^\nu)$  and  $\rho^\nu \leftarrow \alpha \rho^{\nu-1}$ 
23: end while
24: Phase II: Solve a restricted MIP problem
25: Fix the design variables for which consensus is obtained
26: Solve the restricted SMCND model (1)-(5) to obtain a final design  $y^{final}$ 
27: Update best solution,  $y^{Best} \leftarrow y^{final}$  if appropriate.

```

---

- The basic progressive hedging with single scenario subproblems where subproblems are solved using CPLEX (PHS in the following),
- The progressive hedging with multiple scenario subproblems (PHG in the following) proposed in Crainic et al. (2014),
- The reduced cost-based restriction and refinement matheuristic (RCHeur in the following) proposed in Sarayloo et al. (2018b) that currently defines the state-of-the-art heuristic algorithm for the considered problem.

After presenting the two collections of instances and the experimental setting, we start by analyzing the internal performances of the proposed ILPH on the first collection of instances, with small number of scenarios, in Section 5.2. To assess the advantage of the proposed algorithm to efficiently deal with a large number of scenarios, we provide

performance comparisons of the proposed ILPH versus PHS, PHG, RCHeur and CPLEX on the second collection of instances in Section 5.3.

## 5.1 Data and experimental setting

We consider six problem classes (R5-R10) from the set of R instances of the SMCND problem introduced in Crainic et al. (2011, 2014). Each class is characterized by a number of nodes  $|\mathcal{N}|$ , a number of arcs  $|\mathcal{A}|$ , and a number of commodities  $|\mathcal{K}|$ , specified in Table 1. For each instance class, we consider five networks, namely, networks 1, 3, 5, 7, and 9 indicating continuously increasing ratios of fixed to variable costs and total demand to total network capacity. In the first collection of instances, for each of these networks, there are instances with 16, 32 and 64 scenarios with two different levels of correlations 0.2 and 0.8. A total of 150 instances were thus obtained. In the second collection of instances, for each of the networks, there are 10 instances with 1000 scenarios which were generated according to the procedure in Boland et al. (2016).

Algorithms were coded in C++ using IBM-ILOG CPLEX 12.6.1 as the MILP solver. We used the same stopping criteria for the three methods PHS, PHG and ILPH, as in Crainic et al. (2011, 2014):  $N_{Itr} = 1000$ ,  $t_{max} = 8h$ , and  $\gamma = .1$ . The parameter  $N_{Imp}$  was set to 10 for PHS and PHG (Crainic et al. (2011, 2014)) and 4 for ILPH, according to preliminary experiments. These preliminary experiments also conducted at setting the  $\tau$ ,  $N_1$ ,  $p_1$ , and  $N_{\Delta g}$  parameters to the values .95, 3, 2, and  $|\mathcal{K}| * |C_g|$ , respectively. We let  $\hat{y}^0 \leftarrow y^{exp}$  be the initial integer solution at iteration 0 in the *Learn&Optimize* procedure, where  $y^{exp}$  is the optimal solution to the expected value (EV) problem. The EV problem is obtained by replacing the random demand variables by their expected values and solving the resulting deterministic problem. To reduce the time required to complete Phase I, the optimality tolerance parameter of CPLEX was set to 1% when solving the subproblems. This parameter is set to its default value when solving the restricted problem of the second phase. Unless otherwise specified, all other CPLEX parameters were set to their default values since preliminary experiments indicated that these settings yielded better results. All experiments were performed on a Sun Fire X4100 cluster of 16 computers. Each has two 2.6 GHz Dual-Core AMD Opteron processors and 8192 Megabytes of RAM, operating under Solaris 2.10.

## 5.2 Internal performance analysis

We first focus on the first collection of instances (with 16, 32 and 64 scenarios) for which CPLEX is able to provide either the optimal solution or at least a feasible solution within the time limit of 8 hours. In the following, we evaluate different components of the proposed ILPH and analyse their behaviour on these instances.

Table 1: Characteristics of instances

Problem	$ \mathcal{N} $	$ \mathcal{A} $	$ \mathcal{K} $
R05	10	60	25
R06	10	60	50
R07	10	82	10
R08	10	83	25
R09	10	83	50
R10	20	120	40

### 5.2.1 Effect of using *Learn&Optimize*

We first evaluated the effect of using the *Learn&Optimize* on the performance of the proposed ILPH. To do so, we considered a variant of ILPH where the subproblems are solved exactly using CPLEX, which we refer to as ILPH(E). We also investigated the impact of different network characteristics and correlations. The results of these tests are summarized in Table 2. Each row presents the results obtained on 18 instances, which are characterised by different network index values and correlations. We compare the three methods, CPLEX, ILPH and ILPH(E), in terms of optimality gaps (considering the lower bound provided by CPLEX) and computation time (in seconds) denoted by “Gap” and “Time”, respectively.

The results in Table 2 show that using *Learn&Optimize* instead of an exact method improves the performance of the algorithm. We observe that ILPH dominates ILPH(E) in terms of both solution quality (on average, % gap of 1.41 versus 1.77) and computation time (880 versus 3660 seconds). Furthermore, the results show that the proposed ILPH displays a consistent behaviour in dealing with different network characteristics. In particular, for the instances of index 3 and 5, which are considered as the most difficult instances, ILPH is able to reduce the optimality gap of CPLEX by up to 50% within around 30 times faster time (on average, 441 vs 14108 seconds). We also observe that the demand correlation has little impact on how difficult the associated problems are to solve. Overall, these results clearly highlight the significant benefit of the proposed algorithm combining PH and *Learn&Optimize*.

### 5.2.2 Effect of using $\tilde{y}$ and $\hat{y}$

We next study the effect of using the two new important components of the proposed algorithm, i.e., the proposed reference point  $\tilde{y}$  and solution  $\hat{y}$ , on the performance of ILPH. To evaluate the effect of  $\tilde{y}$ , we consider an alternative, which we refer to as ILPH- $(\tilde{y})$  where the traditional reference solution  $\bar{y} \leftarrow \sum_{g \in G} p^g y^g$  is used instead of  $\tilde{y}$ . Also, in columns “ILPH- $(\hat{y})$ ”, we evaluate the effect of using  $\hat{y}$  by replacing it with the EV solution

Table 2: Effect of using *Learn&Optimize*

Index	Corr	# of Ins	CPLEX		ILPH		ILPH(E)	
			Gap	Time	Gap(%)	Time	Gap(%)	Time
1	0.2	18	0.00	340	0.02	65	0.02	161
	0.8	18	0.00	349	0.02	65	0.02	187
3	0.2	18	5.37	14059	2.45	427	2.79	1169
	0.8	18	5.39	14208	2.45	474	2.79	1279
5	0.2	18	2.67	15686	1.70	914	1.79	5797
	0.8	18	2.69	15890	1.72	963	1.79	6004
7	0.2	18	0.31	7588	0.30	811	0.61	4207
	0.8	18	0.31	7759	0.30	861	0.61	4494
9	0.2	18	2.54	14550	2.05	2045	2.97	6180
	0.8	18	2.57	14990	2.09	2165	2.97	6842

( $y^{exp}$ ) as the initial solution for the L&Opt in all iterations of ILPH. The results in Table 3 show that using the proposed  $\tilde{y}$  provides better quality solutions with the average gap of 1.41% within almost half of the computation time, comparing to the ILPH-( $\tilde{y}$ ) where the traditional  $\bar{y}$  is used. We also observed that using the proposed initial solution  $\hat{y}$  in ILPH has a good effect on providing slightly better quality solutions in almost 70% less computation time comparing to the ILPH-( $\hat{y}$ ).

Table 3: Effect of introducing the reference point  $\tilde{y}$  and the design  $\hat{y}$ 

Pro	# of Ins	CPLEX		ILPH		ILPH-( $\tilde{y}$ )		ILPH-( $\hat{y}$ )	
		Gap	Time	Gap(%)	Time	Gap(%)	Time	Gap(%)	Time
R05	30	0.00	1411	0.12	63	0.14	110	0.12	97
R06	30	1.52	11076	0.94	451	1.02	1420	0.98	980
R07	30	0.12	2130	0.31	105	0.35	152	0.32	137
R08	30	0.96	7516	1.08	841	1.08	1764	1.08	1341
R09	30	3.48	16415	1.91	910	1.95	1971	1.91	1710
R10	30	7.01	23557	4.14	4081	4.34	8891	4.18	6170
Avg		2.18	10350	1.41	1075	1.48	2384	1.43	1739

### 5.2.3 Effect of different grouping strategies

Table 4 reports the results obtained by the proposed ILPH when different grouping strategies, i.e., “Random”, “Similar” and “Dissimilar”, are considered, which we refer to as ILPH(R), ILPH(S) and ILPH(D), respectively. The description of these grouping strategies is detailed in Crainic et al. (2014). We provide the comparison results in Table 4. We observed that ILPH(R), ILPH(S) and ILPH(D) provide almost similar optimality gaps of 1.41%, 1.50%, 1.44%, respectively. However, the computation time by ILPH(R)

is 45% and 30% faster than ILPH(S) and ILPH(D), respectively. It is also interesting to note that ILPH(R) is almost 10 times (on average) faster than CPLEX in providing better quality solutions (on average, the gap is 1.41% versus 2.18%).

Table 4: Performance comparison versus CPLEX on first collection of instances

Pro	# of Ins	CPLEX		ILPH(R)		ILPH(S)		ILPH(D)	
		Gap(%)	Time	Gap(%)	Time	Gap(%)	Time	Gap(%)	Time
R05	30	0.00	1411	0.12	118	0.14	146	0.12	136
R06	30	1.52	11076	0.94	951	0.97	1467	0.95	1274
R07	30	0.12	2130	0.31	105	0.31	149	0.31	128
R08	30	0.96	7516	1.08	841	1.12	1638	1.10	1168
R09	30	3.48	16415	1.91	840	2.02	1852	1.92	942
R10	30	7.01	23557	4.14	4081	4.48	6480	4.24	5218
Avg		2.18	10350	1.41	1075	1.50	1955	1.44	1470

The results above are encouraging and demonstrate the potential of ILPH, but the instances are inadequate to fully assess the computational advantage of ILPH. Most instances in this set can be solved to optimality by CPLEX in less than 2 hours. This is not the setting for which a decomposition approach is designed. The ILPH is designed to be used in settings where the instances are large, difficult, and cannot be solved in a reasonable amount of time when providing the MILP formulation to a solver. Indeed, solving the root relaxation may already be computationally prohibitive. In the following subsection, we present results on instances that are more appropriate to show the benefits of ILPH.

### 5.3 Comparative analysis

In this part of experiment, we focus on the difficult instances with 1000 scenarios. We provide the comparative results of the proposed ILPH versus CPLEX, in Tables 5 and 6, progressive hedging-based methods (PHS and PHG) and RCHeur in Table 7, showing the advantage of the proposed method in dealing with such difficult instances. Each row of these tables refers to 10 instances with 1000 scenarios with different characteristics mentioned in Section 5.1.

We compare, in Table 5, the proposed ILPH considering three different group sizes of 20, 50 and 100 scenarios, denoted by, ILPH(20), ILPH(50), and ILPH(100), respectively. We compare the best obtained solution versus that of CPLEX. We report the average optimality gap,  $OptGap$ , that CPLEX is able to provide after a time limit of 8 hours. Note that we consider an optimality gap of 100% for those instances for which CPLEX is unable to provide any feasible solution. In the ‘‘Gap’’ columns, we report the relative difference (%) between the best solutions found by the two algorithms (i.e., upper bounds), denoted

by  $z_{ILPH}$  and  $z_{CPX}$ , by imposing an 8 hour time limit, computed as  $100 * (z_{ILPH} - z_{CPX})/z_{ILPH}$ . The negative values refer to the cases where ILPH provides better solutions than CPLEX. In the last “Time” columns, the respective total computation times are provided.

The results show that ILPH outperforms CPLEX in all the cases with all different group sizes, providing an improvement of at least 24.72% over the best solution obtained by CPLEX, within almost one third of computation time. Comparing the different group sizes, the group size of 100 seems to yield a better performance (especially in the case of difficult instances, e.g., R6 and R9) in terms of both solution quality and computation time. In particular, for R9 instances with an average optimality gap of 66.37%, ILPH(100) provides solutions with an improvement of 63.58% compared to 52.42% when the group size is 20. This is due to the capability of L&Opt to handle subproblems with 100 scenarios, which ultimately leads to good solutions obtained by ILPH.

Table 5: Performance comparison of ILPH versus CPLEX

Pro	CPLEX		Gap(%)			Time		
	OptGap(%)	Time	ILPH(20)	ILPH(50)	ILPH(100)	ILPH(20)	ILPH(50)	ILPH(100)
R05	17.94	25805	-12.80	-12.89	-12.95	11292	12829	10842
R06	47.87	24884	-35.32	-37.12	-37.62	12682	11886	11280
R07	9.70	24063	-5.94	-5.09	-5.05	7952	7292	7494
R08	27.72	29013	-17.12	-18.36	-18.40	8682	8604	9258
R09	66.37	29901	-52.42	-62.42	-63.58	24253	23453	22630
Avg	33.92	26743	-24.72	-27.40	-27.52	12971	12892	12410

To highlight the capability of the proposed ILPH in identifying good solutions quickly, we report the comparative results of ILPH versus CPLEX in a shorter amount of time. Table 6 displays the results obtained by the two methods within 3 hours. We report in column  $ILPH/CPLEX$  the relative difference (%) between the best solutions found by the two algorithms. The negative values refer to the cases where ILPH provides better solutions than CPLEX. In the last columns, we compare the two solution methods based on the percentage of instances for which the considered solution method is able to provide a feasible solution, denoted by “Sol”. The average optimality gap provided by CPLEX is 45.59%, indicating the difficulty of these problems. We observed that ILPH is able to find feasible solutions in all considered instances, while CPLEX is unable to do so in 32% of instances. Furthermore, in terms of solution quality, ILPH provides an average improvement of -39.23% versus CPLEX, which is fascinating.

Table 7 displays the performance results of the proposed ILPH compared to PHS and PHG (after Phase 1), in terms of solution quality and computation time. In order to investigate how the proposed ILPH performs compared to the other two methods, we consider the objective function value of the best solution provided after the first phase ( $y^{Best}$ ), by ILPH, PHS, and PHG. Columns “ $ILPH/PHS$ ” and “ $ILPH/PHG$ ” report the relative improvement (in %), computed as  $100 * (z_{ILPH} - z_{PHS})/z_{ILPH}$  and  $100 * (z_{ILPH} - z_{PHG})/z_{ILPH}$ , respectively. The negative values indicate that ILPH outperforms

Table 6: Efficiency of the ILPH versus CPLEX in finding good solutions quickly

Pro	# of Ins	CPLEX	ILPH/CPLEX (%)	Sol (%)	
		OptGap (%)	3 h	ILPH	CPLEX
R05	10	22.56	-12.62	100	100
R06	10	63.90	-62.31	100	40
R07	10	15.97	-4.34	100	100
R08	10	41.36	-35.03	100	80
R09	10	84.18	-81.84	100	20
Avg		45.59	-39.23	100	68

PHS and PHG. We also report the average computation times in seconds (*Time*) and the percentage of instances with a feasible solution (*Sol.*) for the three methods.

We observed that the proposed ILPH provides solutions with an average improvement of 15.28% within one third of computation time compared to PHS. Also, ILPH dominates PHG providing 12.52% improvement on solution quality within almost half the computation time. Furthermore, ILPH is able to provide a feasible solution in all considered instances, while, PHS and PHG fail to provide any solution in 8% and 6% of the instances, respectively.

Table 7: Performance comparison of ILPH versus PHS and PHG (after Phase 1)

Pro	# of Ins	ILPH/PHS ILPH/PHG		Time			Sol (%)		
		(%)		ILPH	PHS	PHG	ILPH	PHS	PHG
R05	10	-8.45	-6.84	582	5325	4140	100	100	100
R06	10	-10.14	-9.27	4121	19277	15842	100	100	100
R07	10	-10.16	-9.20	1210	3576	2386	100	100	100
R08	10	-5.14	-5.14	13755	14822	14196	100	100	100
R09	10	-42.47	-32.17	7611	30613	21640	100	60	70
Avg		-15.28	-12.52	5456	14728	11640	100	92	94

Finally, Table 8 reports the comparative final results (after Phase II) of ILPH versus that of PHS, PHG, and RCHeur over all 50 instances. The percentages of relative improvements are calculated the same way as the Table 7. Overall, ILPH outperforms PHS, PHG and RCHeur with average relative improvements of 18.81%, 14.12%, and 11.37%, respectively. Moreover, ILPH provides these solution much faster than the other three methods PHS, PHG and RCHeur, with an average computational time of 12470 versus 29645, 23781 and 19360, respectively.

Table 8: Performance comparison of ILPH versus PHS,PHG and RCHeur (final solution)

Pro	# of Ins	ILPH/PHS	ILPH/PHG (%)	ILPH/RCHeur	Time			
					ILPH	PHS	PHG	RCHeur
Avg	50	-18.81	-14.12	-11.37	12470	29645	23781	19360

## 6 Conclusions

This paper explores the development of an efficient optimization approach to address SM-CND problems. We propose a two phase *Integrated Learning and Progressive Hedging* matheuristic to handle a large number of scenarios in the considered context. We generalize the *Learn&Optimize* procedure to efficiently address multi-scenario subproblems at each iteration of the PH algorithm. We further exploit the knowledge learned through the *Learn&Optimize* to improve the performance of the matheuristic. Specifically, we introduce a new reference point in each aggregation step of ILPH by making full use of the knowledge that is gathered regarding the promising design variables when solving the subproblems. By proceeding in this way, we inject the knowledge learned through the heuristic procedure into the PH algorithm leading to the proposed *ILPH* method, which is the main contribution of this paper.

We show, through computational experiments, that the proposed algorithm performs very well in terms of both solution quality and computation time. We provide comparative analysis that show the superiority of the proposed approach versus CPLEX, other PH-based algorithms and matheuristics proposed in the literature. The results indicate that the proposed algorithm provides impressive improvements of 27.40% and 18.81%, when dealing with large instances with 1000 scenarios, versus CPLEX and the basic PH algorithm, respectively. The analysis also indicates that ILPH should be the method of choice where high-quality solutions to instances of stochastic network problems containing a large number of scenarios need to be found quickly.

We conclude by identifying a few possible directions for future research. These include investigating whether the algorithm would be as successful in different optimization problems or other variants of MCND. Indeed, ILPH is a general-purpose algorithm and can be applied to different stochastic programs. Tailored implementations of the proposed PH algorithm can lead to quite effective matheuristics for very large stochastic programming applications. Other research avenues include considering other strategies for updating the penalties within the PH algorithm and other methods for solving the subproblems. Finally, another important research direction is to develop parallel strategies for the ILPH algorithm, which will further amplify its advantages and benefits.

## Acknowledgments

While working on this project, T.G. Crainic was Adjunct Professor with the Department of Computer Science and Operations Research, Université de Montréal. Partial funding for this project has been provided by the Natural Sciences and Engineering Council of Canada (NSERC), through its Discovery Grant program, and by the Fonds Québécois de la Recherche sur la nature et les technologies (FQRNT) through its strategic center infrastructure grant program.

## References

- Alonso-Ayuso, A., Escudero, L. F., Garín, A., Ortuño, M. T., and Pérez, G. (2003). An approach for strategic supply chain planning under uncertainty based on stochastic 0-1 programming. *Journal of Global Optimization*, 26(1):97–124.
- Birge, J. R. and Louveaux, F. (2011). *Introduction to stochastic programming*. Springer Science & Business Media.
- Boland, N., Fischetti, M., Monaci, M., and Savelsbergh, M. (2016). Proximity Benders: a decomposition heuristic for stochastic programs. *Journal of Heuristics*, 22(2):181–198.
- Crainic, T. G. (2000). Service network design in freight transportation. *European Journal of Operational Research*, 122(2):272–288.
- Crainic, T. G., Fu, X., Gendreau, M., Rei, W., and Wallace, S. W. (2011). Progressive hedging-based metaheuristics for stochastic network design. *Networks*, 58(2):114–124.
- Crainic, T. G., Gendreau, M., and Farvolden, J. M. (2000). A simplex-based tabu search method for capacitated network design. *INFORMS Journal on Computing*, 12(3):223–236.
- Crainic, T. G., Hewitt, M., Maggioni, F., and Rei, W. (2016). Partial decomposition strategies for two-stage stochastic integer programs. Publication CIRRELT-2016-37, Centre interuniversitaire de recherche sur les réseaux d’entreprise, la logistique et le transport, Université de Montréal, Montréal, QC, Canada.
- Crainic, T. G., Hewitt, M., and Rei, W. (2014). Scenario grouping in a progressive hedging-based meta-heuristic for stochastic network design. *Computers & Operations Research*, 43:90–99.
- Crainic, T. G., Maggioni, F., Perboli, G., and Rei, W. (2018). Reduced cost-based variable fixing in two-stage stochastic programming. *Annals of Operations Research*.

- Dupačová, J., Consigli, G., and Wallace, S. W. (2000). Scenarios for multistage stochastic programs. *Annals of Operations Research*, 100(1-4):25–53.
- Escudero, L. F., Garín, M. A., Merino, M., and Pérez, G. (2012). An algorithmic framework for solving large-scale multistage stochastic mixed 0–1 problems with nonsymmetric scenario trees. *Computers & Operations Research*, 39(5):1133–1144.
- Ghamlouche, I., Crainic, T. G., and Gendreau, M. (2003). Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design. *Operations Research*, 51(4):655–667.
- Ghamlouche, I., Crainic, T. G., and Gendreau, M. (2004). Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. *Annals of Operations Research*, 131(1-4):109–133.
- Haugen, K. K., Løkketangen, A., and Woodruff, D. L. (2001). Progressive hedging as a meta-heuristic applied to stochastic lot-sizing. *European Journal of Operational Research*, 132(1):116–122.
- Hewitt, M., Nemhauser, G. L., and Savelsbergh, M. W. (2010). Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. *INFORMS Journal on Computing*, 22(2):314–325.
- Higle, J. L. and Wallace, S. W. (2003). Sensitivity analysis and uncertainty in linear programming. *Interfaces*, 33(4):53–60.
- Kall, P., Wallace, S. W., and Kall, P. (1994). *Stochastic programming*. Springer.
- King, A. J. and Wallace, S. W. (2012). *Modeling with stochastic programming*. Springer Science & Business Media.
- Lium, A.-G., Crainic, T. G., and Wallace, S. W. (2009). A study of demand stochasticity in service network design. *Transportation Science*, 43(2):144–157.
- Løkketangen, A. and Woodruff, D. L. (1996). Progressive hedging and tabu search applied to mixed integer (0, 1) multistage stochastic programming. *Journal of Heuristics*, 2(2):111–128.
- Magnanti, T. L. and Wong, R. T. (1984). Network design and transportation planning: Models and algorithms. *Transportation Science*, 18(1):1–55.
- Rahmaniani, R., Crainic, T. G., Gendreau, M., and Rei, W. (2017). The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817.
- Rahmaniani, R., Crainic, T. G., Gendreau, M., and Rei, W. (2018). Accelerating the benders decomposition method: Application to stochastic network design problems. *SIAM Journal on Optimization*, 28(1):875–903.

- Rockafellar, R. T. and Wets, R. J.-B. (1991). Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16(1):119–147.
- Sarayloo, F., Crainic, T. G., and Rei, W. (2018a). A learning-based matheuristic for stochastic multicommodity network design. Publication CIRRELT-2018-12, Centre interuniversitaire de recherche sur les réseaux d’entreprise, la logistique et le transport, Université de Montréal, Montréal, QC, Canada.
- Sarayloo, F., Crainic, T. G., and Rei, W. (2018b). A reduced cost-based restriction and refinement matheuristic for stochastic network design. Publication CIRRELT-2018-32, Centre interuniversitaire de recherche sur les réseaux d’entreprise, la logistique et le transport, Université de Montréal, Montréal, QC, Canada.
- Schütz, P., Tomaszgard, A., and Ahmed, S. (2009). Supply chain design under uncertainty using sample average approximation and dual decomposition. *European Journal of Operational Research*, 199(2):409–419.
- Thapalia, B. K., Crainic, T. G., Kaut, M., and Wallace, S. W. (2011). Single-commodity network design with stochastic demand and multiple sources and sinks. *INFOR: Information Systems and Operational Research*, 49(3):193–211.
- Thapalia, B. K., Crainic, T. G., Kaut, M., and Wallace, S. W. (2012a). Single-commodity network design with random edge capacities. *European Journal of Operational Research*, 220(2):394–403.
- Thapalia, B. K., Wallace, S. W., Kaut, M., and Crainic, T. G. (2012b). Single source single-commodity stochastic network design. *Computational Management Science*, 9(1):139–160.
- Van Slyke, R. M. and Wets, R. (1969). L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17(4):638–663.
- Wallace, S. W. (2000). Decision making under uncertainty: Is sensitivity analysis of any use? *Operations Research*, 48(1):20–25.