A Learning-Based Matheuristic for Stochastic Multicommodity Network Design

Fatemeh Sarayloo
Teodor Gabriel Crainic
Walter Rei

February 2018

CIRRELT-2018-12

A Learning-Based Matheuristic for Stochastic Multicommodity Network Design

Fatemeh Sarayloo[1,2], Teodor Gabriel Crainic[1,3,*], Walter Rei[1,3]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

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

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

Abstract. This paper proposes a solution approach for the Multicommodity Capacitated Fixed-charge Network Design problem with uncertain demand, modeled as a two-stage stochastic program. The proposed learning-based matheuristic combines heuristic-search techniques with mathematical programming. It provides a systematic approach to identify structures of good-quality solutions by gradually considering scenarios and their influences on design decisions. Extensive computational experiments illustrate the efficiency of the proposed matheuristic in obtaining high-quality solutions with limited computational efforts.

Keywords: Stochastic capacitated network design, uncertain multicommodity demand, two-stage formulation, matheuristic.

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.

---

* Corresponding author: teodorgabriel.crainic@cirrelt.net

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2018

1 Introduction

Network Design (ND) defines an important class of combinatorial optimization problems that naturally appear in a large number of applications including transportation, logistics and telecommunications. Given a network where all or a subset of arcs may be used only if selected ("opened") by paying a so-called fixed cost, the *Multicommodity Capacitated Fixed-charge Network Design* (*MCND*) formulation arises when it is required to route at minimum total cost a set of given commodities making up the demand between different pairs of origin and destination nodes of the network. The optimization aims to determine the arcs to select and the commodity flows within the resulting network, the total cost being computed as the sum of the fixed costs of the selected arcs and the cost of transporting the commodities. Surveys on network design may be found in Magnanti and Wong (1984), Minoux (1989), and Crainic (2000).

Critical problem parameters such as demands, costs, and capacities are often uncertain, and many applications require the uncertainty to be explicitly considered when modeling, yielding *Stochastic MCND* (*SMCND*) formulations. We focus on demand uncertainty addressed through two-stage stochastic programs (Birge and Louveaux, 2011), where design decisions are made in the first stage before the actual demand is realized, while second-stage flow-routing decisions adjust the first-stage solution to the observed demand realization. The general goal of SND formulations is to find a single optimal design solution for the range of possible demand realizations. To address such problems, the demand uncertainty is often represented through scenario decomposition, i.e., through a set of values for the uncertain demands, the *scenarios*, together with the associated probabilities. The resulting large-scale mixed integer program (MIP), referred to as the extensive form (EF) in Birge and Louveaux (2011), is difficult to solve exactly in most cases as, first, deterministic network design problems are NP-hard in all but trivial cases (Magnanti and Wong, 1984) and, second, modeling uncertainty with scenarios generally yields very large instances (Crainic et al., 2011). Heuristic solution methods are thus proposed aiming to identify "good"-quality solutions within reasonable computing efforts.

Solving deterministic formulations is generally easier, compared to stochastic ones. Consequently, a number of studies attempted to analyze the information provided by solutions to deterministic variants of stochastic formulations to infer information about the stochastic solutions and simplify addressing the stochastic models. Although solving deterministic formulations cannot replace addressing the stochastic one and solutions to the former may be very bad for the latter, it has been observed that stochastic solutions retain parts of deterministic solutions. Yet, no systematic procedure to identify these parts, providing the basis for efficient meta-heuristics for the MCND may be found in the literature.

We aim to contribute addressing these challenges by proposing an innovative *sys-*

tematic learning mechanism to extract information regarding the solution structure of a stochastic problem out of the solutions of particular combinations of scenarios. In the context of network design, this information takes the form of design decisions that are common to high-quality (i.e., optimal or near optimal) solutions obtained by gradually considering scenarios and their interactions.

We also propose the *Learn-and-Optimize* matheuristic, which jointly makes use of the learning mechanism to infer a set of promising design variables, and a state-of-the-art MIP solver to address a reduced problem. To explore the search space more efficiently and achieve improved results, we apply gradually enlarge the reduced problems to be solved by the MIP solver.

Learning, that is, deriving knowledge relative to the solution structure, is not a new concept in addressing stochastic network design models. Yet, how this knowledge is obtained and exploited are key success factors. The contribution of this paper, therefore, is threefold. First, we introduce and formally describe a new optimization-based learning mechanism to identify the solution structure of stochastic network design problems. Although presented in the context of this complex problem setting, the proposed mechanism can be adapted to many other combinatorial optimization problems. Second, we propose a new matheuristic framework, integrating the proposed learning mechanism, which efficiently obtains high-quality solutions, outperforming a state-of-the-art commercial MIP solver in solution quality and computational efforts, particularly as the instance dimension increases. Third, we present the results of extensive computational experiments to asses the merit and limits of the proposed methodology.

The rest of paper is organized as follows. In We recall the two-stage formulation of the SMCND and briefly review relevant literature in Section 2. Section 3 introduces the main ideas of the learning mechanism we propose. while Section 4 details the matheuristic. We present and analyze the experimental results in Section 5, and provide concluding remarks in Section 6.

2 Problem Description and Literature Review

We first recall the two-stage stochastic formulation (the *a priori* optimization model Birge and Louveaux, 2011) of the stochastic multi-commodity capacitated fixed-cost network design problem. We then present a brief review of the relevant literature.

2.1 Two-stage SMCND formulation

One distinguishes two sets of decisions in an a priori formulation, according to the moment in time, the *stage*, the decision is taken and the type of information available. The first stage corresponds to decisions that need to be taken here-and-now, based on estimations of future demand and prior to the realization of uncertainty. The second stage and its recourse variables correspond to the decisions made repeatedly, given the restrictions imposed by the first stage, once new information is revealed and the uncertainty is resolved. Traditionally, in the case of stochastic two-stage network design problems with uncertain demands, the first stage consists of deciding on the configuration of network, i.e., the design decisions, and the second-stage consists in optimizing the commodity flow distribution of the observed demand, on the restricted configuration imposed by the first stage.

We use the two-stage stochastic formulation of the SMCND by (Crainic et al., 2011). Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be a directed network with $\mathcal{N}$ representing a finite set of nodes and $\mathcal{A}$ a finite set of potential arcs. Let $\mathcal{K}$ be the set of commodities each characterized by a unique pair of origin-destination nodes. Let $\mathcal{S}$ be the set of scenarios, where p^s is the probability of scenario $s \in \mathcal{S}$. We introduce binary variable y_{ij}, which indicates if arc (i,j) is included, or not, in the network in the first stage, where f_{ij} is the fixed cost of doing so. Once the design variables are decided upon, in the second stage, x_{ij}^{ks} represents the amount of commodity k's demand that flows on arc (i,j) in the solution for scenario s. The variable cost per unit of flow on arc (i,j) is denoted by c_{ij}. The mathematical formulation is:

$$\begin{aligned}
\text{minimize} \quad & \sum_{(i,j)\in\mathcal{A}} f_{ij}y_{ij} + \sum_{s\in\mathcal{S}} p^s \sum_{k\in\mathcal{K}} \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij}^{ks} && && (1)\\
\text{subject to} \quad & \sum_{j\in\mathcal{N}^+(i)} x_{ij}^{ks} - \sum_{j\in\mathcal{N}^-(i)} x_{ji}^{ks} = d_i^{ks}, && \forall i \in \mathcal{N},\ \forall k \in \mathcal{K},\ \forall s \in \mathcal{S} && (2)\\
& \sum_{k\in\mathcal{K}} x_{ij}^{ks} \le u_{ij}y_{ij}, && \forall (i,j) \in \mathcal{A},\ \forall s \in \mathcal{S} && (3)\\
& y_{ij} \in \{0,1\}, && \forall (i,j) \in \mathcal{A} && (4)\\
& x_{ij}^{ks} \ge 0, && \forall (i,j) \in \mathcal{A},\ \forall k \in \mathcal{K},\ \forall s \in \mathcal{S} && (5)
\end{aligned}$$

The objective function accounts for the total system cost, consisting of the fixed cost for the included arcs and the expectation of routing costs taken over all the realizations of demand scenarios. Constraints (2) represent the flow conservation equations in each scenario, requiring that demand of commodity $k \in \mathcal{K}$ be routed from its origin node to its destination. Therefore, assuming that ω^{ks} is the demand volume of commodity k in scenario s, the parameter d_i^{ks} is set to ω^{ks} if node i is the origin of commodity k, $-\omega^{ks}$ if node i is the destination of commodity k, and 0 otherwise. Linking Constraints (3) ensure that the same design is used in each scenario and that arc capacity u_{ij} is never violated.

Constraints (4) and (5) are integrality and non-negativity constraints, respectively.

2.2 Literature review

The stochastic network design problem belongs to the class of stochastic mixed integer programs, its complexity stemming not only from stochasticity, but also from the lost convexity of the integrality property.

Most of the existing literature on stochastic network design problems models uncertainty through a given set of scenarios. See Dupačová et al. (2000) and King and Wallace (2012) for an overview of scenario generation methods. The deterministic network design problem is NP-Hard (Magnanti and Wong, 1984) and computationally complex. Scenarios compound the difficulty by significantly increasing the dimension of the instances. Hence, some methods, e.g., the Sample Average Approximation (SAA) approach, includes sampling into the iterative solution procedure (see, e.g., Santoso et al., 2005; Azaron et al., 2008; Schütz et al., 2009; Bidhandi and Yusuff, 2011, for applications to network design).

Once the set of scenarios is defined, standard MIP solvers can be used for small problem instances to directly solve the extensive form of the problem in which all scenarios are considered simultaneously (e.g., Tsiakis et al., 2001; Azaron et al., 2008). For most network design settings and cases, this approach is not appropriate however, as formulations are either too large, or too complex, or both. Decomposition is then often called upon.

The basic idea behind decomposition methods is to divide a large-scale SMIP problem into several subproblems, along scenarios or stages, and solve the smaller-sized subproblems separately in a decomposition-coordination manner. Decomposition approaches generally fall into two groups based on how the two-stage formulation is decomposed. The first group consists of vertical-directive methods that decompose the problem according to scenarios. The Progressive Hedging (PH) method, proposed originally by Rockafellar and Wets (1991), belongs to this category and is the foundation of a number of meta-heuristics for stochastic network design (e.g., Crainic et al., 2011, 2014b; Lanza et al., 2017). In the second group, horizontal-directive methods decompose a stochastic network design problem stage-wise into a master problem for the first-stage variables (i.e., the design variables) and a number of subproblems for the second stage variables (i.e., a network flow problem for each scenario). The L-shaped method Van Slyke and Wets (1969) belongs to this category and has been used for the stochastic network design (e.g., Riis and Andersen, 2000; Smith et al., 2004; Santoso et al., 2005; Bidhandi and Yusuff, 2011; Crainic et al., 2014a, 2016).

The impact of stochasticity on the solution structure and the role solutions to some deterministic problem variants may play in identifying this structure is another significant research direction that has been explored for stochastic network design. It is well known that solutions derived from stochastic programs are, in general, structurally different from those derived from their deterministic counterparts defined by using the mean of demand distributions. (Wallace, 2000; Higle and Wallace, 2003; Lium et al., 2009). A number of studies focusing on discerning similarities and differences between solution structures in deterministic and stochastic problem variants (e.g., Maggioni and Wallace, 2012; Thapalia et al., 2011, 2012a,b; Wang et al., 2016) have also shown, however, that components of the deterministic solutions are to be found in stochastic ones. Despite these investigations into using the partial information provided by deterministic solutions in addressing stochastic models, there is still no methodology to systematically explore it and identify the elements that would guide a meta-heuristic towards good-quality design solutions to large stochastic instances. We propose such a methodology in the next sections.

3 The Learning Mechanism and Heuristic

Several approaches in the literature could be qualified as integrating "learning". Knowledge relative to a stochastic solution may thus be derived from a single deterministic solution only (using the expected value in most cases), as in Maggioni and Wallace (2012) and Wang et al. (2016). One notices, however, that there is little learning in these approaches from the stochastic information present in the scenarios. Scenario-decomposition based methods (e.g., Rockafellar and Wets, 1991; Crainic et al., 2011, 2014b), on the other hand, learn from the multiple solutions provided by solving the deterministic formulations resulting from decomposing along scenarios. Most of the information is lost, however, as solutions are aggregated to compute a so-called consensus solution.

We propose a mechanism to overcome these limitations. We believe that one can derive a good deal of knowledge out of the scenarios, and that better solutions can be obtained more efficiently with a higher level of learning. We aim to provide such a higher level of learning by systematically exploiting the design information obtained by considering not only individual scenarios but also scenario combinations and interactions.

We introduce the concept of *Artificial Demand Scenario* (*ADS*) built out of particular combinations of scenarios. One then learns by iteratively building ADSs, solving the associated problems, and gradually building an image of design variables potentially belonging to good solutions to the stochastic formulation. The result of this learning mechanism is then used to guide the search to higher quality solutions.

We define the Artificial Demand Scenario in Section 3.1, while Sections 3.2 and 3.3 describe the algorithmic components making up the proposed learning mechanism.

3.1 Artificial Demand Scenario

We define an *Artificial Demand Scenario* in this paper as a combination of the demands of two scenarios. Let $\boldsymbol{d(s_\alpha)}$ and $\boldsymbol{d(s_\beta)}$ be the demand vectors of size $|\mathcal{K}|$ of scenarios $s_\alpha, s_\beta \in \mathcal{S}$:

$$\boldsymbol{d(s_\alpha)} = \begin{pmatrix} d_1(s_\alpha) \\ d_2(s_\alpha) \\ d_3(s_\alpha) \\ \vdots \\ d_{|\mathcal{K}|}(s_\alpha) \end{pmatrix} \text{ and } \boldsymbol{d(s_\beta)} = \begin{pmatrix} d_1(s_\beta) \\ d_2(s_\beta) \\ d_3(s_\beta) \\ \vdots \\ d_{|\mathcal{K}|}(s_\beta) \end{pmatrix}.$$

An *Artificial Demand Scenario* $\boldsymbol{\delta(s_\alpha, s_\beta)} \in \boldsymbol{\Delta}(\mathrm{s}_\alpha, \mathrm{s}_\beta)$, $\forall s_\alpha, s_\beta \in \mathcal{S}$ is then defined as a demand vector of the same size, where the element $k = 1, 2, 3, \ldots, |\mathcal{K}|$ contains the demand value associated with commodity k in scenario s_α or scenario s_β, that is,

$$\boldsymbol{\delta(s_\alpha, s_\beta)} = \begin{pmatrix} \delta_1(s_\alpha, s_\beta) \\ \delta_2(s_\alpha, s_\beta) \\ \delta_3(s_\alpha, s_\beta) \\ \vdots \\ \delta_{|\mathcal{K}|}(s_\alpha, s_\beta) \end{pmatrix}, \text{ such that } \ \delta_k(s_\alpha, s_\beta) = d_k(s_\alpha) \vee d_k(s_\beta), \forall k \in \mathcal{K}.$$

To illustrate, consider two demand vectors $\boldsymbol{d(s_1)}$ and $\boldsymbol{d(s_2)}$ containing six commodities each. Three possible artificial demand scenarios are:

$$\boldsymbol{d(s_1)}{=}\begin{pmatrix}10\\20\\60\\30\\90\\70\end{pmatrix}, \quad \boldsymbol{\delta_1(s_1,s_2)}{=}\begin{pmatrix}45\\20\\60\\30\\90\\70\end{pmatrix}, \quad \boldsymbol{\delta_2(s_1,s_2)}{=}\begin{pmatrix}10\\25\\75\\30\\90\\70\end{pmatrix}, \quad \boldsymbol{\delta_3(s_1,s_2)}{=}\begin{pmatrix}10\\20\\75\\85\\95\\70\end{pmatrix}, \quad ..., \quad \begin{pmatrix}45\\25\\75\\85\\95\\15\end{pmatrix}{=}\boldsymbol{d(s_2)}$$

One generates an ADS by selecting a number of demand elements in one of the two scenarios and the rest in the other one. The generation process is thus determined by how many elements are selected and how this selection is performed. In all generality, there are several possible ways to perform the selection and we define $\mathcal{R} = \{r\}$ as the set of *selection rules*. An ADS $\boldsymbol{\delta^{mr}(s_\alpha, s_\beta)} \in \Delta(s_\alpha, s_\beta)$ is then generated by applying the selection rule $r \in \mathcal{R}$ to pick up m elements in $d(s_\beta)$ and $|\mathcal{K}| - m$ elements in $d(s_\alpha)$. Let $\Delta^{mr}(s_\alpha, s_\beta)$ be set of ADSs of type *m-combination*, $m = 1, 2, 3, \ldots, \mathcal{K} - 1$, generated by the selection rule r.

Commodities are an important component of network design problems and, thus, we want the set $\Delta^{mr}(s_\alpha, s_\beta)$, used by the learning mechanism, to be such that each commodity appears in an least one of its ADSs. Define the *operator* $\oplus$ such that

$$\boldsymbol{\delta}'''(\boldsymbol{s_\alpha}, \boldsymbol{s_\beta}) = \boldsymbol{\delta}'(\boldsymbol{s_\alpha}, \boldsymbol{s_\beta}) \oplus \boldsymbol{\delta}''(\boldsymbol{s_\alpha}, \boldsymbol{s_\beta}),\ \boldsymbol{\delta}'(\boldsymbol{s_\alpha}, \boldsymbol{s_\beta}), \boldsymbol{\delta}''(\boldsymbol{s_\alpha}, \boldsymbol{s_\beta}) \in \Delta^{mr}(s_\alpha, s_\beta)$$

yields

$$\delta_k'''(s_\alpha, s_\beta) = \begin{cases} d_k(s_\beta) & \text{if } (\delta_k'(s_\alpha, s_\beta) = d_k(s_\beta)) \vee (\delta_k''(s_\alpha, s_\beta) = d_k(s_\beta)), \\ d_k(s_\alpha) & \text{otherwise,} \end{cases} \tag{6}$$

for all $k \in \mathcal{K}$.

A set $\Delta^{mr}(s_\alpha, s_\beta) = \{\boldsymbol{\delta}_\theta^{mr}(s_\alpha, s_\beta)\}_{\{\theta=1,2,\ldots,N_m\}}$, of cardinality N_m, will then ensure that each commodity is considered in at least one of its ADSs, if

$$\boldsymbol{d(s_\beta)} = \boldsymbol{\delta}_1^{mr}(s_\alpha, s_\beta) \oplus \boldsymbol{\delta}_2^{mr}(s_\alpha, s_\beta) \oplus \ldots \oplus \boldsymbol{\delta}_{N_m}^{mr}(s_\alpha, s_\beta). \tag{7}$$

Note that the minimum cardinality of set $\Delta^{mr}(s_\alpha, s_\beta)$ satisfying relation (7) is $N_m = \lceil \frac{|\mathcal{K}|}{m} \rceil$. To illustrate, in the following case of two scenarios and three ADSs, the set $\{\boldsymbol{\delta}_1, \boldsymbol{\delta}_2, \boldsymbol{\delta}_3\}$ satisfies relation (7), while the set $\{\boldsymbol{\delta}_1, \boldsymbol{\delta}_2, \boldsymbol{\delta}_4\}$ does not.

$$\boldsymbol{d(s_1)} = \begin{pmatrix} 10 \\ 20 \\ 60 \\ 30 \\ 90 \\ 70 \end{pmatrix}, \quad \boldsymbol{\delta}_1 = \begin{pmatrix} 45 \\ 25 \\ 60 \\ 30 \\ 90 \\ 70 \end{pmatrix}, \quad \boldsymbol{\delta}_2 = \begin{pmatrix} 10 \\ 20 \\ 75 \\ 85 \\ 90 \\ 70 \end{pmatrix}, \quad \boldsymbol{\delta}_3 = \begin{pmatrix} 10 \\ 20 \\ 60 \\ 30 \\ 95 \\ 15 \end{pmatrix}, \quad \boldsymbol{\delta}_4 = \begin{pmatrix} 45 \\ 20 \\ 60 \\ 30 \\ 90 \\ 15 \end{pmatrix}, \quad \begin{pmatrix} 45 \\ 25 \\ 75 \\ 85 \\ 95 \\ 15 \end{pmatrix} = \boldsymbol{d(s_2)}$$

Several methods can be used to construct the set $\Delta^{mr}(s_\alpha, s_\beta)$ but, to avoid introducing "noise" in evaluating the learning mechanisms, we use a simple procedure in this paper. Algorithm 1 builds a set $\Delta^{mr}(s_\alpha, s_\beta)$ of minimum cardinality $N_m = \lceil \frac{|\mathcal{K}|}{m} \rceil$, through the random selection of the demand values (i.e., $\mathcal{R} = \{r = \text{random selection}\}$) of two given scenarios. The procedure first builds $N_m - 1$ ADSs (lines 3 - 5) by iteratively selecting m commodities to copy from $d(s_\beta)$. Each commodity may belong to one ADS only. The last ADS is built from the remaining commodities, if any.

3.2 Partial Learning - The Scenario-Pair Case

Given a pair of scenarios $s_\alpha, s_\beta \in \mathcal{S}$ and a feasible design $\bar{y}$, the *partial-learning* mechanism proceeds by exploring the solution characteristics associated to each ADS $\delta(s_\alpha, s_\beta) \in \Delta(s_\alpha, s_\beta)$ to extract information regarding promising design variables. The exploration is performed by solving an *Artificial-Recourse Problem*, $ARP(\delta, \bar{y})$, for each artificial demand scenario $\delta \in \Delta(s_\alpha, s_\beta)$.

Algorithm 1 $Construct(s_\alpha, s_\beta, m, r = \text{random selection})$

```
1: Initialization: $N_m \leftarrow \lceil \frac{|\mathcal{K}|}{m} \rceil$, $\bar{\mathcal{K}} \leftarrow \mathcal{K}$, $\theta \leftarrow 1$;
2: repeat
3:     Randomly choose $m$ commodities $\mathcal{K}^m \subseteq \bar{\mathcal{K}}$, and create ADS $\delta_\theta^{mr}(s_\alpha, s_\beta)$ with the corresponding demand values from scenario $s_\beta$; $\theta \leftarrow \theta + 1$;
4:     Update $\bar{\mathcal{K}}$: $\bar{\mathcal{K}} := \bar{\mathcal{K}} \setminus \mathcal{K}^m$;
5: until $\theta = N_m$
6: Generate $\delta_\theta^{mr}(s_\alpha, s_\beta)$ by choosing the remaining commodities in $\bar{\mathcal{K}}$ out of $d(s_\beta)$;
7: return $\Delta^{mr}(s_\alpha, s_\beta)$
```

To define the $ARP(\delta, \bar{y})$, we separate the set of arcs $\mathcal{A}$ according to the given design $\bar{y}$. Then, $\mathcal{A} = \mathcal{A}^0 \cup \mathcal{A}^1$, where $\mathcal{A}^0 = \{(i,j) | (i,j) \in \mathcal{A}, \bar{y}_{ij} = 0\}$ and $\mathcal{A}^1 = \{(i,j) | (i,j) \in \mathcal{A}, \bar{y}_{ij} = 1\}$ are the sets of closed and open arcs in $\bar{y}$, respectively. We then define a modified arc variable cost $\bar{c}_{ij}$ by linearizing the fixed cost of the closed arcs

$$\bar{c}_{ij} = \begin{cases} c_{ij} + \frac{f_{i,j}}{u_{ij}}, & \forall (i,j) \in A^0, \\ c_{ij}, & \forall (i,j) \in A^1 \end{cases} \tag{8}$$

and solve the $ARP(\delta, \bar{y})$ multi-commodity network flow problem

$$ARP(\delta, \bar{y}): \text{ minimize } \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} \bar{c}_{ij} x_{ij}^k \tag{9}$$

$$\text{subject to} \quad \sum_{j \in \mathcal{N}^+(i)} x_{ij}^k - \sum_{j \in \mathcal{N}^-(i)} x_{ji}^k = \delta_k^i, \quad \forall i \in \mathcal{N}, \ \forall k \in \mathcal{K} \tag{10}$$

$$\sum_{k \in \mathcal{K}} x_{ij}^k \leq u_{ij}, \quad \forall (i,j) \in \mathcal{A} \tag{11}$$

$$x_{ij}^k \geq 0, \quad \forall (i,j) \in \mathcal{A} \setminus A^0, \ \forall k \in \mathcal{K} \tag{12}$$

where

$$\delta_k^i = \begin{cases} \delta_k & \text{if } i = o(k) \\ -\delta_k & \text{if } i = d(k) \\ 0 & \textit{otherwise.} \end{cases} \tag{13}$$

Solving $ARP(\delta, \bar{y})$ yields the solution $x_{ij}(\delta) = \sum_{k \in \mathcal{K}} x_{ij}^k, \forall (ij) \in \mathcal{A}$, with $\mathcal{A}_\delta = \{(i,j) \mid x_{ij}(\delta) > 0\}$. We define a corresponding design solution as $y_{ij}^\delta = 1$, when $x_{ij}(\delta) > 0$, and 0, otherwise. It is noteworthy that some of the arcs in $\mathcal{A}^0$, closed in $\bar{y}$, may be open in y_{ij}^δ to satisfy the demand vector δ. This modification to the design vector following the modification of the recourse problem inspired the procedure and the ARP name. These modifications capture the interactions occurring in the integration of two scenarios within δ, yielding partial information regarding the design arcs required to address the uncertainty captured by the two scenarios. Repeating this procedure for different combinations, artificial scenarios, builds the knowledge we seek.

The partial-learning approach is described in Algorithm 2. The set of Artificial Demand Scenarios, $\Delta^{mr}(s_\alpha, s_\beta)$, was constructed previously using Algorithm 1. We define the *frequency memory*, F'_{ij}, representing how often arc (i, j) has been used in the solutions of the different $ARP(\bar{y}, \delta)$. We also define $\mathcal{A}^\Delta$, the set of design arcs used in at least one ARP, and $\mathcal{A}^{\alpha\text{-}\beta}$, the set of promising design variables to be identified by the procedure.

Algorithm 2 PartialLearning $(\bar{y}, \Delta^{mr}(s_\alpha, s_\beta))$

```
1: Initialization: $F'_{ij} \leftarrow 0, \forall (ij) \in A$, $\mathcal{A}^{\alpha\text{-}\beta} \leftarrow \emptyset$; $\Delta^{mr}(s_\alpha, s_\beta) \leftarrow \emptyset$;
2: repeat
3:     Randomly choose an artificial demand scenario $\delta \in \Delta^{mr}(s_\alpha, s_\beta)$;
4:     Solve $ARP(\delta, \bar{y})$ yielding $x_{ij}(\delta), \forall (ij) \in \mathcal{A}$;
5:     Identify $\mathcal{A}_\delta$ and compute $y^\delta_{ij}$, $\forall (i,j) \in \mathcal{A}_\delta$;
6:     Update $\mathcal{A}^\Delta := \mathcal{A}^\Delta \bigcup \mathcal{A}_\delta$ and $F'_{ij} := F'_{ij} + 1$, for all $(i, j) \in \mathcal{A}_\delta$;
7:     Remove $\delta$ from $\Delta^{mr}(s_\alpha, s_\beta)$;
8: until  $\Delta^{mr}(s_\alpha, s_\beta) = \emptyset$;
9: Normalize frequencies $F'_{ij} := F'_{ij}/max\{F'_{ij} | (i, j) \in \mathcal{A}^\Pi\}$, $\forall (i, j) \in \mathcal{A}^\Delta$;
10: for all  $(i, j) \in \mathcal{A}^\Delta$ do
11:     if  $F'_{ij} \geq \tau'$ then  $\mathcal{A}^{\alpha\text{-}\beta} \leftarrow \mathcal{A}^{\alpha\text{-}\beta} \cup \{(i, j)\}$;
12:     end if
13: end for
14: Return the set of promising design variables $\mathcal{A}^{\alpha\text{-}\beta}$.
```

The main loop (lines 3 to 7) iterates over the artificial demand scenarios in $\Delta^{mr}(s_\alpha, s_\beta)$, each being discarded, line 7, after it has been examined. The procedure stops when the set of artificial demand scenarios becomes empty. The $ARP(\bar{y}, \delta)$ is solved for each $\delta \in \Delta^{mr}(s_\alpha, s_\beta))$, to distribute the demand of δ (line 4). The corresponding design vector is created (line 5), while the set of used design arcs and the frequency memory are updated on line 6. Once artificial demand scenarios in $\boldsymbol{\delta} \in \Delta^{mr}(s_\alpha, s_\beta)$ are treated, the procedure returns the most frequently used arcs for the scenario pair (s_α, s_β), given a threshold τ') (lines 10-13).

3.3 The Learning Procedure

The *learning* mechanism repeatedly applies the partial-learning procedure to various pairs of scenarios to extract global information on promising design variables. The goal is to then use the collected information obtained by the learning mechanism to identify appropriate parts of the solution space where an exact solver may intensify the search (Section 4).

The Learning Procedure, described in Algorithm 3, consists of two phases. The partial-learning procedure (Algorithm 2) is first used for each pair of scenarios in a given set Π (all pairs are considered in the mechanism of this paper), identifying the corresponding promising design variables and frequency vectors. This information is used to gradually build the global set of promising design variables, $\mathcal{A}^{\Pi}$, and frequency vector F. The second phase identifies the set of most promising design arcs, $\mathcal{A}^{\star}$, as the most frequently selected ones (given a threshold τ).

Algorithm 3 Learning($\bar{y}, \Pi, m, r$)

```
1: Initialization: $F_{ij} \leftarrow 0$, $\forall (i,j) \in \mathcal{A}$, $\mathcal{A}^{\Pi} \leftarrow \emptyset$, $\mathcal{A}^{\star} \leftarrow \emptyset$;
2: for all pairs $(s_\alpha, s_\beta) \in \Pi$ do
3:     $\Delta^{mr}(s_\alpha, s_\beta) \leftarrow Construct(s_\alpha, s_\beta, m, r)$;
4:     $\mathcal{A}^{\alpha\text{-}\beta} \leftarrow PartialLearning(\bar{y}, \Delta^{mr}(s_\alpha, s_\beta))$;
5:     Update the frequency memory $F_{ij} := F_{ij} + 1$, $\forall (i,j) \in \mathcal{A}^{\alpha\text{-}\beta}$;
6:     Update $\mathcal{A}^{\Pi} := \mathcal{A}^{\Pi} \bigcup \mathcal{A}^{\alpha\text{-}\beta}$;
7: end for
8: Normalize frequencies $F_{ij} := F_{ij}/max\{F_{ij} | (i,j) \in \mathcal{A}^{\Pi}\}$, $\forall (i,j) \in \mathcal{A}^{\Pi}$;
9: for arc $(i,j) \in \mathcal{A}^{\Pi}$ do
10:     if $F_{ij} \geq \tau$ then $\mathcal{A}^{\star} := \mathcal{A}^{\star} \cup \{(i,j)\}$
11:     end if
12: end for
13: Return the set of promising design variables $\mathcal{A}^{\star}$.
```

4 The Matheuristic

The *Learn&Optimize* matheuristic we propose iteratively executes a *learning step*, to identify a promising set of design variables, and a *partial-optimization step*, where a number of promising variables are fixed and the reduced-size formulation is solved exactly.

Algorithm 4 details the matheuristic, where $\bar{y}$ is an initial design solution, Π, the set of scenario pairs, $\mathcal{M} = \{m_1, m_2, \ldots, m_{MAX}\}$, the set of *m-combinations*, $\mathcal{R}$, the set of *selection rules*, the last three parameters being used to generate artificial demand scenarios, p, the initial percentage of promising variables to fix, $\Delta(p)$, the reduction of the current value of p is the diversification step, and q, the number of consecutive iterations with no improvement activating a diversification step. A computational time limit is the stopping criterion in this version of the algorithm.

At each iteration ν, the matheuristic first performs the procedure of Section 3.3 with current $m_i \in \mathcal{M}$ and $r_l \in \mathcal{R}$ parameters (Line 3), to learn and build statistics on solution characteristics, yielding the set of promising design variables, $\mathcal{A}^{\star}_{\nu}$. The subproblem obtained by fixing p percent of variables in $\mathcal{A}^{\star}_{\nu}$ to the value 1 is then solved (line 4). The global solution is updated when an improvement occurs (lines 5-6). A diversification step is performed when no improvement is achieved after q consecutive iterations (lines

Algorithm 4 Learn&Optimize ($\bar{y}$, Π, $\mathcal{M}$, $\mathcal{R}$, p, $\Delta(p)$, q)

```
1:  Initialization: $\nu \leftarrow 1$, $y^{best} \leftarrow \bar{y}$, $l \leftarrow 1$, $i \leftarrow 1$;
2:  repeat
3:      Learn&Memorize: $\mathcal{A}^{\star}_{\nu} \leftarrow Learning(y^{best}, \Pi, m_i \in \mathcal{M}, r_l \in \mathcal{R})$;
4:      Fix&Optimize: $\bar{y}_{\nu} \leftarrow SubMIPSolve(\mathcal{A}^{\star}_{\nu}, p)$;
5:      if Global update: $\phi(\bar{y}_{\nu}) \leq \phi(y^{best})$ then $y^{best} \leftarrow \bar{y}_{\nu}$;
6:      end if
7:      if Diversification: $y^{best}$ has not been improved in the last $q$ iterations then
8:          $i \leftarrow (i+1) mod\ MAX$;
9:          $l \leftarrow (l+1) mod\ R$;
10:         $p \leftarrow p - \Delta(p)$;
11:     end if
12:       $\nu \leftarrow \nu + 1$ ;
13: until Stopping criterion not satisfied
14: Return the best solution $y^{best}$
```

7-10), by changing the m-combination and the selection rule in the learning step, as well as by decreasing the percentage of variables to fix in the partial optimization step. The last operations expands the solution space of the partial problem to hopefully identify improved solutions.

5 Computational Results

This section presents the results of the computational experiments performed to assess the performance of the proposed matheuristic. We first describe the test instances and experimental settings (Section 5.1). Section 5.2 is dedicated to a sensitivity analysis of algorithm to different initial solutions and ADS types. The performance of the matheuristic in addressing larger instances is analyzed in Section 5.3. Comparative results to CPLEX and a *Local Branching* (*LBr*) matheuristic are presented throughout the section.

5.1 Data and experimental settings

We used 12 problem classes, R4-R15, from the instances of stochastic CMND problem introduced in Crainic et al. (2011). The attributes of each class, in terms of node, arc, and commodity set cardinalities, are given in Table 1. Each class contains five networks with different "ratio" index valued 1, 3, 5, 7, and 9, indicating continuously increasing ratios of fixed-to-variable-cost and total-demand-to-total-network-capacity. For each of these networks, instances are created with 16, 32, and 64 demand scenarios. Demands were assumed to be linearly correlated, and three levels of correlation, 0, 0.2, and 0.8, were considered to create different instances.

Table 1: Characteristics of instances

Problem	$\|\mathcal{N}\|$	$\|\mathcal{A}\|$	$\|\mathcal{K}\|$	Problem	$\|\mathcal{N}\|$	$\|\mathcal{A}\|$	$\|\mathcal{K}\|$
R04	10	60	10	R10	20	120	40
R05	10	60	25	R11	20	120	100
R06	10	60	50	R12	20	120	200
R07	10	82	10	R13	20	220	40
R08	10	83	25	R14	20	220	100
R09	10	83	50	R15	20	220	200

Notice that, while the small R instances, groups R4-R10, were use in revious studies of stochastic network design (e.g. Crainic et al., 2011, 2014b), the large ones, groups R11-R15, were not. We thus used a subset of instances from classes R4-R10 to perform the sensitivity analysis of the algorithm to parameter values, while a more detailed analysis was conducted on classes R11-R15.

Algorithms were implemented in C++. The numerical experiments were performed on a Cisco UCS C200 cluster of 26 computers; each has two 3.07 GHz Intel(R) processors and 96 Gigabytes of RAM, operating under Linux.

We used CPLEX version 12.6.1.0 to solve the deterministic MIP problems, complete and partial. The Local Branching matheuristic (Fischetti and Lodi, 2003) is based on defining a neighborhood of the current incumbent solution by allowing only a few binary variables to flip their value, through the addition of a local branching constraint. The neighborhood is then explored with a branch-and-bound solver. We implemented the LBr by turning on the parameter LBHeur in CPLEX, which invokes a local branching heuristic when it finds a new incumbent. The Learn&Optimize matheuristic was run with the random selection rule in creating artificial demand scenarios. Finally, preliminary experiments helped set the τ, q, and $\Delta(p)$ parameters to 0.8, 4, and 0.05, respectively.

5.2 Sensitivity analysis

Our main goal with this phase of the experiments is to evaluate the impact of using different initial solutions and different types of ADSs on the behavior of the algorithm. We performed this analysis on 22 representative small R instances, R4-R10, which were solved to optimality by CPLEX within 1000 seconds of CPU time. The instances and the computational results (CPU time) for four variants of the matheuristic, corresponding to four combinations of characteristics, are displayed in Table 2. The CPU time required by the final Learn&Optimize metaheuristic, *L&Opt*, CPLEX and the LBr matheuristic are also displayed. Notice that the solution values are not displayed in Table 2 because all variants of L&Opt found the optimal solution reported by CPLEX. Hence, we report the time needed to hit these known optimal solutions only.

Two initial solutions were considered by solving the deterministic network design models with the expected demand, the "Exp" case, and the maximum demand, the "Max" case. Two types of ADS were also considered varying the value of the m parameter, $m = 2$ and $m = \lfloor \frac{\mathcal{K}}{2} \rfloor$, used in Algorithm 4 with the random selection rule. Notice that, $m = 2$ yields a larger cardinality for the generated set of ADSs, $N_m = \lceil \frac{|\mathcal{K}|}{2} \rceil$), compared $m = \lfloor \frac{|\mathcal{K}|}{2} \rfloor$ with a cardinality of 2. The four variants of the L&Opt matheuristic then were:

- Variant1: initial solution "Exp",$m = 2$;
- Variant2: initial solution "Exp", $m = \lfloor \frac{|\mathcal{K}|}{2} \rfloor$;
- Variant3: initial solution "Max", $m = 2$;
- Variant4: initial solution "Max", $m = \lfloor \frac{|\mathcal{K}|}{2} \rfloor$.

The results in Columns 2 to 5 of Table 2 indicate that the proposed learning mechanism and matheuristic are generally not sensitive to the initial solution and the type of ADS, with respect to computational efficiency and solution quality (the optimal solution on the instances used). A more detailed analysis indicates, however, that using "Exp" in computing the initial solution provides slightly better efficiency compared to using the maximum demand. A similar remark may be made with respect to the parameter m, the variants with $m = \lfloor \frac{|\mathcal{K}|}{2} \rfloor$ examining fewer ADS in the course of the algorithm. Following these observations and the preliminary results, the final L&opt algorithm initiates the search from the expected-value solution, and sets the parameter m to iterate over $m = \{\lfloor \frac{|\mathcal{K}|}{2} \rfloor, \lfloor \frac{|\mathcal{K}|}{3} \rfloor, \lfloor \frac{|\mathcal{K}|}{4} \rfloor\}$).

The results of of the L&Opt, with these settings are displayed in Column 6, while Columns 7 and 8 display those of CPLEX and the LBr matheuristic. The last two columns of Table 2 display the comparative performance of the Learn&Optimize matheuristic with respect to CPLEX and LBr, by providing the time ratios calculated as $\frac{t_{CPLEX}}{t_{L\&Opt}}$ and $\frac{t_{LBr}}{t_{L\&Opt}}$. The summation of computation times over all instances for each variant is shown on the last line as "Total" (the average is displayed for the last two columns).

We observe that the proposed L&Opt performs very efficiently by changing the value of parameter m throughout the algorithm, rather than fixing its it to a single value. the proposed matheuristic is also faster than CPLEX and LBr, except for a few very-easy to solve instances requiring less than 30 seconds running time. L&Opt outruns the two other methods by an average factor of 1.82 and 2.7, respectively.

Table 2: Comparison of CPU times

Prob	Variant1	Variant2	Variant3	Variant4	L&Opt	CPLEX	LBr	Time Ratio	
	Time	Time	Time	Time	Time	Time	Time	CPLEX	LBr
R4.1-16	0.51	0.51	0.51	0.51	0.51	0.79	0.38	1.49	0.74
R4.1-32	1.9	1.8	1.95	1.97	1.4	1.4	0.7	1	0.51
R4.1-64	6.70	5.6	6.46	6.71	5.01	5.03	4.3	1.01	0.85
R4.3-16	19.96	14.28	20.33	18.33	7.2	5.20	2.37	0.72	0.3
R4.3-32	50.21	53.92	60	58	37.02	30.23	7.53	0.81	0.19
R4.3-64	184.11	243.94	201	195	181.02	183.3	29.57	1.01	0.15
R4.5-16	53.67	23.86	40	39.54	20.01	19.12	13.86	0.95	0.65
R4.5-32	150.34	67.25	210	202	54.84	196	42.84	3.57	0.77
R4.5-64	398.74	339.7	580	579.1	339.1	685.5	176.73	2.02	0.51
R5.7-16	2.49	2.21	2.32	2.62	2.6	5.53	6.35	2.12	2.6
R5.7-32	8.56	9.20	10.10	10.50	8.56	15.96	18.2	1.86	2.12
R5.7-64	29.40	30	31.3	31.2	29.96	50.73	75.56	1.69	2.5
R7.1-16	0.65	0.63	0.69	0.63	0.61	0.32	0.47	0.52	0.7
R7.1-32	2.43	2.44	2.44	2.23	1.83	0.83	1.18	0.45	0.64
R7.1-64	9.27	8.56	8.9	10.94	4.61	3.64	4.77	0.78	1.03
R7.3-16	23.21	22.68	24.1	22.8	20.21	5.33	8.05	0.26	0.3
R7.3-32	61	59.2	67	65	46.21	19.67	30.5	0.42	0.65
R7.3-64	148	134	160	155	100.89	116.19	197.66	1.15	1.97
R7.5-16	18.88	18.17	18.9	18.86	16.8	15.82	15.91	0.94	0.94
R7.5-32	5.56	5.19	5.51	5.33	5.01	35.02	53.94	6.99	10.6
R7.5-64	17.92	17.85	19.78	17.59	17.01	153.25	315.41	9.01	18.52
R8.5-16	308.20	300.45	350.21	320.51	290.07	402.87	250.61	1.38	0.86
Total	1193.51	1060.99	1471.29	1443.86	900.41	1549.64	1256.89	-	
Avg								1.82	2.17

5.3 Experiments on larger instances

The second set of experiments aimed to assess the behavior and performance of the learning mechanism and the L&Opt metaheuristic on larger instances, not yet tackled in the literature. We 1) compare the performances of L&Opt and CPLEX on those instances; 2) analyze in more depth the behavior of the proposed algorithm, in particular with respect to the impact of demand correlations and instance characteristics (fixed cost and capacity ratios); and 3) illustrate the efficiency of L&Opt in improving the solution through time.

A total of 225 instances, derived from the sets R11 to R15 (able 1), were generated for these experiments . As an illustration of the challenge to address such instances, consider that the deterministic equivalent problems of R15 instances with 64 scenarios consists of 16 142 080 variables and 282 880 constraints. Thus, 500 minutes CPU time was allocated to the three methods for this experiments.

Table 3 gives a general overview of the the computational performance of CPLEX for the instances considered. Column "# Opt." indicates the number of instances solved to optimality (out of the 45 instances in each class) within 500 minutes of CPU time, while Column "# Failures" reports the number of instances for which CPLEX could not solve the root LP relaxation problem within the same time. The difficulty increases with the instance size, class R15 representing the most difficult ones (failure to solve the LP relaxation in 20 out of 45 instances). These results underline the difficulty of even very good commercial MIP solvers to address a large portion of these instances (i.e., 180 out of 225).

Table 3: CPLEX performance on large R instances

Prob.	# Inst.	# Opt.	# Failures
R11	45	18	0
R12	45	18	5
R13	45	9	0
R14	45	0	6
R15	45	0	20
Total	225	45	31

To analyze the performance of L&Opt, we first focus on the 45 instances solved to optimality by CPLEX. Table 4 displays the comparison results, which show that L&Opt was able to find the optimal solutions of all these 45 instances. This indicating that L&Opt performs as well as CPLEX in terms of the number of instances solved to optimality. The computational time to hit the optimal solution are, on average, 56.1 and 126.3 minutes for CPLEX and L&Opt, respectively. Figure 1 illustrates, however, that L&Opt high quality solutions fast, e.g., solutions with average optimality gaps ($\frac{L\&Opt-CPLEX}{CPLEX}*100$) of 1.2%, 0.8%, and 0.74% after 23, 30, and 63.1 minutes, respectively.

Table 4: Performance comparison on easy instances

# Inst.	Opt. CPLEX	Sol. Time	Opt. L&opt	Sol. Time
45	45	56.1	45	126.3

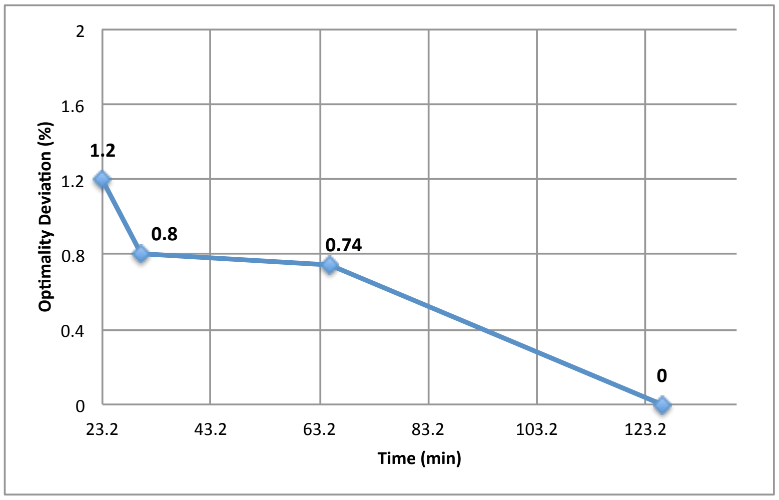

Figure 1: Optimality gap of L&Opt in time

Turning to the 180 instances that CPLEX was not able to solve to optimality, we compared the best results of L&Opt, CPLEX and the LBr matheuristic after 500 minutes

CPU time. Table 5 displays the results. Each line corresponds to a class of instances, Column "# of Ins" indicating the number of instances in each. The two "Wins" column report the percentage of instances for which L&Opt provided better solutions compared to CPLEX and LBr, respectively. Columns 4 and 6 report the average gaps, in %, between L&Opt and the two other methods computed as $\frac{L\&Opt-CPLEX}{L\&Opt} * 100$ and $\frac{L\&Opt-LBr}{L\&Opt} * 100$, respectively. Negative values indicate the superiority of L&Opt over CPLEX and LBr in terms of solution quality.

Table 5: Comparative performance of L&Opt and CPLEX on difficult instances

Prob.	# of	L&Opt/CPLEX		L&Opt/LBr		Time to beat(min)	
	Inst.	Wins(%)	Gap(%)	Wins(%)	Gap(%)	CPLEX	LBr
R11	27	88.8	-9.23 %	88.8	-9.60 %	64.94	61.32
R12	27	77.7	-6.13 %	81.4	-6.77 %	43.51	40.02
R13	36	83.3	-18.58 %	83.3	-23.36 %	60.79	55.2
R14	45	86.6	-15.16 %	86.6	-15.35 %	62.25	60.21
R15	45	100	-23.30 %	100	-23.79 %	70.05	68.32

We also report, in the last two columns, the average time for L&Opt to find a better solution than CPLEX and LBr within 500 min. We observed that, for more than 80% of instances, L&Opt found a better solution compared to those CPLEX and LBr found within 500 minutes, in 70 and 68.4 minutes, respectively, .

Overall, we noted that, for 119 out of 180 instances, L&Opt provided relative improvements of 17.20% and 18.4% in average over the solution found by CPLEX and LBr, respectively. The three methods provided the same solution quality for 30 out of 180 instances. Worth noticing, for the 31 instances for which CPLEX and LBr could not provide any solution after 500 minutes, L&Opt provided a feasible solution within 63 minutes (on average) and continued to improve it through time.

5.3.1 Algorithm behavior.

We continue to analyze the behavior of the learning-based matheuristic algorithm, given various instance characteristics.

We studied the performance of the L&Opt matheuristic and that of CPLEX according to the instance index, which indicates the cost and capacity ratios of the instance. We present the aggregated results of the 15 instances of each type in Table 6 for each instance type and number of scenarios (results computed only when a CPLEX lower bound or feasible solution, as required, was available). Column "Failure" represents the number of instances for which CPLEX could not solve the LP relaxation and, thus, no lower bound, within 500 minutes of CPU time. Column "OptGap" represents the average optimality gap of CPLEX after 500 minutes CPU time. The lat column displays the average gap (in %) between the best solutions found by L&Opt and CPLEX, computed as $\frac{L\&Opt-CPLEX}{L\&Opt} * 100$.

Table 6: Algorithm performance by instance type

Instance Type	|S|	# Inst.	CPLEX Failure	CPLEX OptGap	L&Opt/CPLEX Gap
1	16	15	0	7.6%	-3.75 %
1	32	15	0	12.21%	-5.12 %
1	64	15	2	19.63%	-4.49 %
3	16	15	0	49.2%	-26.38 %
3	32	15	3	58.7%	-30.87 %
3	64	15	9	70.3%	-33.89 %
5	16	15	0	29.1%	-13.42 %
5	32	15	3	38.6%	-22.63 %
5	64	15	7	44.9%	-23.45 %
7	16	15	0	3.63%	-0.42 %
7	32	15	0	8.53%	-2.79 %
7	64	15	3	7.66%	-4.95 %
9	16	15	0	10.77%	-5.40 %
9	32	15	1	22.44%	-11.25 %
9	64	15	3	24.39%	-9.25 %

The results underline the impact of these instance characteristics on the behavior of the algorithms. In particular, they indicate that the instances that are the most difficult to address are not those with the highest fixed cost and capacity ratio, an observation often made for deterministic CMND problems (e.g., Ghamlouche et al., 2003), but rather those with intermediate ratios (e.g., 3 and 5). This is observation is in line with the results of Crainic et al. (2011). This behavior may be caused by the higher number of similar optimal designs that exist for such intermediary deterministic CMND instances, compared to instances with low or high ratios. Then, when demand is stochastic, such instances would display broader differences among scenarios requiring more effort to identify an overall satisfactory, hopefully optimal, solution.

We also analyzed the impact of the scenario correlations, and observed that the demand correlation has little impact on the difficulty to address the problem. Finally, as expected, increasing the number of scenarios makes the problem more difficult to address for both L&Opt and CPLEX.

5.3.2 Improvement through time.

It is also interesting to note how the quality of the solutions evolves over time. We thus let L&Opt and CPLEX run for eight hours and compared the evolution of the two methods.

Table 7 displays the comparative results of L&Opt after two and eight hours (noted L&Opt(2h) and L&Opt(8h), respectively), with respect to those of CPLEX at the end of the eight hours (CPLEX(8h)). The relative gaps were computed as $\frac{L\&Opt-CPLEX}{L\&Opt} * 100$ (negative values indicate superiority of L&Opt) for the instances for which CPLEX found a feasible solution. Columns "Max" and "Avg" display the maximum and average gaps, respectively, of the two comparisons.

The results clearly show that L&Opt not only outperforms CPLEX in producing better solutions for difficult instances, but that it also does so in less computation time. Significant improvements are already observed after two hours of L&Opt, 5.36% on average and up to a maximum of 21.61% for the largest instances. These results are better when the matheuristic is given the same computation time, as shown in the last two columns, with a 15.61% average relative gap and a maximum of 48.76% for the largest instances.

Table 7: Performance comparison between L&Opt and CPLEX through time

Pro	L&Opt(2h)/CPLEX(8h)		L&Opt(8h)/CPLEX(8h)	
	Max(%)	Avg(%)	Max(%)	Avg(%)
R11	-9.86	-2.27	-17.44	-9.23
R12	-5.01	-3.21	-14.27	-6.13
R13	-15.5	-4.71	-28.59	-18.58
R14	-9.82	-6.37	-36.25	-15.16
R15	-21.61	-10.25	-48.76	-23.30

We complete this analysis by illustrating the behavior of the method we propose on the large instances for which CPLEX failed to provide any information (not even a lower bound) in eight hours of CPU time. Figure 2 displays the improvement in solution value obtained in time by L&Opt relative to the initial solution for two instances, 15.3-0-32 and 15.3-0-64. The relative improvement after t hours relative to the initial solution, L&Opt(1h), was calculated as $\frac{L\&Opt(t)-L\&Opt(1h)}{L\&Opt(t)} * 100$. The figure shows that the largest improvement occurs quite rapidly at the beginning of the search, but L&Opt continues to find improving solutions as more time is given.

6 Conclusions

We introduced a learning-based matheuristic for the stochastic fixed charge multi-commodity network design problem with uncertain demands. The innovative learning mechanism systematically explores combinations of scenarios to extract information regarding the solution structure of the stochastic problem. The learning method. The matheuristic builds a global image of the promising structure of the stochastic solution out of the par-

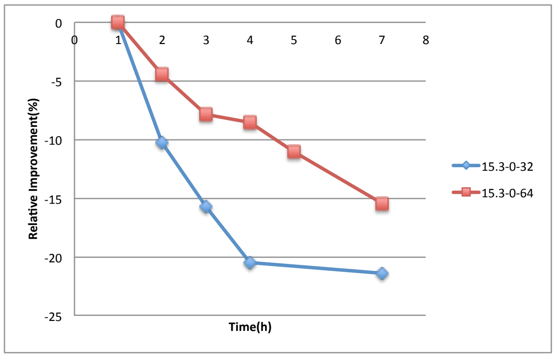

Figure 2: Relative improvement over initial solution in time

tial knowledge produced by the learning mechanism, and exploits it define reduced-size problems that are solved by a MIP solver.

The results of extensive computational experiment showed that the proposed matheuristic performs very well, being highly effective in finding good-quality solutions for the large stochastic network design instances. This is very promising as, although presented in the context of the complex network-design problem setting, the proposed learning mechanism and matheuristic can be adapted to many other stochastic combinatorial optimization problems.

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

Azaron, A., Brown, K., Tarim, S., and Modarres, M. (2008). A multi-objective stochastic programming approach for supply chain design considering risk. *International Journal of Production Economics*, 116(1):129–138.

Bidhandi, H. M. and Yusuff, R. M. (2011). Integrated supply chain planning under uncertainty using an improved stochastic approach. *Applied Mathematical Modelling*, 35(6):2618–2630.

Birge, J. R. and Louveaux, F. (2011). *Introduction to stochastic programming*. Springer.

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., Hewitt, M., and Rei, W. (2014a). Partial decomposition strategies for two-stage stochastic integer programs. Publication CIRRELT-2014-13, 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. (2014b). Scenario grouping in a progressive hedging-based meta-heuristic for stochastic network design. *Computers & Operations Research*, 43(2):90–99.

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.

Dupačová, J., Consigli, G., and Wallace, S. W. (2000). Scenarios for multistage stochastic programs. *Annals of Operations Research*, 100(1-4):25–53.

Fischetti, M. and Lodi, A. (2003). Local branching. *Mathematical programming*, 98(1-3):23–47.

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.

Higle, J. L. and Wallace, S. W. (2003). Sensitivity analysis and uncertainty in linear programming. *Interfaces*, 33(4):53–60.

King, A. J. and Wallace, S. W. (2012). *Modeling with stochastic programming*. Springer Science & Business Media.

Lanza, G., Crainic, T.G., Rei, W., and Ricciardi, N. (2017). Service Network Design Problem with Quality Targets and Stochastic Travel Time. Publication CIRRELT-2017, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, Université de Montréal, Montréal, QC, Canada.

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.

Maggioni, F. and Wallace, S. W. (2012). Analyzing the quality of the expected value solution in stochastic programming. *Annals of Operations Research*, 200(1):37–54.

Magnanti, T. L. and Wong, R. T. (1984). Network design and transportation planning: Models and algorithms. *Transportation Science*, 18(1):1–55.

Minoux, M. (1989). Networks synthesis and optimum network design problems: Models, solution methods and applications. *Networks*, 19(3):313–360.

Riis, M. and Andersen, K. A. (2000). Capacitated network design with uncertain demand. Technical report, Department of Operations Research University of Aarhus.

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.

Santoso, T., Ahmed, S., Goetschalckx, M., and Shapiro, A. (2005). A stochastic programming approach for supply chain network design under uncertainty. *European Journal of Operational Research*, 167(1):96–115.

Schütz, P., Tomasgard, 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.

Smith, J. C., Schaefer, A. J., and Yen, J. W. (2004). A stochastic integer programming approach to solving a synchronous optical network ring design problem. *Networks*, 44(1):12–26.

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.

Tsiakis, P., Shah, N., and Pantelides, C. C. (2001). Design of multi-echelon supply chain networks under demand uncertainty. *Industrial & Engineering Chemistry Research*, 40(16):3585–3604.

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.

Wang, X., Crainic, T.G., and Wallace, S.W. (2016). Stochastic Scheduled Service Network Design: The Value of Deterministic Solutions. Publication CIRRELT-2016-14, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, Université de Montréal, Montréal, QC, Canada.