CIRRELT-2023-04

An Asynchronous Parallel Benders Decomposition Method for Stochastic Network Design Problems

Ragheb Rahmaniani
Teodor Gabriel Crainic
Michel Gendreau
Walter Rei

January 2023

An Asynchronous Parallel Benders Decomposition Method for Stochastic Network Design Problems[ł]

Ragheb Rahmaniani[1], Teodor Gabriel Crainic[2,3,*], Michel Gendreau[2,4], Walter Rei[2,3]

1. Meta Platforms Inc., Seattle, WA, USA
2. Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)
3. School of Management, Université du Québec à Montréal
4. Department of Mathematical and Industrial Engineering, Polytechnique Montréal

Abstract. Benders decomposition (BD) is one of the most popular solution algorithms for stochastic integer programs. The BD method decomposes stochastic problems into one master problem and multiple disjoint subproblems. It thus lends itself readily to parallelization. In almost all studies on the parallelization of this algorithm, the master problem remains idle until every subproblem is solved and vice versa. This can clearly result in an extremely inefficient parallel algorithm due to excessive idle times. On the other hand, relaxing the synchronization requirement can yield a nonconvergent or less efficient algorithm that may even underperform when compared to the sequential version. Addressing these issues, we introduce an asynchronous parallel BD method for stochastic network design problems. We show that the proposed algorithm converges to the global optimum and suggest various acceleration strategies to enhance its performance. We conduct an extensive numerical study on benchmark instances from a generic stochastic network design problem. The results indicate that using up to 20 processors, our asynchronous algorithm is on average 1.4 times faster than the conventional low-level parallel methods.

Keywords: Stochastic programming, network design, Benders decomposition, branch-and-cut, parallel computing.

Acknowledgements. While working on the project, the second author held the UQAM Chair on Intelligent Logistics and Transportation Systems Planning and was Adjunct Professor, Department of Computer Science and Operations Research, Université de Montréal (DIRO), the third was Adjunct Professor, DIRO, and the fourth held the Canada Research Chair in Stochastic Optimization of Transport and Logistics Systems of UQAM. We gratefully acknowledge the financial support provided by the Natural Sciences and Engineering Research Council of Canada (NSERC), through its Discovery Grant program, and by the Fonds de recherche du Québec through its Team Grant program. We also gratefully acknowledge the support of Fonds de recherche du Québec through their CIRRELT strategic infrastructure grants.

[ł]Revised version of the CIRRELT-2021-41

---

[*] Corresponding author:teodorgabriel.crainic@cirrelt.net

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2023

1 Introduction

Two-stage *stochastic integer programming* (SIP) offers a powerful tool to deal with uncertainties in planning problems where the distribution of uncertain parameters is assumed to be known and often characterized with a finite set of discrete scenarios (Birge and Louveaux, 1997). SIP models are, however, often very large in size and very difficult to solve due to the data uncertainty and their combinatorial and nonconvex nature (Ahmed, 2010). Therefore, their solvability constitutes one of the major research streams in the field.

Many efforts have been made to design various decomposition-based algorithms to break SIP models into more manageable pieces. Fortunately, in many cases the SIP models exhibit special structures that can be effectively exploited in various decomposition methods (Ruszczyński, 1997) such as Benders decomposition (BD) (Benders, 1962), also known as the L-shaped method (Van Slyke and Wets, 1969), stochastic decomposition (Higle and Sen, 1991), nested decomposition (Archibald et al., 1999), subgradient decomposition (Sen, 1993), scenario decomposition (Rockafellar and Wets, 1991), disjunctive decomposition (Ntaimo, 2010), etc. Among these methods, BD has become a prominent methodology to address SIPs (Ruszczyński, 1997; Uryasev and Pardalos, 2013).

The BD method decomposes the two-stage SIP model into a *master problem* (MP) and a *subproblem* (SP) for each scenario. It first projects the model on the subspace defined by the first-stage variables. This allows to separate the projected term for each scenario. Then, it builds an equivalent linear model by enumerating all the extreme points and rays of the dual of each projected problem. The extreme rays indicate feasibility requirements for the first-stage variables (i.e., *feasibility cuts*) and the extreme points state the projected cost (i.e., *optimality cuts*). Enumerating all the extreme points and rays of the dual polyhedra is not computationally practical, however. Therefore, a relaxation of the equivalent formulation is performed such that it initially includes no cuts. The cuts are then iteratively generated by sequentially solving the MP and SPs until an optimal solution is found.

Solving the master problem at each iteration for SIP models can be very time-consuming. Therefore, the algorithm is commonly cast into a *branch-and-cut* (B&C) framework in order to avoid solving an integer problem from scratch at each iteration. Thus, a single branch-and-bound tree is built and the cuts are generated at the integer (and possibly some fractional) nodes (Rahmaniani et al., 2017a, 2020). Likewise, solving the SPs at each iteration often corresponds to the most time-consuming part of the algorithm for the SIPs, because a large number of scenarios is often required to properly characterize the uncertain parameters. These SPs are disjoint and can be solved in parallel. Thus, *parallel computing* appears very promising to effectively accelerate solving the SIPs when the BD method is used (Linderoth and Wright, 2003; Li, 2013). Although parallel processing can generally reduce the wall-clock run time, the processors at some point need to exchange information and consolidate the results in order to create work units for the next iteration. In the literature on parallel BD methods, these synchronization points are implemented in every iteration of the algorithm, that is, the processor that solves the MP waits at each iteration after the processors that solve the SPs and vice versa. The need for this frequent synchronization is due to the interdependency of the MP and SPs. This synchronized parallelization, however, causes having one or several idle processors at any given time, particularly when obtaining a master solution can take hours (Yang et al., 2016) or solving the SPs is slow (Linderoth and Wright, 2003). As a result, the efficiency of the parallel execution may decrease as the number of processors increases. It is thus important to design new parallelization schemes for the BD method to obtain a high-performing algorithm.

An evident strategy to design parallel BD methods, specially when the MP becomes the major computational bottleneck, is to implement the BD method in a B&C framework and possibly parallelize the branch-and-bound tree. In this regard, we are only aware of the study by Langer et al. (2013). However, even in such algorithms, the interdependency of the MP and SPs is not addressed and thus, the parallelization may still suffer from excessive idle times due to synchronizing each master processor with the worker processors.

In this article, we aim at developing effective parallelization strategies for the B&C implementation of the BD method. To avoid burdening this article, we limit ourselves to stochastic network design problems where the SPs are optimized in parallel on various *worker* processors and the MP is solved sequentially on a single *master* processor. Our goal is to introduce a parallel framework where the synchronization among the master and worker processors is not required. To the best of our knowledge, this is the first study to address this issue in solving integer programs. Therefore, our study can also serve as a building block toward having more effective multi-master and multi-worker parallel Benders algorithms.

It is important to note that our parallelization framework is different from the existing parallel B&C algorithms. The main emphasis in the latter is on parallelizing the branch-and-bound tree and cuts are commonly generated to accelerate the convergence (Ralphs et al., 2003; Crainic et al., 2006), while in the parallel BD methods, the main emphasis is on parallelizing the SPs for which the cuts, similar to the lazy cut generation for the TSP problem, are *necessary* for convergence. We are not aware of any parallel B&C algorithm where the cuts are necessary for convergence and synchronization has not been applied.

To achieve our goal, we relax the synchronization requirements among the master and worker processors. This means that the master processor may or may not wait for cuts to be generated by the worker processors at each node of the branch-and-bound tree. Although this significantly reduces the idleness of the master and worker processors, it results in an algorithm for which we are unable to prove global convergence. This happens because, in absence of synchronization, the necessary cuts at the integer nodes may not be applied at the right moment. We thus study this issue and show that, with an appropriate B&C design, the algorithm can converge. On the other hand, the asynchronous algorithm may execute a large amount of *redundant work* since the MP keeps branching on the master variables with a *partial* feedback from the SPs. This can cause serious efficiency problems such that the algorithm may even underperform when compared to the sequential variant. Therefore, we propose novel acceleration techniques to obtain an efficient asynchronous parallel BD algorithm. The main contributions of this article are thus severalfold:

- We propose a parallel B&C BD algorithm which alleviates the synchronization requirement of the existing parallel BD algorithms;
- We study convergence properties and various variants of the algorithm;
- We discuss several factors that influence the design and performance of asynchronous parallel BD algorithms and propose a combination of new strategies to accelerate the computational performance of our algorithm;
- We present extensive numerical results on benchmark instances from stochastic network design problem to assess the proposed strategies and algorithms. We also provide guidelines on when to use different variants of the proposed parallelization technique and discuss various fruitful research directions.

The remainder of this article is organized as follows. The problem and the sequential BD algorithm are presented in Section 2. We review parallel BD methods in Section 3, and present our asynchronous parallel algorithm and the proposed acceleration techniques in Sections 4 and 5. Implementation details and numerical results are discussed in Sections 6 and 7. Conclusions and remarks on research perspectives make up the last section.

2 The Benders Decomposition Method

We first recall the notation for the two-stage stochastic network design problem of interest. Then, we present a sequential BD algorithm to solve it.

2.1 Stochastic Network Design Problem

We consider the well-known *Multi-Commodity Capacitated Fixed-charge Network Design Problem* with *Stochastic Demands* (MCFNDSD). This problem naturally appears in many applications (e.g., Klibi et al., 2010) and it is notoriously hard to solve even in its deterministic (Crainic et al., 2021b; Crainic and Gendron, 2021; Crainic and Gendreau, 2021) and sequential-stochastic (Hewitt et al., 2021) forms.

The MCFNDSD is defined on a directed graph consisting a set of nodes $\mathcal{N}$ and a set of potential arcs $\mathcal{A}$. In this problem, a set of commodities $\mathcal{K}$ exist where each commodity $k \in \mathcal{K}$ has an uncertain amount of demand that needs to be routed from its unique origin $O(k) \in \mathcal{N}$ to its unique destination $D(k) \in \mathcal{N}$. We assume that the demand uncertainty b is characterized with a set of discrete scenarios $\mathcal{S}$. Therefore, each commodity $k \in \mathcal{K}$ has a stochastic demand $d_s^k \geq 0$ for each scenario $s \in \mathcal{S}$. We assume that the realization probability ρ_s for scenario $s \in \mathcal{S}$ is known, with $\sum_{s \in \mathcal{S}} \rho_s = 1$. The goal is to select a proper subset of the arcs to meet all the flow requirements at minimum cost. A fixed cost f_a must be paid to select arc $a \in \mathcal{A}$, and a unit cost c_a^k is to be charged to route commodity $k \in \mathcal{K}$ on this arc. There is a capacity limit u_a on each arc $a \in \mathcal{A}$. The objective is to minimize the sum of the fixed costs and the expected flow costs.

To model this stochastic problem, we define binary first-stage variables y_a indicating if arc $a \in \mathcal{A}$ is selected 1 or not 0. To model the second-stage, we define continuous variables $x_a^{k,s} \geq 0$ to reflect the amount of flow on arc $a \in \mathcal{A}$ for commodity $k \in \mathcal{K}$ under realization $s \in \mathcal{S}$. The extensive formulation of MCFNDSD is thus:

$$MCFNDSD = \min_{y \in \{0,1\}^{|\mathcal{A}|}, x \in \Re_+^{|\mathcal{A}||\mathcal{K}||\mathcal{S}|}} \sum_{a \in \mathcal{A}} f_a y_a + \sum_{s \in \mathcal{S}} \rho_s \sum_{k \in \mathcal{K}} \sum_{a \in A} c_a^k x_a^{k,s}, \tag{1}$$

$$\text{s.t:} \sum_{a \in \mathcal{A}(i)^+} x_a^{k,s} - \sum_{a \in \mathcal{A}(i)^-} x_a^{k,s} = \begin{cases} d_s^k & \text{if } i = O(k) \\ -d_s^k & \text{if } i = D(k) \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in \mathcal{N}, k \in \mathcal{K}, s \in \mathcal{S} \tag{2}$$

$$\sum_{k \in \mathcal{K}} x_a^{k,s} \leq u_a y_a \quad \forall a \in \mathcal{A}, s \in \mathcal{S}, \tag{3}$$

where $\mathcal{A}(i)^+$ and $\mathcal{A}(i)^-$ indicate the sets of outward and inward arcs incident to node i, respectively. The objective function minimizes the total fixed costs plus the expected routing costs. Constraints (2) impose the commodity-flow conservation at each node for each scenario. Linking constraints (3) enforce the arc-capacity limit in each scenario.

The following compact notation of the problem is used to ease the presentation:

$$z^* := \min_y \{f^\top y + \sum_{s \in \mathcal{S}} \rho_s Q(y,s) : \ y \in \mathcal{Y}\}, \tag{4}$$

where the recourse problem $Q(y,s)$ (i.e., the flow problem) is defined for each scenario $s \in \mathcal{S}$:

$$Q(y,s) = \min_x \{c_s^\top x : W_s x \geq h_s - T_s y, \ x \in \mathcal{X}_s\}, \tag{5}$$

with $f \in \mathbb{R}^{|\mathcal{A}|}$, $c_s \in \mathbb{R}^{|\mathcal{A}||\mathcal{K}|}$, $l = |\mathcal{A}||\mathcal{K}| + |\mathcal{A}|$, $W_s \in \mathbb{R}^{l \times |\mathcal{A}||\mathcal{K}|}$, $h_s \in \mathbb{R}^l$, $T_s \in \mathbb{R}^{l \times |\mathcal{A}|}$. Here, $\mathcal{X}_s \subseteq \mathbb{R}^{|\mathcal{A}||\mathcal{K}|}$ and $\mathcal{Y} \subseteq \mathbb{R}^{|\mathcal{A}|}$ are nonempty closed sets which define the nature of the x and y decision variables in terms of sign and integrality restrictions. Note that the above notation is more general than the MCFNDSD (where $W_s = W$, $c_s = c$ and $T_s = T$ for all $s \in S$). This is because our solution approach can be applied to a more general case where these parameters are stochastic. We make the conventional assumption that program (4) is bounded, $\mathcal{Y} \equiv \{0,1\}^{|\mathcal{A}|}$, and $\mathcal{X} \equiv \mathbb{R}_+^{|\mathcal{A}||\mathcal{K}|}$. Note that any problem with non-binary integer first-stage variables can be approximated with a binary problem to a desired precision and without increasing the problem size by too much (Zou et al., 2017).

2.2 Sequential Benders Decomposition Method

For a tentative value of the first-stage variables $\bar{y}$, the recourse problem $Q(\bar{y}, s)$ is a continuous linear program. Given a dual variable α associated with the constraint $W_s x \geq h_s - T_s\bar{y}$, the dual of $Q(\bar{y}, s)$ is:

$$(SP(\bar{y}, s)) \;\; Q(\bar{y}, s) = \max_{\alpha} \{(h_s - T_s\bar{y})^\top \alpha : \; {W_s}^\top \alpha \leq c_s, \; \alpha \in \mathbb{R}_+^l\}. \tag{6}$$

The above program is either unbounded or feasible and bounded. In the former case, the $\bar{y}$ solution is infeasible and thus there exists a direction of unboundedness $r_{q,s}$, $q \in F_s$ that satisfies $(h_s - T_s y)^\top r_{q,s} > 0$, where F_s is the set of extreme rays of (6). To assure the feasibility of the y solutions, we need to forbid the directions of unboundedness through imposing $(h_s - T_s y)^\top r_{q,s} \leq 0, \; q \in F_s$, on the y variables, which gives:

$$z^* \;=\; \min_{y \in \mathbb{Z}_+^n} \{f^\top y + \sum_{s \in \mathcal{S}} \rho_s Q(y, s) : \; (h_s - T_s y)^\top r_{q,s} \leq 0 \; \forall s \in \mathcal{S}, q \in F_s\}. \tag{7}$$

In the latter case, the optimal solution of the SP is one of its extreme points $\alpha_{e,s}, \; e \in E_s$, where E_s is the set of extreme points of (6). We can thus rewrite program (7) in an extensive form by interchanging $Q(y, s)$ with its dual $SP(y, s)$

$$\min_{y \in Y} \big\{f^\top y + \sum_{s \in S} \rho_s \max_{e \in E_s} \{(h_s - T_s y)^\top \alpha_{e,s}\} : \; (h_s - T_s y)^\top r_{q,s} \leq 0 \;\;\; s \in S, q \in F_s\big\}, \tag{8}$$

where $Y = \{y \in \mathbb{Z}_+^{|\mathcal{A}|}\}$. Capturing the value of the inner maximization in a single variable θ_s for every $s \in \mathcal{S}$, we can obtain the following equivalent reformulation of (4), called the Benders *master problem* (MP):

$$MP(E_1, ..., E_{|\mathcal{S}|}; F_1, ..., F_{|\mathcal{S}|}) \;:=\; \min_{y \in Y} f^\top y + \sum_{s \in S} \rho_s \theta_s \tag{9}$$

$$(h_s - T_s y)^\top \alpha_{e,s} \leq \theta_s \qquad s \in S, e \in E_s, \tag{10}$$

$$(h_s - T_s y)^\top r_{q,s} \leq 0 \qquad s \in S, q \in F_s. \tag{11}$$

Enumerating all the extreme points E_s and extreme rays F_s for each SP $s \in S$ is computationally burdensome and unnecessary. Thus, Benders (1962) suggested a delayed constraint generation technique to extract the *optimality* (10) and *feasibility* (11) cuts on the fly.

The MP is solved from scratch at each iteration in the classical BD method (Benders, 1962). Currently, however, the method is usually implemented in a B&C framework to avoid solving a MP at each iteration (Naoum-Sawaya and Elhedhli, 2013; Rahmaniani et al., 2017b), and the LP relaxation of the MP is often solved first to quickly tighten the root node (McDaniel and Devine, 1977). Algorithm 1 presents the pseudo-code of this algorithm.

Algorithm 1 : The sequential Branch-and-Benders-cut method

```
1: Create the MP which is the LP relaxation of program (9)-(11) with $E_s = \emptyset$ and $F_s = \emptyset$, $\forall s \in \mathcal{S}$
2: Iteratively solve the $MP$ and add cuts until a stopping criteria is satisfied
3: Add the obtained $MP$ into tree $\mathcal{L}$, set $UB = \infty$
4: while no stopping condition is met do
5:     Select node $l$ from $\mathcal{L}$
6:     Solve this node to get an optimal solution $\bar{y}$ with objective value of $\vartheta^*$
7:     if node was infeasible or $\vartheta^* \geq UB$ then
8:         Prune the node and go to line 4
9:     if $\bar{y}$ is integer then
10:        while a violating cut can be found and the $\bar{y}$ is integer do
11:            for $s \in \mathcal{S}$ do
12:                Solve $SP(\bar{y}, s)$
13:                if $SP(\bar{y}, s)$ was infeasible then
14:                    Extract the extreme ray $r_s$
15:                else
16:                    Extract the optimal extreme point $e_s$
17:            Add the cuts obtained in lines 14 and 16 to the master formulation
18:            Solve this node again to get an optimal solution $\bar{y}$ with objective value of $\vartheta^*$
19:            if node was infeasible or $\vartheta^* \geq UB$ then
20:                Prune the node and go to line 4
21:        if $\bar{y}$ is integer then
22:            Set $UB = \min\{UB, \vartheta^*\}$, prune the node, and go to line 4
23:    Choose a fractional variable from $\bar{y}$ to branch
24:    Create two nodes and add them to $\mathcal{L}$
25:    Remove $l$ from $\mathcal{L}$.
```

The algorithm is applied in two phases. It first starts with solving the LP relaxation of the MP, which initially includes no optimality and feasibility cuts (lines 1 and 2). Thus, it sequentially solves the MP and SPs to generate optimality or feasibility cuts until a stopping condition (e.g., time limit, maximum number of iterations, or optimality gap) is activated . In the second phase, a branch-and-bound tree is created (line 3), where the root node is the MP with the cuts generated in the LP phase. At each iteration, the algorithm selects and solves a node from the pool $\mathcal{L}$ (lines 5 and 6). When the obtained solution is fractional and the node cannot be pruned (line 7), a branching occurs to create and add two new nodes to the pool (lines 23 and 24). When the node cannot be pruned (line 7) and its solution is integer (line 9), the algorithm iteratively adds cuts until it can either prune that node, the node solution becomes fractional, or no more violated cut can be found (lines 10 to 20). This process (lines 5 to 25) repeats until a stopping condition is satisfied or the node pool $\mathcal{L}$ becomes empty. Note that, at line 22, the upper bound is updated, i.e., the potential incumbent is accepted if $\bar{y}$ is an integer solution that satisfies all the Benders cuts and the associated cost is lower than the current incumbent value.

Our developments are based on Algorithm 1, which we refer to as the *Branch-and-Benders-cut* (B&BC) method. More specifically, our goal is to generate the cuts in parallel and avoid the inner while-loop so as to improve the overall efficiency. Various acceleration techniques are proposed to boost the BD algorithm. To avoid burdening this article, we refer the reader to Rahmaniani et al. (2017a) for a complete overview of the classical acceleration techniques. In the next section, we present an extensive review of the parallel BD methods.

3 Parallelization Strategies and Previous Work

The BD method lends itself readily to parallelization as it creates a MP and many disjoint SPs. Thus, efforts have been made to take advantage of parallel computing to accelerate the method. To the best of our knowledge, existing BD parallel variants follow the master-worker parallel programming model. We classify and review such methods in this section.

3.1 Master-worker parallelization strategies

Almost all the existing parallel BD methods can be summarized as follows: *The MP is assigned to a processor, the "master", which also coordinates other processors, the "workers", which solve the SPs. At each iteration, the solution obtained from solving the MP is broadcast to the worker processors. They then return the objective values and the cuts obtained from solving the SPs to the master. The cuts are added to the MP model and the same procedure repeats.* Such master-worker parallelization schemes are known as *low-level parallelism* as they do not modify the algorithmic logic or the search space (Crainic and Toulouse, 1998).

In many cases, the number of processors is less than the number of SPs and some

SPs may be more time consuming than others. Therefore, it is important to take into account how work units are allocated to the processors to avoid having idle processors.

To create work units for the next iteration, information must be exchanged among the processors, which necessitates some sort of communication to share information. The type of communications strongly influences the design of parallel BD algorithms. For example, if communications are synchronized, the only difference between the parallel algorithm and the sequential one lies in solving the SPs in parallel. Whereas, in asynchronous communications, the processors are less interdependent and many new factors must be taken into account while designing the parallelization framework. For example, after broadcasting each master solution, the master processor may or may not wait for cuts before branching. Also, some of the SPs associated with the current master solution may remain unsolved because the master processor may generate a new solution before the worker processors evaluate all the current SPs. Likewise, the worker processors may generate many cuts (associated to the current and/or previous solutions) while the master processor is solving the MP. Therefore, it is important to decide when to solve the MP, which solution to use in generating cuts, which SP to solve now, which cut to apply to the MP, and so on. We thus define a three-dimension taxonomy, depicted in Figure 1, to capture all these factors:

- ***Communication***: defines whether the processors at each iteration stop to communicate (*synchronous*) or not (*asynchronous*);
- ***Allocation/Scheduling***: determines how the SPs and the MP are assigned to the processors and when they are solved. These decisions can be made either *dynamically* (D) or *statically* (S). In the dynamic work allocation, the decisions regarding when and where to solve each problem are taken during the course of the algorithm to ensure a balanced load among the processors, e.g., using a first-in-first-out (FIFO) rule. In the static work allocation, the decisions are made beforehand and each processor is given a static set of problems to solve repeatedly. Thus, S/D, for example, means static allocation and dynamic scheduling;
- ***Pool Management*** characterizes the strategies used to manage the pool of solutions generated by the master problem (denoted by $S_1, ..., S_I$) and the pool of cuts extracted from the subproblems (denoted by $C_1, ..., C_J$), where I and J are the number of possible strategies to manage each pool, respectively.

A combination of various alternatives for these components yields a parallel BD algorithm. Proper strategies in each dimension need to be defined such that the overall *idle times* and amount of *redundant work* are minimized.

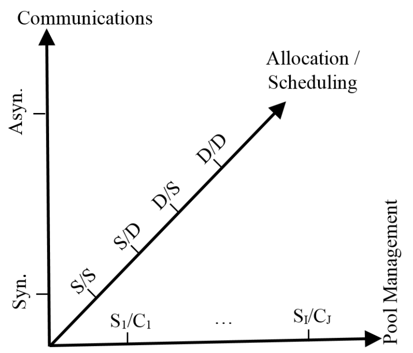

Figure 1: Taxonomy of the master-worker parallel Benders decomposition methods

3.2 Previous work

Most of the parallel BD algorithms are developed for stochastic problems as the decomposition creates multiple disjoint SPs (Ariyawansa and Hudson, 1991; Wolf and Koberstein, 2013; Tarvin et al., 2016). Although parallelization of the BD method for SIP models seems natural, the number of existing parallel variants of this method is limited. This is mainly due to the interdependency of the MP and the SPs, i.e., the MP needs feedback from the SPs before being able to execute its next iteration and vice versa. The literature discusses some of the strategies and algorithmic challenges arising from parallelizing the BD method.

Dantzig et al. (1991) considered a dynamic work allocation strategy in which the next idle processor gets the next SP based on a FIFO strategy until all the SPs are solved and the MP can be recomputed with the new cuts. The efficiency (measured in time reduction) of this parallel algorithm did not exceed 60% even on machines with 64 processors. Similarly, Li (2013) observed that dynamic work allocation is superior to static work allocation, because it reduces idle times and it also saves executing extra work. For example, if an SP is infeasible, the evaluation of the SPs remaining in the queue will be terminated and the feasibility cut generation scheme will be launched.

Nielsen and Zenios (1997) exploited the structural similarities of the SPs by applying an interior point algorithm on a fine-grained parallel machine. They decided to sequentially solve the MP to optimality from scratch at each iteration because it could easily be handled by means of an interior point method. Vladimirou (1998) implemented a partial-cut aggregation strategy to reduce the communication overheads.

The decomposition has been modified in some studies to better suit the parallelization scheme. Dempster and Thompson (1998) proposed a parallel nested algorithm for multistage stochastic programs. The authors used stage aggregation techniques to increase the size of the nodes and therefore the time spent on calculations relative to the time spent communicating between processors. In a similar spirit, Chermakani (2015) observed that when the number of SPs is considerably larger than the number of available processors, so that some SPs must be solved sequentially, it may be better to aggregate some of them. Latorre et al. (2009) modified the decomposition scheme for multi-stage stochastic programs such that the subsets of nodes assigned to each SP may overlap. This, unlike former studies, allows to noticeably reduce the dependency among SPs at each iteration, because they can be solved at the same time.

All the previously reviewed studies implement synchronized parallelism. Moritsch et al. (2001) proposed a prototype for an asynchronous nested optimization algorithm. However, no specific strategy or numerical results were presented. Linderoth and Wright (2003) implemented an asynchronous communication scheme in which the MP is re-optimized as soon as a portion of the cuts are generated. Testing this algorithm on LP stochastic programs with up to 10^7 scenarios, the authors observed a time reduction up to 83.5% on a computational grid compared to the sequential variant.

The speedup rates of low-level parallelizations are often limited. Pacqueau et al. (2012) observed that solving SPs accounts for more than 70% of the total time requirement, while they merely observed a speedup ratio up to 45% with 4 processors. Furthermore, low-level parallelism improves the efficiency when the MP solving does not dominate the solution process. Yang et al. (2016) observed that the benefit of parallelism fades away with the scale of the problem because the computational effort is dominated by solving the MP. To cope with this issue, Langer et al. (2013) proposed to parallelize the branch-and-bound tree where each processor solves a partition of the tree. To further improve the scalability, the authors also proposed to solve the SPs in parallel, where a FIFO queue is used in picking and solving the SPs. Moreover, in this parallelization, each node after evaluation is added to a waiting list until all its cuts are generated. Langer et al. (2013) observed that a better performance in solving aircraft allocation problem can be obtained when the master processors share the cuts.

Some of the common knowledge for the sequential algorithm may not apply to its parallel variants. For example, Wolf and Koberstein (2013) pointed out that the single-cut version benefits from parallelization more than the multi-cut version because more SPs need to be solved in the former case. They also observed that the single-cut method may outperform the multi-cut variant because it requires less cuts in general, although it executes a larger number of iterations. To conclude this part, we summarize the literature in Table 1.

We observe that the literature on parallel BD algorithms is sparse. Few studies consider integer programs and only one has implemented the parallel algorithm in the B&C format. Moreover, only two studies consider asynchronous communications and these are developed for linear continuous problems. Also, we realized that synchronized parallelism is suitable when the MP can be solved quickly and the SPs have roughly the same runtime, but it constitutes the major computational bottleneck of the algorithm.

Table 1: Summary of Benders-based parallel algorithms

Reference	Problem class		Implementation		Communication		Work allocation		Scheduling		Pool management	
	LP	MIP	Classic	B&C	Syn.	Asyn.	Dynamic	static	Dynamic	Static	Solution	Cut
Chermakani (2015)	+		+		+			+		+		
Dantzig et al. (1991)	+		+		+		+			+		
Dempster and Thompson (1998)	+		+		+			+		+		
Langer et al. (2013)		+		+	+		+			+		+
Latorre et al. (2009)		+	+		+		+			+		
Li (2013)		+	+		+		+			+		
Linderoth and Wright (2003)	+		+			+		+	+		+	+
Moritsch et al. (2001)	+		+			+		+	+		+	
Nielsen and Zenios (1997)	+		+		+			+		+		
Pacqueau et al. (2012)		+	+					+		+		
Vladimirou (1998)	+		+		+			+		+		
Wolf and Koberstein (2013)	+	+	+		+		+			+		
Yang et al. (2016)		+	+		+			+		+		
Mateo et al. (2018)	+		+		+			+		+		

Moreover, none of these studies addresses the interdependency of the MP and SPs, particularly for SIPs and when the BD method is casted into a B&C framework. Thus, the reviewed strategies are not directly applicable to develop a truly efficient parallelization of the BD method for stochastic network design problems.

4 Asynchronous Parallel Benders Decomposition Algorithm

The synchronization requirement between master and worker processors increases the overheads due to the excessive idle times. This is particularly evident when solving any of the problems (MP or SP) is noticeably more time consuming than the others.

Our strategy to deal with this issue is to obtain an asynchronous algorithm by relaxing the interdependency among the master and worker processors. To do so, the master processor is required to wait only for $100\gamma\%$ of the cuts from worker processors before executing its next iteration, where $0 \leq \gamma \leq 1$ and $\gamma = 0$ defines a fully asynchronous execution. The term "asynchronism" in the following hence refers to any algorithm variant with $\gamma < 1$.

For any $\gamma < 1$, the B&BC method may fail to converge and also, it may increase the amount of redundant work such that the parallel algorithm might underperform when compared to the sequential algorithm. In this section, we address these issues to obtain an effective and convergent asynchronous parallel Benders algorithm . Towards this end, we detail the considerations and propose modifications to Algorithm 1 and conclude the section with a complete pseudo-code of the proposed asynchronous algorithm.

4.1 Convergence

To ensure convergence of the B&BC method, the inner while-loop in Algorithm 1 cannot be stopped prematurely. Otherwise, an incumbent solution might be accepted (line 22), which does not necessarily satisfy all the Benders cuts and, thus, the optimal value might be underestimated. For this reason, the asynchronous execution of the master and subproblems (i.e., for any γ <1) may compromise the convergence as it applies only a subset of the cuts at each iteration of the while-loop. To cope with this issue, we need to keep track of the evaluated SPs for each master solution and update the incumbent value only when all the SPs are evaluated and the obtained bound is lower than the current incumbent value.

In the B&BC, the incumbent value is actively used to prune nodes. In the asynchronous algorithm, depending on the γ value, potential incumbent solutions are partially evaluated when their associated node is being processed. This creates a delay between finding an incumbent solution and obtaining its true value. Thus, to reduce the exploration of nodes which might have been pruned if we had the true incumbent value at the right moment, we suggest using the best-first search strategy in exploring the tree.

Asynchronous algorithms raise another issue when the integer node solution remains unchanged, i.e., the $\bar{y}$ solutions at lines 9 and 21 are the same after applying γ% of the cuts. Note that it is not guaranteed that every SP produces a violated cut for a given master solution. In this case, we cannot prune the node based on the integrality rule at line 22 unless we are sure that the $\bar{y}$ satisfies all the Benders cuts. This can be alleviated by keeping such nodes in the pool until all their associated SPs are solved or at least one of the corresponding violated cuts is generated. We, however, propose the use of combinatorial cuts of the following form to easily and effectively remedy this issue:

$$\sum_{i\in\{1,..,n\}:\bar{y}_i=0} y_i + \sum_{i\in\{1,..,n\}:\bar{y}_i=1} (1-y_i) \geq 1, \tag{12}$$

to forbid the regeneration of the current integer solution $\bar{y}$. This cut family is particularly helpful in the context of the asynchronous algorithm because it eliminates the current solution by (1) making the current node infeasible, (2) generating another integer solution, or (3) leading to the generation of a new fractional solution. In the first case, the node can be pruned by the infeasibility rule. This does not affect the convergence because no other feasible solution can be extracted from that node. The second case is in fact a desirable situation as it may yield a better incumbent value. This case, however, indicates that we need to add the combinatorial cuts within the inner while-loop at line 17. In the third case, the node will be added to the pool of active nodes, which does not affect the convergence properties. As a result, we can practically set $\gamma = 0$, i.e., the master processor does not need to wait for any cut and maintains the convergence.

Last but not least, the convergence does not depend on the number of subproblems and the proposed asynchronous algorithm can still converge to an optimal solution when

$|S| = 1$. However, we anticipate that the benefits of parallelization within the context of the BD method will be more pronounced when solving stochastic problems. This is the case in the present paper, where the strategy is applied to solve stochastic network design problems.

4.2 Communications

In our parallel algorithm, each processor continuously executes its assigned work units (e.g., branching on the master variables y or solving an SP) and shares the generated information. The message communicated by the master processor contains the current master solution. Each message sent by a worker processor contains the dual information and objective values obtained from solving an SP. The information sharing must take place without any waiting in order to minimize the idle times.

To do so, we make use of asynchronous message passing communications where each processor has its own buffer, i.e., local memory. This allows each processor to write its shareable information onto a buffer without waiting for the other processor(s) (i.e., the receiver) to actually receive the message. Thus, after sending the message, the processor returns immediately and continues executing the next work unit. Similarly, to avoid waiting in receiving information, each processor checks its buffer to see if there is any new message recorded by another processor. If there is new information in the buffer, the processor reads and processes it (which also cleans the buffer) and continues with the next step. If there is no information to read from the buffer, the processor continues with executing the next work unit without waiting.

At some point, however, a processor may need to wait for new information before being able to execute the next step. This waiting for a worker processor can only happen when there are no more SPs to evaluate. For the master processor, waiting is controlled by the γ parameter, on which we will further elaborate in Section 4.5.2.

4.3 Work Allocation Strategies

When the number of subproblems is greater than the number of processors, one needs to decide how the work is allocated to the processors. Two main strategies can be used in assigning SPs (i.e., work units) to worker processors. The first revolves around dynamically assigning the SPs to workers, i.e., give the next SP to the next available processor. In the second, static, strategy, each SP is assigned to an specific worker processor decided a priori.

We adopt a static work allocation because the SPs obtained from SIP models usually have the same level of difficulty, ensuring the workload to be balanced. Moreover, static work allocation allows for a more effective use of the re-optimization tools of the black box solvers and avoids the overhead of relocating code and data. In addition, when a dynamic work allocation within an asynchronous algorithm requires an additional processor to merely coordinate the worker processors, while it could be used to do more useful work.

Our static work allocation assigns an (almost) equal number of SPs to each processor in order to avoid the additional overheads and complexities associated with the load balancing. To further improve the re-optimization capabilities, we also try to assign similar SPs to the same processor. The similarity of two SPs is measured as the Euclidean distance among the random parameters:

Definition 1 *Given the $l \times 3$-dimensional matrix $w_s = (W_s, T_s, h_s)$ associated to each scenario $s \in S$, the Euclidean distance between two scenarios $s, s' \in S$ (two SPs) is computed as* $\sqrt{\sum_{j\in l}\sum_{k\in\{0,1,2\}}(w_s^{j,k} - w_{s'}^{j,k})^2}$.

Note that, grouping and assigning SPs to the worker processors is done by the master processor once at the beginning of the algorithm.

4.4 Pool Management Strategies

In our asynchronous algorithm, each processor continuously generates and shares information. The master processor keeps generating first-stage solutions and the worker processors keep evaluating the SPs to obtain cuts and bounds. Therefore, when the master processor is reading the buffer there might be many cuts in the memory. These cuts can be associated to different solutions and different SPs. Likewise, the master processor may share new solutions with the worker processors before they finish evaluating the SPs associated with the current solution. Therefore, each worker processor must have a pool to keep the generated master solutions, called *solution pool*, and the master processor must have a pool that contains the generated cuts, called *cut pool*. Properly managing these pools is of crucial importance to the performance of the asynchronous algorithm.

4.4.1 Solution Pool Management

Each worker processor has a pool containing one or several master solutions. All the SPs assigned to the worker processor must be solved for every solution in this pool. The order of selecting these solutions is important. For instance, the cuts associated with a more recent solution might be more effective in improving the lower bound, while the objective value obtained from an older solution might be even more important in updating the incumbent value. Therefore, the worker processor needs to decide which solution to choose at the current step and then decide to evaluate which one of its associated unevaluated SPs. The latter is a scheduling decision which we address in Section 4.5.1. We suggest the following strategies to select solutions from the solution pool:

S1: choose solutions randomly;

S2: choose solutions based on the *first-in-first-out* (FIFO) rule;

S3: choose solutions based on the *last-in-first-out* (LIFO) rule.

In all these strategies, we always give a lower priority to infeasible solutions. This means that if an SP associated to a solution is infeasible (i.e., the dual is unbounded) that solution will not be selected unless there is no other solution in the pool. This is because an infeasible solution does not yield an incumbent value and it has little impact on the lower bound. Last but not least, we use the *cut improvement* notion to identify the solutions which are no longer required to be evaluated, see Rei et al. (2009) for more information.

4.4.2 Cut Pool Management

The worker processors may generate many cuts, which are not all worth adding to or keeping in the master formulation. We thus define some strategies to effectively manage the cuts generated by the worker processors. In this regard, the master processor reads and stores all the cuts from the buffer. Then, it selects an appropriate subset of them to be added to the MP, removes some unviolated cuts from the master formulation, and deletes cuts from the cut pool. Note that removing unviolating cuts from the MP is a safe procedure (i.e., it does not affect convergence), since they can be regenerated if needed.

A cut from the cut pool is added to the master formulation if its relative violation (the absolute violation of the cut at the current solution divided by the 2-norm of the cut coefficients) is at least δ, otherwise it is discarded. Moreover, if there is more than one cut for a recourse variable $\theta_s,\ s \in \mathcal{S}$, in the pool, we only add the one that violates the current solution the most and keep the rest in the pool.

Cut removal can be computationally expensive as it changes the simplex basis and slows re-optimization. Thus, we execute the following routine after termination of the LP phase ()line 2 of Algorithm 1) to identify the dominated cuts. At iteration t, the hyperplane $(h_s - T_s y)^\top \alpha_s^t$ is generated to bound the recourse problem s at solution $\bar{y}^t$. Following a heuristic notion, another generated cut $(h_s - T_s y)^\top \alpha_s^j$, $j \neq t$, which bounds the recourse variable θ_s tighter at $\bar{y}^t$, i.e., $h_s^\top(\alpha_s^j - \alpha_s^t) + (\alpha_s^t - \alpha_s^j)^\top T_s \bar{y}^t \geq 0$, flags the possibility of removing cut $(h_s - T_s y)^\top \alpha_s^t$ without deteriorating the approximated value function for SP $s \in \mathcal{S}$. We apply the same procedure for the feasibility cuts. Note that, executing this test for all $y \in Y$ yields an exact method to identify the dominated cuts (Pfeiffer et al., 2012). In Phase II (lines 4 to 25), our algorithm removes any cut that might become slack.

4.5 Scheduling Strategies

In this section, we address two important questions which largely affect our asynchronous algorithm: 1) when to solve the MP, 2) which SP to solve next.

4.5.1 Sequencing the SPs

Once the worker processor chooses the next master solution to evaluate, it also needs to decide which one of the associated SPs to evaluate next. Notice that, the sequence of choosing a master solution and an associated SP is repeated at each step, as new information can be obtained after solving an SP and, thus, the algorithm can make better decisions in selecting the next master solution. In this regard, we suggest the following strategies:

SP1: Randomly choose an unevaluated SPs associated with the chosen master solution;

SP2: One may not need to solve all SPs associated to an infeasible master solution. Thus, we assign a *criticality* counter to each SP, which increases by one each time the SP becomes infeasible. Then, an SP with the highest criticality value is selected.

SP3: This strategy is a hybrid of SP1 and SP2, where SPs with higher criticality values have a higher chance to be selected.

4.5.2 Scheduling the MP

A key feature of the asynchronous algorithm lies in being independent of the worker processors to return their evaluation before it executes the next step. Recall that we control the waiting portion of the master processor with the γ parameter. However, the cuts that the master processor reads from the buffer can be associated with the current and/or previous solutions. Therefore, we need to be more specific when saying “the master processor waits only for $\gamma\%$ of the cuts”. We thus highlight the following two strategies:

MP1: The master processor waits until $\gamma|\mathcal{S}|$ cuts associated with the current master solution are added to the master formulation;

MP2: Master processor waits until $\gamma|\mathcal{S}|$ cuts associated with any master solution are added to the master formulation or no more unevaluated solution exists.

Note that, these strategies reduce to the parallel synchronized method for γ=1 (Langer et al., 2013), and are the same when γ=0. We computationally tune and examine these three strategies with different values of γ (Section 7.2.1) Note that in these strategies only violated cuts are added to the MP formulation (Section 4.4.2).

4.6 Overall Algorithm

We present the overall framework of the proposed asynchronous parallel BD method. Similar to Algorithm 1, the proposed method starts from the LP phase for which its pseudo code is presented in Algorithm 2.

Algorithm 2 : LP phase of the asynchronous parallel Benders method

```
1:  Create the MP which is the LP relaxation of program (9)-(11) with $E_s = \emptyset$ and
    $F_s = \emptyset$, $\forall s \in \mathcal{S}$ on the master processor
2:  Group scenarios and assign each cluster to a worker processor (Section 4.3)
3:  $LB = -\infty$, $UB = \infty$, and $t = 0$
4:  while no stopping condition is met do
5:      Solve the MP to get an optimal solution $\bar{y}$
6:      Broadcast the $\bar{y}$ solution and set $t \leftarrow t + 1$
7:      Wait until the appropriate cuts are read from the buffer (Section 4.5.2)
8:      Chose and add cuts to the MP (Section 4.4.2)
9:      Update the $UB$ (if possible) and set $LB$ equal to the current MP's objective value.
10: Clean the dominated cuts (Section 4.4.2).
```

Algorithm 2 first initiates the LP MP, groups the SPs, and assigns each cluster of SPs to a worker processor. The master processor iteratively solves the MP and adds cuts until a stopping condition is met. After solving the MP, the master solution is broadcast to all other processors provided that the solution is not the same as the previous iteration (note that the MP can generate the same solution if the applied cut subset is not violated by the current solution). After broadcasting the current solution, the master processor returns immediately and checks for the information shared by the worker processors. After retrieving the appropriate information, it adds the appropriate cuts to the master formulation, updates the bounds, and repeats the same procedure.

Every worker processor executes the same process but independently (i.e., no communications among worker processors). Algorithm 3 displays the pseudo code of this process.

Algorithm 3 The process executed on each worker processor

```
1:  Receive the set of assigned SPs; Set $\mathcal{P} = \emptyset$.
2:  Create the dual SP formulation(s)
3:  **while** *no termination signal is received* **do**
4:      Retrieve each available $\bar{y}$ solution in the buffer and set $\mathcal{P} = \mathcal{P} \cup \bar{y}$, if any
5:      **if** $\mathcal{P} = \emptyset$ **then**
6:          Wait until a new master solution is broadcast
7:      Select a master solution $\bar{y}$ from the pool $\mathcal{P}$
8:      Select an unevaluated SP associated with $\bar{y}$
9:      Update and solve the dual formulation for this SP
10:     Send a message to the master processor with the cut and bound information
11:     **if** all SPs associated with $\bar{y}$ are evaluated **then**
12:         Remove $\bar{y}$ from the pool, i.e., $\mathcal{P} = \mathcal{P} \backslash \bar{y}$
```

In Algorithm 3, each worker processor receives its unique set of SPs from the master processor. At each step, the worker processor checks if a new master solution is broadcast. If so, it retrieves and stores the solution in a pool. If the solution pool is empty, it waits until a new one is broadcast by the master processor. Next, it chooses a solution from the pool based on the appropriate *solution management* strategy. Then, it selects (based on the appropriate *solving SPs* strategy) one of the unevaluated SPs associated with the chosen solution and evaluates it. Finally, the processor sends a non-blocking message to the master processor containing the generated information (i.e., cut and bound).

In the second phase, the worker processors keep executing the same procedure, but the master processor starts branching on the master variables. The pseudo code of the overall algorithm is presented in Algorithm 4.

At each step, the master processor reads the new information from the buffer before choosing and evaluating an open node.

After selecting a node from the pool $\mathcal{L}$, if it cannot be pruned and its solution is integral, the solution will be sent to the worker processors. Then, the master processor may or may not stay idle for feedback from the worker processors. Unless all the cuts are added to the MP, it will add a complementary combinatorial cut to the formulation to ensure convergence (lines 14 and 15). If upon adding the new cuts, the master solution changes but remains integral, we repeat the same procedure (lines 22 and 23). In any case if the node solution is fractional, branching on the fractional variables occurs and the resulting child nodes are added to the set of active nodes $\mathcal{L}$ (lines 24 and 25). Finally, if one of the stopping criteria is satisfied, the master processor broadcasts the termination signal and exits.

5 Acceleration Strategies

The performance of the BD method vastly depends on the feedback it receives from the SPs at each iteration. The asynchronous parallel BD method may entail a large number of iterations and a considerable amount of redundant work as it executes the next iteration with *partial feedback* on its current solution. As a result, the lower bound may progress slowly since only a subset of the cuts are applied to the MP at each iteration. Moreover, the asynchronism may increase the computational cost of each iteration since the MP grows large at a faster pace. Finally, the proof of convergence may also be delayed due to the unavailability of the function value of the master solutions for all SPs.

To overcome these drawbacks, we propose a number of acceleration strategies. Note, one may use many of the classical acceleration techniques to improve the performance of the asynchronous algorithm. We discuss such techniques in Section 6.2. In this section, we present the novel strategies that we designed for the (asynchronous) parallel implementation of the algorithm. It should be noted that the proposed strategies in Sections 5.1 to 5.3 are applicable only when there are multiple subproblems as in the case of SIPs.

5.1 Cut Aggregation

When the number of SPs is larger than the number of first-stage variables, it is not numerically efficient to add to the MP a cut per SP (Trukhanov et al., 2010). Thus, we group the SPs into $|\mathcal{D}|$ clusters using the *k-mean++* algorithm (Arthur and Vassilvitskii, 2007), where $\mathcal{D}$ is the set of clusters with cardinality $|\mathcal{D}|$. Note that number of SPs in each cluster may not be equal. In the synchronized and sequential BD algorithms, one can define a single recourse variable for each cluster and aggregate the generated cuts in that cluster into a single cut. However, this procedure is not applicable to the asynchronous algorithm, since it may only solve a subset of the SPs in each cluster.

To alleviate this issue, we define a recourse variable for each SP. Then, we add a cut of the form: $\sum_{s\in\hat{\mathcal{S}}_d} \rho_s\theta_s \geq \sum_{s\in\hat{\mathcal{S}}_d} \rho_s\big(h_s - T_s y\big)^\top \alpha_s$ for cluster $d \in \mathcal{D}$, where $\hat{\mathcal{S}}_d$ indicates the set of evaluated SPs in this cluster at the current iteration. Note that, we can update this inequality in the following iterations when the remaining SPs in cluster d are evaluated. However, this process requires modifying the MP, which is not computationally efficient. We thus aggregate the cuts for the current solution separately from those associated with the previous iterations.

5.2 Creating Artificial SPs

The master processor does not wait for all the SPs associated with the current solution to be solved. Thus, having good cuts that can represent the unevaluated SPs is important to improve the efficiency of the asynchronous algorithm. To this end, we extend the idea presented by Crainic et al. (2021a) in order to create artificial scenarios. Note, Crainic et al. (2021a) used artificial scenarios to tighten the MP formulation while we propose using them to generate valid optimality and feasibility cuts. The SP associated with each artificial scenario is then solved to bound the recourse variables for a set of SPs., given:

Assumption 1 *We assume that the problem has a fixed recourse property and the objective coefficients are deterministic, i.e.,* $W_s = W$ *and* $c_s = c$, $\forall s \in \mathcal{S}$.

We cluster the SPs into $|\mathcal{G}|$ groups according to the similarity measure (Crainic et al., 2021a). Note that the cardinality of clusters may not be equal. Then, to generate an artificial SP for cluster $g \in \mathcal{G}$, we set $h_g = \sum_{s\in\mathcal{S}_g} \beta_s h_s$ and $T_g = \sum_{s\in\mathcal{S}_g} \beta_s T_s$, where $\mathcal{S}_g$ is the set of scenarios in cluster $g \in \mathcal{G}$ with $\mathcal{S} = \cup_{g\in\mathcal{G}}\mathcal{S}_g$, and β_s is the weight associated with scenario $s \in \mathcal{S}_g$ such that $\sum_{s\in\mathcal{S}_g} \beta_s = 1$.

Proposition 1 (Proof in A) *Any extreme point* α^g *and extreme ray* r^g *of the artificial subproblem* $g \in \mathcal{G}$ *gives a valid optimality cut* $\sum_{s\in\mathcal{S}_g} \beta_s\theta_s \geq (h_g - T_g y)^\top \alpha^g$ *or a feasibility cut* $0 \geq (h_g - T_g y)^\top r^g$.

An important issue in deriving the artificial cuts lies in setting the β weights. The following theorem suggests how to set these weights to obtain the maximum bound improvement.

Theorem 1 (Proof in C) *The maximum bound improvement by the artificial scenario* $g \in \mathcal{G}$ *is attained when the convex combination weight* β_s *for scenario* $s \in \mathcal{S}_g$ *is* $\dfrac{\rho_s}{\sum_{s\in\mathcal{S}_g}\rho_s}$.

Note that we make use of the artificial scenarios to bound the recourse variable of those SPs which remain unevaluated at the present iteration. The following corollary suggests a strategy to further tighten the associated cuts.

Corollary 1 *Let* $\bar{\mathcal{S}}_g$ *be the set of evaluated SPs in cluster* $g \in \mathcal{G}$ *at the current iteration. Then solving the artificial SP using a smaller set* $\mathcal{S}_g \backslash \bar{\mathcal{S}}_g$ *yields a tighter cut for the remaining SPs than that generated from the artificial SP associated with* $\mathcal{S}_g$. *This follows from the aggregation step in the proof of Proposition 1.*

From Corollary 1, we observe that there is no need to apply the scenario creation strategy within the synchronized or sequential algorithms, because the disaggregated cuts associated to each SP are available at each iteration.

5.3 Extracting Global Upper Bounds from Fractional Solutions

The upper bounds obtained from fractional master solutions (e.g., line 2 of Algorithm 1) are not valid for the original integer problem. Due to the high importance of the upper bound value in pruning nodes of the branch-and-bound tree, we suggest a practical strategy to extract valid incumbent values from fractional solutions. This strategy requires

Assumption 2 *If $f_i \geq 0$ and, for a given feasible $\bar{y}$, $\hat{y}$ is also feasible if $\hat{y}_i \geq \bar{y}_i$ for all $i \in \{1,...,n\}$, then $Q(\bar{y}, s) \geq Q(\hat{y}, s), s \in \mathcal{S}$.*

Proposition 2 (Proof in B) *Given a feasible fractional master solution $\bar{y}$, i.e., $\bar{y} \in Y \bigcap_{s\in\mathcal{S}}\{W_s x \geq h_s - T_s\bar{y}$, for some $x \in \mathbb{R}^m_+\}$, and the associated recourse costs ν^*_s for every $s \in \mathcal{S}$, under Assumption 2, a global upper bound can be obtained from $f^\top\lceil\bar{y}\rceil + \sum_{s\in\mathcal{S}} \rho_s \nu^*_s$, where $\lceil\rceil$ is a roundup function.*

Note that, even when Assumption 2 does not hold for a given problem, the overall algorithm still converges to an optimal solution without the results of Proposition 2.

5.4 Hybrid Parallelization

We observed that, when solving the MP and SPs is quick, re-optimizing the MP with partial feedback from the worker processors can yield efficiency drawbacks. This is particularly the case during the LP phase of our asynchronous parallel algorithm. For this reason, we propose to use the synchronized algorithm to solve the LP relaxation of the problem (i.e., set $\gamma = 1$ in line 7 of Algorithm 2) and explore the branch-and-bound tree with the asynchronous algorithm. We refer to this strategy as *hybrid parallelism.*

6 Implementation Details

We solve all LP and MILP problems using IBM ILOG CPLEX 12.7.1. All programs are coded in C++. The code is compiled with g++ 4.8.1 performed on Intel Xeon E7-8837

CPUs running at 2.67GHz with 54 cores and 64GB memory under a Linux operating system. The B&C algorithm was also implemented using CPLEX's user and lazy legacy cut callable libraries. We solve the extensive formulation with CPLEX's default setting and, following IBM's guidelines, we turned off the presolve features for our Benders-based algorithms.

We used the *openMPI* 1.8.6 to manage the communications. The openMPI library provides a convenient implementation of the standard message passing interface. It creates a pool of processors by running the executable file of our C++ code where the number of processors is an input argument. From this pool, we select the processor with rank 0 as the master processor and the remaining processors will act as the workers. The master processor creates a CPLEX instance of the MP, assigns SPs to the workers and manages the communications. Master processor broadcasts any new master solution to the workers without any delay and it reads the new messages from the queue after each iteration. Each worker has its own local memory which includes the pool of unevaluated master solutions and a vector of CPLEX instances for the assigned SPs. Note that having one CPLEX instance per SP may require additional memory but it reduces the runtime because it better utilizes the warm start capabilities and it entails less updates to a CPLEX instance. Whenever a worker solves a SP, it immediately sends the solution back to the master processor and then checks its queue for new master solutions. Note that each message is a vector of doubles with predefined length and no message is exchanged among the workers. The queues follow a FIFO policy and a message remains in the queue until the destination processor reads it. To avoid blocking any processor, asynchronous messages are passed and received using the *MPI::Comm::ISend* and *MPI::Comm::Irecv* routines of the openMPI library. To check the queue for new messages, the *MPI::Test* routine is used. Finally, the termination signal is broadcasted by the master processor upon which each worker releases its memory and exits the process. The interested reader can find complete details of this implementation at the following sample code repository https://github.com/Ragheb2464/async_BD.

6.1 Test Instances

To conduct the computational tests, we have used the **R** instances, which are widely used in the literature, e.g., Chouman et al. (2017); Rahmaniani et al. (2017b); Crainic et al. (2021a, 2011); Boland et al. (2016). These instances have up to 64 scenarios. To generate a larger number of scenarios (i.e., 1000 scenarios), we have followed a procedure similar to the one used by Boland et al. (2016). For the numerical assessment of the strategies, we have considered a subset of the instances: r04-r10 with correlation of 0.2 and cost/capacity ratio of 1, 3, and 9 which yields 21 instances. This subset of the **R** family corresponds to the instances most commonly tackled in the literature (Rahmaniani et al., 2017b; Crainic et al., 2021a). The description of these instances is given in E. Finally, for the sake of implementation simplicity, we introduce the complete recourse property to the formulation by adding a dummy "outsourcing" arc with large routing cost between each O-D pair.

6.2 Classical Acceleration Strategies

To further accelerate the B&BC algorithms, we also incorporated some of the best known classical acceleration strategies. We have thus implemented the following techniques in all of our algorithms: (i) warm start strategy , which for the first few iterations replaces the current master solution with that obtained from a convex combination with a core-point (Fischetti et al., 2017), (ii) valid inequalities for the MP and valid inequalities for each SP (Rahmaniani et al., 2017b), and (iii) Pareto-optimal cuts (Magnanti and Wong, 1981). Finally, as local branching was shown to effectively accelerate the BD method (Rei et al., 2009), we turn on the local branching of CPLEX in the second phase of our B&BC algorithms.

6.3 Implementation of the Asynchronous Algorithm

We discussed various search strategies whose combination gives a very large number of algorithms to test. Presenting the numerical results for all of them is clearly beyond the scope of this article. For this reason, we provide some insights based on our preliminary studies for those strategies which we do not present numerical results for

With respect to the *solution management* strategies of Section 4.4.1, we observed that LIFO outperforms both the FIFO and random strategies. The main reason is that the Benders method, as a "dual algorithm", is very sensitive to the feedback on its current solution. In both the FIFO and random selection strategies, "older" solutions are usually selected. As a result, the generated cuts are dominated or they have a very limited impact on the current MP. Thus, the MP generates a solution which is not very different (in terms of quality) from the previous iteration(s) and the lower bound progresses very slowly. Furthermore, in both strategies, the upper bound improves at a slower pace compared to LIFO, although they might update the upper bound more frequently at the initial iterations. This is because the FIFO and random selection strategies need a much larger number of iterations to actually find a high quality feasible solution that gives a tight bound. As a result, we only consider the LIFO as the solution selection strategy in the following.

With respect to *solving SPs*, we realized that the random selection of the SPs yields a better performance. This is because the random selection of the SPs ensures diversity of the cuts applied to the MP. In the ordering-based strategy SP2, the order of solving the SPs reaches a stationary state after a few iterations. Thus, the algorithm rarely bounds the recourse variables of the SPs with the least priority. Hence, the lower bound progresses slowly. Notice that each worker processor selects a master solution based on the LIFO strategy and not every SP associated with that solution might be solved before the master processor broadcasts a new solution. Finally, the random selection based on the criticality weights SP3 tends to perform better than a pure random strategy. This is because higher priority is given to the indicator SPs and the diversity in the cuts applied to the MP is maintained. Therefore, we consider this strategy, i.e., SP3, when selecting SPs for evaluation.

Finally, considering the proposed strategies in Section 4.5.2, we make use of MP1, i.e., waiting until $\gamma|S|$ cuts associated with the current solution are added to the MP. This is because we can use the combinatorial cuts that are violated by any integer master solution. We thus avoid the additional waiting times as required in the MP2 strategy. Moreover, we observed that applying cuts associated with the current solution is important, due to the same reasons that justified the LIFO strategy.

6.4 Stopping Criteria and Search Parameters

In solving each stochastic instance, we have set the stopping optimality gap at 1%. The total time limit is set at 2 hours. To solve the LP relaxation of the problem at the root node, we have considered half of the maximum running time limit. The parallel variants are run on 5 processors unless otherwise specified. One of the processors solves the MP and the rest are assigned to solving the SPs. In the cut generation strategy, τ, λ, and δ values are set to 0.5%, 10^{-1}, and 10^{-3}, respectively.

7 Computational Results

We present the computational assessments of the proposed parallelization strategies in two steps. We first study different versions of our parallel B&BC algorithms to evaluate the limitations and impact of the proposed acceleration techniques. The second part of the analysis is devoted to test the speedup and scalability of our parallel algorithms. Note, all the results in Section 7.1, through 7.2.3, are obtained on 5 processors. In the last Section 7.3, we study the performance with a varying number of processors.

7.1 Synchronized Parallel Algorithm

We first investigate the impact of the proposed acceleration strategies, namely the cut management, cut aggregation, and cut generation in the context of the synchronous parallel B&BC algorithm. Note that, we activate the proposed strategies one by one to observe their cumulative impact over the performance. Thus, the basic algorithm in each section is the best variant obtained from the previous subsection.

7.1.1 Cut Management

We first study the impact of the cut management strategy. We compare the method proposed in Section 4.4.2 to that commonly used in the literature, where cuts with a slack value greater than 100 are removed (e.g., Pacqueau et al., 2012). Table 2 reports the comparisons metrics in terms of the average running time in seconds, optimality gap in percentages, and number of removed cuts, shown with #Cut, for each cut management strategy. The results are obtained from running the complete algorithm (i.e., both the first and second phase).

Table 2: Numerical results for various cut management strategies

	NoCutManagement		**SlackCutManagement**			**DominanceCutManagement**		
	Time(ss)	Gap(%)	Time(ss)	Gap(%)	#Cut	Time(ss)	Gap(%)	#Cut
r04	2422.60	1.00	2424.57	1.26	151.00	1802.97	0.36	1616.00
r05	1133.64	0.47	517.22	0.47	164.33	500.64	0.47	942.67
r06	3270.45	1.76	2656.59	2.06	602.33	2601.43	2.12	22.00
r07	2437.16	3.32	2439.03	2.37	13.67	2141.70	2.06	1599.33
r08	2534.89	3.36	2513.26	3.56	50.67	2315.18	2.40	292.00
r09	4898.53	4.01	4737.94	4.24	163.67	4780.70	4.25	868.67
r10	4977.60	7.77	4927.27	5.25	645.00	4931.94	6.23	108.00
Ave.	**3096.41**	**3.10**	**2887.98**	**2.75**	**255.81**	**2724.94**	**2.56**	**778.38**

It is important to note that none of the methods deteriorates the LP bound at the root node. Moreover, all the algorithms of this section have solved the same number of instances (i.e., 61.90% of the considered instances). We observe from Table 2 that cleaning up the useless cuts yields a positive impact on performance. The slack-based strategy keeps many of the useless cuts in the MP while the proposed method removes a much larger number of them. Thus, for small and medium instances, we observe a clear advantage of the proposed method. However, for larger instances, the impact of removing cuts on the run time is less significant because the algorithm reaches the time limit. We note that most of the removed cuts are associated with early iterations. Finally, we observe that there is no direct relation between the number of cuts removed and the performance of the B&BC method in terms of average run time and gap. This is probably due to the heuristic nature of both strategies.

7.1.2 Cut Aggregation

In this part, we study the impact of the cut aggregation strategy for the synchronized parallel algorithm with the dominance based cut management. We ran the algorithm with 11 different cluster sizes in order to study the impact of various cut aggregation levels on the run time and optimality gap. The comparative results in terms of total running time are depicted in Figure 2. Note that the value on each column gives the average optimality gap in percentage.

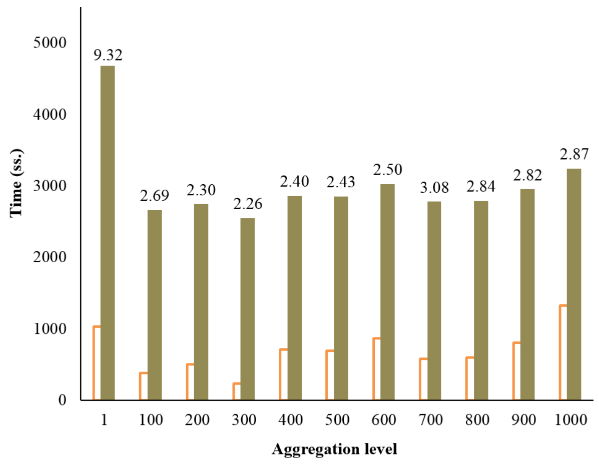

Figure 2: Impact of cluster size on the synchronized-algorithm performance (the white bars show the average run time for the solved instances)

We observe from Figure 2 that neither of the two conventional strategies, i.e., single cut (column labeled “1”) and multi-cut (column labeled “1000”), gives the best results, although the latter performs noticeably better than the former. The best aggregation level is associated with a cluster size of 300. All aggregation levels are able to solve 66.67% of the same instances within the 2-hour time limit, except for the single cut method which could only solve 61.90% of the instances. In addition, we observe that the difference in running time among some aggregation levels is small. This is because the algorithm reaches the maximum run time limit for 33.33% of the instances. Thus, if we only consider the instances that are solved by all aggregation levels, we observe more significant differences in the running times. These results are presented by the white bars in Figure 2.

7.2 Asynchronous Parallel Algorithm

We study the proposed strategies for the asynchronous parallel algorithm. We first study different synchronization levels as controlled by the γ parameter. Then, we analyze the proposed acceleration techniques.

7.2.1 The Synchronization Level

Figure 3 displays the impact of the γ value on the running time. The valuesion this figure represent the average optimality gaps in percentage.

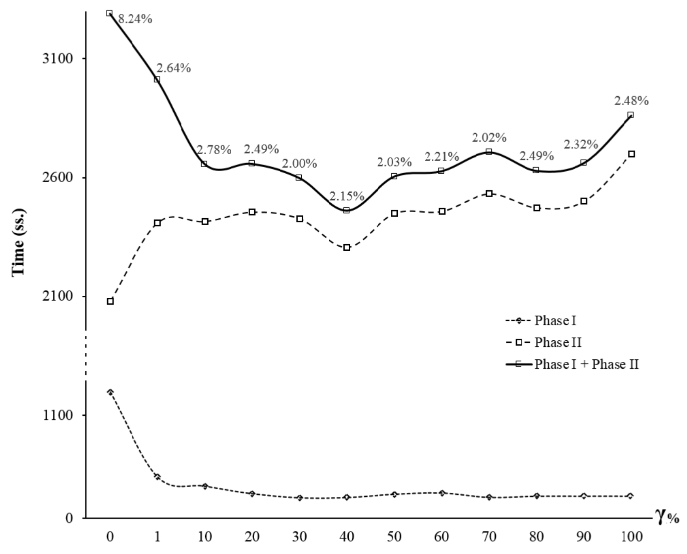

Figure 3: Effect of the master-processor waiting (γ value) on the performance of the algorithm and its phases

We observe that the best performance (i.e., lowest run time and optimality gap) lies neither at 100% nor at 0%. It is always better to wait for some percentage of cuts in between. A reasonable compromise in performance is attained when the master processor waits until 40% of the SPs associated with the current solution are solved. This synchronization level solves 66.67% of the instances with an average optimality gap of 2.15% in 2465.26 seconds. This is comparable to the synchronous algorithm without the cut generation strategies in terms of run time, optimality gap, and number of solved instances (Figure 2).

We observe that waiting for a larger portion of the SPs is more computationally beneficial when solving the LP relaxation of the MP, while the contrary is true in the second phase of the algorithm. In the LP phase, we gain nothing from quickly resolving the MP since doing that would generally yield a much larger number of iterations to converge as we are only applying partial feedbacks to the MP. In the second phase, however, we can perform useful work (i.e., evaluating open nodes of the search tree) while the cuts are being generated. We thus consider $\gamma = 100\%$ and 0% in the first and second phase of our hybrid algorithm.

7.2.2 Artificial Scenario Creation

We study the impact of creating artificial scenarios on the asynchronous parallel algorithm with $\gamma = 40\%$. To do so, we have created one artificial SP per worker processor. Each time a worker receives a new solution, it first solves and returns the associated artificial cut with that solution. Through the figures of Table 3, we investigate the impact of the artificial SPs on the convergence of our asynchronous algorithm in terms of run time, optimality gap and number of solved instances.

Table 3: Impact of the artificial subproblems on the performance of the asynchronous algorithm

	NoArtificialSP			NumberOfArtificialSPs=$\|\mathcal{P}\|$		
	Time(ss.)	Gap(%)	Sol.(%)	Time(ss.)	Gap(%)	Sol.(%)
r04	689.50	0.75	100.00	759.39	0.73	100.00
r05	274.84	0.51	100.00	286.64	0.42	100.00
r06	2518.45	1.86	66.67	2546.88	1.48	66.67
r07	1442.27	0.84	66.67	1412.31	0.90	66.67
r08	2521.60	2.65	66.67	2466.25	2.02	66.67
r09	4898.42	3.98	33.33	4859.83	3.46	66.67
r10	4911.77	5.07	33.33	4880.33	4.53	33.33
Ave.	2465.26	2.25	66.67	2458.80	1.93	71.43

We observe in Table 3 that the creation of artificial scenarios increases the percentage of the solved instances by 4.76% (i.e., 1 instance) while the average run time remains almost unchanged. The running time with artificial SPs increases for small instances and for larger instances the time improvement is not very much noticeable. In small problems, additional time is spent on generating cuts while they could have been solved quicker. For larger instances, in many cases, the algorithm reaches the time limit and thus the improvement on the average time does not appear significant. We observe, however, that the average optimality gap has been reduced as the additional cuts improve the lower bound. Thus, the artificial SPs are a valid strategy to accelerate the asynchronous algorithm.

7.2.3 Cut Aggregation

The conclusions of Section 7.1.1 relative to the cut-management strategies also apply here. With respect to aggregating cuts, we study its impact on the asynchronous parallel algorithm with $\gamma = 40\%$ and one artificial SP per processor. Figure 4 reports this impact, in terms of running time and optimality gap, for different cluster sizes. The value on each bar shows the average optimality gap in percentage.

We observe that clustering reduces the average optimality gap and the run time of the algorithm. The best aggregation level, the one that produces the lowest run time and optimality gap, is reached for 500 clusters. Notice that this number is greater than

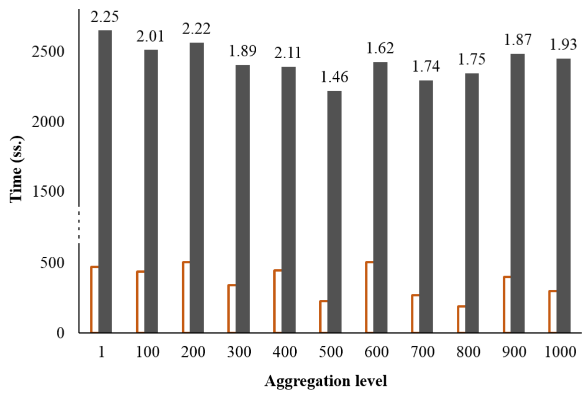

Figure 4: Impact of the cut aggregation level on the asynchronous-algorithm performance (the white bars show the average run time for the solved instances)

the best value found for the synchronized algorithm, i.e., a cluster size of 300. With this aggregation level, our asynchronous algorithm solves 76.19% of the instances with an optimality gap of 1.46% in 2219.42 seconds and thus it outperforms the synchronized algorithm.

7.3 Speedup of the Parallel Algorithms

We compare our parallel algorithms on 2, 3, 5, 10, 15, and 20 processors (Figure 5). To have a clear sense of the speedups, we considered only those instances that the sequential algorithm could solve within a 10-hour time limit (i.e., 15 instances). Note that the value on each bar indicates the speedup ratio calculated as the time of the best sequential algorithm divided by the time of the parallel algorithm.

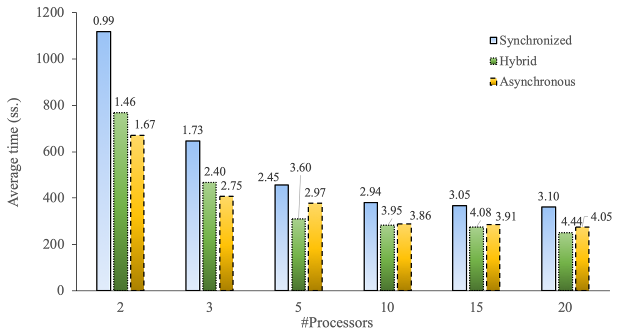

Figure 5: Speedup rates of our parallel Benders decomposition algorithms

The first interesting observation is related to the use of 2 processors, i.e., 1 worker and 1 master processor. In this case, the synchronized algorithm performs slightly slower than the sequential one because of the cut generation and communication overheads. Our asynchronous algorithm, on the contrary, performs much better, which is clearly due to the better use of the processors since the two processors are less dependent. Also, we observe that the hybrid algorithm is better than the synchronized one, although they are roughly the same during the LP phase of the algorithm. Thus, the speedup rate in our hybrid algorithm must be due to its asynchronous phase. This indicates that our asynchronous method has noticeably reduced the computational bottleneck at the master level.

The speedups do not increase monotonically with the number of processors. This is because the system is constrained by the Amdahl's law and increasing the number of worker processors does not alleviate the bottleneck at the master processor, although it significantly accelerates the cut generation cycle. Particularly, in phase II of the algorithm, the worker processors are not efficiently used and the heavy work is carried out by a single processor.

Our third observation concerns the hybrid method. It reaches better speedusp than the synchronized method because of using non-blocking communications in phase II, which is the most time consuming part of the algorithm; see Figure 3. The advantage of the hybrid algorithm over the asynchronous method becomes more evident for larger instances in which solving the LP relaxation is noticeably time consuming. Moreover, we observe that its efficiency exceeds, or becomes closer to, the asynchronous method as the number of processors increases. This justifies the development of the hybrid algorithm.

8 Conclusions and Remarks

We studied parallelization strategies for the Benders decomposition method in which the subproblems are concurrently solved on different processors and the master problem on a single processor. We implemented the algorithm in a B&C framework and presented synchronous, asynchronous, and hybrid parallelization frameworks along with various acceleration strategies. We designed the asynchronous algorithm such that it alleviates the interdependency of the master and subproblems.

We reported computational results on hard benchmark instances from stochastic network design problems with 1000 scenarios. We observed that the hybrid algorithm generally reaches a better performance compared to the sequential, synchronized, and asynchronous methods. However, the overall parallel algorithms did not reach a linear speedup. We also observed that our parallelizations did not scale with the number of processors. The main reason for this is the fact that the most significant computational bottleneck of the parallel algorithm is solving the master problem sequentially on a single processor. All in all, the proposed asynchronous and hybrid algorithms displayed a better performance compared to the synchronous method.

This research opens the way for a number of interesting issues to be considered in future works. First and foremost, the master problem in our algorithm needs to be solved in parallel in order to further reduce the idle time and reach a more scalable algorithm. Second, it is worthwhile to study the proposed cut propagation strategy for the problems where generating a single optimality cut is significantly time consuming. Third, some of the well-known acceleration strategies for the Benders method need to be revisited in order to be properly applied in the parallel environment. An example would be the partial decomposition strategy of Crainic et al. (2021a). Heuristics are widely used to accelerate the BD method, but we are not aware of any integration of these methods in a parallel (cooperative) framework. Last but not least, it will be worthwhile to evaluate the proposed algorithm on different problems to further assess its computational advantage.

Acknowledgments

While working on the project, the second author held the UQAM Chair on Intelligent Logistics and Transportation Systems Planning and was Adjunct Professor, Department of Computer Science and Operations Research, Université de Montréal (DIRO), the third was Adjunct Professor, DIRO, and the fourth held the Canada Research Chair in Stochastic Optimization of Transport and Logistics Systems of UQAM. We gratefully acknowledge the financial support provided by the Natural Sciences and Engineering Research Council of Canada (NSERC), through its Discovery Grant program, and by the Fonds de recherche du Québec through its Team Grant program. We also gratefully acknowledge the support of Fonds de recherche du Québec through their CIRRELT strategic infrastructure grants.

References

S. Ahmed. Two-stage stochastic integer programming: A brief introduction. In J. J. Cochran, L. A. Cox, P. Keskinocak, J. P. Kharoufeh, and J. C. Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*, pages 1–10. John Wiley & Sons, 2010.

T. W. Archibald, C. S. Buchanan, K. I. M. McKinnon, and L. C. Thomas. Nested Benders decomposition and dynamic programming for reservoir optimisation. *Journal of the Operational Research Society*, 50(5):468–479, 1999.

K. A. Ariyawansa and D. D. Hudson. Performance of a benchmark parallel implementation of the Van Slyke and Wets algorithm for two-stage stochastic programs on the sequent/balance. *Concurrency: Practice and Experience*, 3(2):109–128, 1991. ISSN 1040-3108.

D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

J. R. Birge and F. Louveaux. *Introduction to stochastic programming.* Springer, New York, 1997.

N. Boland, M. Fischetti, M. Monaci, and M. Savelsbergh. Proximity Benders: a decomposition heuristic for stochastic programs. *Journal of Heuristics*, 22(2):181–198, 2016.

D. P. Chermakani. Optimal aggregation of blocks into subproblems in linear programs with block-diagonal-structure. *Available at https://arxiv.org/ftp/arxiv/papers/1507/1507.05753.pdf*, 2015.

M. Chouman, T. G. Crainic, and B. Gendron. Commodity representations and cut-set-based inequalities for multicommodity capacitated fixed-charge network design. *Transportation Science*, 51(2):650–667, 2017.

T. G. Crainic and M. Toulouse. Parallel Metaheuristics. In T. G. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, pages 205–251. Springer, Boston, MA, 1998.

T. G. Crainic, B. Le Cun, and C. Roucairol. Parallel Branch-and-Bound algorithms. In E.-G. Talbi, editor, *Parallel Combinatorial Optimization*, chapter 1, pages 1–28. John Wiley & Sons, 2006.

T. G. Crainic, X. Fu, M. Gendreau, W. Rei, and S. W. Wallace. Progressive hedging-based metaheuristics for stochastic network design. *Networks*, 58(2):114–124, 2011.

T. G. Crainic, M. Hewitt, F. Maggioni, and W. Rei. Partial Benders decomposition: General methodology and application to stochastic network design. *Transportation Science*, 55(32):414–435, 2021a.

T.G. Crainic and M. Gendreau. Heuristics and Metaheuristics for Fixed-Charge Network Design. In T.G. Crainic, M. Gendreau, and B. Gendron, editors, *Network Design with Applications in Transportation and Logistics*, chapter 4, pages 91–138. Springer, Boston, 2021.

T.G. Crainic and B. Gendron. Exact Methods for Fixed-Charge Network Design. In T.G. Crainic, M. Gendreau, and B. Gendron, editors, *Network Design with Applications in Transportation and Logistics*, chapter 3, pages 29–89. Springer, Boston, 2021.

T.G. Crainic, M. Gendreau, and B. Gendron. Fixed-Charge Network Design Problems. In T.G. Crainic, M. Gendreau, and B. Gendron, editors, *Network Design with Applications in Transportation and Logistics*, chapter 2, pages 15–28. Springer, Boston, 2021b.

G. B. Dantzig, J. K. Ho, and G. Infanger. Solving stochastic linear programs on a hypercube multicomputer. Technical Report ADA240443, DTIC Document, August 1991.

M. A. H. Dempster and R. T. Thompson. Parallelization and aggregation ofnested Benders decomposition. *Annals of Operations Research*, 81(0):163–188, Jun 1998.

M. Fischetti, I. Ljubic, and M. Sinnl. Redesigning Benders decomposition for large-scale facility location. *Management Science*, 63(7):2049–2395, 2017. doi: 10.1287/mnsc.2016.2461.

M. Hewitt, W. Rei, and S.W. Wallace. Stochastic Network Design. In T.G. Crainic, M. Gendreau, and B. Gendron, editors, *Network Design with Applications in Transportation and Logistics*, chapter 10, pages 283–315. Springer, Boston, 2021.

J. L. Higle and S. Sen. Stochastic decomposition: An algorithm for two-stage linear programs with recourse. *Mathematics of Operations Research*, 16(3):650–669, 1991.

W. Klibi, A. Martel, and A. Guitouni. The design of robust value-creating supply chain networks: A critical review. *European Journal of Operational Research*, 203(2):283 – 293, 2010.

A. Langer, R. Venkataraman, U. Palekar, and L. V. Kale. Parallel branch-and-bound for two-stage stochastic integer optimization. In *20th Annual International Conference on High Performance Computing*, pages 266–275, Dec 2013.

J. M. Latorre, S. Cerisola, A. Ramos, and R. Palacios. Analysis of stochastic problem decomposition algorithms in computational grids. *Annals of Operations Research*, 166 (1):355–373, 2009.

X. Li. Parallel nonconvex generalized Benders decomposition for natural gas production network planning under uncertainty. *Computers & Chemical Engineering*, 55:97 – 108, 2013.

J. Linderoth and S. Wright. Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, 24(2-3):207–250, 2003.

T. L. Magnanti and R. T. Wong. Accelerating Benders decomposition: Algorithmic enhancement and model selection criteria. *Operations Research*, 29(3):464–484, 1981.

J. Mateo, L. M. Plà, F. Solsona, and A. Pagès. A scalable parallel implementation of the cluster Benders decomposition algorithm. *Cluster Computing*, Dec 2018. ISSN 1573-7543. doi: 10.1007/s10586-018-2878-4.

D. McDaniel and M. Devine. A modified Benders' partitioning algorithm for mixed integer programming. *Management Science*, 24(3):312–319, 1977.

H. W. Moritsch, G. C. Pflug, and M. Siomak. Asynchronous nested optimization algorithms and their parallel implementation. *Wuhan University Journal of Natural Sciences*, 6(1):560–567, Mar 2001.

J. Naoum-Sawaya and S. Elhedhli. An interior-point Benders based branch-and-cut algorithm for mixed integer programs. *Annals of Operations Research*, 210(1):33–55, 2013.

S. S. Nielsen and S. A. Zenios. Scalable parallel Benders decomposition for stochastic linear programming. *Parallel Computing*, 23(8):1069–1088, 1997.

L. Ntaimo. Disjunctive decomposition for two-stage stochastic mixed-binary programs with random recourse. *Operations Research*, 58(1):229–243, 2010.

R. Pacqueau, S. Francois, and H. Le Nguyen. A fast and accurate algorithm for stochastic integer programming, appllied to stochastic shift scheduling. Publication G-2012-29, Groupe d'études et de recherche en analyse des décisions (GERAD), Université de Montréal, Montréal, QC, Canada, 2012.

L. Pfeiffer, R. Apparigliato, and S. Auchapt. Two methods of pruning Benders' cuts and their application to the management of a gas portfolio. *Optimization Online, Available at http://www.optimization-online.org/DB_FILE/2012/11/3683.pdf*, 2012.

R. Rahmaniani, T. G. Crainic, M. Gendreau, and W. Rei. The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801 – 817, 2017a.

R. Rahmaniani, T. G. Crainic, M. Gendreau, and W. Rei. A Benders decomposition method for two-stage stochastic network design problems. Publication CIRRELT-2017-22, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, Université de Montréal, Montréal, QC, Canada, 2017b.

R. Rahmaniani, S. Ahmed, T. G. Crainic, M. Gendreau, and W. Rei. The Benders dual decomposition method. *Operations Research*, 68(3):878–895, 2020.

T. Ralphs, L. Ladányi, and M. Saltzman. Parallel branch, cut, and price for large-scale discrete optimization. *Mathematical Programming*, 98(1):253–280, Sep 2003.

W. Rei, J.-F. Cordeau, M. Gendreau, and P. Soriano. Accelerating Benders decomposition by local branching. *INFORMS Journal on Computing*, 21(2):333–345, 2009.

R. T. Rockafellar and R. J.-B. Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16(1):119–147, 1991.

A. Ruszczyński. Decomposition methods in stochastic programming. *Mathematical Programming*, 79(1):333–353, 1997.

S. Sen. Subgradient decomposition and differentiability of the recourse function of a two stage stochastic linear program. *Operations Research Letters*, 13(3):143 – 148, 1993.

D. A. Tarvin, R. K. Wood, and A. M. Newman. Benders decomposition: Solving binary master problems by enumeration. *Operations Research Letters*, 44(1):80–85, 2016.

S. Trukhanov, L. Ntaimo, and A. Schaefer. Adaptive multicut aggregation for two-stage stochastic linear programs with recourse. *European Journal of Operational Research*, 206(2):395–406, 2010.

S. Uryasev and P. M. Pardalos. *Stochastic optimization: algorithms and applications*, volume 54. Springer Science & Business Media, 2013.

R. M. Van Slyke and R. Wets. L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17(4):638–663, 1969.

H. Vladimirou. Computational assessment of distributed decomposition methods for stochastic linear programs. *European Journal of Operational Research*, 108(3):653–670, 1998.

C. Wolf and A. Koberstein. Dynamic sequencing and cut consolidation for the parallel hybrid-cut nested l-shaped method. *European Journal of Operational Research*, 230 (1):143 – 156, 2013.

H. Yang, J. N. Gupta, L. Yu, and L. Zheng. An improved L-shaped method for solving process flexibility design problems. *Mathematical Problems in Engineering*, 2016, 2016.

J. Zou, S. Ahmed, and X. A. Sun. Multistage stochastic unit commitment using stochastic dual dynamic integer programming. *Optimization Online, Available at http://www.optimization-online.org/DB_HTML/2017/05/6003.html*, 2017.

A Proof of Proposition 1

If $|\mathcal{S}_g| = 1$, the results follows immediately from the disaggregated version of the BD method Van Slyke and Wets (1969). To prove the validity of the cuts for $1 < |\mathcal{S}_g| \leq |\mathcal{S}|$, we need to show that the derived SP gives a valid lower approximation of the aggregated recourse variables for any $y \in Y$.

$$\theta_s \geq \min_{x \in \mathbb{R}^m_+} \{c^\top x_s : \; W x_s \geq h_s - T_s y\} \qquad s \in \mathcal{S}_g$$

$$\beta_s \geq 0 \; \rightarrow \; \beta_s \theta_s \geq \min_{x \in \mathbb{R}^m_+} \{\beta_s c^\top x_s : \; W x_s \geq h_s - T_s y\} \qquad s \in \mathcal{S}_g$$

$$\sum_{s \in \mathcal{S}_g} \beta_s \theta_s \geq \min_{x \in \mathbb{R}^{m|\mathcal{S}|}_+} \{\sum_{s \in \mathcal{S}_g} \beta_s c^\top x_s : W x_s \geq h_s - T_s y, \; s \in \mathcal{S}_g\} \geq$$

$$\min_{x \in \mathbb{R}^{m|\mathcal{S}|}_+} \{c^\top \sum_{s \in \mathcal{S}_g} \beta_s x_s : \sum_{s \in \mathcal{S}_g} W \beta_s x_s \geq \sum_{s \in \mathcal{S}_g} \beta_s (h_s - T_s y)\}$$

The last inequality holds because we have aggregated the constraints using convex combination weights $1 \geq \beta_s \geq 0$ such that $\sum_{s \in \mathcal{S}_g} \beta_s = 1$. We next consider a variable transformation $\sum_{s \in \mathcal{S}_g} \beta_s = 1, \; x_g = \sum_{s \in \mathcal{S}_g} \beta_s x_s, \; h_g = \sum_{s \in \mathcal{S}_g} \beta_s h_s, \; T_g = \sum_{s \in \mathcal{S}_g} \beta_s T_s$. Thus,

$$\sum_{s \in \mathcal{S}_g} \beta_s \theta_s \geq \min_{x \in \mathbb{R}^m_+} \{c^\top x_g : \; W x_g \geq h_g - T_g y\} = \max_{\alpha \in \mathbb{R}^l_+} \{(h_g - T_g y)^\top \alpha : \; W^\top \alpha \leq c\} \quad \forall y \in Y.$$

Also, we observe that the dual polyhedron is identical to a regular dual SP which indicates the validity of the feasibility cut. □

B Proof of Proposition 2

It is trivial to observe that $\lceil \bar{y} \rceil \in Y$ due to the assumption. Let assume that the recourse cost associated with the rounded solution $\lceil \bar{y} \rceil$ is known and given by ν'_s for each $s \in \mathcal{S}$. Thus, for this integer solution, a valid upper bound can be calculated from $f^\top \lceil \bar{y} \rceil + \sum_{s \in \mathcal{S}} \rho_s \nu'_s \geq z^*$. On the other hand, based on the assumption, we have $\nu^*_s = Q(\bar{y}, s) \geq Q(\lceil \bar{y} \rceil, s) = \nu'_s$. Thus, $f^\top \lceil \bar{y} \rceil + \sum_{s \in \mathcal{S}} \rho_s \nu^*_s \geq f^\top \lceil \bar{y} \rceil + \sum_{s \in \mathcal{S}} \rho_s \nu'_s \geq z^*$. □

C Proof of Theorem 1

For an arbitrary first-stage solution $y \in Y$, let $\sigma^g = (h_g - T_g y)^\top \alpha_g$ be the right hand side of the optimality cut generated from artificial scenario $g \in \mathcal{G}$ and let $\theta^*_s = (h_s - T_s y)^\top \alpha^{i^*}_s$, where $i^* \in \arg\max_{i\in E^t_s}(h_s - T_s y)^\top \alpha^i_s$ for which E^t_s is the set of optimality cuts associated with scenario s at iteration t. If the cut from the artificial SP is violated by the y solution, it means that $\sigma^g > \sum_{s\in\mathcal{S}_g}\beta_s\theta^*_s$. For a given y solution, the MP can be separated for each cluster $g \in \mathcal{G}$ and can be defined as:

$$\begin{aligned}\min \sum_{s\in\mathcal{S}_g} \rho_s\theta_s & \\ \sum_{s\in\mathcal{S}_g} \beta_s\theta_s \geq \sigma^g & \\ \theta_s \geq \theta^*_s \qquad & s \in \mathcal{S}_g.\end{aligned}$$

It can be shown that the above formulation has a knapsack structure and since $\sigma^g > \sum_{s\in\mathcal{S}_g}\beta_s\theta^*_s$, the recourse variable $\theta_{\tilde{s}}$ takes the extra violation $\sigma^g - \sum_{s\in\mathcal{S}_g}\beta_s\theta^*_s > 0$, where $\tilde{s} \in \arg\min_{s\in\mathcal{S}_g}\frac{\rho_s}{\beta_s}$. This gives $\theta_{\tilde{s}} = \theta^*_{\tilde{s}} + \frac{1}{\beta_{\tilde{s}}}(\sigma^g - \sum_{s\in\mathcal{S}_g}\beta_s\theta^*_s)$ which results in a lower bound improvement of $\Delta := \frac{\rho_{\tilde{s}}}{\beta_{\tilde{s}}}(\sigma^g - \sum_{s\in\mathcal{S}_g}\beta_s\theta^*_s)$ at the given y solution. The maximum value of Δ is achieved when $\beta_s = \rho_s$. Considering $\sum_{s\in\mathcal{S}_g}\beta_s = 1$ and $\mathcal{S}_\}\leq |\mathcal{S}|$, the maximum value of Δ is achieve at $\beta_s = \frac{\rho_s}{\sum_{s\in\mathcal{S}_g}\rho_s}$. $\square$

D Restricted MILP Problem

This problem is derived by considering the extensive formulation for a small subset of scenarios and fixing variables to 1 if their current value is greater than $1 - \hat{\lambda}$ and to 0 if it is less than $\hat{\lambda}$.

E Test Instances

Table 4 presents the details for the instances used in this article. These instances can be found at https://github.com/Ragheb2464/async_BD/tree/main/instances-R.

Table 4: Attributes of the instance classes

Name	$\|N\|$	$\|A\|$	$\|K\|$	$\|\Omega\|$	Cost/Capacity Ratio	Correlation	#Instances
r04	10	60	10	1000	1, 3, 9	0.2	3
r05	10	60	25	1000	1, 3, 9	0.2	3
r06	10	60	50	1000	1, 3, 9	0.2	3
r07	10	82	10	1000	1, 3, 9	0.2	3
r08	10	83	25	1000	1, 3, 9	0.2	3
r09	10	83	50	1000	1, 3, 9	0.2	3
r10	20	120	40	1000	1, 3, 9	0.2	3
r11	20	120	100	1000	1, 3, 9	0.2	3

F Numerical results of the sequential algorithm

To make fair comparisons, we have incorporated the techniques that we developed in sections 4.4.2 , 5.1 and 5.3 into our sequential method (presented in Algorithm 1). We have also incorporated the classical acceleration techniques which we have used in our parallel algorithms, see section 6.

In this appendix, we present some numerical results to complement our numerical assessments in section 7. In Figure 6, we thus study impact of the cut aggregation over the sequential algorithm. For each aggregation level, we have ran the algorithm for 2 hours. The value on each column indicates the average optimality gap in percentages.

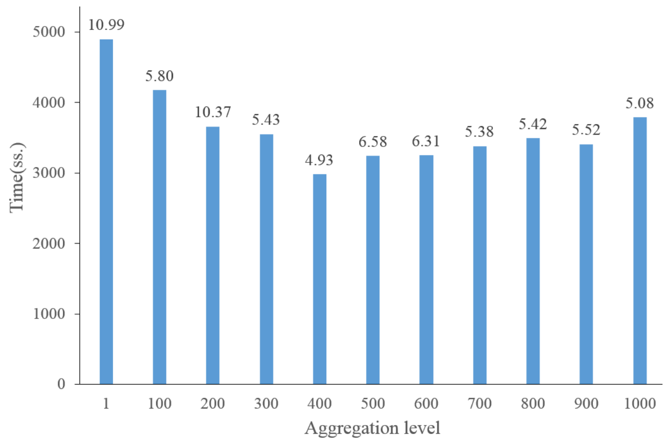

Figure 6: Comparison of various cut aggregation levels for the sequential B&BC method

Comparing the results to those of the synchronized algorithm, see Figure 2, we observe that a larger number of clusters gives the best results for the sequential algorithm. This is because the sequential algorithm performs less iterations in the same amount of time and thus, it is more sensitive to the loss of information in the aggregation step. Moreover, comparing Figures 3 and 6, we observe that full asynchronism (i.e., $\gamma = 0$) underperforms even when compared to the sequential algorithm.

We next present the numerical performance of the sequential algorithm for a run time limit of 10 hours. The results are presented in Table 5.

Table 5: Numerical results of the sequential B&BC algorithm

	#Instance	Time to solve LP	Total Time	Gap(%)	Sol.(%)
r04	3	31.43	1098.21	0.36	100.00
r05	3	159.20	798.77	0.47	100.00
r06	3	775.99	14671.58	1.23	66.67
r07	3	34.81	12087.66	0.76	66.67
r08	3	224.44	12259.00	2.34	66.67
r09	3	1188.19	24270.19	2.79	33.33
r10	3	3582.01	24521.04	6.80	33.33
Ave.	**3**	**856.58**	**12815.21**	**2.11**	**66.67**

Algorithm 4 : The asynchronous parallel Benders method

```
1: Let the obtained $MP$ from Algorithm 2 be the root node of tree $\mathcal{L}$, set $UB = \infty$,
   and let $\epsilon_{opt}$ to be the optimality tolerance.
2: while no stopping condition is met do
3:     Read all the messages in the buffer
4:     Add the cuts (if any) to the MP according to the scheduling (Section 4.5) and
   selection (Section 4.4.2) strategies
5:     Update the $UB$, if possible
6:     Select node $l$ from $\mathcal{L}$
7:     Solve node $l$ to get an optimal solution $\bar{y}$ with objective value of $\vartheta^*$
8:     if node $l$ is infeasible or $\vartheta^* \geq UB$ then
9:         Prune the node (i.e., $\mathcal{L} = \mathcal{L} \backslash l$) and go to line 2
10:    if $\bar{y}$ is integer then
11:        Broadcast $\bar{y}$
12:        Wait for cuts according to the selected scheduling strategy (Section 4.5.2)
13:        Add the new cuts, if any
14:        if no cut is violated by $\bar{y}$ and not all cuts associated with $\bar{y}$ are generated
   then
15:            Add a combinatorial cut
16:        Solve this node again and get the objective value $\vartheta^*$
17:        if node was infeasible or $\vartheta^* \geq UB$ then
18:            Prune the node and go to line 2
19:        if the current node solution is the same as $\bar{y}$ then
20:            if all cuts associated with $\bar{y}$ are added to the MP then
21:                Prune node $l$, and go to line 2
22:        else if current node solution is integer then
23:            Set $\bar{y}$ equal to the current node solution and go to line 11
24:    Remove node $l$ from $\mathcal{L}$ and choose a fractional variable from $\bar{y}$ for branching
25:    Create two nodes and add them to $\mathcal{L}$.
26: Broadcast the termination signal.
```