CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

The Asynchronous Benders Decomposition Method

Ragheb Rahmaniani
Teodor Gabriel Crainic
Michel Gendreau
Walter Rei

January 2018

CIRRELT-2018-07

Bureaux de Montréal :
Université de Montréal
Pavillon André-Aisenstadt
C.P. 6128, succursale Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

The Asynchronous Benders Decomposition Method

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

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

[2] Department of Mathematics and Industrial Engineering, Polytechnique Montréal, P.O. Box 6079, Station Centre-Ville, Montréal, Canada H3C 3A7

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

Abstract. The Benders decomposition method is a widely used approach in addressing stochastic integer programs with recourse. The available parallel variants of this method are based on a synchronized master-slave implementation in which the slave processors wait until the master problem is solved, and vice versa. This parallelization may, however, suffer from significant load imbalance, particularly when an iteration of the master problem can take hours. We thus propose parallelization strategies for the Benders decomposition algorithm in a branch-and-cut framework. To further reduce the idle times, we relax the synchronization requirements among the master and slave processors. However, the combination of the asynchronous communications and branch-and-cut implementation results in an algorithm for which we are unable to prove its global convergence and it may even underperform the sequential algorithm. Therefore, we study the convergence of the algorithm and propose various acceleration strategies to obtain an effective parallel method. We conduct an extensive numerical study on benchmark instances from stochastic network design problems. The results indicate that our asynchronous algorithm reaches higher speedup rates compared to the conventional parallel methods. We also show that it is several orders of magnitude faster than the state-of-the-art solvers.

Keywords: Benders decomposition, branch-and-cut, stochastic integer programming, parallel computing.

Acknowledgements. Partial funding for this project has been provided by the Natural Sciences and Engineering Council of Canada (NSERC), through its Discovery Grant program and by the Fonds de recherche du Québec through its Team Grant program. We also gratefully acknowledge the support of Fonds de recherche du Québec through their strategic infrastructure grants.

---

* Corresponding author: teodorgabriel.crainic@cirrelt.net

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

1. Introduction

Stochastic integer programming (SIP) offers a powerful tool to deal with uncertainties in planning problems where the distribution of the uncertain parameters is assumed to be known and often characterized with a finite set of discrete scenarios (Birge and Louveaux 1997). We consider here the special two-stage case of SIPs where decisions are made in two stages. In the *first-stage* the decision maker must make a decision now, while not knowing the exact outcome of the uncertain parameters. In the *second-stage* when the parameters have become known, the decision maker can take recourse actions to adjust his/her plan accordingly.

SIP models are usually very large in size and very difficult to solve due to the data uncertainty and their combinatorial nature (Ahmed 2010). However, they exhibit special structures amenable to decomposition methods (Ruszczyński 1997). Therefore, efforts have been made to design various decomposition-based algorithms for these problems, e.g., Benders decomposition (BD) (Benders 1962) also known as 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), and disjunctive decomposition (Ntaimo 2010), etc. Among these methods, the BD has become a prominent methodology to address stochastic programs with recourse (Ruszczyński 1997, Uryasev and Pardalos 2013).

In the BD method the model is projected onto the subspace defined by the first-stage variables. The projected term is then replaced by its dual counterpart. Accordingly, an equivalent model is built by iteratively enumerating the extreme points and rays of the dual program, referred to as the *subproblem* (SP). At each iteration, one solves a *master problem* (MP), which initially includes no constraints except for those imposed on the first-stage variables. The obtained solution is fixed in the SP. If the resulting SP is feasible, the corresponding optimal extreme point is used to generate an *optimality cut*. Otherwise, an extreme ray is extracted to generate a *feasibility cut*. The generated cut is then added to the MP and this process repeats until an optimal solution is found. If the MP is a mixed-integer program, 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 build and the cuts are generated at the integer (and possibly some fractional) nodes (Rahmaniani et al. 2017a).

The SP decomposes by scenario. Solving these scenario SPs at each iteration usually corresponds to the most time-consuming part of the algorithm, because a large number of scenarios are often required to properly set the value of uncertain parameters. These SPs are disjoint and can be solved in parallel. Thus, *parallel computing* appears very promising to effectively accelerate solution of SIP problems when the BD method is used (Linderoth and Wright 2003, Li 2013).

Although parallel processing saves time, the processors at some point require to exchange information and consolidate the results to create work units for next iteration. In the parallel BD methods of the literature, these points are implemented using synchronized communications among the processor that solves the MP and the processors that solve the SPs. This, however, causes having one or several idle processors at any given time, particularly when an iteration of the MP can take hours (Yang et al. 2016). In this case, the efficiency of the parallel execution may decrease as the number of the processors increases. It is thus important to design new parallelization schemes for the BD method to obtain a high-performing algorithm for the SIPs.

In this article we aim at developing effective parallelization strategies for the B&C implementation of the BD method. To the best of our knowledge, this is the first article to consider parallelization of the BD method in a B&C framework. It is important to note that in this case, the resulting parallel algorithm is different from the existing parallel B&C algorithms. In the parallel B&C methods the main emphasis is on parallelizing the branch-and-bound tree and the cuts are only 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 and the cuts are necessary for the convergence. Thus, the parallelization strategies of the B&C algorithms cannot easily be translated into parallel BD methods and, we focus on the strategies in which the SPs are optimized in parallel on various *slave* processors and the MP is solved sequentially on a single *master* processor.

To realize our goal, we relax the synchronization requirements among the master and slave processors. This means that the algorithm waits only for a small portion of the cuts at each (integer) node of the branch-and-bound tree. Although this significantly reduces the idle times, it results in an algorithm for which we are unable to prove its global convergence. This happens because, in absence of a rigid synchronization, the necessary cuts at the integer nodes may not be generated 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 executes its next iteration with a *partial* feedback from the SPs. This can cause serious efficiency issues such that the algorithm may even underperform the sequential variant. Therefore, we propose various acceleration techniques to obtain an efficient asynchronous BD algorithm. The main contributions of this article are thus severalfold:

- Proposing an effective Benders-based *asynchronous* parallel B&C algorithm for two-stage SIP problems. We also present the *synchronized* algorithm and an *hybrid* of the two algorithms;
- Studying the convergence of these algorithms and proposing strategies to accelerate their numerical performance. In this regard, we revisit some of the classical acceleration techniques to properly fit into our parallel frameworks and we propose novel ones;

- Presenting extensive numerical results on benchmark instances to assess the proposed strategies and algorithms. We provide guidelines on how to properly use the parallelization techniques and discuss various fruitful research directions.

The remainder of this article is organized as follows. In section 2, the problem of interest and the sequential BD algorithm are presented. In section 3, we classify and review parallel BD methods. We present our synchronized, asynchronous, and hybrid parallel algorithms in sections 4, 5, and 6, respectively. We discuss the implementation details and numerical results in sections 7 and 8. Finally, conclusions and future remarks are summarized in the last section.

2. The Benders Decomposition Method

In this section, we first recall the two-stage stochastic problem of interest. Then, we present a sequential BD algorithm to solve it.

2.1. Two-Stage Stochastic integer programming

In two-stage stochastic programming, the uncertainty is observed only once and decisions are made before and after observing the uncertainties. The common practice is to approximate the probability distribution of random variables by a discrete probability distribution with a finite support. This gives a finite set of scenarios, each representing a possible realization of the random events. Given the scenario set $\mathcal{S}$ and occurrence probability $\rho_s > 0$ for each $s \in \mathcal{S}$ such that $\sum_{s\in\mathcal{S}} \rho_s = 1$, a canonical representation of a two-stage stochastic program is:

$$z^* := \quad \min_y \{f^\top y + \sum_{s\in\mathcal{S}} \rho_s Q(y,s) :\ By \geq b,\ y \in \mathcal{Y}\} \tag{1}$$

where for each scenario $s \in \mathcal{S}$

$$Q(y,s) = \quad \min_x \{c_s^\top x : W_s x \geq h_s - T_s y,\ x \in \mathcal{X}\} \tag{2}$$

with $f \in \mathbb{R}^n$, $B \in \mathbb{R}^{k\times n}$, $b \in \mathbb{R}^k$, $c_s \in \mathbb{R}^m$, $W_s \in \mathbb{R}^{l\times m}$, $h_s \in \mathbb{R}^l$, $T_s \in \mathbb{R}^{l\times n}$. Here, $\mathcal{X} \subseteq \mathbb{R}^m$ and $\mathcal{Y} \subseteq \mathbb{R}^n$ are nonempty closed subsets which define the nature of the x and y decision variables in terms of sign and integerality restrictions. In this article we assume that $\mathcal{Y} = \mathbb{Z}^n_+$ and $\mathcal{X} = \mathbb{R}^m_+$. In this program, y represents the first-stage decisions and x represents the second-stage decisions. We thus seek a feasible solution that minimizes the first-stage cost $f^\top y$ plus the expected cost of the second-stage decisions.

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 constraint $W_s x \geq h_s - T_s \bar{y}$, the dual of $Q(\bar{y}, s)$ is

$$(SP(\bar{y},s))\ \ Q(\bar{y},s) = \ \max_\alpha \{(h_s - T_s\bar{y})^\top \alpha :\ {W_s}^\top \alpha \leq c_s,\ \alpha \in \mathbb{R}^l_+\} \tag{3}$$

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 (3). 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

$$z^* \ = \ \min_{y \in \mathbb{Z}^n_+} \{ f^\top y + \sum_{s \in \mathcal{S}} \rho_s Q(y,s) : \ By \geq b,\ (h_s - T_s y)^\top r_{q,s} \leq 0 \ \forall s \in \mathcal{S}, q \in F_s \} \tag{4}$$

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 (3). We can thus rewrite program (4) in an extensive form by interchanging $Q(y,s)$ with its dual, i.e., $SP(y,s)$

$$\min_{y \in Y} \big\{ f^\top y + \sum_{s \in S} \rho_s \max_{e \in E_s} \{ (h_s - T_s y)^\top \alpha_{e,s} \} : \ (h_s - T_s y)^\top r_{q,s} \leq 0 \quad s \in S, q \in F_s \big\} \tag{5}$$

where $Y = \{ y : By \geq b, y \in \mathbb{Z}^n_+ \}$. If we capture 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 (1), called Benders *master problem* (MP):

$$MP(E_1, ..., E_{|S|}; F_1, ..., F_{|S|}) \ := \ \min_{y \in Y} f^\top y + \sum_{s \in S} \rho_s \theta_s \tag{6}$$

$$(h_s - T_s y)^\top \alpha_{e,s} \leq \theta_s \qquad s \in S, e \in E_s \tag{7}$$

$$(h_s - T_s y)^\top r_{q,s} \leq 0 \qquad s \in S, q \in F_s \tag{8}$$

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 strategy to generate the *optimality* (7) and *feasibility* (8) cuts on the fly. In the classical BD method, the MP is solved from scratch at each iteration. However, nowadays, this 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). Moreover, the LP relaxation of the MP is often solved first to quickly tighten the root node which enables the Benders method to perform more efficiently (McDaniel and Devine 1977). Algorithm 1 presents the pseudo-code of this algorithm.

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, for a certain number of iterations, it sequentially solves the MP and SPs to generate optimality and feasibility cuts. In the second phase, a branch-and-bound tree is created (line 3), where the root node is the MP with the generated cuts at the first phase. At each iteration, the algorithm selects and solves a node from the pool L (lines 5 and 6). If the obtained solution is fractional and the node cannot be

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

1: Create the MP which is the LP relaxation of program (6)-(8) with $E_s = \emptyset$ and $F_s = \emptyset$, $\forall s \in \mathcal{S}$
2: Iteratively solve the MP and add cuts
3: Add the obtained MP into tree L, set $UB = \infty$, $LB = -\infty$, and ϵ_{opt} to the optimality tolerance
4: **while** $UB - LB > \epsilon_{opt}$ **and** $L \neq \emptyset$ **do**
5: Select a node from L
6: Solve this node to get an optimal solution $\bar{y}$ with objective value of ϑ^*
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: Add a feasibility cut and go to line 16
15: **else**
16: Extract the optimal extreme point e_s
17: Add new optimality cut(s) using the extreme points obtained in line 15
18: Solve this node again to get an optimal solution $\bar{y}$ with objective value of ϑ^*
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: Set LB as the minimum objective values of the nodes in L
24: Choose a fractional variable from $\bar{y}$ to branch
25: Create two nodes and add them to L.

pruned (line 7), a branching occurs to create and add two new nodes to the pool (lines 24 and 25). If 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 solution becomes fractional, or no more violated cut can be found (lines 10 to 20). This process (lines 5 to 25) repeats until the bounds collide or the node pool becomes empty. Note that in line 22 the upper bound is updated, i.e., the potential incumbent is accepted, if the $\bar{y}$ is an integer solution that satisfies all the Benders cuts.

Our developments in this article are based on Algorithm 1, which we refer to as the *Branch-and-Benders-cut* (B&BC) method. Various studies have been developed to accelerate the BD algorithm.

To avoid burdening this article, we refer the reader to Rahmaniani et al. (2017a) for a complete treatment of this topic.

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 in accelerating this method. To the best of our knowledge, all the existing parallel variants of this method follow the master-slave parallel programming model. We classify and review such methods in this part.

3.1. Master-slave parallelization strategies

The existing parallel BD method can be summarized as follows: *The MP is assigned to a processor, the "master", which also coordinates other processors, the "slaves", which solve the SPs. At each iteration, the solution obtained from solving the MP is broadcast to the slave processors. They then return the objective values and the cuts obtained from solving the SPs to the master and the same procedure repeats.* Such master-slave 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 the processors is less than the number of the 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 next iteration, information must be exchanged among the processors. In a parallel environment, this necessitates having 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. When communications are asynchronous, the processors are less interdependent. After broadcasting each MP solution, the master processor waits only for a subset of the cuts before re-optimizing the MP. For this reason, some SPs may remain unsolved which results in having a pool of unevaluated SPs. On the other hand, the slave processors continuously evaluate SPs. As a result, many cuts might be available when the master processor requests cuts. 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? We thus define a three-dimension taxonomy, depicted in Figure 1, that captures 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 in a *dynamic* (D) or *static* (S) fashion. In the former, the decisions regarding when and where to solve each problem are

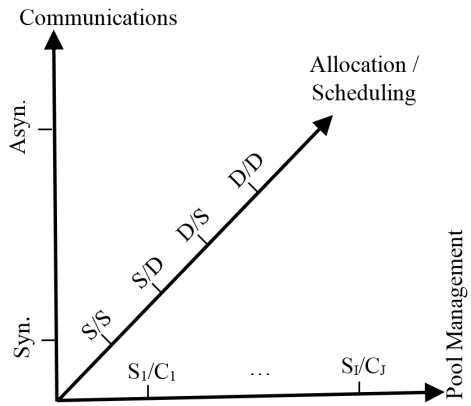

Figure 1 Taxonomy of the master-slave parallel Benders decomposition methods

taken during the course of the algorithm. In the latter, the decisions are made beforehand. Thus, S/D, for example, means static allocation and dynamic scheduling;

- ***Pool Management*** implies the strategies used to manage the pool of solutions (denoted by $S_1, ..., S_I$) and the pool of cuts (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* is minimized.

3.2. Previous work

Although parallelization of the BD method seems natural, number of the existing parallel variants of this method are very limited. This is due to the interdependency of MP and SPs, i.e., MP needs feedback from SPs before being able to execute its next iteration and vice versa. Moreover, due to dependency of this framework on having multiple SPs, most of the parallel BD algorithms are developed for stochastic problems. However, since solving SPs is significantly time consuming, it has proven effective in various cases (Ariyawansa and Hudson 1991, Wolf and Koberstein 2013, Tarvin et al. 2016). The literature discusses some of the strategies and algorithmic challenges arising from this parallelization approach.

Dantzig et al. (1991) considered a dynamic work allocation strategy in which the next idle processor gets the next SP based on a first-in-first-out strategy until all the SPs are solved and the MP can be recomputed with the new cuts. The efficiency 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 also it saves executing extra work. For example, if a 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 can easily be handled by means of an interior point method. Vladimirou (1998) implemented a partial-cut aggregation strategy to reduce the communication overheads.

In some studies the decomposition has been modified to better suit the parallelization scheme. Dempster and Thompson (1998) proposed a parallel nested algorithm for multi-stage 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 reviewed studies implement a synchronized parallelism. Moritsch et al. (2001) proposed an 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 the 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 an speedup ratio up to 45% with 4 processors. Furthermore, low-level parallelism improves the efficiency if 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.

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 no one has

Table 1 Summary of the Benders-based parallel algorithms

Reference	Problem class		Implementation		Communication		Work allocation		Scheduling		Pool management	
	LP	MIP	Classic	B&C	Syn.	Asyn.	Dynamic	static	Synamic	Static	Solution	Cut
Chermakani (2015)	+		+		+			+		+		
Dantzig et al. (1991)	+		+		+		+			+		
Dempster and Thompson (1998)	+		+		+			+		+		
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)		+	+		+			+		+		

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 if the MP can be solved quickly and the SPs constitute the major computational bottleneck of the algorithm. Thus, the reviewed strategies are not directly applicable to develop an efficient parallelization of the BD method when an iteration of the MP can take hours.

4. Synchronized Parallel Benders Decomposition Algorithm

In the parallel synchronous version of Algorithm 1, each time the MP finds a new solution to generate cuts, it is broadcast to all other processors. The master processor waits for the slave processors to solve their assigned SPs. When the current solution is evaluated, the cuts are added to the MP. Then, the master processor continues with the next iteration until it finds another appropriate solution to generate cuts for which the same process is repeated. Thus, the only difference of this parallel algorithm with the sequential one lies in solving the SPs in parallel, i.e., the for-loop in line 11 of Algorithm 1.

To assign the SPs to the processors, given $|\mathcal{P}|$ slave processors and $|\mathcal{S}|$ scenario SPs, we group SPs and assign each group to a processor such that each processors receives a roughly equal number of SPs. This follows from the fact that in two-stage stochastic programming, the SPs have usually the same level of resolution difficulty. Thus by equally distributing them among the slave processors, load-balancing strategies are not required. In addition, we try to assign similar SPs to a processor, because this helps to more effectively take advantage of the re-optimization tools of our LP solver. The similarity is measured as the Euclidean distance of random parameters.

4.1. Cut aggregation

When the number of SPs is larger than number of the first-stage variables, it is not numerically the best strategy to add to the MP a cut per SP as in line 17 of Algorithm 1 (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 the number of SPs in each

cluster may not be equal. Then, for each cluster we define a single recourse variable and aggregate the generated cuts in that cluster into a single cut.

4.2. Upper bound from fractional points

The upper bounds obtained from fractional master solutions (e.g., line 2 of Algorithm 1) are not valid for the original problem. Due to 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 the following assumptions: (i) $f_i \geq 0$, (ii) for a given feasible $\bar{y}$, $\hat{y}$ is also feasible if $\hat{y}_i \geq \bar{y}_i$ for all $i \in \{1, ..., n\}$, and (iii) $Q(\bar{y}, s) \geq Q(\hat{y}, s), s \in \mathcal{S}$.

PROPOSITION 1. *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 $s \in \mathcal{S}$, 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.*

Proof. See Appendix A.

4.3. Cut generation

Conventionally, generating cuts at fractional nodes of the B&BC algorithms, except for the root node, is ignored. This is, to a very large extent, due to the excessive time required to generate them (Botton et al. 2013). However, in parallel B&BC methods we have more computational power at disposal. Thus, we study some strategies to generate cuts at fractional nodes of the branch-and-bound tree that we refer to as *cut generation* strategies.

The first strategy is to generate Benders cuts at the fractional nodes. Accordingly, we generate cuts for the first Γ fractional nodes, where $0 < \Gamma \leq \infty$ and 0 being the root node for which we always generate cuts. There are two possibilities at each fractional node: (i) generate at most one round of cuts per node or (ii) call upon the cut generation module as long as the local bound at the current node improves by more than $\tau\%$.

The second class of strategies rounds the fractional solutions to derive integer ones which can be used to obtain valid upper bounds and cuts. Specifically: (i) round the fractional values to the closest integer point; (ii) round upward any value greater than λ and downward otherwise, where λ is a positive digit in $[0, 0.5)$; (iii) solve a restricted MILP program to round the fractional values. 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 $1 - \lambda$ and to 0 if it is less than $\hat{\lambda}$.

Many of the dispatched solutions to the slave processors are infeasible. Thus, the last strategy revolves around a repair heuristic to restore the feasibility of such solutions. In many applications feasibility of the solutions satisfies the following monotonicity property: given a feasible solution $\bar{y}$, $\hat{y}$ is as well feasible if $\hat{y}_i \geq \bar{y}_i$ for $i \in \{1, ..., n\}$. The repair heuristic thus solves a restricted version of

the problem with objective function of $\triangle(\bar{y}) := \sum_{i\in\{1,\dots,n\}}(1-\bar{y}_i)y_i$, where variables are bounded from below by their current value, i.e., $y \geq \bar{y}$, to maintain the structure of current solution and solve this problem faster.

4.4. Cut management

The B&BC method, particularly with the cut generation strategies, may generate many cuts which are not all worth adding to or keeping in the master formulation. Thus, a cut corresponding to a fractional solution is added to the master formulation if its relative violation (the absolute violation of the cut divided by the 2-norm of the cut coefficients) is at least 10^{-3}, otherwise it is discarded.

Cut removal can computationally be expensive as it disturbs our LP solver, i.e., CPLEX. Thus, we execute the following routine only following the termination of the first phase to identify the dominated cuts. At iteration t, cut $(h_s - T_s y)^\top \alpha_s^t$ is generated to primarily bound the recourse problem s at solution $\bar{y}^t$. Following a heuristic notion, if there is another generated cut $(h_s - T_s y)^\top \alpha_s^j$, such that $j \neq t$, that bounds the recourse variable θ_s tighter at $\bar{y}^t$, i.e., $h_s^\top(\alpha_s^j - \alpha_s^t) + (\alpha_s^t - \alpha_s^j)^\top T_s \bar{y}^t \geq 0$, it flags the possibility of removing cut $(h_s - T_s y)^\top \alpha_s^t$ without deteriorating the approximated value function. We apply the same procedure for the feasibility cuts. Note that executing this test for all $y \in Y$ would yield an exact method to identify dominated cuts (Pfeiffer et al. 2012). In the second phase, our solver shall remove any cut that might become a slack.

5. Asynchronous Parallel Benders Decomposition Algorithm

The synchronization requirement between master and slave processors increases the overheads due to the excessive idle times. This is particularly evident when solving any of the problems is noticeably more time consuming than others. A natural way to overcome this issue is to loosen the synchronization requirement among the master and slave processors. To achieve this, the master processor is required to wait only for $\gamma|\mathcal{S}|\%$ of the cuts from slave processors before executing its next iteration (i.e., resolving the MP in the first phase or exploring the tree in the second phase), where $0 \leq \gamma \leq 1$. However, 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 the sequential algorithm. These are the main issues in developing an effective asynchronous parallel Benders algorithm which we address them in this section.

5.1. 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 in line 22 which does not necessarily satisfy all the Benders cuts. For this reason, the asynchronism may compromise the convergence of the algorithm as it applies only a subset of the cuts at each iteration of the

while-loop. In fact, the applied cuts (which can be zero cuts based on the γ value) may not affect the current integer solution and thus, resolving the associated node in line 18 can cause the algorithm to break the loop. While the cuts which are not yet generated may override the current node solution by changing it to a fractional solution, rendering it infeasible, generating another integer solution or changing its cost. Therefore, these uncertainties regarding the potential incumbent solutions need to be taken into account in order to maintain the global convergence.

To cope with this, we use two techniques. First, we make sure that the global upper bound is updated only when a reliable value is obtained, e.g., all SPs associated to a potential incumbent solution have been evaluated. Second, to overcome the uncertainties associated with the integer nodes for *binary* master variables, we make use of combinatorial cuts with the following form: $\sum_{i\in\{1,..,n\}:\bar{y}_i=0} y_i + \sum_{i\in\{1,..,n\}:\bar{y}_i=1}(1-y_i) \geq 1$, to forbid regeneration of the current integer solution $\bar{y}$. This cut eliminates the current solution by (1) making the current node infeasible, (2) generate 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 upper bound value. In the third case, the node will be added to the pool of active nodes.

Finally, for *general integer* master variables, the node uncertainty must be handled through synchronization. Thus, if the applied cuts do not affect the current integer solution, we need to wait until a violating cut associated with that solution is generated or all cuts have been applied.

5.2. Search techniques

In this part we propose some strategies to specify the scheduling and pool management decisions.

5.2.1. Solution and cut pool management Considering the previously partially evaluated solutions and the new one at the current iteration, we need to decide which solution to choose and evaluate its associated (unevaluated) SPs. At each iteration, the master process broadcasts its solution to all slave processors. Each slave processor stores this solution in a pool and follows one of the following strategies to pick the appropriate one:

S1: chooses solutions based on the *first-in-first-out* (FIFO) rule;

S2: chooses solutions based on the *last-in-first-out* (LIFO) rule;

S3: chooses solutions in the pool randomly. We experiment with two selection rules: (1) each solution in the pool has an equal chance to be selected, (2) each solution is assigned a weight of $\frac{1}{1+t}$, where t is number of the iterations since that specific solution has been generated, so that more recent solutions have higher chance to be selected.

Moreover, 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. Finally, we make use of the same techniques outlined in subsection 4.4 to manage the cut pool.

5.2.2. Solving SPs We have implemented static work allocation because by equally distributing the scenario SPs, every process is almost equally loaded. Once the solution is chosen, we need to decide the order by which the associated SPs will be evaluated, because we may not evaluate all of them and it is important to give higher priority to those which tighten the MP formulation the most. The following strategies are considered:

SP1: randomly choose the SPs;

SP2: assign a weight to each SP and then randomly select one based on the roulette wheel rule. The weights are set equal to the normalized demands for each SP.

SP3: we observe that if a solution is infeasible, we may not need to solve all its SPs. This strategy first orders the SPs based on their demand sum and then assigns to each SP a *criticality* counter which increases by one each time that the SP is infeasible. Then, a SP with the highest criticality value is selected.

5.2.3. Solving the MP This dimension specifies the waiting portion of the master processor before it re-optimizes the MP. We have proposed the following strategies:

MP1: the master processor waits for at least $\gamma|S|\%$ new cuts at each iteration;

MP2: the master processor waits for $\gamma|S|\%$ of the cuts associated with the current solution;

MP3: this strategy is the same as the MP2 strategy, but with a mechanism to synchronize the processors according to the current state of the algorithm. In this regard, if the cuts added to the MP fail to affect the lower bound and/or regenerate the same solution, the MP waits until all the cuts associated with the current solution are delivered.

5.3. Cut aggregation

The asynchronous algorithm may only solve a subset of the SPs in each cluster. For this reason, cut aggregation as introduced in section 4.1 is not applicable in the context of our asynchronous parallel algorithm. 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 present iteration. Note that we could 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.

5.4. Partial information

In general, the BD method is very sensitive to the feedback on its current solution. The asynchronous variant thus entails a larger number of iterations and a considerable amount of redundant work as it executes the next iteration with *partial information* on its current solution. In fact, the asynchronism may increase the computational cost of each iteration since the MP grows large at a

faster pace. Moreover, the lower bound may progress slower since merely a subset of the cuts are applied to the MP at each iteration. The proof of convergence may as well be delayed due to the unavailability of the upper bound, especially when one wishes to solve the problem to a certain optimality gap. To overcome these drawbacks, we propose two classes of strategies acknowledging the fact that we solve a SP in order to: (i) calculate an upper bound and (ii) generate a cut.

The upper bound is used to assess the convergence and prune nodes. In our asynchronous method the delay in calculating the upper bound is usually due to long intervals between visiting two integer nodes, where we conventionally collect the feedbacks from the slave processors. Thus, by designing a module that can receive information from slave processors at every node of the search tree, we can overcome this issue and update the upper bound with minimal delay.

The generated cuts are used to cutoff the subregion of the MP defined by the set of complicating variables. To address the issue related to having cuts merely for a subset of the recourse variables at each iteration, we have designed the following two strategies: (i) creating artificial SPs and (ii) propagating the generated cuts.

5.4.1. Creating artificial SPs The MP does not wait for all SPs to be solved. Thus, having good cuts that can represent the unevaluated SPs is important to improve the efficiency of our asynchronous algorithm. To this end, we propose to create a set of artificial scenarios. The SP associated with each artificial scenario will then be solved to generate a valid cut to bound the recourse cost of the scenarios that remain unevaluated at the current iteration.

We assume that the recourse matrix and the costs are deterministic, i.e., $c_s = c$ and $W_s = W$, $\forall s \in \mathcal{S}$. We cluster the SPs into $|\mathcal{G}|$ groups according to the similarity measure. Note that the cardinality of clusters may not be equal. Then, to generate the artificial SP, $g \in \mathcal{G}$, 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 2. *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 $0 \geq (h_g - T_g y)^\top r^g$.*

Proof. See Appendix B.

An important issue in deriving the artificial cuts lies in setting the β weights. The following theorem suggests an optimal way to set these weights.

THEOREM 1. *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 $\dfrac{\rho_s}{\sum_{s\in\mathcal{S}_g} \rho_s}$.*

Proof. See Appendix C.

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. This follows the aggregation step in the proof of Proposition 2.*

From corollary 1, we observe that there is no need to apply the scenario creation strategy in the synchronized and sequential algorithms.

5.4.2. Cut propagation In this part, we propose to propagate the generated cuts in order to derive additional approximate cuts for the set of unevaluated SPs. We make the same assumption on the problem's structure as in section 5.4.1.

PROPOSITION 3. *Given an optimality cut $\theta_s \geq (h_s - T_s y)^\top \alpha_s$ associated with the SP $s \in \mathcal{S}$, we can generate a valid optimality cut $\theta_{s'} \geq (h_{s'} - T_{s'} y)^\top \alpha_s$ for SP $s' \in \mathcal{S} : s' \neq s$ without solving SP s'.*

Proof. Based on our assumption, the dual SP (3) becomes $\max_{\alpha \in \mathbb{R}^l_+} \{(h_s - T_s \bar{y})^\top \alpha : \ W^\top \alpha \leq c\}$ for every $s \in \mathcal{S}$. Thus, all SPs have the same dual polyhedron for which each of its extreme points and rays gives a valid cut. □

It is not computationally viable to propagate any dual solution due to time consuming calculations and handling requirements. We thus generate propagated cuts only for those recourse variables whose associated SP remains unevaluated at current iteration. Also, if a dual value yields a cut which is not violated by the current solution, it will not be used for propagation. Finally, for SP s' we generate a propagated cut using the solution from SP s if the latter has the greatest *dominance* value over the former in the set of evaluated SPs at the current iteration. Note that the dominance value of SP s over SP s' is calculated as the total number of the elements $j \in \{1, ..., l\}$ for which $h_s^j \geq h_{s'}^j$ and ${T_s^j}^\top y \geq {T_{s'}^j}^\top y$ for every $y \in Y$ (Crainic et al. 2016).

5.5. The overall framework

The overall framework of the proposed asynchronous algorithm with its various components is depicted in Figure 2.

Each processor starts with an initialization step where a series of tasks are executed. At the master processor, the MP is created, scenarios are grouped (based on similarity criteria) and assigned to each slave processor, and the recourse variables are clustered for the cut aggregation purpose. Each slave processor waits to receive the assigned scenarios and then creates the SPs.

In the first phase, each time that the LP relaxation of MP is solved, the solution is broadcast to all slave processors. Then, the master processor waits to receive the feedback from these processors

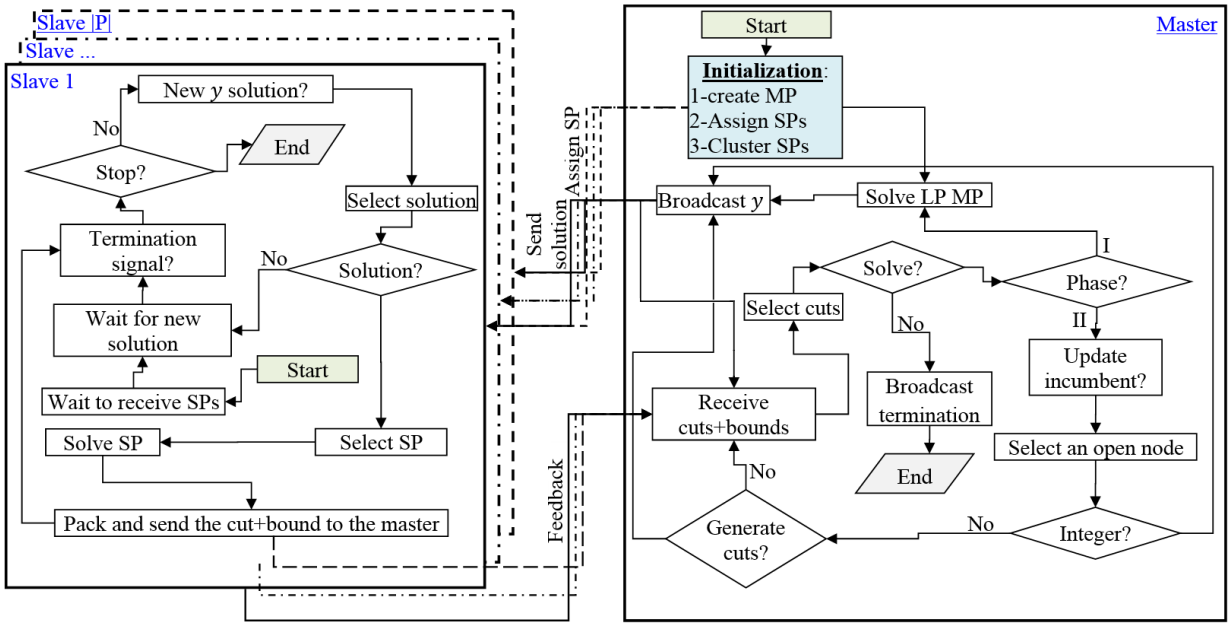

Figure 2 Flowchart of the asynchronous parallel method

based on the chosen scheme from the *solving MP* strategies. In the next step, cuts from the pool are selected and aggregated using the appropriate strategy for *cut management*. If the first phase is not yet optimized, we repeat the same steps. Otherwise, we proceed with the branching phase.

In the second phase, we check if the incumbent solution can be updated before choosing and evaluating an open node. If the selected node is integer, we broadcast the solution to all processors. However, if the node solution is fractional, we need to decide if we want to generate cuts associated with that solution based on the cut generation strategies of section 8.1.3. Before evaluating the selected node, we check the cut pool to see if new cuts exist or not. If the selected node is fractional, we do not wait for cuts to be generated. For the integer nodes, however, we wait for feedback from slave processors depending on the *solving MP* strategy that we are using. Finally, if one of the stopping criteria is satisfied, we broadcast the termination signal to stop the algorithm.

All slave processors execute the same process but independently. At each step, they check if a new master solution is broadcast. If so, they receive the solution and store it in a pool. However, if there is no solution in the queue, they do not wait and proceed by choosing a solution from the pool based on the appropriate *solution management* strategy. If the solution pool is empty for any slave processor, it waits until a new one or the termination signal is broadcast. At next step, they select one of their local SPs based on the *solving SPs* strategy and evaluate it. Finally, each processor sends back to the master processors the generated information (i.e., cut and bound). If no termination signal has been received, the process repeats.

6. Hybrid Parallel Benders Decomposition Algorithm

We observed that when solving the MP and SPs is quick, re-optimizing the MP with partial feedback from the slave processors can yield efficiency drawbacks. This is particularly the case during the first phase of our asynchronous algorithm. For this reason, we hybridize the two previous parallelization strategies by solving the first phase with the synchronized and the second phase with the asynchronous strategies. We refer to this strategy as *hybrid parallelism*.

7. Implementation Details

We solve all LP and MILP problems using IBM ILOG CPLEX 12.7. All programs are coded in C++ environment. The code is compiled with g++ 4.8.1 performed on Intel Xeon E7-8837 CPUs running at 2.67GHz with 64GB memory under a Linux operating system. The B&C algorithm was also implemented using CPLEX's callable libraries. We solve the extensive formulation with CPLEX's default setting, and turn off presolve features for our Benders-based algorithms.

To further accelerate the presented B&BC algorithms, we also incorporate some classical acceleration strategies. We have thus implemented the following techniques in all of our algorithms: (i) warm start strategy, (ii) valid inequalities for the MP, (iii) valid inequalities for each SP, and (iv) Pareto-optimal cuts. The complete details of these strategies can be found in Rahmaniani et al. (2017b). 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.

7.1. Test instances

To test our method, we address the well-known *Multi-Commodity Capacitated Fixed-charge Network Design Problem* with *Stochastic Demand* (MCFNDSD). This problem naturally appears in many applications (Klibi et al. 2010) and it is notoriously hard to solve (Costa 2005, Crainic et al. 2011). The complete detail of this problem is given in Appendix D. To conduct the numerical 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. (2016, 2011), Boland et al. (2016). These instances have up to 64 scenarios. To generate a larger number of 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, or 9 which accounts for 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. 2016). In the following part of the computational experiments, where we study the performance of our method versus alternative methods, we have considered a larger number of instances, i.e., r04-r11 accounting for 200 instances. The description of these instances is given in Appendix E.

7.2. 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 length of this article. For this reason, we provide some insights for those strategies which we do not intent to presented numerical results for.

With respect to *solution management* strategies, we observed that LIFO outperforms both FIFO and random strategies. The main reason is that Benders method, as "a dual algorithm", is very sensitive to the feedback on its current solution. In the both 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 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 in 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 henceforth only consider the LIFO as solution selection strategy.

With respect to *solving SPs*, we realized that, on average, the random selection of the SPs performs better than other strategies. This is because the random selection of the SPs ensures the diversity of the cuts applied to the MP. In ordering based strategies, after a certain number of iterations, a specific set of the recourse variables is bounded at each iteration. Hence, the lower bound progresses slowly and the algorithm performs poorly. Finally, the random selection based on the criticality weights tends to perform better than pure random selection. This is because higher priority is given to the indicator SPs and the diversity of the cuts applied to the MP is maintained. Therefore, we consider this strategy to select SPs for evaluation.

Finally, we consider the proposed strategies in the *solving MP* dimension. We have decided to use both MP2 and MP3. This is because of two reasons. First, we observed that applying cuts associated with the current solution is necessary. Otherwise, the algorithm performs poorly due to the same reasons pertinent to the FIFO strategy. Second, with adoptive synchronization points the late convergence proof in the first phase of the algorithm will be alleviated significantly. On the contrary, synchronization causes overheads during the second phase of the algorithm because master processor can proceed with evaluating the open nodes of the branch-and-bound tree while the cuts are being generated. Therefore, we use MP3 in the first and MP2 in the second phase of our algorithm.

7.3. 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. In the cut generation strategy, τ, λ, and $\hat{\lambda}$ values are set to 0.5%, 10^{-1}, and 10^{-2} . The set of scenarios in the feasibility repair problem are chosen based on their demand sum. We have sorted the scenarios based on their total demand and chosen the first 5 scenarios.

8. Computational Results

In this section, we present the numerical assessments of the proposed parallelization strategies. 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. Finally, we conduct a comparison between our exact algorithms and CPLEX to benchmark their performance.

8.1. Synchronized parallel algorithm

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

8.1.1. Cut management We first study the impact of the cut management strategy. We compare the method proposed in section 4.4 to that commonly used in the literature, where cuts with high slack value are removed, see, e.g., Pacqueau et al. (2012). In Table 2, the average running time in seconds, optimality gap in percentages, and number of removed cuts, shown with #Cut, are reported for each cut management strategy.

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. We observe from Table 2 that cleaning up the useless cuts yields positive impact on the performance.

The slack based notion 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 running is less significant because the algorithm reaches the time limit. We here 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 the cuts removed and the performance of the B&BC method. This is because of the heuristic nature of both strategies.

8.1.2. Cut aggregation We ran the algorithm with 11 different cluster sizes in order to study the impact of various cut aggregation levels on the performance. The comparative results in terms of total running time are depicted in Figure 3. Note that the value on each column gives the average optimality gap in percentages.

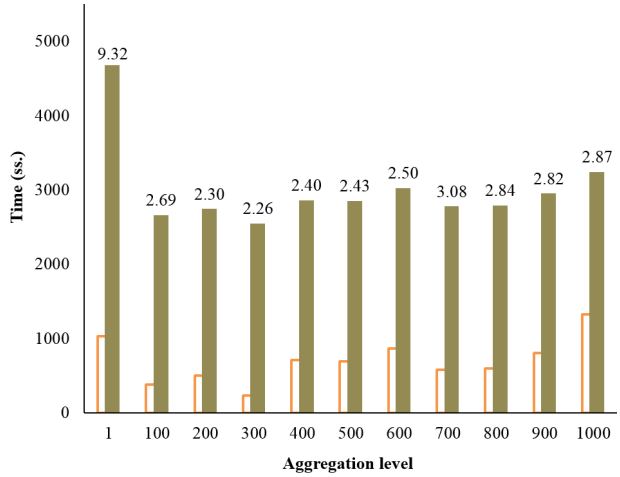

Figure 3 Comparison of various cut aggregation levels for the synchronized parallel algorithm

We observe from Figure 3 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 instances within the 2 hours 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 empty half bars in Figure 3.

Comparing the results to those of the sequential algorithm, see Figure 8 in Appendix F, we observe that a larger cluster size 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.

8.1.3. Cut generation strategy We next study the cut generation strategies of section 4.3. For computational purposes, we have set Γ to 10, 100 and ∞ and compare the results to the case for which we generate cuts merely at the root node, i.e., $\Gamma = 0$. For each Γ value, we have reported numerical results for two cut generation strategies: "Single" indicates generating cuts once per node and "Multi" indicates generating cuts for each node as long as the lower bound improves by more than 0.5%. Also, we have studied two cases for each strategy: (i) the fractional solutions are directly used and (ii) the feasibility of the infeasible ones is restored. The numerical results are summarized in Table 3.

Table 3 Numerical results for various cut generation strategies

Γ	Strategy	NoFeasibilityRestoration			FeasibilityRestoration		
		Time(ss)	Gap(%)	Sol.(%)	Time(ss)	Gap(%)	Sol.(%)
∞	Single	2737.66	20.19	57.14	2694.15	10.95	66.67
∞	Multi	3045.31	29.38	57.14	3164.12	24.80	61.90
100	Single	2538.98	16.27	52.38	2611.14	10.58	66.67
100	Multi	2999.89	25.00	57.14	3050.19	24.84	66.67
10	Single	**2398.36**	**2.16**	**71.43**	2405.82	2.41	71.43
10	Multi	2612.17	2.48	71.43	2583.14	2.40	71.43
0	-	2549.10	2.26	66.67	-	-	-

Here "Sol.(%)" gives percentage of the solved instances. From Table 3 we observe that generating cuts for many fractional nodes considerably increases the optimality gap. This happens because of two main reasons. First, generating cuts for many nodes is computationally expensive and second, it causes considerable handling costs. For the same reasons, the "Multi" strategy underperforms the one with the "Single" module. On the contrary, generating cuts merely for the first 10 nodes performs better or equal to generating cuts only at the root node, such that the average running time and optimality gap reduce by 150.74 seconds and 0.1%, while the percentage of solved instances increases by 4.76%.

We next examine the strategies proposed to round the fractional solutions. The first strategy, denoted *Round*, rounds each fractional value to its closest integer value. The second strategy, denoted *Upward*, rounds each fractional value greater and equal than 0.1 to 1 and 0 otherwise. The third strategy employs a restricted MIP to find a close integer feasible solution to the current fractional solution. The numerical results are summarized in Table 4.

Note that, following the observations in Table 3, we have applied these strategies only at the first 10 fractional nodes. Comparing Tables 3 and 4, we observe that the rounding strategies do not perform better than directly using the fractional solutions. The main reason is that the cuts

Table 4 Numerical results for the rounding based cut generation strategies

Strategy	NoFeasibilityRestoration			FeasibilityRestoration		
	Time(ss)	Gap(%)	Sol.(%)	Time(ss)	Gap(%)	Sol.(%)
MIP	-	-	-	**2486.78**	**2.31**	**71.43**
Round	2629.98	2.20	61.90	2591.03	2.25	61.90
Upward	2579.79	2.35	66.67	2606.85	2.44	71.43
$\Gamma = 0$	2549.10	2.26	66.67	-	-	-

associated to the rounded solutions do not, in general, affect the local bounds. In addition, CPLEX's default heuristics and local branching are active in our implementations. This makes the marginal impact of the rounding strategies negligible.

8.2. Asynchronous parallel algorithm

In this part, 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.

8.2.1. The synchronization level Figure 4 depicts the impact of the γ value on the running time. The values on this figure represent the average optimality gaps in percentage.

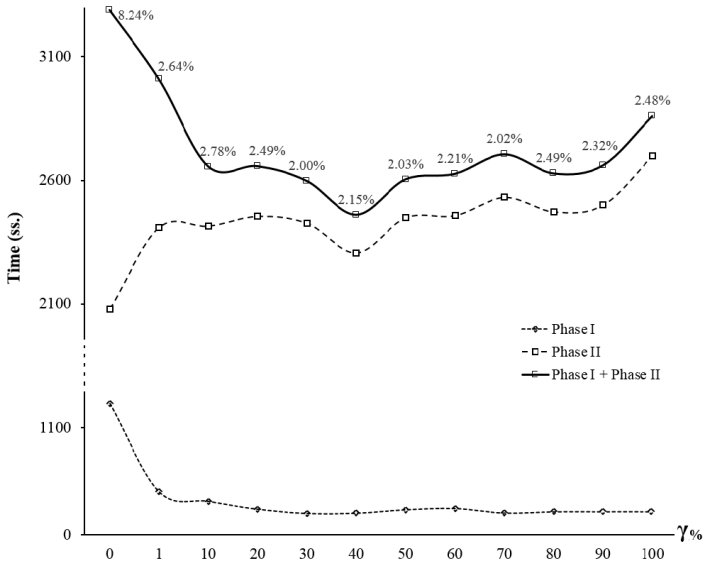

Figure 4 Effect of waiting portion of the master processor (gamma value) on the performance

We observe that the best performance lies neither at 100% nor at 1%. It is always numerically better to wait for some percentage in between. Moreover, comparing Figures 4 and 8, we observe that full asynchronism (i.e., $\gamma = 0$) underperforms even the sequential algorithm. The best result 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 the average optimality

gap of 2.15% in 2465.26 seconds. This is comparable to the synchronous algorithm without the cut generation strategies, see Figure 3.

We observe that waiting for a larger portion of the SPs is more desirable when solving the LP relaxation of the MP, while the contrary is true in the second phase of the algorithm. In the former case, we gain nothing from quickly resolving the MP since no other useful task can be executed. In the latter case, 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.

8.2.2. Scenario creation In Table 5 we investigate the impact of the artificial SPs on the convergence of our asynchronous algorithm.

Table 5 Impact of the artificial subproblems on 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 5 that the creation of artificial scenarios increases the percentage of the solved instances by 4.76% while the average time is 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 show. We observe, however, that the average optimality gap has been reduced. Thus, the artificial SPs are a valid strategy to accelerate the asynchronous algorithm.

8.2.3. Cut aggregation Figure 5 reports the impact of the proposed cut aggregation scheme for different cluster sizes. The value on each bar shows the average optimality gap in percentages.

We observe that clustering improves the convergence of the algorithm. The best aggregation level is reached at the size of 500, which is larger than 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, which outperforms the synchronized algorithm. Moreover, the single cut method does not perform much differently from other aggregation levels. This is because of introducing a recourse variable for each SP in the aggregated cuts.

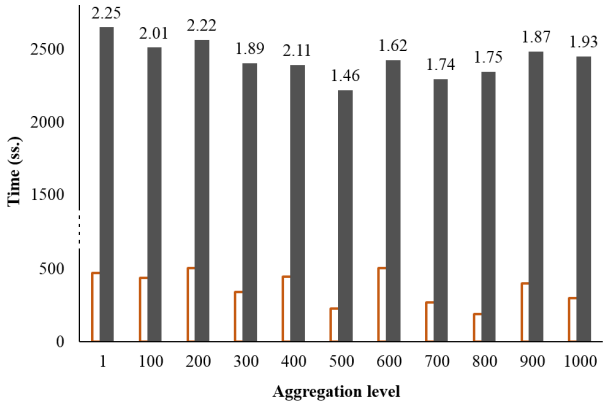

Figure 5 **Comparison of different cut aggregation levels in context of the asynchronous method**

8.2.4. Cut propagation To assess the usefulness of the proposed cut propagation strategy, we compare our algorithm with and without the cut propagation module. Table 6 summarizes the results in terms of average time requirement and number of iterations. Note that the cut propagation is performed merely at the first phase of the algorithm. We have thus reported the time required and number of iterations to find the optimal LP solution.

Table 6 **Impact of the cut propagation on the LP phase of the asynchronous algorithm**

	WithoutPropagation		WithPropagation	
	Time(ss.)	#Iter.	Time(ss.)	#Iter.
r04	29.62	17.00	111.24	13.67
r05	50.88	21.33	213.86	14.00
r06	307.41	32.67	1377.90	18.67
r07	19.98	15.67	286.50	14.00
r08	59.61	21.00	415.22	15.33
r09	546.43	30.67	1085.35	16.00
r10	1158.31	41.67	1591.48	23.00
Ave.	310.32	25.71	725.94	16.38

We observe from Table 6 that the cut propagation reduces number of the major iterations. However, it increases the time requirement to optimize the LP relaxation of the problem. This is partially due to the additional time we need to spend on handling the propagated cuts and checking their violation. In addition, we can solve the LP relaxation of the considered instances within the considered time limit and the cut propagation does not further tighten this bound. We thus believe that the proposed cut propagation strategy can be very useful for problems with a fairly small number of hard-to-solve SPs rather than many of easy-to-solve SPs. Since this is not the case in our problem, we do not include this strategy in our method.

8.3. Speedup of the parallel algorithms

In Figure 6, we compare our parallel algorithms on 2, 3, 5, 10, 15 and 20 processors. In this experiment, we have considered only those instances that the sequential algorithm could solve

within a 10-hour limit in order to have a clear sense of the speedups. Note that the value on each bar indicates the speedup ratio calculated as Sequential time/Parallel time.

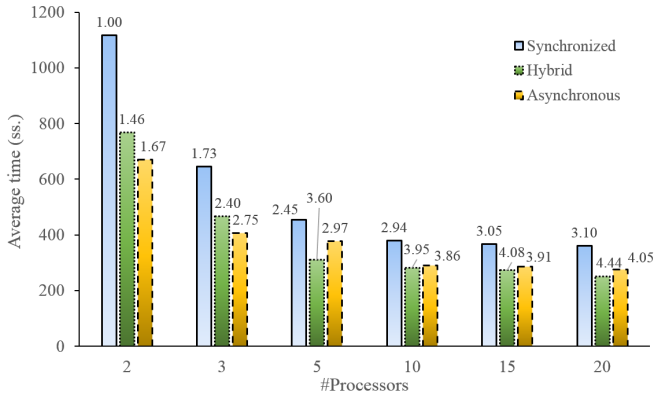

Figure 6 Speedup rates of our parallel Benders decomposition algorithms

The first interesting observation is related to the use of 2 processors, i.e., 1 slave and 1 master processor. In this case, the synchronized algorithm performs worse than the sequential algorithm because of the cut generation which slows down the algorithm. Our asynchronous algorithm, on the contrary, performs much better. This is clearly due to the better use of the processors in our asynchronous method 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 first phase of the algorithm. Thus, the speedup rate in our hybrid algorithm must be due to its asynchronous part. 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 the processors. This is because increasing the number of slave processors does not alleviate the bottleneck at the master processor, although it significantly accelerates the cut generation cycle. We observe that the synchronous algorithm reaches super linear speedup during the LP phase of the algorithm, while it fails to reach even linear speedup overall. This is due to the second phase of the algorithm where the slave processors are not efficiently used and the heavy work is carried out by a single processor, i.e., master. The same situation also applies to the asynchronous and hybrid algorithms.

Our third observation concerns the hybrid method. It outperforms the synchronized method because of using non-blocking communications in the branching phase, which is the most time consuming part of the algorithm; see Figure 4. 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.

8.4. Comparison with CPLEX

In this part, we compare our asynchronous and hybrid algorithms versus the latest version of CPLEX for all instances, see Table 8 in Appendix E. In doing so, we ran our algorithms until reaching the same optimality gap which is obtained by CPLEX after 10 hours. Note that all algorithms are run on 15 processors. The average speedup rates are reported in Figure 7. These values are rounded to the closest integer point and they are obtained from dividing CPLEX's time requirement to that of our methods.

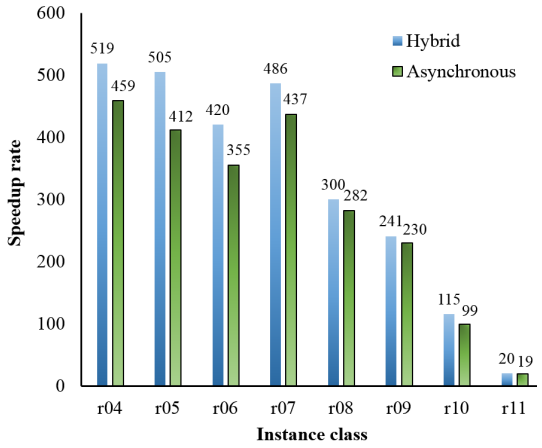

Figure 7 The speedup rate of our asynchronous and hybrid parallel algorithm compared to CPLEX

The asynchronous and hybrid parallel algorithms are, on average, 286.48 and 325.69 times faster than CPLEX in finding a solution of equal or better quality. We observe that the speedup for larger instances is smaller because solving their LP relaxation takes up to one hour. This makes the speedup rates in scale of 10. Moreover, the hybrid parallelization reaches a better speedup rates since it solves the LP relaxation faster.

To complete the comparative study in this section, we compare the optimality gaps obtained by our parallel algorithms to that of CPLEX for a run time limit of 2 hours. Note that if CPLEX fails to find a feasible solution, we have set the optimality gap at 100%. The average optimality gap for each instance class is summarized in Table 7.

We observe from Table 7 that CPLEX fails to handle even small instances of the considered stochastic problem. Furthermore, the hybrid algorithm reaches better optimality gaps than the asynchronous method for larger instances.

9. 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.

Table 7 The average optimality gap obtained by each method after 2 hours

	CPLEX	Asynchronous	Hybrid
r04	14.49	0.33	0.34
r05	31.84	0.48	0.46
r06	56.11	2.00	2.75
r07	33.73	0.34	0.50
r08	43.90	2.18	2.48
r09	58.07	3.25	2.72
r10	56.15	3.58	2.28
r11	86.86	7.65	6.82
Ave.	**47.64**	**2.48**	**2.29**

We implemented the algorithm in a B&C framework and presented synchronous, asynchronous and hybrid parallelization frameworks along with various acceleration strategies.

Reporting numerical results on hard benchmark instances from stochastic network design problems with 1000 scenarios, we observed (super-)linear speedups when the master problem does not computationally dominate the algorithm. This is particularly true when solving the LP relaxation of the master problem. In similar cases, the synchronized method performed generally better than the asynchronous algorithm. On the larger instances, our parallel algorithms did not reach a linear speedup. Moreover, we observed that our parallelizations did not scale with the number of the 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. Also, the proposed asynchronous and hybrid algorithms displayed a better performance compared to the synchronous method. Compared to CPLEX, they were more than 286 times faster to obtain the same optimality gap. In addition, for the same time limit, they could also obtain optimality gaps that were more than 19 times lower than CPLEX.

This research opens the way for a number of interesting issues to be considered in future works. First and foremost, the master problem needs to be solved in parallel in order to reach a scalable algorithm. Second, it is worthwhile to study the proposed cut propagation strategy for problems in which 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. (2016). Last but not least, 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.

Acknowledgment

Partial funding for this project has been provided by the Natural Sciences and Engineering Council of Canada (NSERC), through its Discovery Grant program and by the Fonds de recherche du Québec through its Team Grant program. We also gratefully acknowledge the support of Fonds de recherche du Québec through their strategic infrastructure grants.

References

Ahmed, Shabbir. 2010. Two-stage stochastic integer programming: A brief introduction. James J. Cochran, Louis A. Cox, Pinar Keskinocak, Jeffrey P. Kharoufeh, J. Cole Smith, eds., *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, 1–10.

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

Ariyawansa, K. A., D. D. Hudson. 1991. 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** 109–128.

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

Benders, Jacques F. 1962. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik* **4** 238–252.

Birge, John R, Francois Louveaux. 1997. *Introduction to stochastic programming*. Springer, New York.

Boland, Natashia, Matteo Fischetti, Michele Monaci, Martin Savelsbergh. 2016. Proximity Benders: a decomposition heuristic for stochastic programs. *Journal of Heuristics* **22** 181–198.

Botton, Quentin, Bernard Fortz, Luis Gouveia, Michael Poss. 2013. Benders decomposition for the hop-constrained survivable network design problem. *INFORMS journal on computing* **25** 13–26.

Chermakani, Deepak Ponvel. 2015. 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* .

Chouman, Mervat, Teodor Gabriel Crainic, Bernard Gendron. 2017. Commodity representations and cut-set-based inequalities for multicommodity capacitated fixed-charge network design. *Transportation Science* **51** 650–667.

Costa, Alysson M. 2005. A survey on Benders decomposition applied to fixed-charge network design problems. *Computers & operations research* **32** 1429–1450.

Crainic, Teodor Gabriel, Xiaorui Fu, Michel Gendreau, Walter Rei, Stein W Wallace. 2011. Progressive hedging-based metaheuristics for stochastic network design. *Networks* **58** 114–124.

Crainic, Teodor Gabriel, Mike Hewitt, Walter Rei. 2016. Partial Benders decomposition strategies for two-stage stochastic integer programs. Publication CIRRELT-2016-37, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, Université de Montréal, Montréal, QC, Canada.

Crainic, Teodor Gabriel, Bertrand Le Cun, Catherine Roucairol. 2006. Parallel Branch-and-Bound algorithms. El-Ghazali Talbi, ed., *Parallel Combinatorial Optimization*, chap. 1. John Wiley & Sons, 1–28.

Crainic, Teodor Gabriel, Michel Toulouse. 1998. Parallel metaheuristics. Teodor Gabriel Crainic, Gilbert Laporte, eds., *Fleet Management and Logistics*. Springer, Boston, MA, 205–251.

Dantzig, George B, James K Ho, Gerd Infanger. 1991. Solving stochastic linear programs on a hypercube multicomputer. Tech. Rep. ADA240443, DTIC Document.

Dempster, M. A. H., R. T. Thompson. 1998. Parallelization and aggregation ofnested benders decomposition. *Annals of Operations Research* **81** 163–188.

Higle, Julia L., Suvrajeet Sen. 1991. Stochastic decomposition: An algorithm for two-stage linear programs with recourse. *Mathematics of Operations Research* **16** 650–669.

Klibi, Walid, Alain Martel, Adel Guitouni. 2010. The design of robust value-creating supply chain networks: A critical review. *European Journal of Operational Research* **203** 283 – 293.

Latorre, Jesús M, Santiago Cerisola, Andrés Ramos, Rafael Palacios. 2009. Analysis of stochastic problem decomposition algorithms in computational grids. *Annals of Operations Research* **166** 355–373.

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

Linderoth, Jeff, Stephen Wright. 2003. Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications* **24** 207–250.

McDaniel, Dale, Mike Devine. 1977. A modified Benders' partitioning algorithm for mixed integer programming. *Management Science* **24** 312–319.

Moritsch, Hans W., G. Ch. Pflug, M. Siomak. 2001. Asynchronous nested optimization algorithms and their parallel implementation. *Wuhan University Journal of Natural Sciences* **6** 560–567.

Naoum-Sawaya, Joe, Samir Elhedhli. 2013. An interior-point Benders based branch-and-cut algorithm for mixed integer programs. *Annals of Operations Research* **210** 33–55.

Nielsen, Soren S, Stavros A Zenios. 1997. Scalable parallel Benders decomposition for stochastic linear programming. *Parallel Computing* **23** 1069–1088.

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

Pacqueau, Remi, SoumisS Francois, Hoang Le Nguyen. 2012. 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.

Pfeiffer, Laurent, Romain Apparigliato, Sophie Auchapt. 2012. 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* .

Rahmaniani, Ragheb, Teodor Gabriel Crainic, Michel Gendreau, Walter Rei. 2017a. The benders decomposition algorithm: A literature review. *European Journal of Operational Research* **259** 801 – 817.

Rahmaniani, Ragheb, Teodor Gabriel Crainic, Michel Gendreau, Walter Rei. 2017b. 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.

Ralphs, T.K., L. Ladányi, M.J. Saltzman. 2003. Parallel branch, cut, and price for large-scale discrete optimization. *Mathematical Programming* **98** 253–280.

Rei, Walter, Jean-Franois Cordeau, Michel Gendreau, Patrick Soriano. 2009. Accelerating benders decomposition by local branching. *INFORMS Journal on Computing* **21** 333–345.

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

Ruszczyński, Andrzej. 1997. Decomposition methods in stochastic programming. *Mathematical Programming* **79** 333–353.

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

Tarvin, D. Antony, R. Kevin Wood, Alexandra M. Newman. 2016. Benders decomposition: Solving binary master problems by enumeration. *Operations Research Letters* **44** 80 – 85.

Trukhanov, Svyatoslav, Lewis Ntaimo, Andrew Schaefer. 2010. Adaptive multicut aggregation for two-stage stochastic linear programs with recourse. *European Journal of Operational Research* **206** 395–406.

Uryasev, Stanislav, Panos M Pardalos. 2013. *Stochastic optimization: algorithms and applications*, vol. 54. Springer Science & Business Media.

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

Vladimirou, Hercules. 1998. Computational assessment of distributed decomposition methods for stochastic linear programs. *European Journal of Operational Research* **108** 653–670.

Wolf, Christian, Achim Koberstein. 2013. Dynamic sequencing and cut consolidation for the parallel hybrid-cut nested l-shaped method. *European Journal of Operational Research* **230** 143 – 156.

Yang, Huasheng, Jatinder ND Gupta, Lina Yu, Li Zheng. 2016. An improved L-shaped method for solving process flexibility design problems. *Mathematical Problems in Engineering* **2016**.

Appendix A: Proof of Proposition 1

It is trivial to observe that $\lceil \bar{y} \rceil \in Y$ due to assumption (ii). Let assume that the recourse cost associated to 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 assumptions (ii) and (iii), we have $\nu^*_s = Q(\bar{y}, s) \geq Q(\lceil \bar{y} \rceil, s) = \nu'_s$. Thus, $f^\top \lceil \bar{y} \rceil + \sum_{s\in\mathcal{S}} \rho_s \nu^*_s \geq f^\top \lceil \bar{y} \rceil + \sum_{s\in\mathcal{S}} \rho_s \nu'_s \geq z^*$.

Appendix B: Proof of Proposition 2

If $|\mathcal{S}_g| = 1$, the results follows immediately from the disaggregated version of the BD method Van Slyke and Wets (1969). To proof 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 $y \in Y$.

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

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

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

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,

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

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

Appendix 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 $\theta_s^* = (h_s - T_s y)^\top \alpha_s^{i^*}$, where $i^* \in \arg\max_{i \in E_s^t} (h_s - T_s y)^\top \alpha_s^i$ for which E_s^t is the set of optimality cuts associated to 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 $\mathcal{S}_g \leq |\mathcal{S}|$, the maximum value of Δ is achieve at $\beta_s = \frac{\rho_s}{\sum_{s \in \mathcal{S}_g} \rho_s}$.

Appendix D: Stochastic network design problem

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 is characterized with a set of discrete scenarios $\mathcal{S}$. Therefore, each commodity $k \in \mathcal{K}$ has a stochastic demand which is denoted by $d_s^k \geq 0$ for each possible scenario $s \in \mathcal{S}$. We assume that the realization probability for each scenario $s \in \mathcal{S}$ is known and denoted by ρ_s such that $\sum_{s \in \mathcal{S}} \rho_s = 1$. The goal is to select a proper subset of the arcs to meet all the flow requirements at minimum cost. To use arc $a \in \mathcal{A}$ we need to pay a fixed cost of f_a units and to route a unit of commodity $k \in \mathcal{K}$ on this arc c_a^k units will be charged. In addition, there is a capacity limit u_a on each arc $a \in \mathcal{A}$. Thus, the objective function is to minimize sum of the fixed costs and the expected flow costs.

To model this stochastic problem we define the binary first-stage variables y_a indicating if arc $a \in \mathcal{A}$ is used 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:

$$MCFNDSD = \min_{y \in \{0,1\}^{|\mathcal{A}|}, x \in \Re_+^{|\mathcal{A}||\mathcal{K}||S|}} \sum_{a \in \mathcal{A}} f_a y_a + \sum_{s \in \mathcal{S}} \rho_s \sum_{k \in \mathcal{K}} \sum_{a \in A} c_a^k x_a^{k,s} \tag{9}$$

$$\text{s.t:} \sum_{a \in \mathcal{A}(i)^+} x_a^{k,s} - \sum_{a \in \mathcal{A}(i)^-} x_a^{k,s} = \begin{cases} d_s^k & \text{if } i = O(k) \\ -d_s^k & \text{if } i = D(k) \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \mathcal{N}, k \in \mathcal{K}, s \in \mathcal{S} \tag{10}$$

$$\sum_{k \in \mathcal{K}} x_a^{k,s} \leq u_a y_a \qquad \forall a \in \mathcal{A}, s \in \mathcal{S}, \tag{11}$$

where $\mathcal{A}(i)^+$ and $\mathcal{A}(i)^-$ indicate the set of outward and inward arcs incident to node i. The objective function minimizes the total fixed costs plus the expected routing costs. For each scenario, constraint set (10) imposes the flow conservation requirements for each commodity and node. Constraints (11) enforce the capacity limit on each arc in every scenario. To introduce the complete recourse property for the above formulation, we add a dummy arc between each O-D pair with large routing cost as an outsourcing strategy.

Appendix E: Test instances

Table 8 presents the detail of the used instances problems from **R** family which we used in this article.

Table 8 Attributes of the instance classes

Name	$\|N\|$	$\|A\|$	$\|K\|$	$\|\Omega\|$	Cost/Capacity Ratio	Correlation	#Instances
r04	10	60	10	1000	1, 3, 5, 7, 9	0, 0.2, 0.4, 0.6, 0.8	25
r05	10	60	25	1000	1, 3, 5, 7, 9	0, 0.2, 0.4, 0.6, 0.8	25
r06	10	60	50	1000	1, 3, 5, 7, 9	0, 0.2, 0.4, 0.6, 0.8	25
r07	10	82	10	1000	1, 3, 5, 7, 9	0, 0.2, 0.4, 0.6, 0.8	25
r08	10	83	25	1000	1, 3, 5, 7, 9	0, 0.2, 0.4, 0.6, 0.8	25
r09	10	83	50	1000	1, 3, 5, 7, 9	0, 0.2, 0.4, 0.6, 0.8	25
r10	20	120	40	1000	1, 3, 5, 7, 9	0, 0.2, 0.4, 0.6, 0.8	25
r11	20	120	100	1000	1, 3, 5, 7, 9	0, 0.2, 0.4, 0.6, 0.8	25

Appendix F: Numerical results of the sequential algorithm

To make fair comparisons, we have incorporated the techniques that we developed in sections 4.1, 4.2, and 4.4 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 7.

In this appendix, we present some numerical results to complement our numerical assessments in section 8. In Figure 8, 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.

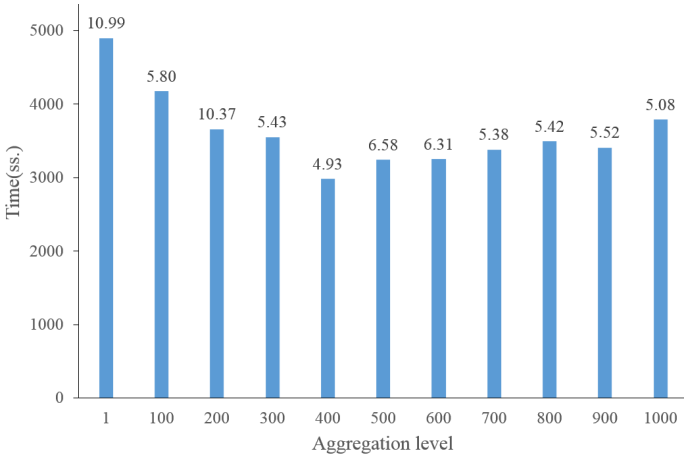

Figure 8 Comparison of various cut aggregation levels for the sequential B&BC method

We next present the numerical performance of the sequential algorithm for a run time limit of 10 hours. The results are presented in Table 9.

Table 9 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**