TOP (2021) 29:781–798
https://doi.org/10.1007/s11750-021-00593-2

ORIGINAL PAPER

A column-and-constraint generation algorithm for two-stage stochastic programming problems

Denise D. Tönissen[1] · **Joachim J. Arts**[2] · **Zuo-Jun Max Shen**[3]

Received: 16 March 2020 / Accepted: 21 January 2021 / Published online: 16 February 2021

Abstract

This paper presents a column-and-constraint generation algorithm for two-stage stochastic programming problems. A distinctive feature of the algorithm is that it does not assume fixed recourse and as a consequence the values and dimensions of the recourse matrix can be uncertain. The proposed algorithm contains multi-cut (partial) Benders decomposition and the deterministic equivalent model as special cases and can be used to trade-off computational speed and memory requirements. The algorithm outperforms multi-cut (partial) Benders decomposition in computational time and the deterministic equivalent model in memory requirements for a maintenance location routing problem. In addition, for instances with a large number of scenarios, the algorithm outperforms the deterministic equivalent model in both computational time and memory requirements. Furthermore, we present an adaptive relative tolerance for instances for which the solution time of the master problem is the bottleneck and the slave problems can be solved relatively efficiently. The adaptive relative tolerance is large in early iterations and converges to zero for the final iteration(s) of the algorithm. The combination of this relative adaptive tolerance with the proposed algorithm decreases the computational time of our instances even further.

Keywords Stochastic programming · Column-and-constraint generation · Benders decomposition · Facility location

Mathematics Subject Classification 90-08 · 90C15 · 90B80 · 49M27

✉ Denise D. Tönissen
d.d.tonissen@vu.nl

Extended author information available on the last page of the article

1 Introduction

Two-stage stochastic programming problems are often solved by Benders decomposition (Benders 1962) (BD), also known as the L-shaped decomposition method in the stochastic programming literature (van Slyke and Wets 1969). BD is used in many different types of two-stage stochastic programming problems such as supply chain planning (You and Grossmann 2013), production routing (Adulyasak et al. 2015), and hub location (Contreras et al. 2011). Many variations of BD exist, but the most relevant to this paper are multi-cut Benders decomposition (MCBD) and partial Benders decomposition (PBD) (Crainic et al. 2020). MCBD generates multiple cuts per iteration that approximate the second-stage values for each scenario separately. Birge and Louveaux (1988) show that such a framework can greatly increase the speed of convergence. PBD includes the deterministic equivalent model (DEM) of a subset of the scenarios directly into the master problem resulting in a stronger master problem that can lead to faster convergence. Rahmaniani et al. (2017) give an overview of variants and acceleration strategies of BD.

Recently, it was shown that column-and-constraint generation (C&CG) algorithms (also called row-and-column or scenario generation) work very well for two-stage robust optimization. Zeng and Zhao (2013) show that a C&CG procedure performs an order of magnitude faster than BD for a two-stage robust location transportation problem with demand levels in a polyhedral uncertainty set. Tönissen et al. (2019) show that a C&CG algorithm outperforms BD and MCBD with two orders of magnitude for a two-stage robust maintenance location routing problem for rolling stock with discrete scenarios. Furthermore, C&CG procedures have been applied to two-stage robust facility location (Chan et al. 2017; An et al. 2014), unit commitment (An and Zeng 2015; Zhao and Zeng 2012), and distribution network design (Lee et al. 2015).

In this paper, we demonstrate that a C&CG algorithm can be used for two-stage stochastic programming problems. This C&CG algorithm is an iterative algorithm, similar to BD, that decomposes the problem into a master problem and a second-stage (slave) problem for each scenario. In each iteration, the constraints and variables of one or more scenarios from the DEM are added to the master problem and the other scenarios are used to generate Benders optimality cuts. The C&CG algorithm reduces to multi-cut partial Benders decomposition (MCPBD) when the master problem contains a subset of the scenarios, and the iterative addition of scenarios is stopped. Furthermore, the algorithm reduces to the DEM when all scenarios are added to the master problem and to MCBD when no scenarios are added to the master problem. From this it can be concluded that the proposed algorithm reaches optimality in a finite number of iterations and should in principle perform equally well or better than all algorithms that it contains as special cases.

The benefit of adding the scenarios iteratively is that second-stage information can be used to find and add "important" scenarios to the master problem. Besides the iterative addition of scenarios, an important difference between the PBD algorithm of Crainic et al. (2020) and this paper is that Crainic et al. (2020) assume

fixed recourse. We do not assume fixed recourse and as a consequence the values and dimensions of our recourse matrix can vary from one scenario to another. The acceleration strategies (e.g., artificial scenarios) presented in Crainic et al. (2020) do not apply to problems where the recourse matrix is not fixed. Thus our algorithm applies to a broader class of two-stage stochastic programming problems. When the recourse matrix of a problem does have a fixed size, the algorithm in this paper can be applied in conjunction with the ideas of Crainic et al. (2020).

The models' dimensions in the proposed algorithm are much smaller than those of the full DEM. So, for instances with a large number of scenarios, the C&CG algorithm outperforms the plain use of any state-of-the-art MIP solver in memory requirements and computational time. We demonstrate this by performing computational experiments on a maintenance location routing problem. Furthermore, for large problems where the solution time of the master problem is the bottleneck and the second-stage (slave) problems can be solved relatively efficiently, we introduce an adaptive relative tolerance for solving the master problem. The effectiveness of the adoptive relative tolerance is also demonstrated by performing, computational experiments on a maintenance location routing problem.

In Geoffrion and Graves (1974), it was already shown that BD can be used with any feasible master problem solution. Consequently, changing the (relative) tolerance for the master problem in BD is not new. However, for our C&CG algorithm, using an adaptive relative tolerance is important because it decreases the computational time by allowing the algorithm to add scenarios faster while spending less time on closing the gap of the master problem. The proposed relative tolerance is defined as $\frac{\text{UB-LB}}{\text{LB}}$. It is large (e.g., 0.25) when the gap between the lower bound (master problem) and upper bound (incumbent solution) of the algorithm is large and converges to zero for the final iteration(s) of the algorithm. Therefore, the C&CG algorithm with adaptive relative tolerance solves problems to optimality.

The main contributions of the work are as follows:

- We develop a C&CG algorithm for two-stage stochastic programming problems.
- We show that an adaptive relative tolerance for the master problem can be used to decrease the computation time.
- We showcase the algorithm on a maintenance location routing problem for rolling stock. We show that this algorithm can be used to trade-off computational speed and memory requirements and that it can perform better than MC(P)BD or the DEM.

The paper starts with an explanation of the C&CG algorithm that can be used for two-stage stochastic problems in general. In Sect. 3, we explain the maintenance location routing problem for rolling stock and formulate it as two-stage optimization problem. In Sect. 4, we present the C&CG algorithm for the maintenance location routing problem. In Sect. 5, we present our experimental results of the C&CG algorithm for our maintenance location routing problem and we end the paper with a conclusion.

2 A column-and-constraint generation algorithm

Consider a two-stage stochastic programming problem with a discrete set of scenarios D, where p_d is the probability that scenario $d \in D$ occurs. This set can either be given or created using a sample average approximation for a distribution. Let $F_y \subseteq \mathbb{R}^{n-p} \times \mathbb{Z}^p$ and $F_{xd} \subseteq \mathbb{R}^{m_d}$ be the feasible regions containing mixed integer and linear variables, respectively. The first-stage decision is represented by $\boldsymbol{y} \in F_y$ and the second-stage decisions by $\boldsymbol{x}_d \in F_{xd}$. Define the uncertain matrices $\boldsymbol{W}_d$, $\boldsymbol{T}_d$, and the vector $\boldsymbol{h}_d$ that can be uncertain. The problem can be expressed as follows:

$$\min \left\{ \boldsymbol{c}^T \boldsymbol{y} + \textstyle\sum_{d \in D} p_d Q(\boldsymbol{y}, d) | \boldsymbol{A}\boldsymbol{y} \leq \boldsymbol{b}, \boldsymbol{y} \in F_y \right\},$$

where

$$Q(\boldsymbol{y}, d) = \min \left\{ \boldsymbol{h}_d^T \boldsymbol{x}_d | \boldsymbol{W}_d \boldsymbol{y} + \boldsymbol{T}_d \boldsymbol{x}_d \leq \boldsymbol{e}_d, \boldsymbol{x}_d \in F_{xd} \right\}.$$

The relatively complete recourse assumption is used, i.e., the second-stage problem is feasible for any $\boldsymbol{y}$ that is feasible for the first-stage problem.

This two-stage problem can be reformulated to the following DEM that contains all scenarios:

$$\begin{aligned}
\min \quad & \boldsymbol{c}^T \boldsymbol{y} + \textstyle\sum_{d \in D} p_d \boldsymbol{h}_d^T \boldsymbol{x}_d \\
\text{s.t.} \quad & \boldsymbol{A}\boldsymbol{y} \leq \boldsymbol{b}, \\
& \boldsymbol{W}_d \boldsymbol{y} + \boldsymbol{T}_d \boldsymbol{x}_d \leq \boldsymbol{e}_d \quad \forall d \in D, \\
& \boldsymbol{x}_d \in F_{xd} \quad \forall d \in D, \\
& \boldsymbol{y} \quad \in F_y.
\end{aligned}$$

This problem is computationally difficult to solve in terms of CPU and memory requirements when there are many scenarios. Our proposed algorithm solves this problem using the DEM for a few important scenarios and using MCBD for the remaining scenarios. We split the scenario set D in two sets. The set D_0 contains the important scenarios for which the constraints and objective functions are directly included in the master problem and the set D_1 contains the other scenarios for which we use Benders optimality cuts. Information from the second-stage problem is used to move scenarios from D_1 to D_0.

The iteration counter is set at $i = 0$ and we let $(\hat{\boldsymbol{y}}, \hat{\boldsymbol{x}})$ denote the incumbent solution. Furthermore, we define $\boldsymbol{y}^i$ as the optimal solution of the master problem and $\boldsymbol{x}^i$ as the optimal solution for the slave (i.e., second-stage) problems for iteration i. The lower bound (LB) is initially set at $-\infty$ and the upper bound (UB) at ∞. The algorithm consists of the following steps:

1. Solve the master problem. The master problem is the DEM for the objective function and constraints related to the y-variables (i.e., the first-stage submodel) and the x-variables (i.e., the second-stage submodel) for the scenario set D_0, supplemented by optimality cuts representing the input from the x-variables for the scenario set D_1.

$$\begin{aligned} \text{LB} = \min\ & \boldsymbol{c}^T\boldsymbol{y} + \sum_{d\in D_0} p_d \boldsymbol{h}_d^T \boldsymbol{x}_d + \sum_{d\in D_1} p_d \theta_d \\ \text{s.t.}\ & \boldsymbol{A}\boldsymbol{y} \le \boldsymbol{b}, \end{aligned} \tag{1}$$

$$\boldsymbol{W}_d\boldsymbol{y} + \boldsymbol{T}_d\boldsymbol{x}_d \le \boldsymbol{e}_d \quad \forall d \in D_0, \tag{2}$$

$$\left(\boldsymbol{r}_k^d\right)^T \boldsymbol{y} + \theta_d \ge v_k^d \quad k = 1, \ldots, i, \quad d \in D_1, \tag{3}$$

$$\theta_d \ge 0 \quad \forall d \in D_1, \tag{4}$$

$$\boldsymbol{x}_d \ge 0 \quad \forall d \in D_0, \tag{5}$$

$$\boldsymbol{y} \quad \in F_y. \tag{6}$$

The variable θ_d represents the second-stage cost for scenario $d \in D_1$. Constraints (3) are the optimality cuts that approximate $\theta_d \ge Q(\boldsymbol{y}^i, d)\, \forall d \in D_1$ and these cuts are only added to the master problem from iteration 1. The coefficients $\boldsymbol{r}_{i+1}^d$ and v_{i+1}^d are determined from the duals of the second-stage problem in such a way that $\sum_{d\in D_1} \boldsymbol{r}_{i+1}^d \boldsymbol{y}^i + v_{i+1}^d = \sum_{d\in D_1} p_d Q(\boldsymbol{y}^i, d)$. The details of the generation of $\boldsymbol{r}_{i+1}^d$ and v_{i+1}^d are explained in Step 5.

2. Calculate the objective value $\boldsymbol{c}^T\boldsymbol{y}^i + \sum_{d\in D} p_d Q(\boldsymbol{y}^i, d)$ and get $\boldsymbol{x}^i$ by solving the second-stage problem $Q(\boldsymbol{y}^i, d)$ for each scenario $d \in D$. If the objective value is smaller than the upper bound UB, we update the UB with the new objective value and set the incumbent solution at $(\hat{\boldsymbol{y}}, \hat{\boldsymbol{x}}) = (\boldsymbol{y}^i, \boldsymbol{x}^i)$.
3. Update sets D_0 and D_1. For this step, there are a few options that can differ per iteration i:

   - Add $\arg\max_{d\in D_1} Q(\boldsymbol{y}^i, d)$ to set D_0 and remove it from D_1.
   - Pick more than one scenario per iteration. Possible choices are the worst, best, middle scenario when the scenarios are sorted in ascending second-stage cost.
   - Do nothing.

   There are many other (problem dependent) possibilities. Note that new constraints are added to the master problem when a scenario is moved from set D_1 to D_0. Furthermore, all Benders optimality cuts for that scenario and the corresponding θ_d variable are removed from the master problem.
4. If $\frac{\text{UB-LB}}{\text{LB}} < \delta$, where $\delta > 0$ is a pre-specified tolerance, the algorithm stops and returns $(\hat{\boldsymbol{y}}, \hat{\boldsymbol{x}})$ as a solution and the corresponding UB that has a relative optimality gap of no more than δ. Otherwise, the algorithm proceeds to Step 5.
5. Let $\boldsymbol{\pi}_{id}$ be the dual variables for the constraints of the second-stage problem for scenarios $d \in D_1$. The cut coefficients are

$$\boldsymbol{r}_{i+1}^d = \left(\boldsymbol{\pi}_i^d\right)^T \boldsymbol{W}_d$$

and

$$v_{i+1}^{d} = \left(\boldsymbol{\pi}_{i}^{d}\right)^{T} \boldsymbol{e}_{d}.$$

Update $i = i + 1$ and go back to Step 1.

As mentioned in Sect. 1, MC(P)BD and the DEM are special cases of this algorithm. The C&CG algorithm reduces to the DEM when the algorithm starts with $D_0 = D$ and $D_1 = \emptyset$. When for every iteration the "do nothing" action is chosen for Step 3, we have MCBD when $D_0 = \emptyset$ and MCPBD when $D_0 \neq \emptyset$. As a consequence, the C&CG algorithm reaches optimality in a finite number of iterations and should in principle perform equally well or better than the algorithms included as special cases.

2.1 Adaptive relative tolerance for the master problem

The master problem is a MIP that can become very large and has to be solved multiple times. Most of the solution time of MIP problems often lies in proving that a solution is optimal rather than finding a feasible solution. As a consequence, it is frequently observed that algorithms that have higher optimality tolerances find (close to) optimal solutions of which the quality is yet unproven. It will suffice for our algorithm if the master problem is not solved to optimality in each iteration of the algorithm as long as it is solved to optimality in the final iteration. Therefore, it is computationally beneficial (faster) to allow larger optimality tolerances in the solution of the master problem initially. We propose that the master problem solution algorithm terminates when a tolerance is reached and we allow this tolerance to adapt each time that the master problem is solved. This tolerance should be large initially and adapt to be small eventually. Our proposed tolerance for the master problem depends on the LB and UB of each iteration of the algorithm. When the difference between the LB and UB is large, the proposed adaptive tolerance should also be large. In later iterations, when the gap between the LB and UB is almost closed, the proposed adaptive tolerance is small.

The adaptive relative tolerance has two input parameters: the initial_tolerance (which can be large, e.g., 0.25) and the end_tolerance (which approaches zero, e.g., 10^{-4}). The initial_tolerance represents the starting value of our adaptive tolerance, while the end_tolerance represents the terminating tolerance. The proposed adaptive relative tolerance is defined as follows:

$$\max\left\{\min\left\{\text{initial\_tolerance}, 0.5\frac{\text{UB-LB}}{\text{LB}}\right\}, \text{end\_tolerance}\right\}. \tag{7}$$

This adaptive tolerance decreases quickly as the lower bound and incumbent solution become close. Yet this adaptive tolerance is larger initially such that the master problem need not be solved to proven optimality initially.

3 Maintenance location routing for rolling stock

Our algorithm will be tested on the two-stage stochastic maintenance location routing problem (SMLRP) that was introduced in Tönissen et al. (2019). The C&CG algorithm does not consider the special structure of the SMLPR and the reason for choosing this problem is that it is large scale with binary first-stage decisions and continuous second-stage decisions. Consequently, this problem can benefit from the structure of (MC)BD, but it has convergence problems, while the DEM is quickly solved but runs out of memory for large instances. The C&CG algorithm can be used to trade-off the computational time and memory requirements to provide a suitable algorithm for instances of most sizes.

The SMLRP is described concisely below to make this paper self-contained, but we refer to Tönissen et al. (2019) for the details. Furthermore, in Sect. 4, we specify the C&CG algorithm for the SMLRP to make our results easier to reproduce. The reader that is only interested in the general application of the C&CG algorithm can skip ahead to Sect. 5.

The SMLRP is a facility location problem that has a set of candidate facilities and costs to open these facilities. Different from most facility location problems, our customers (train units) have to travel over a railway network to reach a facility. The transportation costs of these train units are more complicated than can be modeled by fixed allocation costs. The routing costs of train units to maintenance facilities is calculated by the solution of the second stage maintenance routing problem. The facility location problem and the maintenance routing problem cannot be solved separately because the difficulty with which a facility can be reached depends on the railway infrastructure and the line plan which can vary from one scenario to the next.

A line plan is a set of paths in the railway network that is operated by a rolling stock type with a given frequency. Figure 1 gives an example of a railway network and a line plan on that network. The nodes in the right-hand side are the potential begin and end station of a line, called end stations. Train units can move from one line to another when the lines have coinciding end stations. The example line plan on the right hand-side of Fig. 1 has two train types. The first, denoted by a, is an intercity train that skips the small stations, while train type b is a regional train that stops at every station. When a train unit requires maintenance, it has

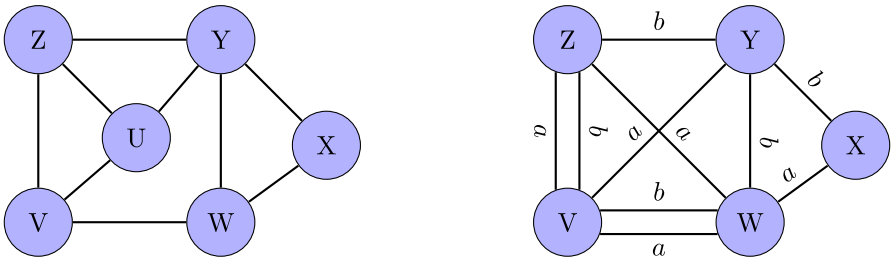

Fig. 1 The railway network on the left and a line plan on the right

to visit a maintenance facility. A train unit requires such maintenance approximately four times every year. It can visit the maintenance facility by deadheading (driving without passengers) incurring deadheading costs or by interchanging the destinations at the end stations of a line with other train units of the same rolling stock type until the maintenance location is reached. For example, in the right-hand side of Fig. 1, a train unit can interchange from line (Y, Z, b) to line (Z, V, b) while an interchange from (Y, Z, b) to line (Z, W, a) is not possible because the rolling stock types do not match.

The line plan is not always the same and changes over time to meet changing travel demands. For example, it is possible to start an additional line for rolling stock type b between node W and X. Such changes in the line plan are captured in discrete scenarios.

The question is now where to locate maintenance facilities for the rolling stock. This is a two-stage stochastic programming problem, with the facility locations as binary first-stage decisions and the allocations and routing of train units to the facilities as second-stage decisions. Formally, we are given a railway network $G = (N_P, E_P)$, consisting of rails E_P and stations N_P. Furthermore, we are given a set of discrete scenarios $d \in D$, in which each scenario defines a line plan: a set of lines $L^d \ \forall d \in D$, with, for each line, the rolling stock type rolling stock, the annual maintenance frequency, the coordination cost for each interchange, and the deadheading cost from each line to each facility. In addition, the line plan specifies the end stations $S^d \subseteq N_P \ \forall d \in D$, their location in the rail network, the possible interchanges, the maximum number of annual interchanges that can be performed at each end station ($g_s^d \ \forall s \in S^d, d \in D$), and the maximum number of interchanges that can be performed on the railway network ($G^d \ d \in D$).

The maintenance routing of train units can be formulated as a flow model that uses a flow graph. This flow graph $G_F^{dy} = (N_F^{dy}, A_F^{dy})$ is constructed as follows for each scenario $d \in D$ and first-stage decision $\boldsymbol{y}$:

- A node is created for every line and the set with these nodes is denoted as $N_L^d \subset N_F^{dy}$.
- A source $\mathcal{S}$ is created that is connected with a directed arc to each node in N_L^d.
- An directed arc with as cost the interchange coordination cost is created between the line nodes where an interchange is possible. The set of these interchange arcs is denoted by $A_I^d \subset A_F^{dy}$.
- A node is created for every candidate facility and the set with these nodes is denoted by N_C. Each node in N_C is connected with an arc to the sink $\mathcal{T}$.
- For each node $n \in N_L^d$, an arc to node $n \in N_C$ is created. The cost of this arc is the deadheading cost of the line to the facility that the nodes represents. The set of these incoming facility arcs is denoted by $A_C^d \subset A_F^{dy}$.

Figure 2 shows the application of this transformation. The left-hand side in Fig. 1 depicts a line plan for the railway graph, where a and b are the rolling stock types and 0-9 are the line numbers. The right-hand side depicts the routing flow graph $G_F^{dy} = (N_F^{dy}, A_F^{dy})$ for the line plan on the left when in the given $\boldsymbol{y}$, facilities $\{X, W, Z\}$ are opened and the others are closed.

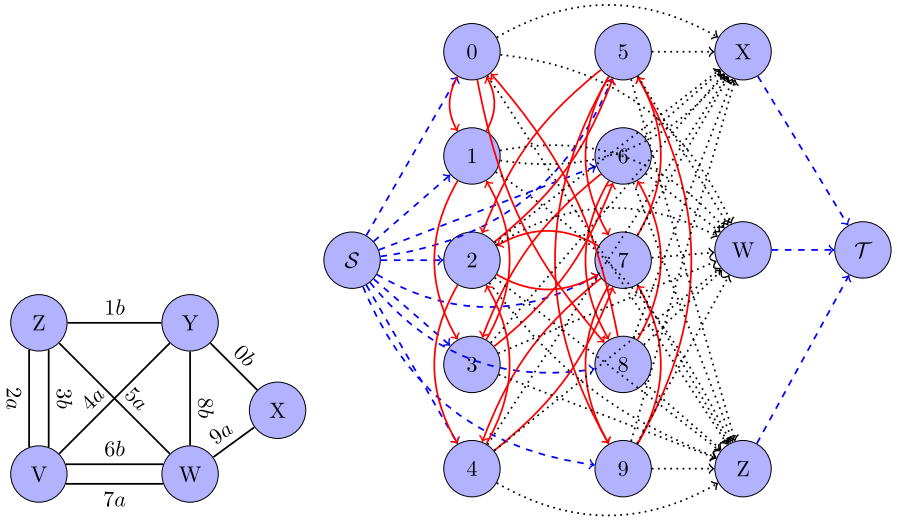

Fig. 2 On the left, a possible line plan on the railway graph in Fig. 1. On the right, the resulting flow graph ($G_F^{dy} = (N_F^{dy}, A_F^{dy})$). The arcs from and to the source $\mathcal{S}$ and sink $\mathcal{T}$ are dashed blue, the interchange arcs (A_I) solid red and the arcs to the facilities (A_O) are dotted black

The first-stage binary decision variables $\boldsymbol{y} \in \{0,1\}^{|N_C|}$ equal 1 when a facility is opened and 0 otherwise. The second-stage decisions are the flow through arcs $a \in A_F^{dy}$ denoted by $z(a) \in \mathbb{R}_0^+$ and the cost are denoted by $c(a)$. Let $\delta_{\text{in}}^d(n)$ and $\delta_{\text{out}}^d(n)$ be the set of in-going and out-going arcs of node n for scenario d. The set of arcs that represent the interchanges going through end station s for scenario d is denoted by A_s^d. The weights $w_d \; \forall d \in D$ denote the expected fraction of time that a line plan is used during the lifetime of the facilities.

The two-stage stochastic programming model can be formulated as

$$\begin{aligned} &\text{(SMLRP)} \quad \min \sum_{n \in N_C} c_n y_n + \sum_{d \in D} w_d \text{IMRP}(\boldsymbol{y}, d) \\ &\text{s.t.} \sum_{n \in N_F} y_n q_n \geq \max_{d \in D} \sum_{l \in L^d} m_l^d, \end{aligned} \tag{8}$$

$$y_n \in \{0,1\} \quad \forall n \in N_C, \tag{9}$$

where

$$\begin{aligned} &\text{IMRP}(\boldsymbol{y}, d) = \min \sum_{a \in A_I^d \cup A_C^d} c(a) z(a) \\ &\text{s.t.} \sum_{a \in \delta_{\text{in}}^d(n)} z(a) \leq y_n q_n \quad \forall n \in N_C, \end{aligned} \tag{10}$$

$$\sum_{a \in \delta^d_{\text{in}}(n)} z(a) = \sum_{a \in \delta^d_{\text{out}}(n)} z(a) \quad \forall n \in N_F^{dy} \backslash \{\mathcal{S}, \mathcal{T}\}, \tag{11}$$

$$z(a) = m_l^d \quad \forall l \in L^d, \quad a \in \delta^d_{\text{in}}(n_l^d) \backslash A_I, \tag{12}$$

$$\sum_{a \in A_s^d} z(a) \leq g_s^d \quad \forall s \in S^d, \tag{13}$$

$$\sum_{a \in A_I^d} z(a) \leq G^d, \tag{14}$$

$$z(a) \geq 0 \quad \forall a \in A_F^{dy}. \tag{15}$$

The objective is to optimize the annual facility cost in combination with the average maintenance routing cost. Constraint (8) guarantees that the combined capacity for the opened facilities is sufficient to handle all maintenance visits. This constraint ensures relative complete recourse. Constraint (10) enforce the capacity of a maintenance facility. Constraint (11) is the flow conservation constraint, while Constraint (12) guarantees that all maintenance visits are assigned to a facility. Constraints (13) and (14) are the end station and budget interchange capacity constraints.

Note that there is a different G_F^{dy} for each combination of first-stage solution $\boldsymbol{y}$ and scenario d. Consequently, the values and dimensions of the recourse matrices are different for each scenario and the artificial scenario method proposed by Crainic et al. (2020) does not apply here.

4 A column-and-constraint generation algorithm for the SMLRP

The C&CG algorithm requires one large maintenance routing graph for all scenarios that are included in $D_0 \subseteq D$. This graph $G_M^{D_0} = (N_M^{D_0}, A_M^{D_0})$ can be built with the following steps:

- A node is created for each scenario and for each line. The set with all line nodes belonging to a scenario $d \in D_0$ is denoted N_L^d.
- A source $\mathcal{S}$ is created that is shared for all scenarios. The source is connected with an arc to each node in $\bigcup_{d \in D_0} N_L^d$.
- An arc is created between every line where an interchange is possible where the cost is the interchange coordination cost. The set with all interchanges is denoted $A_I^{D_0}$.
- A node is created for every candidate facility and each of these nodes is connected with an arc to the sink $\mathcal{T}$. The set of candidate facility nodes is denoted by N_C.

- An arc is created from each node $n \in \bigcup_{d \in D_0} N_L^d$ to each facility. The cost of this arc is the deadheading cost of the line to the facility.

The C&CG algorithm for the SMLRP from Sect. 3 can now be expressed as follows:

1. Construct the graph $G_M^{D_0} = (N_M^{D_0}, A_M^{D_0})$ for all scenarios in D_0 and solve the master problem. The master problem is the IMIP for all scenarios in D_0 and the first-stage problem for all scenarios in D_1 with optimality cuts representing input from the second stage.

$$\text{LB} = \min_{y,\theta} \sum_{n \in N_C} c_n y_n + \sum_{d \in D_0} w_d \sum_{a \in A_I^d \cup A_C^d} c(a) z(a) + \sum_{d \in D_1} w_d \theta_d$$

subject to

$$\sum_{a \in \delta_{\text{in}}^d(n)} z(a) \leq y_n q_n \quad \forall d \in D_0, \quad \forall n \in N_C, \tag{16}$$

$$\sum_{a \in \delta_{\text{in}}(n)} z(a) = \sum_{a \in \delta_{\text{out}}(n)} z(a) \quad \forall n \in N_M^{D_0} \backslash \{\mathcal{S}, \mathcal{T}\}, \tag{17}$$

$$z(a) = m_l^d \quad \forall d \in D_0, \quad \forall l \in L^d, \quad a \in \delta_{\text{in}}^d(n_l^d) \backslash A_I^{D_0}, \tag{18}$$

$$\sum_{a \in A_s^d} z(a) \leq g_s^d \quad \forall d \in D_0, \quad \forall s \in S^d, \tag{19}$$

$$\sum_{a \in A_I^d} z(a) \leq G^d \quad \forall d \in D_0, \tag{20}$$

$$\sum_{n \in N_C} y_n q_n \geq \max_{d \in D} \sum_{l \in L^d} m_l^d, \tag{21}$$

$$\theta_d \geq \sum_{n \in N_C} a_{kn}^d y_n + b_k^d \quad k = 1, \ldots, i, \quad d \in D_1, \tag{22}$$

$$\theta_d \geq 0 \quad \forall d \in D_1, \tag{23}$$

$$z(a) \geq 0 \quad \forall a \in A_F^{D_0}, \tag{24}$$

$$y_n \in \{0, 1\} \quad \forall n \in N_C. \tag{25}$$

Constraints (16) to (19) are equal to the DEM of the SMLRP for the scenarios in D_0. The cost of the scenarios in D_1 are approximated by MCBD. Constraint (21) guarantees that the opened facilities have sufficient capacity. The variable θ_d represents the maintenance routing costs for scenario $d \in D_1$, and Constraint (22) approximates the constraints $\theta \geq \sum_{d \in D_1} w_d \text{IMRP}(\boldsymbol{y}, d)$. Optimality Constraint (22) is only added to the master problem from iteration 1 and are ignored the first time that it is solved. The coefficients $a_{i+1,n}$ and b_{i+1} are determined from the duals of $\text{IMRP}(\boldsymbol{y}, d)$ in such a way that $\sum_{d \in D_1} \left(\sum_{n \in N_C} a^d_{kn} y^i + b^d_k \right) = \sum_{d \in D_1} w_d \text{IMRP}(\boldsymbol{y}, d)$. The generation of the variables $a^d_{i+1,n}$ and b^d_{i+1} will be explained in Step 5.

2. Calculate the objective value $\text{SMLRP}(\boldsymbol{y}^i) = \boldsymbol{c}^T \boldsymbol{y}^i + \sum_{d \in D} w_d \text{IMRP}(\boldsymbol{y}, d)$ and get $\boldsymbol{x}^i$. If $\text{SMLRP}(\boldsymbol{y}^i)$ is smaller than the upper bound, we update the upper bound with the new objective value and set the incumbent solution at $(\hat{\boldsymbol{y}}, \hat{\boldsymbol{x}}) = (\boldsymbol{y}^i, \boldsymbol{x}^i)$.
3. For this step there are a few options:

   - Add $\arg\max_{d \in D} \text{IMRP}(\boldsymbol{y}, d)$ to set D_0 and remove it from D_1.
   - Pick more than one scenario per iteration. Possible choices are the worst, best, middle scenario when the scenarios are sorted in ascending second-stage cost.
   - Do nothing.

   Note that the transfer of a scenario from D_1 to D_0 both removes and creates constraints. When the scenario is removed from D_1, Constraints (22) and (23) associated with that scenario are removed, while the addition to D_0 creates new constraints ((16)–(20), (24)).
4. If $\frac{\text{UB-LB}}{\text{LB}} < \delta$, where $\delta > 0$ is a pre-specified tolerance, the algorithm stops and returns $(\hat{\boldsymbol{y}}, \hat{\boldsymbol{x}})$ as the optimal solution and UB as the optimal objective value. Otherwise the algorithm proceeds to Step 5.
5. Let $\mu^d_{in}, \nu^d_{inl}, \pi^d_{il}, \phi^d_{is}$ and ω^d_i be the dual variables for $\text{IMRP}(\boldsymbol{y}^i, d)$. The cut coefficients are

$$a^d_{i+1,n} = \mu^d_{in} q^d_n \quad \forall n \in N_F, \quad \forall d \in D_1$$

   and

$$b^d_{i+1} = \sum_{l \in L^d} \pi^d_{il} m^d_l + \sum_{s \in S^d} \phi^d_{is} g^d_s + \omega^d_i G^d \quad \forall d \in D_1.$$

   Update $i = i + 1$ and go back to Step 1.

5 Experimental results

Java with CPLEX library version 12.6.3 on a laptop with an Intel Core i7-4710MQ Quad Core 2.5 GHz processor with 32 GB of RAM is used for the computational experiments. As stopping criterion (δ) a relative tolerance ($\frac{\text{UB-LB}}{\text{LB}}$) of 10^{-4} is used. For Step 3 of our algorithm, we tested the first two addition policies and found that

the best addition method is to always add the scenario with the highest second-stage cost. We also tested strategies such as adding a scenario every 3, 5, or 10 iterations and found that adding a scenario each iteration is the best strategy for the SMLRP. These experiments indicated that the C&CG algorithm finds a good UB early and that the scenarios added to the master problem guide the first-stage problem to the optimal solution. Consequently, C&CG makes the use of some acceleration techniques commonly used in BD such as a trust region and a good upper bound less important. Furthermore, it is known from Tönissen et al. (2019), that cut strengthening does not decrease the computational time for the SMLRP. Because of these reasons, we decided to keep our experiments and results simple by not combining our algorithm with accelerating techniques for BD.

5.1 Matrix size versus computational speed

For these experiments, we used the computational SMLRP instances from Tönissen et al. (2019). This set consists of 15 instances that have 10, 25, and 50 candidate facilities for each 2^x scenarios, where x is varied between 0 and 15. We first compare different variations of the C&CG algorithm on 15 instances, with 25 facilities and 128 scenarios. We show these results in Table 1. C&CG x% is the C&CG algorithm where we start with x% of the scenarios in D_0 and in each iteration the scenario with the highest second-stage cost from the set D_1 is moved to D_0. MCPBD x% is MCPBD, where we start with x% of the scenarios in D_0, but no scenarios are added afterwards. We add the scenarios with the highest weights to D_0 and select scenarios randomly when they have equal weight. We denote the average number of iterations the algorithm takes, the average number of constraints and columns in the last iteration, the master problem matrix size (the number of constraints times the number of columns) in millions, the average number of scenarios in set D_0 after termination of the algorithm, and the average solution time in seconds. The matrix size gives an indication of the size and the memory requirements of the problem, although a sparse representation is used by CPLEX.

Table 1 C&CG and MC(P)BD results for the SMLRP for instances with 25 candidate facilities and 128 scenarios

Strategy	Iterations	Constraints	Columns	Size ($\times 10^6$)	Scens	Time (s)
DEM	1	12,097	120,199	1454	128	34
C&CG 50%	4	6693	63,915	428	68	98
C&CG 25%	9	4652	37,898	176	41	125
C&CG 10%	14	4075	26,048	106	27	152
C&CG 0%	19	4331	20,813	90	20	178
MCPBD 50%	9	6527	59,552	389	64	216
MCPBD 25%	36	6274	28,801	181	32	300
MCPBD 10%	53	7213	12,297	89	13	523
MCBD	95	12,134	153	1.9	0	1095

All C&CG strategies perform computationally better than the MCPBD strategies, while the used number of scenarios and matrix size is only slightly higher. The fastest strategy is DEM, but the size of the matrix is extremely large compared to the other strategies. The fastest C&CG strategy is C&CG 50%, but the size of the matrix is still large, while computationally it is less than two times faster than C&CG 0%. C&CG 0% uses more than three times fewer scenarios in the master problem, less memory, and is computationally faster than MCPBD 50%. The C&CG strategies have a nice trade-off between the matrix size and the solution speed. The slowest strategy is MCBD, but the size of the problem is the smallest with only 1.9 million possible entries in the matrix. Even when we include all kinds of accelerations (see Table 6, page 29 in Tönissen et al. 2019) to the MCBD strategy, C&CG performs better. In addition, MCBD performs much better than regular BD (Tönissen et al. 2019).

Table 2 shows the results when we decrease or increase the number of facilities to 10 and 50 facilities, respectively. In addition, it again shows a trade-off between computational speed and time. The C&CG strategies are slower, but the matrix size is significantly reduced compared to the DEM. Furthermore, based on Tables 1 and 2, it can be concluded that C&CG outperforms MC(P)BD. Unfortunately, not all instances with 50 facilities could be solved within 18,000 s (5 h).

5.2 The influence of the number of scenarios

In this section, we test the influence of the number of scenarios. The experiments are only done with the instances with ten candidate facility locations due to time constraints. Some of the instances with only 1 scenario cannot be solved by MCBD within an hour and almost none of the instances with 512 scenarios can be solved within an hour. MCBD is, therefore omitted from further results.

Table 2 C&CG and MC(P)BD results for the SMLRP for instances with 10, 25, and 50 candidate facilities and 128 scenarios

Strategy	10 Facilities		25 Facilities		50 Facilities		
	Size ($\times 10^6$)	Time (s)	Size ($\times 10^6$)	Time (s)	Size ($\times 10^6$)	Time (s)	Fails
DEM	162	1	1454	34	7267	743	0
C&CG 50%	44	1	428	98	2901	4525	1
C&CG 25%	14	1	176	125	2114	7014	4
C&CG 10%	5	1	106	152	2048	7971	4
C&CG 0%	2	1	90	178	2249	8400	4
MCPBD 50%	42	1	389	216	2147	8411	4
MCPBD 25%	13	1	181	300	957	13,350	10
MCPBD 10%	4	2	89	523	–	> 18,000	> 10
MCBD	0.2	2	2	1095	–	> 18,000	> 10

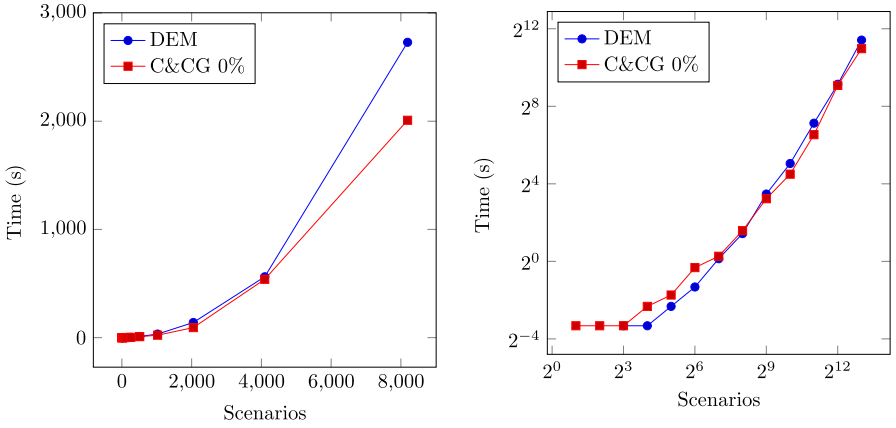

Fig. 3 Behavior of the computational time for the C&CG 0% strategy and DEM for different number of scenarios at different scales

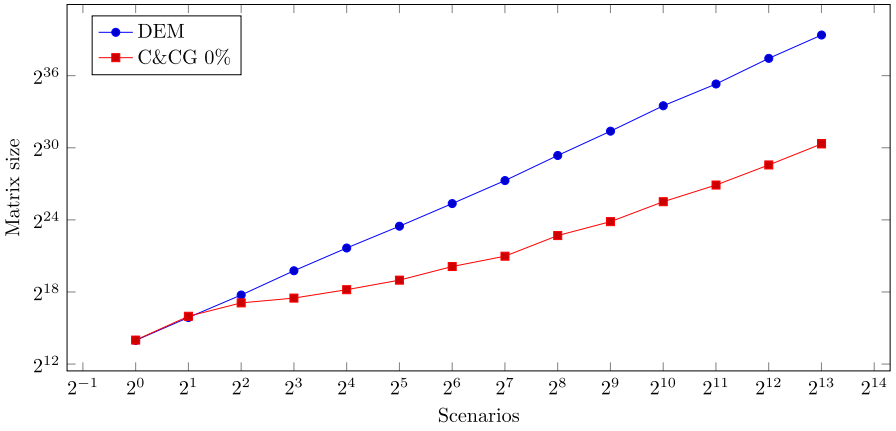

Fig. 4 Behavior of the C&CG 0% strategy and DEM for different number of scenarios at a logarithmic scale

Furthermore, because it was shown in Sect. 5.1 that C&CG outperforms MCPBD, we also exclude MCPBD.

Figures 3 and 4 show the average computational time and matrix size for 15 instances with ten facilities for a different number of scenarios. These figures do not show the results of C&CG 10%, 20%, and 50% because they are always exactly in between the lines of the DEM and the C&CG 0% strategy. According to the Wilcoxon signed rank test (p value $= 6.338 \times 10^{-8}$), the C&CG 0% strategy is computationally faster than the DEM for instances with 512 or more scenarios.

Figure 4 clearly shows that the results of the matrix size and indirectly the memory requirements are highly in favor of the C&CG 0% strategy.

Table 3 C&CG results with adaptive relative tolerance for the SMLRP for instances with 10, 25, and 50 candidate facilities and 128 scenarios

strategy	10 facilities		25 facilities		50 facilities		
	Size ($\times 10^6$)	Time (s)	Size ($\times 10^6$)	Time (s)	Size ($\times 10^6$)	Time (s)	Fails
C&CG 0%	2	1	90	178	2249	8820	4
C&CG 0% A	3	1	107	157	2762	6877	4
C&CG 50%	44	1	428	98	2901	4525	1
C&CG A 50%	47	1	440	97	2487	3092	1

5.3 Adaptive relative tolerance

In our experiments we use an initial_tolerance of 0.25 and an end_tolerance of 10^{-4}. Consequently, the adaptive tolerance is defined as follows: $\max\{\min\{0.25, 0.5\frac{\text{UB-LB}}{\text{LB}}\}, 10^{-4}\}$. We combined this tolerance with the C&CG 0% and the C&CG 50% strategies and tested it on the instances sets with 128 scenarios and 10, 25, and 50 facilities, respectively. The results are shown in Table 3, where an *A* behind the strategy name indicates that the proposed adaptive tolerance has been used. It can be seen that using the proposed adaptive relative tolerance decreases the computational time. As expected, the more facilities (binary decision variables) the instance has, the larger the decrease in computational time due to the adaptive tolerance. However, the adaptive relative tolerance also slightly increases the matrix size for most instances, most likely because of weaker cuts and consequently more iterations.

6 Conclusion

We presented an algorithm to solve large-scale two-stage stochastic programs that finds a middle ground between the DEM and BD. The algorithm can add scenarios to the master problem in any iteration to find a balance between computational speed and memory requirements. We showcased the approach on instances of the SMLRP and found that our algorithm can solve instances that MC(P)BD cannot solve within reasonable time. Furthermore, our algorithm outperforms DEM in memory requirements and, when the number of scenarios is large enough, also in computational speed. Finally, the adaptive relative tolerance can be used to further trade-off computational time and memory requirements. Using an adaptive relative tolerance can be useful for instances with many binary variables for which the master problem is the bottleneck.

Acknowledgements The study was funded by Nedtrain. Furthermore, the authors thank Geert-Jan van Houtum and Nicole Perez-Becker for giving valuable input.

References

Adulyasak Y, Cordeau J-F, Jans R (2015) Benders decomposition for production routing under demand uncertainty. Oper Res 63(4):851–867

An Y, Zeng B (2015) Exploring the modeling capacity of two-stage robust optimization: variants of robust unit commitment model. IEEE Trans Power Syst 30(1):109–122

An Y, Zeng B, Zhang Y, Zhao L (2014) Reliable p-median facility location problem: two-stage robust models and algorithms. Transp Res Part B Methodol 64:54–72

Benders JF (1962) Partitioning procedures for solving mixed-variables programming problems. Numer Math 4(1):238–252

Birge J, Louveaux F (1988) A multicut algorithm for two-stage stochastic linear programs. Eur J Oper Res 34(3):384–392

Chan TC, Shen Z-JM, Siddiq A (2017) Robust defibrillator deployment under cardiac arrest location uncertainty via row-and-column generation. Oper Res 66:358–379

Contreras I, Cordeau J-F, Laporte G (2011) Stochastic uncapacitated hub location. Eur J Oper Res 212(3):518–528

Crainic TG, Hewitt M, Maggioni F, Rei W (2020) Partial benders decomposition: general methodology and application to stochastic network design. Transp Sci

Geoffrion AM, Graves GW (1974) Multicommodity distribution system design by Benders decomposition. Manage Sci 20(5):822–844

Lee C, Liu C, Mehrotra S, Bie Z (2015) Robust distribution network reconfiguration. IEEE Trans Smart Grid 6(2):836–842

Rahmaniani R, Crainic TG, Gendreau M, Rei W (2017) The Benders decomposition algorithm: a literature review. Eur J Oper Res 259(3):801–817

Tönissen DD, Arts JJ, Shen Z-J (2019) Maintenance location routing for rolling stock under line and fleet planning uncertainty. Transp Sci 53(5):1252–1270

van Slyke RM, Wets R (1969) L-shaped linear programs with applications to optimal control and stochastic programming. SIAM J Appl Math 17(4):638–663

You F, Grossmann IE (2013) Multicut Benders decomposition algorithm for process supply chain planning under uncertainty. Ann Oper Res 210(1):191–211

Zeng B, Zhao L (2013) Solving two-stage robust optimization problems using a column-and-constraint generation method. Oper Res Lett 41(5):457–461

Zhao L, Zeng B (2012) Robust unit commitment problem with demand response and wind energy. In: 2012 IEEE power and energy society general meeting. IEEE, pp 1–8

Authors and Affiliations

Denise D. Tönissen[1] · Joachim J. Arts[2] · Zuo-Jun Max Shen[3]

Joachim J. Arts
joachim.arts@uni.lu

Zuo-Jun Max Shen
maxshen@berkeley.edu

[1] Department of Operations Analytics, School of Business and Economics, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

[2] Centre for Logistics and Supply Chain Management, University of Luxembourg, Luxembourg, Luxembourg

[3] Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720, USA