Dynamic sequencing and cut consolidation for the parallel hybrid-cut nested L-shaped method

Christian Wolf[a], Achim Koberstein[b,*]

[a] *DS&OR Lab, University of Paderborn, Warburger Str. 100,33098 Paderborn, Germany*
[b] *Goethe University Frankfurt, Grüneburgplatz 1, 60323 Frankfurt am Main, Germany*

Abstract

The Nested L-shaped method is used to solve two- and multi-stage linear stochastic programs with recourse, which can have integer variables on the first stage. In this paper we present and evaluate a cut consolidation technique and a dynamic sequencing protocol to accelerate the solution process. Furthermore, we present a parallelized implementation of the algorithm, which is developed within the COIN-OR framework. We show on a test set of 48 two-stage and 42 multi-stage problems, that both of the developed techniques lead to significant speed ups in computation time.

Keywords: Stochastic programming, Nested L-shaped method, Sequencing protocols, Cut consolidation

1. Introduction

Many real world applications can be modeled as a multi-stage stochastic program with recourse, e.g. in applications from supply chain planning, electricity and finance (cf. Wallace & Ziemba, 2005). An algorithm to solve two-stage stochastic linear programs with discrete and finite distributions is the L-shaped method developed by Van Slyke & Wets (1969) which is an adaption of Benders decomposition (cf. Benders, 1962) to two-stage recourse problems. It can be used in a nested application to solve multi-stage stochastic programs with recourse (cf. Birge, 1985). The algorithmic improvement of the L-shaped method is highly relevant and ongoing research. Recent achievements include a variant of the algorithm with aggregated cuts introduced by Trukhanov et al. (2010) and the generation of tighter feasibility cuts (see Aranburu et al., 2011). Zverovich et al. (2010) compare alternative variantes of the L-shaped method, namely, a regularized version based on work of Ruszczyński (1986) and the level decomposition method developed by Fábián & Szőke (2006). They find that regularized versions outperform non-reguralized versions on many of their test models.

In this paper we propose further algorithmic techniques that can improve the performance of the parallel nested L-shaped method and its variants. The Nested L-shaped

*Corresponding author
Email addresses: christian.wolf@dsor.de (Christian Wolf), koberstein@wiwi.uni-frankfurt.de (Achim Koberstein)

algorithm can decide at every stage other then the first or the last in which direction it should push information. Information is pushed up the tree by feasibility or optimality cuts to the ancestor problems. The solution to the current problem can be passed down the tree to form new right hand sides for the successor problems. The algorithm decides where to push the information according to a tree-traversing strategy, a so called sequencing protocol. Several studies showed that the sequencing protocol itself has an impact on the solution time (cf. Gassmann, 1990; Morton, 1996; Altenstedt, 2003). We propose a new dynamic sequencing protocol that leads to faster solution times compared with the well known sequencing protocols FastForwardFastBack and ϵ-FastBack.

Depending on the level of cut aggregation, a certain number of cuts is added to the subproblems at every iteration of the algorithm. Especially for the multi-cut case (see Birge & Louveaux, 1988) this can become prohibitive in both memory usage and solution time. An approach to reduce the computational burden is the removal of previously added cuts. Ruszczyński & Shapiro (2002) showed, that there is no easy way to keep the number of cuts bounded. However, they also point out, that it is possible to delete inactive cuts, when the objective value of the master problem strictly increases. We suggest a cut removal strategy that not only removes inactive cuts, but retains an aggregated cut so that not all information contained in the removed cuts is lost. Our results show that the removal of old and inactive cuts can lead to shorter solution times on many problems. Furthermore, we investigate the scenario aggregation technique recently employed by Trukhanov et al. (2010) for the two-stage case for the multi-stage case. We evaluate the mentioned techniques using our own parallel implementation of the nested L-shaped method that is partly based upon and embedded into the COIN-OR project (see Lougee-Heimer, 2003).

The remainder of the paper is organized as follows. Section 2 describes a basic variant of the Nested L-shaped method. Our new sequencing protocol and cut consolidation techniques are presented in Section 3. The parallel implementation is explained in Section 4. In Section 5 we evaluate the presented techniques and implementation based on a computational study. We conclude with a brief summary of our main results and an outlook on future research opportunities in Section 6.

2. Nested Benders decomposition

This paper is focussed on two- and multi-stage stochastic programs with recourse with discrete and finite distributions. For a general introduction to stochastic programming see Birge & Louveaux (1997) and Shapiro et al. (2009), for an overview of applications and implemented solution algorithms see Wallace & Ziemba (2005) and Kall & Mayer (2010), respectively. In order to fix our notation and as an introduction for the non-expert we start with describing a basic version of the Nested Benders decomposition algorithm.

We can formulate the multi-stage stochastic program with recourse with discrete and finite distributions as follows:

$$\min c_t^i x_t^i + Q_t(x_t^i) : W_t^i x_t^i = h_t^i - T_t^i x_{t-1}^{a(i,t)} \tag{1}$$

$$= \min c_t^i x_t^i + \theta_t^i : W_t^i x_t^i = h_t^i - T_t^i x_{t-1}^{a(i,t)}, \theta_t^i \geq Q_t(x_t^i) \tag{2}$$

where

$$Q_t(x_t^i) := \min \sum_{j \in d(i,t)} p_{t+1}^j c_{t+1}^j x_{t+1}^j + Q_{t+1}(x_{t+1}^j) \tag{3}$$

$$\text{s.t.} \quad W_{t+1}^j x_{t+1}^j = h_{t+1}^j - T_{t+1}^j x_t^i \tag{4}$$

$$\begin{aligned} l_t^i &\leq x_t^i \leq u_t^i \quad , \\ l_{t+1}^j &\leq x_{t+1}^j \leq u_{t+1}^j \quad , \quad j \in d(i,t) \end{aligned}$$

and $Q_{T-1}(\cdot) = 0$, for $t = 0$ and $i = 0$[1]. This is a recursive, node-based formulation, where the subscripts t denote the stage and the superscripts i denote the node of the scenario tree. We use the notion of ancestor $a(i,t) \in V$ and descendants $d(i,t) \subseteq V$ of a node $i \in V$ at stage t in the scenario tree to refer to the respective parent and child nodes. The set of descendants of the nodes at the last stage is empty. $x_t^i \in R^{n_t}$ is the vector of decision variables, $c_t^i \in R^{n_t}$ is the cost vector and $h_t^i \in R^{m_t}$ is the right-hand side vector, all for nodes $i \in \{0, \ldots, K_t - 1\}$ at stage t. p_t^i denotes the node probability. The technology matrices $T_t^i \in R^{m_t, n_{t-1}}$ of stage t belong to the decision variables of stage $t-1$, whereas the recourse matrices $W_t^i \in R^{m_t, n_t}$ belong to the decision variables of the same stage.

The Nested Benders decomposition algorithm (see Dempster & Thompson, 1998; Birge & Louveaux, 1997; Gassmann, 1990; Ruszczyński & Shapiro, 2002, for further descriptions) constructs an outer linear approximation θ_t^i of the recourse function $Q_t^i(\cdot)$ in consecutive iterations via cutting planes at every node of the scenario tree. To obtain a LP formulation of this approach the restriction $\theta_t^i \geq Q_t(x_t^i)$ is removed from the problem (1) and the approximation term θ_t^i is added instead of $Q_t(\cdot)$ to the objective function. Subproblems at stage t do no longer rely explicitly on all later stage problems. The cutting planes that bound the approximation variables are *optimality cuts*. *Feasibility cuts* restrict the solution set to solutions that are feasible for all subproblems.

The subproblem formulation (1) is hence transformed to the following subproblem approximation problem:

$$\min c_t^i x_t^i + \theta_t^i : W_t^i x_t^i = h_t^i - T_t^i x_{t-1}^{a(i,t)} \tag{5}$$

$$D_{t,r}^i x_t^i \geq d_{t,r}^i, r \in \{1, \ldots, it\} \setminus \mathcal{F}_t^i(it) \tag{6}$$

$$E_{t,s}^i x_t^i + \theta_t^i \geq e_{t,s}^i, s \in \mathcal{F}_t^i(it) \tag{7}$$

$$l_t^i \leq x_t^i \leq u_t^i \tag{8}$$

where (7) are the optimality and (6) are the feasibility cuts. $\mathcal{F}_t^i(it)$ contains all iterations where optimality cuts were added to the current problem, up to the current iteration it.

The single linear approximation θ_t^i of the recourse function can be split into up to A_t^i approximation terms $\theta_{t,k}^i$, so called *aggregates*, that form a partition of the descendant node set $d(i,t)$:

$$\bigcup_{k=1}^{A_t^i} S_{t,k}^i = d(i,t), S_{t,k}^i \cap S_{t,l}^i = \emptyset, \forall l \neq k$$

[1] For $t = 0$, we drop $T_0^i x_{-1}^{a(i,0)}$ from the constraints as there is no prior solution to the first stage.

Each partition $S^i_{t,k}$ contains distinct nodes from the descendant node set. The multi-cut method (cf. Birge & Louveaux, 1988) is a special case, where the number of partitions is equal to the number of descendant nodes. In this case each partition consist of exactly one descendant node. If only one partition is used, we have the single-cut method. This notation can thus be used to refer to the hybrid method introduced by Trukhanov et al. (2010) as well as the single and multi-cut methods which are just two special cases of the hybrid method.

If a subproblem is solved to optimality, it has a primal and dual optimal solution. With dual feasible solutions $(\pi^j_{t+1}, \rho^j_{t+1}, \sigma^j_{t+1}, \lambda^j_{t+1}, \mu^j_{t+1})$ to all the descendants problems of the current node, $j \in d(i,t)$, we can compute an optimality cut $E^i_{t,k,s}x^i_t + \theta^i_{t,k} \geq e^i_{t,k,it}$ with

$$E^i_{t,k,s} = \sum_{j \in S^i_{t,k}} \frac{p^j_{t+1}}{p^i_t} (\pi^j_{t+1})^T T^j_{t+1} \tag{9}$$

$$e^i_{t,k,s} = \sum_{j \in S^i_{t,k}} \frac{p^j_{t+1}}{p^i_t} \Big[(\pi^j_{t+1})^T h^j_{t+1} + (\rho^j_{t+1})^T d^j_{t+1} + \tag{10}$$
$$(\sigma^j_{t+1})^T e^j_{t+1} + (\lambda^j_{t+1})^T l^j_{t+1} + (\mu^j_{t+1})^T u^j_{t+1} \Big],$$

where d^j_{t+1} denotes the vector of all feasibility cut right-hand-side values and e^j_{t+1} the vector of all optimality cut right-hand-side values, respectively, for problem j at stage $t+1$. The dual values π^j_{t+1} correspond to the original rows of the problem, ρ^j_{t+1} to the feasibility cuts and σ^j_{t+1} to the optimality cuts. λ^j_{t+1} corresponds to the lower bounds of the variables and μ^j_{t+1} to the upper bounds of the variables. If a subproblem is infeasible, and the dual is unbounded, a dual ray $(\pi^j_{t+1}, \rho^j_{t+1}, \sigma^j_{t+1}, \lambda^j_{t+1}, \mu^j_{t+1})$ exists for the problem $j \in d(i,t)$. Using this ray we can then compute a feasibility cut $D^i_{t,r}x^i_t \geq d^i_{t,r}$ with

$$D^i_{t,r} = (\pi^j_{t+1})^T T^j_{t+1} \tag{11}$$

$$d^i_{t,r} = (\pi^j_{t+1})^T h^j_{t+1} + (\sigma^j_{t+1})^T e^j_{t+1} + (\rho^j_{t+1})^T d^j_{t+1} + (\lambda^j_{t+1})^T l^j_{t+1} + (\mu^j_{t+1})^T u^j_{t+1}, \tag{12}$$

with D^j_{t+1} and d^j_{t+1} defined as above. If the primal subproblem is unbounded, the overall problem is unbounded.

With these definitions we can now formulate the subproblem with scenario aggregation $P(i,t)$ for node i at stage t:

$$\min c^i_t x^i_t + \sum_{k=0}^{A^i_t} \theta^i_{t,k} : W^i_t x^i_t = h^i_t - T^i_t x^{a(i,t)}_{t-1} \tag{13}$$

$$D^i_{t,r}x^i_t \geq d^i_{t,r}, r \in \{1, \ldots, it\} \setminus \mathcal{F}^i_t(it) \tag{14}$$

$$E^i_{t,k,s}x^i_t + \theta^i_{t,k} \geq e^i_{t,k,s}, s \in \mathcal{F}^i_t(it), k = 1, \ldots, A^i_t \tag{15}$$

$$l^i_t \leq x^i_t \leq u^i_t \tag{16}$$

If more than one descendant problem is infeasible at a given iteration, it is in principle possible to generate more than one feasibility cut, e.g. one cut for every infeasible node. We formally state the general Nested L-shaped method as follows:

1. Set $t = 0$, $i = 0$,$it = 0$,$lb = -\infty$,$ub = \infty$, $dir = forward$. Initialize all θ variables with a coefficient of 0 in the corresponding objective function.
2. If $|ub - lb| < \epsilon_{gap}$ or $|ub - lb| / \left(|lb| + 1e^{-10}\right) < \epsilon_{gap}$, stop.
3. Solve problem $P(i, t)$.
   - If infeasible and $t = 0$ stop, problem is infeasible.
   - If infeasible and $t > 0$ store the dual ray $\pi_t^i, \rho_t^i, \sigma_t^i, \lambda_t^i, \mu_t^i$ and compute a feasibility cut (14) for problem $P(a(i, t), t - 1)$. Set $dir = backward$ and go to step 7.
   - If feasible and $t = 0$ set lb to the objective value of $P(i, t)$.
   - If feasible and $t < T - 1$, store the dual values $\pi_t^i, \rho_t^i, \sigma_t^i, \lambda_t^i, \mu_t^i$ and the primal values x_t^i. If $i < K_t - 1$, set $i = i + 1$ and go to step 3. If $i = K_t - 1$ set $i = 0$.
4. Call sequencing protocol to decide the direction.
5. For all nodes $j \in d(i, t)$ and for all partitions $k = 0, \ldots, A_{t,it}^j$
   - Compute optimality cut coefficients $E_{t-1,k,it}^j$ (9) and right hand side values $e_{t-1,k,it}^j$ (10) to form an optimality cut (15) for aggregate k and problem P(t-1,j).
   - Test if generated cut should be added to the problem. If this is the first optimality cut for this aggregate, set the corresponding objective coefficient to 1.
   - If $i < K_t - 1$, set $i = i + 1$ and go to step 5, else set $i = 0$.
6. If $t = T - 1$, compute temporary upper bound $temp_u b$ by summing up $p_{t'}^{i'} c_{t'}^{i'} x_{t'}^{i'}$ for $t' = 0, \ldots, T - 1, i' = 0, \ldots, K_{t'}$. If $temp_u b < ub$, set $ub = temp_u b$.
7. If $dir = forward$, set $t = t + 1$, else set $t = t - 1$. Go to step 2

3. Dynamic sequencing and cut consolidation

3.1. A dynamic sequencing protocol

After all problems of a certain stage have been solved, a decision has to be made whether to move back up the tree and thereby give information in the form of optimality or feasibility cuts to the previous stage or to proceed to the next stage with the new solution from the current stage as input which modifies the right-hand-side of the problems at that stage. Sequencing protocols formulate rules for how this decision is made. Sequencing protocols are only needed for multi-stage problems, as there is no choice in which direction to go for the two-stage case.

At the first stage of a multi-stage problem it is only possible to move to the next stage and pass the current solution down the tree. At the last stage it is only possible to solve all the subproblems and generate optimality and/or feasibility cuts for the previous stage. When a subproblem was found to be infeasible at a stage, the algorithm moves back to the first stage, i.e. the direction is backward. Three common strategies were developed by Gassmann (1990). The first strategy is the Fast-Forward-Fast-Back (FFFB) or Fastpass strategy that goes down the whole tree and back up to the root from there. This is called a full sweep, consisting of a full forward and backward sweep. A forward sweep solves all subproblems from stage 0 to stage $T - 1$. A backward sweep consists

of adding cuts to all subproblems at stages $T-2$ to 0, by solving problems from stage $T-2$ up to stage 1. This is repeated until the algorithm finishes, i.e. the gap between upper and lower bound is small enough.

The Fast-Forward (FF) strategy tries to move forward or down the tree whenever possible. It only goes up the tree when the current stage is solved to optimality with respect to the current primal information, i.e. the gap between the lower and upper bound at the current stage is less than a small tolerance ϵ_{gap}. The Fast-Back (FB) strategy does the opposite, it tries to move back up the tree whenever possible. It only moves down a further stage, if no new optimality cuts can be generated at the current stage or the gap is below an ϵ_{gap}. It requires an initialization period because it needs an initial approximation of the recourse function at every stage.

An evaluation conducted by Gassmann (1990) showed that out of the three strategies, the FFFB strategy is the best. Morton (1996) comes to the same conclusion, adding that an ϵ-Fast-Back strategy reached comparable performance. He introduced the notions of absolute error and discrepancy, which we describe below. In the ϵ-Fast-Forward strategy, the algorithm goes back up the tree, when the absolute error is smaller than $\epsilon \cdot \min(|LB|, |UB|)$. The ϵ-Fast-Back strategy goes further down the tree, when the discrepancy is smaller than $\epsilon \cdot \min(|LB|, |UB|)$ instead of a fixed ϵ_{gap}. The discrepancy for stage t is defined as

$$Disc(t) = \sum_{i=0}^{K_t-1} \left[p_t^i c_t^i x_t^i + \sum_{l=1}^{A_t^i} \theta_{t,l}^i \right] - \sum_{j=0}^{K_{t-1}-1} \sum_{l=1}^{A_{t-1}^j} \theta_{t-1,l}^j, \tag{17}$$

which is the difference between the approximation of the recourse functions for stage t plus the probability weighted sum of the objective functions at stage t and the approximation of the recourse functions at stage $t+1$. The absolute error for stage t is defined as

$$AbsErr(t) = \sum_{j=t+1}^{T-1} \sum_{i=0}^{K_j} p_j^i c_j^i x_j^i - \sum_{i=0}^{K_t-1} \sum_{l=1}^{A_t^i} \theta_{t,l}^i, \tag{18}$$

which is the difference between the probability weighted sum of objective functions of all stages after stage t and the approximation of the recourse functions at stage t. To be able to compute the absolute error for stage t, all stages $t' > t$ have to be solved. Hence, it is only possible to compute the absolute error after a full forward sweep or during a backward sweep, whereas the discrepancy can also be computed during a forward sweep.

Another simple strategy, the bouncing strategy introduced by Altenstedt (2003), is to solve the problem up to stage $t, t < T-1$, return to the first stage and then do a full iteration. The stage t is also called the *bouncing* stage, as the algorithm changes direction at that stage. The motivation for this protocol is the observation, that the algorithm spends most of the time at later stages, in particular the last stage. This is mostly due to the large number of scenarios at the last stage which correspond to subproblems that have to be solved on this stage. The idea of the bouncing strategy is to perform partial iterations to achieve better solutions to the later stage subproblems and thereby reduce the overall number of major iterations. However, it is not clear which stage is the best bouncing stage a priori.

Our dynamic strategy uses a bouncing stage too, but in a different manner. We declare a stage *critical* to enforce a full sweep after the algorithm reached this stage.

This is done to prevent a cycle to the first stage and back that does not improve the solution much, but costs computation time. The strategy is dynamic because it declares the critical stage after the first full sweep and because the threshold that is used to decide the direction is adapted to the current gap of the algorithm. This is a major advantage over existing strategies which have to be adjusted to specific model instances. Our strategy can be summarized as follows:

1. Do a full sweep. Repeat until no new feasibility cuts have been generated.
2. Determine a critical stage ct.
3. Solve stage 0 problem, set $t = 0$.
4. Set $t = t + 1$, go to stage t and solve problems at stage t.
5. If a problem is infeasible, do a backward sweep and go to step 3.
6. If $Disc(t)$ is lower than current $ForwardThreshold$, go to step 7. Otherwise do a backward sweep and go to step 3.
7. If $t == ct$, do a full sweep and go to step 3. Else go to step 4.

The critical stage is determined by the first sweep of the algorithm in which no problem was found infeasible. The wall clock time the algorithm stays in each stage is measured. We calculate the wall clock time of all stages and the cumulated wall clock time for every stage. If the cumulated wall clock time for stage k divided by the overall wall clock time is greater than a predefined value, e.g. 0.1, the stage k is declared the critical stage. This critical stage heuristic is used to prevent spending time generating cuts for the first stages without getting new dual information from the last stage.

We do not use an absolute value as $ForwardThreshold$, but a relative value compared to the absolute value of the current gap between lower and upper bound. Thus the threshold adjusts along with the absolute gap. We propose to set $ForwardThreshold = 10^{\log_{10}(UB-LB)-1}$, which is essentially equal to the number of digits of the current absolute gap. In contrast to this setting the ϵ-FastBack strategy uses the minimum of $|LB|$ and $|UB|$ times ϵ as a threshold. This works fine, until either the lower or upper bound has a value of zero. If this is the case, $Disc(t)$ is usually greater than the threshold, namely zero. The algorithm concludes that it should generate new optimality cuts in a backward sweep to improve the discrepancy. But the threshold remains at zero, so the algorithm does not terminate. We observed this behavior for some of our test set problems.

3.2. Cut consolidation

In each iteration of the algorithm where all subproblems are feasible new optimality cuts are added to the corresponding master problem. In the multi-stage case, this leads to a growing number of cuts in the respective master problems. In the regularized decomposition method (cf. Ruszczyński, 1986) the algorithm keeps only a limited number of cuts, instead of adding new and keeping all old cuts in the problem. It would be preferable with respect to computational efficiency and memory requirements to keep only the cuts in the problem, that are needed to solve the overall problem to optimality. Unfortunately, as the nested Benders algorithm proceeds in generating new cutting planes to approximate the recourse function, there is no reliable rule to determine which cut can be safely removed (cf. Ruszczyński, 1997). The simple deletion of old inactive cuts can therefore lead to the recomputation of those cuts in later iterations (cf. Ruszczyński

& Shapiro, 2002). Trukhanov et al. (2010) note that the removal of cuts only lead to a reduced memory usage, but had no other effect, e.g., on runtime. We propose a method that reduces the number of cuts significantly, but keeps most of the information that was contained in these cuts. In the case of pure multi-cut, we added the following cuts in some previous iteration s to a node i at stage t:

$$E^i_{t,k,s} x^i_t + \theta^i_{t,k} \geq e^i_{t,k,s}, k = 1, \ldots, A^i_t$$

If all these cuts become redundant, i.e. their corresponding dual values are zero, we can generate a new single cut out of these multiple cuts by just summing up the existing cuts

$$\sum_{k=1}^{A^i_t} E^i_{t,k,s} x^i_t + \theta^i_{t,k} \geq \sum_{k=1}^{A^i_t} e_{t,k,s}.$$

The only difference compared to a common single cut is the sum of aggregate variables $\sum_{k=1}^{A^i_t} \theta^i_{t,k}$ instead of a single aggregate variable θ^i_t. We then replace all the cuts in iteration s with the newly generated single cut. Thereby the number of redundant cuts that needs to be stored in the master problem is reduced from as many as A^i_t to one for iteration s. We call this technique *cut consolidation*. The trade-off between information loss due to aggregation and memory and computational gains due to smaller problems is evaluated in Section 5.

The decision when to aggregates cuts of one iteration into a single cut is called a *cut consolidation scheme*. Our scheme is controlled via two threshold values that guide the consolidation. The first threshold value $ConsecInactive$ specifies the number of iterations a cut needs to be consecutively inactive, before it is marked as removable. The second threshold value $RelativeActive$ specifies how many cuts must be marked as removable before any cuts are consolidated. In our implementation, we set $ConsecInactive = 5$ and $RelativeActive = 0.95$.

If the scheme is too aggressive towards reducing the number of cuts, this can lead to longer overall solution times as cuts are consolidated that the algorithm needs to bound some aggregate and thus recomputes later on. The following pseudo-code summarizes our cut consolidation scheme, which is performed at each iteration it while solving a node i at stage t:

1. Set $num\_inac_s = 0$ for all $s \in \mathcal{F}(it)$.
2. For all non-aggregated optimality cuts k and all (previous) iterations $s \in \mathcal{F}(it)$:
   (a) If the current dual value of cut k is zero, set $ic_{k,s} = ic_{k,s} + 1$. Otherwise, set $ic_{k,s} = 0$.
   (b) If $ic_{k,s} > ConsecInactive$ set $num\_inac_s = num\_inac_s + 1$.
3. For all iterations $s \in \mathcal{F}(it)$:
   If $num\_inac_s > RelativeActive \cdot A^i_t$, aggregate all inactive cuts to a single cut. Remove old cuts from problem, add new cut and resolve the problem. Store warm start.

In the description above, the variables $ic_{k,s}$ count for how many consecutive iterations a cut k generated in iteration s is inactive (step 2(a)). These variables have to be initialized with zero before calling the scheme for the first time. If a counter $ic_{k,s}$ exceeds the threshold $ConsecInactive$ the corresponding cut is marked as removable and the

counter num_inac_s, which counts the number of removable cuts which were generated in a certain iteration s, is increased (step 2(b)). If the number of removable cuts, which were generated in an interation s exceeds the relative treshold $RelativeActive \cdot A_t^i$ all the removable cuts generated in iteration s are consolidated (step 3).

3.3. A note on adaptive aggregation

Trukhanov et al. (2010) show that for their sample of problems it is preferable to choose an aggregation between pure single- and multi-cut. They note, that "a good level of cut aggregation is not known a priori". For this reason they devise the adaptive multi aggregation algorithm that decreases the number of aggregates during the runtime of the algorithm. However, in our view the description of the aggregation scheme "Redundancy Threshold" in their paper contains an inconsistency, which would let the algorithm to be equivalent to the multi-cut method. This is due to the following observation: every aggregate is a free variable with an objective function coefficient of one in the master problem. Minimization of the master problem thus strives to minimize the objective function and the value of this variable. Optimality cuts that are added for an aggregate bound the aggregate variable, so that the problem as a whole is not unbounded. This is noted in their algorithm, as aggregates without optimality cuts are ignored in the computation. The scheme "Redundancy Threshold" explicitly requires that *all* optimality cuts corresponding to an aggregate are redundant, i.e., their dual values are zero. But if all cuts were redundant, the aggregate would not be bounded and thus the problem would be unbounded. As the problem is supposed to have an optimal solution or to be infeasible, this situation can never occur. Therefore, no aggregates would be aggregated and thus the algorithm would behave like the multi-cut method. Furthermore, the scheme still requires a fixed value for the maximal number of scenarios that can get aggregated into a single aggregate. It follows that this value is a bound on the minimal number of aggregates the algorithm should use. For a problem with 1000 scenarios, for instance, a value of 10 is equivalent to at least 100 aggregates, as maximal 10 scenarios can be aggregated into one aggregate. This value has to be determined a priori, which is in principle as difficult as to find a good value for the number of aggregates in the static method. In their computational study, the static approach is superior compared to the adaptive approach for all the investigated problems. Also, Trukhanov et al. (2010) state that they tried several further adaptive schemes which turned out to be inferior to the static version. Due to these considerations we did not pursue further strategies for an adaptive algorithm.

4. Implementation

Our algorithm is implemented in C++ and it is embedded into the COIN-OR project (Lougee-Heimer, 2003), using the Stochastic Modeling Interface (SMI) and the OpenSolverInterface (OSI) in particular. SMI is used to read in problems in the SMPS format (Birge et al., 1987) and to store the scenario tree in memory as well as to query the stored data. OSI provides access to different linear and mixed-integer programming solvers via a common interface. In the remainder of this chapter we describe further design decisions and parameter settings of our implementation.

4.1. Tolerances

A practical issue that arises during implementation is numerical stability of the algorithm. We receive computational results from a third party LP solver that are not necessarily exact. To use the results, tolerances are used in the algorithm and the employed LP solvers. We use a zero tolerance of 10^{-8}, so all values that are smaller than 10^{-8} are treated as zero. This applies to all the values that the LP solver returns such as primal and dual solutions. In addition we use a relative gap tolerance ϵ_{gap} of 10^{-6} between upper and lower bound to decide if we can stop the algorithm. If no tolerances are used, the algorithm does not converge well on practical model instances.

4.2. Parallelization of the algorithm

The Nested L-shaped method lends itself to parallelization, as all the subproblems that have to be solved in a stage are independent of one another and can thus be solved in parallel. Several different parallelization approaches have been proposed in the research literature for different network and computing architectures. Distributed computing techniques were used by (Birge et al., 1996; Dempster & Consigli, 1998; Moritsch et al., 2001). A subtree-wise parallelization, i.e. different nodes in the network process different subtrees, was developed by Birge et al. (1996). According to the authors, this approach works well, but requires "careful attention to processor load balancing". Another option is to parallelize the task of solving subproblems and let a master node compute the cuts (cf. Dempster & Thompson, 1999). An asynchronous protocol is devised by (Moritsch et al., 2001). Vladimirou & Zenios (1999) give an overview over parallel algorithms to solve stochastic programs other than Nested Benders decomposition. If a distributed computing environment is used, the communication overhead has to be kept in mind for the design of the algorithm. Our algorithm is suited for single workstation computers with several processor cores and avoids the communication overhead by the use of shared memory. Our parallel implementation of the Nested Benders algorithm can be described as follows:

1. Initialization. Set $cs = 0, dir = forward, lb = -\infty, ub = \infty$. Set up thread-pool.
2. If absolute or relative gap between lb and ub is smaller than ϵ_{gap}, stop.
3. If $cs = 0$, set $dir = forward$. If $cs = T - 1$, set $dir = backward$
4. Iterate over all the nodes i of the current stage cs and add the task `HandleSubproblem(i)` to the thread-pool. Wait until all tasks are finished.
5. Call sequencing protocol to determine direction dir.
6. If no subproblem was infeasible and $dir == backward$, iterate over all the nodes i of the stage $cs-1$ and add the task `AggregateOptimalityCut(i)` to the thread-pool. Wait until all tasks are finished.
7. If $cs = 0$ and no subproblem was infeasible, update lb. If $cs = T - 1$ and no subproblem was infeasible, update ub, if possible.
8. If a subproblem was infeasible and $cs = 0$ stop, problem is infeasible. If $dir = forward$ set $cs = cs + 1$, else set $cs = cs - 1$. Go to step 1.

The lower bound lb is the objective function value of the first stage problem $c_0x_0 + \sum_{k=0}^{A_0^0} \theta_{0,k}^0$. The upper bound ub is the sum over the weighted objective function values of all nodes $\sum_{t=0}^{T-1} \sum_{i=0}^{K_t-1} p_t^i c_t^i x_t^i$, excluding the approximation terms. We use both an

absolute gap $ub - lb$ and a relative gap $|ub - lb|/(|ub| + 1 \cdot 10^{-10})$ as a stopping criterion. In the following we give a description of the subroutines used in the above pseudo-code.

The subroutine `HandleSubproblem(i)` consists of the following steps:

1. Build the model from stored data or reuse existing LP solver representation if possible.
2. Load warm start information into solver, if available.
3. Solve the subproblem.
4. If the subproblem is feasible, store warm start and necessary data and compute optimality cut coefficients.
5. If the subproblem is infeasible, and no feasibility cut was generated so far, generate a feasibility cut (14).
6. If $dir = forward$ call `CutConsolidation(i)`

The subroutine `AggregateOptimalityCut(i)` consists of the following steps:

1. For each partition of the descendants of node i
   (a) Initialize optimality cut coefficients with zero.
   (b) For each partition member
      i. Add the already computed optimality cut coefficients.
   (c) Add the computed optimality cut (15) to the node, if it is not redundant.

The subroutine `CutConsolidation(v)` consists of the following steps:

1. Gather data necessary for aggregation scheme.
2. Let CutConsolidation scheme decide which cuts to consolidate.
3. Consolidate cuts and remove old cuts from the problem while adding the newly aggregated ones.
4. Resolve the problem and store optimal basis.

4.3. Warm start strategies

The algorithm benefits from the capabilities of modern solvers to start the LP solution method from a given basis. Two possibilities exist. Either the optimal basis from the previous solve of the subproblem can be used as a warm start for the current solve. Or the optimal basis of another subproblem already solved within this iteration can be used as a warm start. Our experiments show that the superior warm start strategy is problem dependent. The default is to store a warm start for every subproblem.

4.4. LP solver strategies

State-of-the art LP solvers offer three algorithmic choices for the solution of the LP subproblems, namely the primal simplex algorithm, the dual simplex algorithm and an interior point method. The advantage of the simplex methods is the warm start capability, i.e. restarting the simplex algorithm from a previously stored basis. When cuts are added to a problem a previously optimal basis loses primal feasibility but remains dual feasible. Therefore only a small number of pivots may be needed by the dual simplex algorithm to reach a new optimal solution, compared with a solution process without a warm start. An interior point method usually solves problems faster than the simplex methods, but it has no warm-start capability and delivers non-basic solutions. Due

to the iterative nature of the Nested Benders algorithm that changes subproblems in each iteration by addition of new cuts or new right hand side values, the warm-start capability of the simplex solvers is of paramount importance. Due to this capability we did not explore other possibilities like bunching procedures which were implemented by Gassmann (1990) for example. We decided to use the dual simplex method, due to the mentioned advantages. A technical point in addition: the dual values that are needed for the cut generation depend on the algorithmic choice and on the use of a warm start basis, for problems with multiple optimal (and/or degenerate) dual solutions. Each method may reach different optimal dual solutions, so different cuts can be generated depending on the LP solution method, which then leads to different numbers of iterations of the algorithm.

5. Computational results

5.1. Setting and test models

In order to evaluate our algorithm, we assembled a test set of 48 two-stage and 42 multi-stage problems. An overview of the problems with their main characteristics is given in the appendix in Tables A.2 and A.3, respectively. We included problems from the following test sets (excluding all problems that solve in under 0.2 seconds by the deterministic equivalent solver to prevent outliers):

- the POSTS test set (Holmes, 1995) containing two- and multi-stage problems,
- the WATSON test set (Consigli & Dempster, 1998) containing multi-stage stochastic linear problems with up to 2688 scenarios and ten stages,
- the Slptestset collection (Ariyawansa & Felt, 2004),
- some random problems using the generator *genslp* (Kall & Mayer, 1998) and
- an integrated strategic production planning and financial hedging problem (Koberstein et al., 2012).

The problems from the Slptestset collection were also used in the solver study by Zverovich et al. (2010). The problems 20-term, gbd, LandS, storm and ssn are two-stage problems with a huge number of scenarios, up to $6 \cdot 10^{81}$. Linderoth et al. (2006)[2] solved these problems with a Sample Average Approximation (SAA) method on a cluster. Trukhanov et al. (2010) sampled versions of the 20-term, ssn and storm problems with $1000, 2000$ and 3000 scenarios each. We did the same for the sake of comparison. In addition we were able to solve gbd and LandS directly, as they only contain up to 1000000 scenarios. The random problems are two-stage stochastic linear programs with up to 10000 scenarios and a relatively dense matrix with up to 10% density. The problems taken from Koberstein et al. (2012) consider location, product allocation, capacity and financial hedging decisions under exchange rate uncertainty. The first-stage contains binary variables, all variables at later stages are continuous.

[2] `http://pages.cs.wisc.edu/~swright/stochastic/sampling/`

We conducted our test runs on an Intel i5-750 processor with four cores and 12 GB RAM, running on Windows 7 64-bit. If not mentioned otherwise, the algorithm uses all of the four cores. Cplex 12.3 64-bit was employed to solve the linear problems and deterministic equivalent problems. The interior point method (IPM) uses all four threads, whereas the dual simplex can only use one thread, so we chose IPM as the solver for the deterministic equivalent, without crossover. A time limit of three hours was imposed on all solution runs. We use one solver instance for every thread and an additional instance for the first stage problem.

We present wall clock solution times for different problems. Due to the possibility of setting several parameters, we define a base case algorithm (*BC*), which uses all cores available, single-cut aggregation, FFFB sequencing protocol, and no cut consolidation. The dual simplex method is used to solve all the subproblems. It uses the warm start capability and stores an optimal basis at every node. If no warm start basis is available for a problem, e.g. in the first iteration, the algorithm uses an optimal basis of another subproblem, if available. The remainder of this section is devided into two parts, the first is devoted to cut consolidation, the second to sequencing protocols.

5.2. Cut consolidation

In this section, we evaluate the cut consolidation strategy described in Section 3.2. In particular, we investigate its interaction with the hybrid algorithm devised by Trukhanov et al. (2010). For this purpose, we compare solution times of five different variants of our implementation of the Benders decomposition algorithm, namely, the base-case multi-cut algorithm (*Multi*), the base-case multi-cut algorithm with cut consolidation (*Multi-CC*), the base-case single-cut algorithm (*Single)*, the base-case algorithm with number of aggregates set to 10% of the number of childs of the root node (*Hybrid*) and the same but with cut consolidation (*Hybrid-CC*). As an additional information, we also include the deterministic equivalent solved with Cplex IPM (*DEQ*). All algorithms were run in parallel mode with four cores.

The results are visualized in Figure 1 as a performance profile[3]. It clearly shows that cut consolidation leads to a significant improvement in performance. Furthermore, the two hybrid methods outperform the single and multi-cut version of the algorithms, which confirms the results of Trukhanov et al. (2010). Among the hybrid variants the one with cut consolidation (*Hybrid-CC*) strictly dominates all other methods, i.e., it is able to solve the biggest fraction problems within a certain multiple τ of the best possible solution time for all meaningful τ. Surprisingly, on our testset, the single cut method (*Single*) outperforms both variants of the multi-cut method. We think, that this is in part due to the parallelization of the algorithm. The single cut method performs the greatest number of iterations compared to the other algorithms. Therefore more problems in the second

[3]Performance profiles were introduced by Dolan & Moré (2002) and allow for an easy visual comparison of different solution methods with regard to a given set of test problems. Method A outperforms method B if the A's graph is strictly above and left of B's graph. Formally, a performance profile is defined as the distribution function of a performance metric. We measured the wall clock solution time $t_{p,m}$ for every method $m \in M$ and problem $p \in P$. The minimal solution time $tmin_p = \min\{t_{p,m} : m \in M\}$ is used as a baseline for comparing the different methods via a performance ratio $r_{p,m} = \frac{t_{p,m}}{tmin_p}$. The distribution function for a method is then defined as $\rho_m(\tau) = \frac{|\{p \in P | r_{p,m} \leq \tau\}|}{|P|}$. Thus $\rho_m(\tau)$ denotes the percentage of problems method m can solve within a factor τ of the fastest method for these problems.

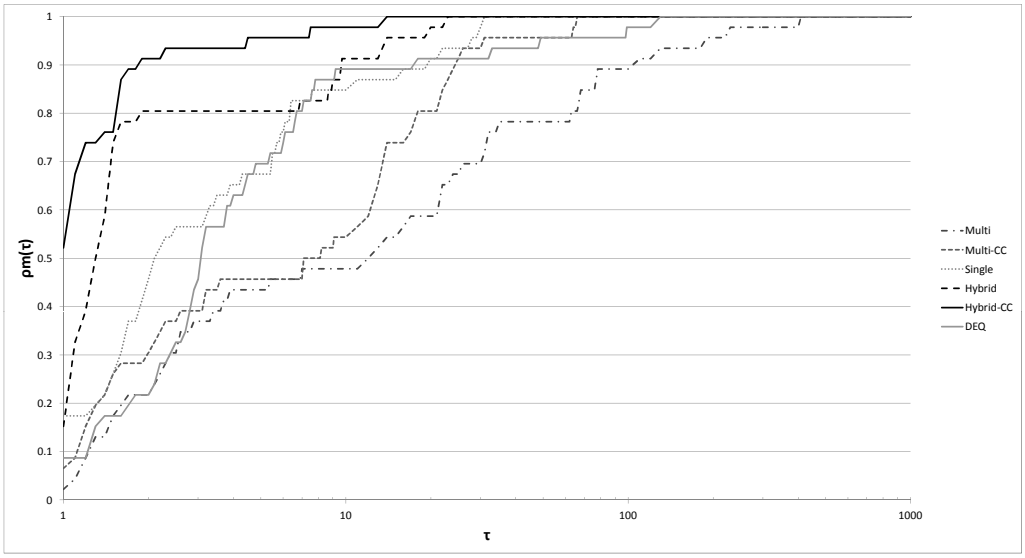

Figure 1: Performance Profile of the methods tested on our test set. Multi is the base-case multi-cut algorithm, Multi-CC is the base-case multi-cut algorithm with Cut Consolidation. Single is the base-case single-cut algorithm. Hybrid is the base-case algorithm with number of aggregates set to 10% of the number of childs of the root node. Hybrid-CC is the same but with Cut Consolidation. DEQ is the deterministic equivalent solved with Cplex IPM. All algorithms were run in parallel mode with four cores.

stage have to be solved, repeatedly. The speed-up of the second stage and cut generation is nearly linear. The single-cut method can therefore benefit more from parallelization than the multi-cut method. A detailed tabular presentation of the results is given in the appendix in Table A.4.

In order to investigate the interaction between cut aggregation and consolidation, we conducted further test runs with different fixed number of aggregates on a selected subset of the test problems. Detailed results are given in the appendix in Table A.5. Table A.6 displays results for the problem instance rand2_10000, where the effect of the number of aggregates is pronounced, due to the large number of scenarios. BC denotes the base case algorithm without and CC the version with cut consolidation. Again, it can be seen, that cut consolidation reduces the running times considerably in most cases. Furthermore, it can be observed that cut consolidation is particularly useful for many aggregates, but is already superior for aggregate sizes starting with 50. If a problem is solved with a lower number of aggregates, cuts may get aggregated that the algorithm would have used otherwise. The results show that the usage of cut consolidation eases the ex ante choice of the "right" number of aggregates. The running times for the problems with cut consolidation do not vary as much compared to the base case algorithm. Any choice between 100 and 500 for the problem 20_term_1000 (cf. Table A.5) for example leads to similar running times, whereas the same does not hold for the BC-algorithm.

Figure 2 further details the results of the CC-algorithm from Table A.6 in the appendix. It can be observed that a higher number of aggregates leads to fewer iterations, but not in general to a decrease in overall solution time. The time that is spent in the

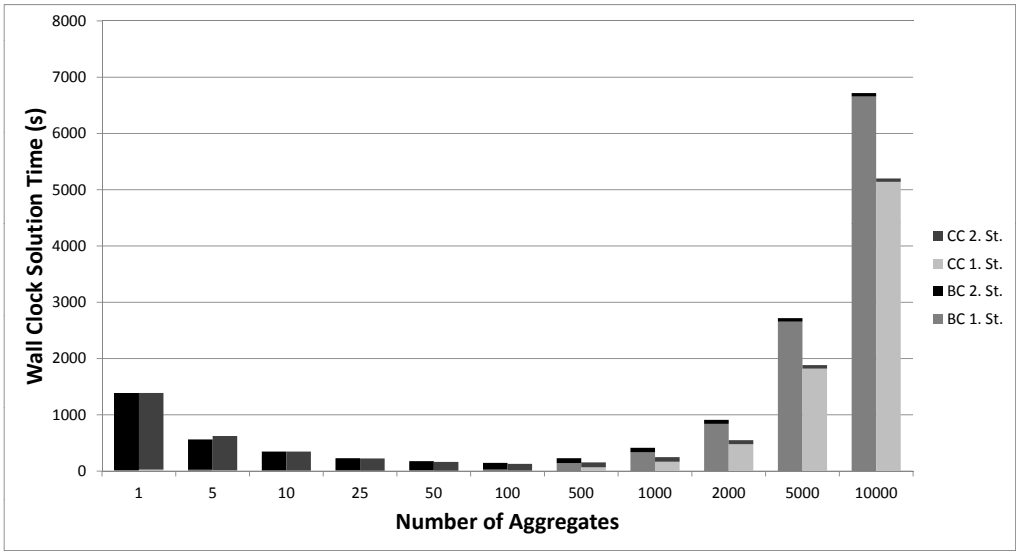

Figure 2: Computational results for rand2_10000 problem with different number of aggregates without and with CutConsolidation. The trade-off in terms of absolute time spent in solving first- and second-stage problems can be seen via the bars.

first stage master problem increases with the number of aggregates as it becomes harder to solve. The time that is spent in the second stage subproblems decreases with the number of aggregates, mostly because the number of iterations decreases. The parallelization effect mentioned above is also present in this case.

We investigate the impact of the threshold variable $ConsecInactive$ for the rand2_10000 problem in Figure 3. The efficacy of cut consolidation depends upon the removal of cuts that are of no more use to the algorithm. If cuts are removed late, the first-stage problem grows bigger and is harder to solve. If cuts are removed early, the algorithm needs more iterations overall. Cut consolidation is influenced by the number of aggregates the algorithm uses, as can be seen in Figure 3. For a low number of aggregates, cut consolidation has an adverse effect on the running time, that is more pronounced when the threshold is small. On the other hand, for a higher number of aggregates, a smaller threshold is advantageous.

5.3. Dynamic sequencing

In this section, we compare the sequencing strategy developed in Section 3.1 to other existing protocols. Most multi-stage problems in our test set only have up to sixteen child nodes, one problem has forty child nodes. As in preliminary tests, the use of scenario aggregation did not influence the solution time much on our test set, we set the number of aggregates to only 10% of the average number of child nodes in the scenario tree (but at least to 1).

Figure 4 shows the performance profile with regards to the 42 multi-stage problems in our test set. The dynamic protocol significantly outperforms all other protocols. To be more specific, it is the fastest protocol on half of the problems and it is able to solve

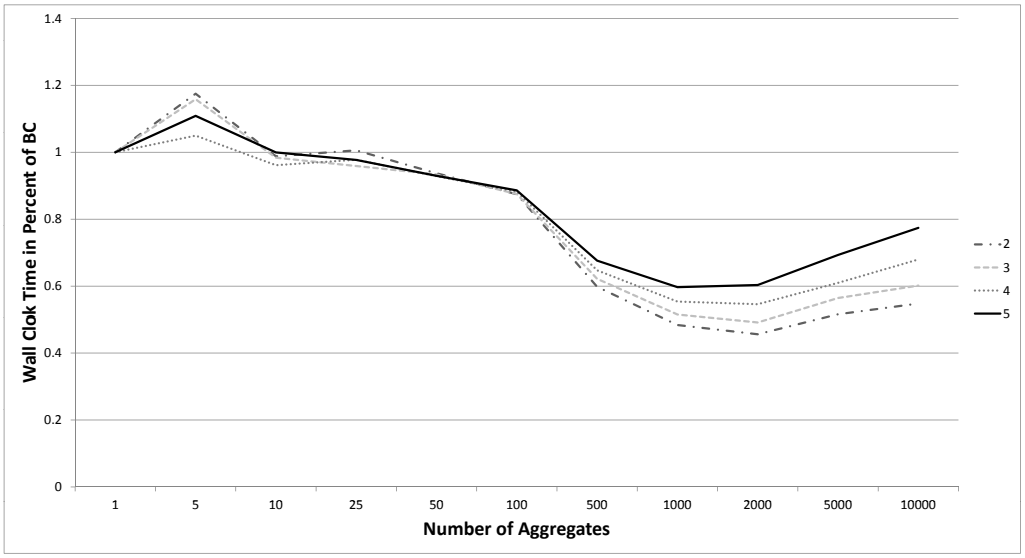

Figure 3: Impact of the choice of $ConsecInactive$ on running time. The y-axis shows relative wall clock solution times in percent of the base-case algorithm without cut-consolidation. The different graphs represent present different choices for $ConsecInactive$.

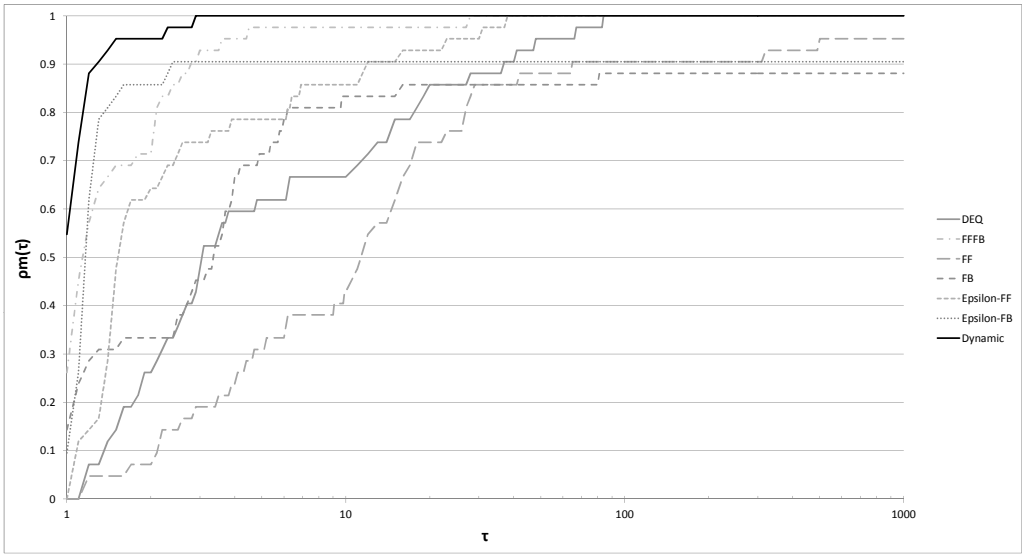

Figure 4: Performance Profile of the different sequencing protocls tested on our test set. The algorithm is the base-case algorithm with number of aggregates chosen to 10% of the number of child nodes, only the sequencing protocols differs. Dynamic is our new sequencing protocol. FFFB is the FastForward-FastBack protocol, FF is FastForward, Epsilon-FF is $\epsilon-$FastForward, FB is FastBack, Epsilon-FB is the $\epsilon-$FastBack protocol. ϵ was chosen to be 0.064. For comparison, DEQ is the deterministic equivalent solved with Cplex IPM. All algorithms were run in parallel mode with four cores.

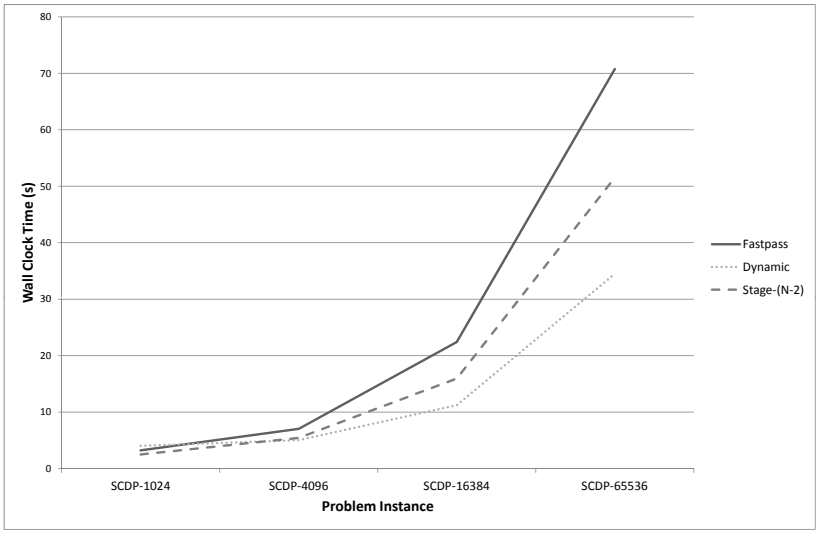

Figure 5: Wall Clock Solution Times for different sequencing protocols and different SCDP problem instances. Stage-(N-2) refers to the static bouncing protocol with bouncing stage chosen to number of stages for the problem instance minus two.

all problems within a τ of three. Overall the FastForwardFastBack protocol is the second best protocol, but in the worst case it is twelve time slower than the best protocal (τ=12). $\epsilon-$FF is also able to solve all problems, but is slower than the other two protocols. FB and $\epsilon-$FB are both unable to solve the financial hedging problems (Koberstein et al., 2012) in our test suite. The ϵ-variants are superior to their non-epsilon variants. The DEQ solution times show, that the usage of a specialized solution method for multi-stage stochastic programs is preferable to a general purpose solver. Detailed results on all 42 problems are listed in the appendix in Table A.7.

In Figure 5 we analyse the impact of the dynamic sequencing protocoll for different numbers of scenarios on the strategic network design problems with integrated financial hedging (Koberstein et al., 2012). In this figure, we present solution times only for the single-cut method, as it was the best aggregate choice for these problems. The use of the dynamic sequencing protocol (*Dynamic*) leads to a significant reduction in solution time for all problem instances, except for the smallest problem with 1024 scenarios and five stages compared with the FFFB method. The $\epsilon-$FB and FB protocol can not be applied to this problem, as the upper bound becomes zero during the solution process which leads to a not-ending algorithm, see Section 3 for a further discussion. With growing number of scenarios, the predominance of the dynamic protocol increases considerably.

5.4. Overall assessment

We compared our algorithm against the deterministic equivalent problem, where the root node is solved with the Cplex barrier method. The dual simplex is not competitive on our test set, as it is not parallelized and can thus not take advantage of the four cores. Selected results are presented in Table 1. Only the instances of one problem class, the 20_term problems, are solved faster with Cplex barrier than with the Parallel Nested

Table 1: Wall clock solution times for the deterministic equivalent solved with Cplex barrier (using all cores) and the best solution time for parallel Nested Benders (PNB) with dynamic sequencing protocol, problem dependent number of scenario aggregates and Cut Consolidation.

Problem	DEQ (s)	Best PNB (s)	PNB/DEQ (%)
storm-1000	26.35	4.51	17
storm-2000	64.08	9.58	15
storm-3000	135.77	14.16	10
ssn-1000	33.15	12.25	37
ssn-2000	83.86	28.11	34
ssn-3000	135.68	41.44	31
20_term-1000	11.53	62.25	542
20_term-2000	23.61	128.77	545
20_term-3000	40.30	191.11	474
SCDP-1024	3.49	4.03	115
SCDP-4096	34.75	4.44	13
SCDP-16384	398.36	11.22	3
SCDP-65536	5335.34	34.51	1
SCDP-64000	578.97	9.28	2
pltexpA5_16	752.24	8.87	1
rand2_10000	648.80	130.35	20

Benders (PNB) with the Dynamic sequencing protocol and Cut Consolidation enabled. The most significant advantage achieves PNB when solving the mixed-binary first stage SCDP problems. These results confirm that the use of specialized solution methods for stochastic linear programs with recourse is justified for two-stage problems (Zverovich et al., 2010) and for multi-stage problems as opposed to the use of the deterministic equivalent. This holds true especially for problems with a huge number of scenarios that can not be solved via the deterministic equivalent because of memory constraints.

6. Conclusions and further work

In this paper we presented and evaluated a cut consolidation technique and a dynamic sequencing protocol to speed up the solution of two- and multi-stage stochastic programming problems via solution algorithms based on Benders' decomposition. We showed on a test set of 47 two-stage and 42 multi-stage problems, that both of these techniques lead to significant speed ups in computation time. Contrary to what the literature suggested (Birge & Louveaux, 1997), FastForwardFastBack is not the fastest protocol. The removal of optimality cuts from the problem makes the right choice of an aggregation level less critical. It also speeds up the solution process due to smaller subproblems, contrary to what was previously observed (Trukhanov et al., 2010). In addition it reduces memory requirements. The acceleration techniques presented here can also be useful for sampling methods such as SAA (Kleywegt et al., 2002), as they involve the solution of multi-stage stochastic programs. The effect of cut aggregation on other methods that are based upon Benders decomposition, e.g. regularized decomposition (Ruszczyński, 1986), trust region (Linderoth & Wright, 2003) or level decomposition (Fábián & Szőke, 2006), should also

be investigated in the future. It is not yet clear how to choose an ideal number of aggregates for a specific problem a priori. Also the partitioning of the descendant nodes into aggregates can be guided via more sophisticated methods than the static partitioning we used.

Appendix A.

Table A.2: Test set: two-stage problems

Instanz	Scenarios	Cols	Rows	Cols 2nd st.	Rows 2nd st.	Cols DEQ	Rows DEQ	NZ DEQ
4node-256 [4]	256	52	14	186	74	47,668	18,958	120,063
4node-512 [4]	512	52	14	186	74	95,284	37,902	239,871
4node-1024 [4]	1,024	52	14	186	74	190,516	75,790	479,487
4node-2048 [4]	2,048	52	14	186	74	380,980	151,566	958,719
4node-4096 [4]	4,096	52	14	186	74	761,908	303,118	1,917,183
4node-8192 [4]	8,192	52	14	186	74	1,523,764	606,222	3,834,111
4node-16384 [4]	16,384	52	14	186	74	3,047,476	1,212,430	7,667,967
4node-32768 [4]	32,768	52	14	186	74	6,094,900	2,424,846	15,335,679
env-1200 [4]	1,200	49	48	49	48	58,849	57,648	172,932
env-1875 [4]	1,875	49	48	49	48	91,924	90,048	270,132
env-3780 [4]	3,780	49	48	49	48	185,269	181,488	544,452
env-5292 [4]	5,292	49	48	49	48	259,357	254,064	762,180
env-loose [4]	5	49	48	49	48	294	288	852
env-lrge [4]	8,232	49	48	49	48	403,417	395,184	1,185,540
env-xlrge [4]	32,928	49	48	49	48	1,613,521	1,580,592	4,741,764
phone [4]	32,768	8	1	85	23	2,785,288	753,665	9,863,176
phone-int [4]	32,768	8	1	85	23	2,785,288	753,665	9,863,176
stormG2_27 [5]	27	121	185	1,259	528	34,114	14,441	90,903
stormG2_125 [5]	125	121	185	1,259	528	157,496	66,185	418,321
stormG2_1000 [5]	1,000	121	185	1,259	528	1,259,121	528,185	3,341,696
20term-1000 [6]	1,000	63	3	764	124	764,063	124,003	4,488,063
20term-2000 [6]	2,000	63	3	764	124	1,528,063	248,003	8,976,063
20term-3000 [6]	3,000	63	3	764	124	2,292,063	372,003	13,464,063
SSN-1000 [6]	1,000	89	1	706	175	706,089	175,001	2,373,089
SSN-2000 [6]	2,000	89	1	706	175	1,412,089	350,001	4,746,089
SSN-3000 [6]	3,000	89	1	706	175	2,118,089	525,001	7,119,089
storm-1000 [6]	1,000	121	185	1,259	528	1,259,121	528,185	3,341,696
storm-2000 [6]	2,000	121	185	1,259	528	2,518,121	1,056,185	6,682,696

[4] Ariyawansa & Felt (2004)
[5] Holmes (1995)
[6] Linderoth et al. (2006)
[7] Kall & Mayer (1998)
[8] Koberstein et al. (2011)

Table A.2: (continued)

Problem	Scenarios	Cols	Rows	Cols 2nd st.	Rows 2nd st.	Cols DEQ	Rows DEQ	NZ DEQ
storm-3000 [6]	3,000	121	185	1,259	528	3,777,121	1,584,185	10,023,696
rand0_2000 [7]	2,000	100	50	50	25	100,100	50,050	754,501
rand0_4000 [7]	4,000	100	50	50	25	200,100	100,050	1,508,501
rand0_6000 [7]	6,000	100	50	50	25	300,100	150,050	2,262,501
rand0_8000 [7]	8,000	100	50	50	25	400,100	200,050	3,016,501
rand0_10000 [7]	10,000	100	50	50	25	500,100	250,050	3,770,501
rand1_2000 [7]	2,000	200	100	100	50	200,200	100,100	3,006,001
rand1_4000 [7]	4,000	200	100	100	50	400,200	200,100	6,010,001
rand1_6000 [7]	6,000	200	100	100	50	600,200	300,100	9,014,001
rand1_8000 [7]	8,000	200	100	100	50	800,200	400,100	12,018,001
rand1_10000 [7]	10,000	200	100	100	50	1,000,200	500,100	15,022,001
rand2_2000 [7]	2,000	300	150	150	75	300,300	150,150	6,758,501
rand2_4000 [7]	4,000	300	150	150	75	600,300	300,150	13,512,501
rand2_6000 [7]	6,000	300	150	150	75	900,300	450,150	20,266,501
saphir_1000 [8]	1,000	53	32	3,924	8,678	3,924,053	8,678,032	22,733,103
saphir_500 [8]	500	53	32	3,924	8,678	1,962,053	4,339,032	11,366,603
saphir_100 [8]	100	53	32	3,924	8,678	392,453	867,832	2,273,403
saphir_50 [8]	50	53	32	3,924	8,678	196,253	433,932	1,136,753

Table A.3: Test set: multi-stage problems. †denotes instances where the scenario tree is not symmetric.

Problem	Scen St	Scen	St	Bin/ 1st St	Cols	Rows	Cols DEQ	Rows DEQ	NZ DEQ
sgpf3y5 [9]	5	625	3	0	87	38	39,867	30,458	103,090
sgpf3y6 [9]	5	3,125	4	0	87	38	199,242	152,333	515,590
sgpf3y7 [9]	5	15,625	5	0	87	38	996,117	761,708	2,578,090
sgpf5y5 [9]	5	625	5	0	139	62	61,759	49,202	165,570
sgpf5y6 [9]	5	3,125	6	0	139	62	308,634	246,077	828,070
sgpf5y7 [9]	5	15,625	7	0	139	62	1,543,009	1,230,452	4,140,570
pltexpA3_16 [10]	16	256	3	0	188	62	74,172	28,350	150,801
pltexpA4_6 [10]	6	216	4	0	188	62	70,364	26,894	143,059
pltexpA4_16 [10]	16	4,096	4	0	188	62	1,188,284	454,334	2,415,889
pltexpA5_6 [10]	6	1,296	5	0	188	62	422,876	161,678	859,747

[9] Ariyawansa & Felt (2004)
[10] Holmes (1995)
[11] Consigli & Dempster (1998)
[12] Koberstein et al. (2012)

Table A.3: (continued)

Problem	Scen St	Scen	St	Bin/ 1st St	Cols	Rows	Cols DEQ	Rows DEQ	NZ DEQ
pltexpA5_16 [10]	16	65,536	5	0	188	62	19,014,076	7,270,078	38,657,297
pltexpA6_6 [10]	6	7,776	6	0	188	62	2,537,948	970,382	5,159,875
pltexpA7_6 [10]	6	279,936	7	0	188	62	15,228,380	5,822,606	30,960,643
SCDP-1024 [12]	4	1,024	6	6	95	49	55,939	41,397	248,801
SCDP-4096 [12]	4	4,096	7	6	95	49	223,811	165,621	1,000,929
SCDP-16384 [12]	4	16,384	8	6	95	49	895,299	662,517	4,009,441
SCDP-65536 [12]	4	65,536	9	6	95	49	3,581,251	2,650,101	16,043,489
SCDP-64000 [12]	40	64,000	4	6	83	45	2,448,923	1,910,325	10,574,919
WAT_C_10_64 [11]	†	64	10	0	15	11	28,097	15,101	72,648
WAT_C_10_128 [11]	†	128	10	0	15	11	49,153	26,237	128,648
WAT_C_10_256 [11]	†	256	10	0	15	11	82,177	43,517	218,888
WAT_C_10_512 [11]	†	512	10	0	15	11	128,001	67,069	350,728
WAT_C_10_768 [11]	†	768	10	0	15	11	191,994	100,598	526,078
WAT_C_10_1024 [11]	†	1,024	10	0	15	11	255,987	134,127	701,428
WAT_C_10_1152 [11]	†	1,152	10	0	15	11	287,949	150,869	789,028
WAT_C_10_1536 [11]	†	1,536	10	0	15	11	383,927	201,155	1,052,028
WAT_C_10_1920 [11]	†	1,920	10	0	15	11	479,905	251,441	1,315,028
WAT_C_10_2304 [11]	†	2,304	10	0	15	11	575,883	301,727	1,578,028
WAT_C_10_2688 [11]	†	2,688	10	0	15	11	671,861	352,013	1,841,028
WAT_I_10_64 [11]	†	64	10	0	15	11	28,097	15,101	72,648
WAT_I_10_128 [11]	†	128	10	0	15	11	49,153	26,237	128,648
WAT_I_10_256 [11]	†	256	10	0	15	11	82,177	43,517	218,888
WAT_I_10_512 [11]	†	512	10	0	15	11	128,001	67,069	350,728
WAT_I_10_768 [11]	†	768	10	0	15	11	191,994	100,598	526,078
WAT_I_10_1024 [11]	†	1,024	10	0	15	11	255,987	134,127	701,428
WAT_I_10_1152 [11]	†	1,152	10	0	15	11	287,949	150,869	789,028
WAT_I_10_1536 [11]	†	1,536	10	0	15	11	383,927	201,155	1,052,028
WAT_I_10_1920 [11]	†	1,920	10	0	15	11	479,905	251,441	1,315,028
fxm3_6 [10]	6	36	3	0	114	92	9,492	6,200	54,589
fxm3_16 [10]	16	256	3	0	114	92	64,162	41,340	370,839
fxm4_6 [10]	6	216	4	0	114	92	30,732	22,400	248,989
fxm4_16 [10]	16	4,096	4	0	114	92	517,282	386,940	4,518,039

Table A.4: Evaluation of cut consolidation on two-stage problems.

Problem	Multi	Multi-CC	Single	Hybrid	Hybrid-CC	DEQ
4node-256	0.91	0.75	1.37	0.60	0.56	0.70
4node-512	1.41	1.25	2.68	1.48	1.23	1.51
4node-1024	4.84	2.85	6.61	1.93	3.09	3.39
4node-2048	9.98	6.94	15.02	4.80	5.36	8.08
4node-4096	32.29	19.02	33.07	6.04	6.16	18.97

Table A.4: (continued)

Problem	Multi	Multi-CC	Single	Hybrid	Hybrid-CC	DEQ
4node-8192	45.70	41.70	75.05	12.15	11.85	35.58
4node-16384	731.92	294.63	162.85	23.52	21.48	68.47
4node-32768	4244.80	2071.07	371.82	84.09	68.38	86.31
env-1200	29.06	3.08	0.44	3.00	0.66	7.51
env-1875	91.11	10.00	0.90	7.78	1.25	3.37
env-3780	196.27	21.94	1.61	15.60	2.44	9.65
env-5292	426.89	38.09	2.25	31.20	3.46	14.67
env-lrge	774.39	78.54	3.38	64.20	6.09	420.64
env-xlrge	5146.58	813.10	12.78	285.04	28.51	1253.88
env-first	0.23	0.18	0.16	0.16	0.16	7.56
phone	7.36	7.41	4.09	3.31	3.32	25.55
phone-int	47.54	47.72	6.80	6.93	6.87	224.27
stormG2_27	0.16	0.18	0.25	0.21	0.24	1.10
stormG2_125	0.63	0.63	1.03	0.51	0.53	2.73
stormG2_1000	8.31	7.89	8.17	3.59	3.59	27.17
20term-1000	343.47	92.31	241.98	102.06	153.20	11.35
20term-2000	1836.42	284.58	448.67	225.78	174.92	23.61
20term-3000	3079.14	550.19	612.13	563.05	178.39	40.30
SSN-1000	24.44	16.20	500.30	22.44	19.05	33.15
SSN-2000	83.43	49.73	967.30	45.50	32.30	86.86
SSN-3000	162.42	95.89	1341.00	72.13	48.34	135.68
storm-1000	9.52	8.82	8.70	4.56	4.38	26.35
storm-2000	26.96	24.69	15.90	10.03	9.62	64.08
storm-3000	53.84	46.90	23.46	15.85	14.77	135.77
rand0_2000	24.03	23.80	4.46	2.40	2.18	10.32
rand0_4000	68.60	68.69	7.40	5.72	5.01	22.20
rand0_6000	705.30	687.44	17.37	12.62	10.42	41.45
rand0_8000	388.46	381.43	24.81	22.60	18.13	42.69
rand0_10000	716.77	692.26	56.88	42.71	30.74	66.92
rand1_2000	175.78	140.19	71.38	14.66	11.27	31.21
rand1_4000	587.00	476.14	163.67	39.83	26.95	73.38
rand1_6000	1157.05	951.99	194.64	66.60	45.29	139.62
rand1_8000	2376.99	1916.89	286.94	112.45	75.19	183.18
rand1_10000	3415.44	2677.04	361.09	171.22	109.68	476.45
rand2_2000	460.74	334.45	396.74	56.21	36.91	79.05
rand2_4000	1044.44	785.92	360.98	93.02	63.40	187.41
rand2_6000	2341.80	1825.06	623.22	159.32	106.79	304.90
rand2_8000	3609.11	2847.64	752.73	226.33	151.48	463.59
rand2_10000	6718.72	5200.79	1389.76	417.33	248.80	648.84
saphir-1000	319.59	336.58	483.48	313.53	262.19	975.46
saphir-500	139.83	131.31	202.67	147.35	139.69	406.82
saphir-100	37.30	31.18	45.24	40.35	34.55	42.05
saphir-50	23.59	18.36	32.19	30.55	25.10	16.44

Table A.4: (continued)

Problem	Multi	Multi-CC	Single	Hybrid	Hybrid-CC	DEQ
Sum	41730.51	24163.42	10345.41	3592.33	2213.97	6990.21
Arithmetic mean	869.39	503.40	215.53	74.84	46.12	145.63
Geometric mean	108.43	62.54	33.24	19.69	13.14	40.82

Table A.5: Comparison of hybrid Benders' algorithm with (CC) and without (BC) cut consolidation. Wall clock solution times in seconds with different number of aggregates.

Problem	# Agg.	BC	# It.	CC	# It.	CC/BC(%)
20_term-1000	1	250.22	1727	250.22	1727	100
20_term-1000	5	198.38	1036	222.68	1279	112
20_term-1000	10	157.24	792	181.49	1010	115
20_term-1000	25	163.31	629	139.92	729	86
20_term-1000	50	117.51	412	111.63	535	95
20_term-1000	100	103.15	287	75.07	333	73
20_term-1000	200	122.55	219	62.25	231	51
20_term-1000	500	187.79	144	74.49	166	40
20_term-1000	1000	343.66	99	86.24	97	25
20_term-2000	1	459.07	1751	460.16	1751	100
20_term-2000	5	395.47	1154	405.76	1244	103
20_term-2000	10	338.62	810	245.68	870	73
20_term-2000	25	291.59	664	255.62	757	88
20_term-2000	50	178.96	419	188.20	531	105
20_term-2000	100	220.29	363	157.21	396	71
20_term-2000	200	236.53	262	170.70	343	72
20_term-2000	500	346.15	183	128.77	183	37
20_term-2000	1000	611.31	127	162.44	139	27
20_term-2000	2000	1969.59	111	426.85	113	22
20_term-3000	1	706.44	1752	705.46	1745	100
20_term-3000	5	442.52	995	567.99	1243	128
20_term-3000	10	466.60	908	514.54	1093	110
20_term-3000	25	453.16	744	449.73	957	99
20_term-3000	50	266.91	465	361.31	711	135
20_term-3000	100	356.78	424	310.10	563	87
20_term-3000	200	298.13	309	223.70	360	75
20_term-3000	500	450.20	208	191.11	224	42
20_term-3000	1000	849.95	167	243.78	168	29
20_term-3000	2000	1330.06	139	368.68	152	28
20_term-3000	3000	6137.71	117	685.65	102	11
ssn-1000	1	503.82	2877	503.72	2877	100

Table A.5: (continued)

Problem	# Agg.	BC	# It.	CC	# It.	CC/BC(%)
ssn-1000	5	178.05	767	234.20	1024	132
ssn-1000	10	102.25	396	102.76	442	100
ssn-1000	25	54.67	184	48.67	191	89
ssn-1000	50	33.06	104	27.99	99	85
ssn-1000	100	22.74	62	19.88	68	87
ssn-1000	200	18.82	45	13.53	44	72
ssn-1000	500	18.28	29	12.25	30	67
ssn-1000	1000	24.29	24	16.80	24	69
ssn-2000	1	952.84	2897	953.45	2897	101
ssn-2000	5	397.62	939	409.86	1012	103
ssn-2000	10	248.03	524	236.84	543	93
ssn-2000	25	124.67	233	129.57	257	104
ssn-2000	50	81.98	138	73.17	137	89
ssn-2000	100	53.58	81	47.59	84	89
ssn-2000	200	45.61	56	33.40	54	73
ssn-2000	500	41.95	36	28.11	39	67
ssn-2000	1000	51.03	30	28.86	27	57
ssn-2000	2000	85.61	24	55.14	24	64
ssn-3000	1	1341.18	2899	1342.64	2899	100
ssn-3000	5	581.09	938	615.00	1020	106
ssn-3000	10	385.22	565	399.90	606	104
ssn-3000	25	203.24	261	203.78	284	100
ssn-3000	50	132.17	158	122.59	159	93
ssn-3000	100	93.24	98	78.46	97	84
ssn-3000	200	71.22	64	57.74	68	81
ssn-3000	500	67.49	42	41.44	41	61
ssn-3000	1000	80.13	33	48.61	34	61
ssn-3000	2000	97.13	28	58.44	28	60
ssn-3000	3000	162.66	25	107.63	24	66
storm-1000	1	8.95	59	8.95	59	100
storm-1000	5	7.15	44	6.95	42	97
storm-1000	10	6.14	36	6.18	36	101
storm-1000	25	5.12	29	5.25	30	103
storm-1000	50	4.79	25	4.69	25	98
storm-1000	100	4.59	22	4.51	22	98
storm-1000	200	4.97	20	4.70	20	95
storm-1000	500	6.62	17	5.91	17	89
storm-1000	1000	9.70	14	8.75	14	90
storm-2000	1	16.29	52	16.35	52	100
storm-2000	5	15.32	48	14.67	45	96
storm-2000	10	14.13	44	13.63	42	97

Table A.6: Wall Clock solution times and iteration numbers for problem rand2_10000 with different number of aggregates. The time is split between first and second stage wall clock solution time. BC is the algorithm without cut consolidation, CC the algorithm with cut consolidation.

#Agg.	BC(s)	1.St.(s)	2.St.(s)	#It.	CC(s)	1.St.(s)	2.St.(s)	#It.
1	1386.69	27.02	1361.26	995	1386.69	27.02	1361.26	995
5	563.80	23.06	539.34	360	625.12	12.98	610.76	422
10	348.03	15.58	331.07	210	347.89	8.75	337.75	213
25	230.92	15.99	213.55	117	225.65	7.90	216.37	119
50	177.98	18.61	158.00	77	165.46	7.54	156.54	77
100	147.26	26.27	119.62	53	130.50	9.61	119.50	52
500	231.15	141.74	88.01	31	156.30	67.09	87.84	32
1000	417.33	334.49	80.26	26	249.12	167.08	80.65	26
2000	912.34	839.81	71.15	22	550.51	478.09	71.06	22
5000	2720.01	2655.22	63.40	17	1885.22	1820.67	63.17	18
10000	6718.72	6656.01	61.33	15	5202.05	5139.82	60.86	15

Table A.5: (continued)

Problem	# Agg.	BC	# It.	CC	# It.	CC/BC(%)
storm-2000	25	10.99	31	11.05	31	101
storm-2000	50	10.67	30	10.63	30	100
storm-2000	100	10.57	27	10.67	28	101
storm-2000	200	10.16	23	9.58	23	94
storm-2000	500	12.58	20	11.17	20	89
storm-2000	1000	15.19	16	13.37	16	88
storm-2000	2000	26.97	14	23.53	14	87
storm-3000	1	23.50	53	23.50	53	100
storm-3000	5	21.25	47	23.55	53	111
storm-3000	10	19.11	42	19.54	42	102
storm-3000	25	17.92	38	17.50	37	98
storm-3000	50	16.20	33	16.13	33	100
storm-3000	100	14.82	28	15.06	29	102
storm-3000	200	14.85	25	14.66	26	99
storm-3000	500	15.61	19	14.16	19	91
storm-3000	1000	22.28	18	19.24	18	86
storm-3000	2000	29.08	16	26.64	16	92
storm-3000	3000	53.73	14	44.59	14	83

Table A.7: Comparison of different sequencing protocols for the Parallel Nested Benders decomposition. Wall clock solution times in seconds. ‡ denotes a solution run that took longer than the time limit of three hours. † denotes a solution run that took more than the maximal number of iterations (=5000).

Problem	**DEQ**	**FFFB**	**FF**	**FB**	**ϵ-FF**	**ϵ-FB**	**Dynamic**
sgpf3y5	0.22	0.09	0.22	0.08	0.11	0.09	0.08
sgpf3y6	1.23	0.55	1.45	0.44	0.63	0.42	0.53
sgpf3y7	41.42	2.17	13.38	174.71	3.27	2.36	2.17
sgpf5y5	0.30	0.07	0.10	0.07	0.10	0.06	0.07
sgpf5y6	3.24	0.35	0.59	0.27	0.55	0.32	0.37
sgpf5y7	19.59	1.36	2.90	1.39	2.90	1.33	1.51
fxm3_6	0.23	0.20	0.23	0.21	0.21	0.21	0.20
fxm3_16	1.24	0.92	1.00	0.90	0.90	0.92	0.91
fxm4_6	0.67	0.26	0.37	0.18	0.26	0.23	0.21
fxm4_16	17.80	26.15	2.49	0.96	2.47	2.13	1.44
pltexpA3_16	0.97	0.14	0.33	0.07	0.13	0.06	0.05
pltexpA4_6	0.59	0.19	3.89	0.14	0.58	0.10	0.09
pltexpA4_16	26.26	1.97	36.15	0.56	6.65	0.60	0.59
pltexpA5_6	5.90	0.83	124.28	6.18	6.05	0.40	0.90
pltexpA5_16	740.46	39.70	4373.22	10.53	202.32	10.02	8.83
pltexpA6_6	49.51	5.30	9263.15	17.47	55.16	2.02	1.81
pltexpA7_6	403.93	30.20	‡	†	411.36	12.01	11.06
WAT_C_10_64	0.41	0.82	10.13	1.86	1.33	0.84	0.35
WAT_C_10_128	0.74	0.35	3.89	0.95	0.52	0.52	0.40
WAT_C_10_256	1.15	0.63	7.77	1.67	0.88	0.73	0.65
WAT_C_10_512	1.87	0.30	2.75	1.81	0.36	0.37	0.86
WAT_C_10_768	2.96	1.32	18.88	3.28	1.90	1.53	1.32
WAT_C_10_1024	4.52	1.82	40.88	6.09	2.43	1.86	1.83
WAT_C_10_1152	5.55	1.81	32.32	4.46	2.78	2.00	1.97
WAT_C_10_1536	7.52	2.54	26.55	8.41	3.10	2.93	2.59
WAT_C_10_1920	10.46	2.98	81.29	10.62	4.25	3.74	3.53
WAT_C_10_2304	12.96	4.70	111.78	17.64	6.06	5.05	4.22
WAT_C_10_2688	14.75	4.63	111.81	17.03	6.18	5.35	4.26
WAT_I_10_64	0.41	0.28	2.78	1.73	0.37	0.42	0.31
WAT_I_10_128	0.78	0.49	7.58	2.45	0.69	0.51	0.43
WAT_I_10_256	1.12	0.75	8.40	2.85	1.10	0.91	0.73
WAT_I_10_512	2.01	0.98	14.64	3.06	1.28	1.14	0.96
WAT_I_10_768	3.43	1.45	19.75	4.74	1.84	1.64	1.31
WAT_I_10_1024	4.61	2.22	29.89	8.67	2.55	2.30	1.79
WAT_I_10_1152	5.55	4.75	18.49	13.44	4.88	5.39	5.07
WAT_I_10_1536	7.67	6.16	29.55	21.04	6.12	6.66	5.77
WAT_I_10_1920	10.18	6.91	28.73	25.36	7.10	7.42	6.58
scdp-1024	3.49	2.63	8.10	†	4.47	†	2.01

Table A.7: (continued)

Problem	DEQ	FFFB	FF	FB	ϵ-FF	ϵ-FB	Dynamic
scdp-4096	34.75	6.33	37.97	†	20.08	†	3.17
scdp-16384	398.36	21.34	101.00	†	68.54	†	9.95
scdp-65536	5335.34	66.69	474.86	†	362.31	†	31.85
scdp-64000	578.97	15.34	40.92	9.02	28.50	10.93	8.77
Sum[13]	1587.23	141.48	14472.53	380.34	366.51	83.51	73.46
Arithmetic mean[13]	42.90	3.82	391.15	10.28	9.91	2.26	1.99
Geometric mean[13]	3.85	1.22	10.19	2.33	1.79	1.03	0.95

References

Altenstedt, F. (2003). *Aspects on asset liability management via stochastic programming*. Ph.D. thesis Chalmers University of Technology and Göteborg University.

Aranburu, L., Escudero, L., & Garín, M. (2011). A so-called Cluster Benders Decomposition approach for solving two-stage stochastic linear problems. *TOP*, (pp. 1–17).

Ariyawansa, K., & Felt, A. (2004). On a new collection of stochastic linear programming test problems. *INFORMS Journal on Computing*, *16*, 291–299.

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

Birge, J. R. (1985). Decomposition and Partitioning Methods for Multistage Stochastic Linear Programs. *Operations Research*, *33*, 989–1007.

Birge, J. R., Dempster, M., Gassmann, H. I., Gunn, E., King, A. J., & Wallace, S. W. (1987). A standard input format for multiperiod stochastic linear programs. *COAL newsletter*, *17*, 1–19.

Birge, J. R., Donohue, C. J., Holmes, D. F., & Svintsitski, O. G. (1996). A parallel implementation of the nested decomposition algorithm for multistage stochastic linear programs. *Mathematical Programming*, *75*, 327–352.

Birge, J. R., & Louveaux, F. V. (1988). A multicut algorithm for two-stage stochastic linear programs. *European Journal of Operational Research*, *34*, 384–392.

Birge, J. R., & Louveaux, F. V. (1997). *Introduction to Stochastic Programming*. Springer Verlag.

Consigli, G., & Dempster, M. (1998). Dynamic Stochastic Programming For Asset-liability Management. *SSRN Electronic Journal*, .

Dempster, M., & Consigli, G. (1998). Dynamic stochastic programming for asset-liability management. *Annals of Operations Research*, *81*, 131 – 161.

Dempster, M., & Thompson, R. (1998). Parallelization and Aggregation of Nested Benders Decomposition. *Annals of Operations Research*, *81*, 163–188.

Dempster, M., & Thompson, R. (1999). EVPI-based importance sampling solution proceduresfor multistage stochastic linear programmeson parallel MIMD architectures. *Annals of Operations Research*, *90*, 161–184.

Dolan, E. D., & Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, *91*, 201–213.

Fábián, C. I., & Szőke, Z. (2006). Solving two-stage stochastic programming problems with level decomposition. *Computational Management Science*, *4*, 313–353.

Gassmann, H. I. (1990). MSLiP: A computer code for the multistage stochastic linear programming problem. *Mathematical Programming*, *47*, 407–423.

Holmes, D. (1995). A (PO)rtable (S)tochastic programming (T)est (S)et (POSTS).

Kall, P., & Mayer, J. (1998). On testing SLP codes with SLP-IOR. *New Trends in Mathematical Programming: Homage to Steven Vajda*, (pp. 115–135).

Kall, P., & Mayer, J. (2010). *Stochastic Linear Programming: Models, Theory, and Computation*. Springer.

[13] problems pltexpA7_6, scdp-1024, scdp-4096, scdp-16384 and scdp-65536 excluded

Kleywegt, A., Shapiro, A., & Homem-de Mello, T. (2002). The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, *12*, 479–502.

Koberstein, A., Lucas, C., Wolf, C., & König, D. (2011). Modelling and optimising risk in the strategic planning problem of local distribution companies. *The Journal of Energy Markets*, *4*, 47–68.

Koberstein, A., Lukas, E., & Naumann, M. (2012). Integrated strategic planning of global production networks and financial hedging under uncertain exchange rates. Working paper.

Linderoth, J., Shapiro, A., & Wright, S. (2006). The empirical behavior of sampling methods for stochastic programming. *Annals of Operations Research*, *142*, 215–241.

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

Lougee-Heimer, R. (2003). The Common Optimization Interface for Operations Research. *IBM Journal of Research and Development*, *47*, 57–66.

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

Morton, D. P. (1996). An enhanced decomposition algorithm for multistage stochastic hydroelectric scheduling. *Annals of Operations Research*, *64*, 211–235.

Ruszczyński, A. (1986). A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical programming*, *35*, 309–333.

Ruszczyński, A. (1997). Accelerating the regularized decomposition method for two stage stochastic linear problems. *European Journal of Operational Research*, *101*, 328–342.

Ruszczyński, A., & Shapiro, A. (2002). *Stochastic Programming*. Elsevier.

Shapiro, A., Dentcheva, D., & Ruszczyński, A. (2009). *Lectures on stochastic programming: Modeling and Theory*.

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

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

Vladimirou, H., & Zenios, S. A. (1999). Scalable parallel computations for large-scale stochastic programming. *Annals of Operations Research*, *90*, 87–129.

Wallace, S., & Ziemba, W. T. (Eds.) (2005). *Applications of stochastic programming*. Society for Industrial Mathematics.

Zverovich, V., Fábián, C., Ellison, F., & Mitra, G. (2010). A computational study of a solver system for processing two-stage stochastic linear programming problems. *Stochastic Programming E-Print Series (SPEPS)*, (pp. 1–34).