A BRANCH AND BOUND ALGORIHM TO SOLVE LARGE SCALE MULTISTAGE STOCHASTIC PROGRAMS

Brianna Christian and Selen Cremaschi*
Department of Chemical Engineering, Auburn University
Auburn, AL 36849

Abstract

The growth in computation complexity of multistage stochastic programs (MSSPs) with problem size often prevents its application to real-world size problems. In this work, we present a branch and bound algorithm capable of reducing the resource requirements for the generation and solution of large-scale MSSPs. Using the algorithm to solve four instances of the pharmaceutical R&D pipeline management problem revealed that the algorithm used significantly less memory compared to the deterministic equivalent solutions.

Keywords

Multistage Stochastic Programming, Knapsack Decomposition, Progressive Hedging, Endogenous Uncertainty, Branch and Bound

Introduction

Applications of optimization under uncertainty in chemical process industry cover a broad range of problems including production planning and scheduling, location and transportation planning, product and process design, and resource allocation. One approach that is used to model and solve these problems is stochastic programming (SP). In SP, a scenario based framework is used to explicitly account for uncertainty. The main components of a SP are (1) a set of scenarios representative of the outcomes of the uncertain parameters, and (2) stages where actions can be taken. When all uncertain parameters can be considered independent, scenarios are generated using the Cartesian product of the realizations of each parameter (Apap and Grossmann, 2015). Stochastic programs can be classified based on the number of decision stages. Problems with a single decision stage, after which uncertainty is realized, are called two-stage SPs. When uncertainty is revealed gradually over multiple decision stages the problem is called a multistage SP (MSSP) (Birge and Louveaux, 2011). The stages in MSSPs are typically tied to time periods, where uncertainty is revealed at different time periods and the decisions are made sequentially based on available information.

The uncertainty in SPs can be grouped into two broad categories, endogenous and exogenous. The realization of exogenous uncertain parameters is not affected by the decisions. In contrast, decisions impact endogenous uncertain parameters. This impact can either determine when the uncertainty is resolved or change the distribution(s) of the uncertain parameter(s).

At the beginning of the planning horizon and before any decisions are made, all scenarios in a SP are indistinguishable. As uncertainty is revealed, either through decisions or naturally, the scenarios begin to be distinguishable. Once a set of scenarios is differentiable from the rest, decisions for it may be made independently. To avoid using unrealized values of the uncertain parameter(s) when making decisions, non-anticipativity

* selen-cremaschi@auburn.edu

constraints (NACs) are introduced to the SP formulation. The NACs of a MSSP with endogenous uncertainty also depend on the decisions.

Most real-world-size MSSP problems with endogenous uncertainty are computationally intractable due to the space complexity of the problem caused by exponential growth in the size of the variables and the number of NACs. The number of scenarios increases with the number of uncertain parameters and the number of realizations associated with each uncertain parameter. Several approaches have been developed to address this complexity. One approach relaxes the NACs using Lagrangean-relaxation. The problem is then solved using a duality based approach. (Goel and Grossmann, 2004; Tarhan et al., 2013). Gupta and Grossmann (2014) recently presented an improvement to this approach by incorporating a scenario grouping strategy. Colvin and Maravelias (2010) increased the size of the pharmaceutical R&D pipeline management problem that can be solved by using a branch and cut algorithm that gradually added NACs. Solak et al. (2010) used a sample average approximation approach, where candidate solutions were generated using subsets of the full scenario set. Jiang et al. (2016) introduced a set of cutting plane inequalities to strengthen the linear relaxation of the general multistage stochastic unit commitment problem.

This paper presents a branch and bound algorithm that uses progressive hedging (PH) combined with the knapsack decomposition algorithm (KDA) (Christian and Cremaschi, 2015) to solve large-scale MSSPs with endogenous uncertainty. We test the performance of the algorithm on pharmaceutical R&D pipeline clinical trial planning problem. The algorithm solves the problems using less random-access memory (RAM) than the deterministic equivalent MSSPs.

The Branch and Bound Algorithm

The algorithm is summarized in Fig. 1. At the initialization step, the values for the relative gap between the upper bound and the lower bound (α), and the tolerance (ε) are set. Next, the iteration count, *i*, is set to zero. The algorithm starts by generating a feasible solution, φ_i, using the KDA, which is a heuristic algorithm that solves the original MSSP by decomposing it into a series of knapsack problems. The Equivalent Expected Net Present Value (EENPV) for the KDA solution provides the initial lower bound, LB_0. The algorithm next determines the branching variable(s) by comparing the values of decision variables that have been fixed in the current branch to values of decision variables in the KDA solution, φ_i. From the decisions variables in φ_i that have not yet been fixed in the current branch, the ones that occur at the earliest time period are selected as the branching variable(s).

Assuming that there is one binary branching variable, two new linear programs (LPs) are generated; in one, the branching variable takes the value of one, in the other, the value of zero. In both LPs, the values of decision variables that were fixed in the parent branch are carried over. The solution of each of these LPs provides an upper bound, U_n, for each branch *n*. The LPs are added to the set of active branches, **N**, and the parent branch is removed. After determining the upper bound for each branch, the algorithm determines the upper bound for the problem, UB_i, for iteration *i*. It is defined as $\max\{U_n \ \forall n \in \mathbf{N}\}$, and Q is the set of fixed decisions corresponding to the upper bound of the problem, UB_i.

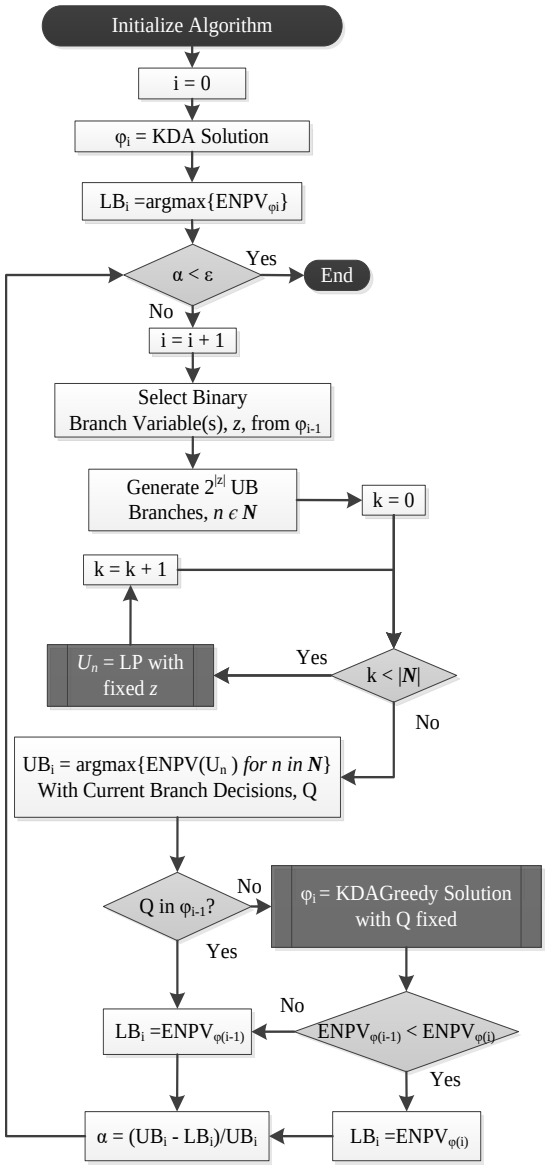

Figure 1. The branch and bound algorithm

The algorithm continues by comparing the decisions in Q with values of the decision variables in the KDA solution, φ_{i-1}. If the values of the decision variables match then the lower bound of the problem, LB_i, is equal to the lower bound of the previous iteration, LB_{i-1}. Otherwise, the algorithm solves the MSSP where the values of the decision variables in Q are fixed using the KDAGreedy algorithm, and generates a new feasible solution, φ_i. If the EENPV of φ_i is greater than LB_{i-1}, the lower bound, LB_i, takes the value

of the EENPV of φ_i. If the EENPV of the KDA solution is lower than LB_{i-1}, the value of LB_i is set equal to LB_{i-1}. The algorithm updates α using LB_i and UB_i. If α is lower than ε or the maximum iteration count is reached, the algorithm terminates. Otherwise, the algorithm increments the iteration count and selects new branching variables. At termination, φ_i provides the solution of the MSSP at a relative gap of α.

A Progressive Hedging Upper Bound

The LPs used for determining the upper bound are solved using the Progressive Hedging (PH) approach originally presented in Rockafellar and Wets (1991) and adapted by Watson and Woodruff (2011). The PH approach is proven to converge to the optimum of convex MSSPs with exogenous uncertainty. It decomposes the MSSP into individual scenario quadratic programs (QPs) with a modified objective function, and uses the solutions of these QPs to converge to the MSSP solution. This scenario-wise approach allows solving linear MSSPs with exogenous uncertainty without generating the full MSSP.

The SP formulation with exogenous uncertainty can be written as $\min cx + \sum_{s\in S} p_s(f_s y_s)$, $s.t.\ (x, y_s) \in Q_s \forall s \in S$ where c is the cost associated with the constrained decision vector x, p_s is the probability that the scenario s will occur, and f_s represents the cost of the scenario specific decisions y_s. The decision vector x represents the decisions which must be identical in all scenarios. By writing the decision vector as a single variable enforceable in all scenarios, Watson and Woodruff (2011) implicitly enforce the NACs.

The PH algorithm used in this work is given in Fig. 2. After initializing the iteration counter (k) to zero, the algorithm solves the deterministic optimization problem for each scenario, finding $x^{(k)}$ (Fig. 2, Step 2). Next, the average values for the decision vector, , and the weights, $w_s^{(k)}$, are calculated (Fig. 2,Steps 3 and 4). In Step 5, the iteration counter is incremented, and new QPs are constructed using the values of $\bar{x}^{(k)}$ and $w_s^{(k)}$. Solutions of these QPs are used to update the values of $\bar{x}^{(k)}$ and $w_s^{(k)}$ (Fig. 2, Steps 7 and 8). The convergence of the algorithm is checked in Step 10 using the value of $g^{(k)}$ calculated in Step 9 (Fig. 2). If the algorithm is within ε, it terminates. Otherwise, it returns to Step 5.

The PH algorithm requires the knowledge of the scenarios and their differentiation time periods. The problems of interest in this work are MSSPs with integer decision variables and endogenous uncertainty, where the differentiating events can be specified but not when and if they would occur. To ensure that the solutions obtained by the PH algorithm are true upper bounds for these problems, the integrality constraints of the MSSP are relaxed, and appropriate upper and lower bounds for these variables are introduced. Next, all NACs except the current-stage NACs are removed, which yields a two-stage SP. For the problems considered in this work, the values of the uncertain endogenous parameters are realized gradually as the corresponding series of decisions are taken. To ensure that the 'current-stage' NACs are always enforced, we used a set of problem specific logical statements that tracks the value of differentiating decision variables and determines if any uncertainty realizations have occurred. The solutions obtained by the PH algorithm (Fig. 2) to these relaxed two-stage SPs are used to update the upper bounds of the MSSP (Fig. 1) at each iteration.

1. $k \coloneqq 0$
2. $For\ all\ s \in S,$
   $x_s^{(k)} \coloneqq argmin(cx + f_s y_s) : (x, y_s) \in \boldsymbol{Q}_s$
3. $\bar{x}^{(k)} \coloneqq \sum_{s\in S} \Pr(s) x_s^{(k)}$
4. $w_s^{(k)} \coloneqq \rho\left(x_s^{(k)} - \bar{x}^{(k)}\right)$
5. $k \coloneqq k + 1$
6. $For\ all\ s \in S,$
   $x_s^{(k)} \coloneqq argmin\left(cx + w_s^{(k-1)} x + {}^{\rho}/_{2} \left\|x - \bar{x}^{(k)}\right\|^2 + f_s y_s\right) : (x, y_s) \in \boldsymbol{Q}_s$
7. $\bar{x}^{(k)} \coloneqq \sum_{s\in S} \Pr(s) x_s^{(k)}$
8. $For\ all\ s \in S, w_s^{(k)} \coloneqq w_s^{(k-1)} + \rho\left(x_s^{(k)} - \bar{x}^{(k)}\right)$
9. $g^{(k)} \coloneqq \sum_{s\in S} \Pr(s) \left\|x - \bar{x}^{(k)}\right\|$
10. $If\ g^{(k)} < \varepsilon, then\ go\ to\ 5. Otherwise\ terminate$

Figure 2. The progressive hedging algorithm (Watson and Woodruff, 2011)

Updating the Lower Bound Using the KDAGreedy Algorithm

The lower bound of the problem is found using a modified version of the KDA (Christian and Cremaschi, 2015). The original KDA uses a series of knapsack problems to find a feasible solution for MSSPs with endogenous uncertainty. The KDA starts by decomposing the decision variables into a set of items. Each item has an associated value and weight. The value of the item is based on the expected potential gains from the associated decision variable. The weight of the item corresponds to the resource requirements associated with the decision variable. The KDA starts by packing an initial knapsack with the items based on overall weight limitations. The selected items are used to determine the value of the decision variables at the first time period in the planning horizon. The uncertainty associated with those decision variables are realized, and the KDA generates a new knapsack problem for each realization. Based on the realizations, the algorithm decides which items are eligible to be considered in each of the newly created knapsack problems, and solves them. Solutions determine the values of the decision variables, which in turn results in realizations of associated uncertain parameters. The KDA algorithm continues until the end of the planning horizon. To ensure that the KDA solution does not over-utilize resources early in the planning horizon, the algorithm introduces a heuristic overscheduling constraint. The constraint prevents selection of items if there are not sufficient resources (i.e., maximum weight) for potential knapsack problems that may be generated due to the realizations of the item in consideration in the future. Details of the KDA can be found elsewhere (Christian and Cremaschi, 2015).

The original KDA is modified by removing the heuristic overscheduling constraint and changing when new knapsack problems are generated. Removing this constraint allows the KDAGreedy algorithm to pack any eligible item in any knapsack while keeping the feasibility of its solution. In the original KDA, new knapsacks are only generated after all uncertainty associated with selected items was realized. In the KDAGreedy algorithm, new knapsack problems are generated at every time period allowing non-zero decision variable values if there are enough resources at any time period.

Case Studies – Pharmaceutical R&D Pipeline Management Problem

We use the branch and bound algorithm to solve four instances of the pharmaceutical R&D pipeline management problem. A brief overview of the pharmaceutical R&D pipeline management problem is provided in the section below. Values for the constants in each case can be found in Christian and Cremaschi (2015). The branch and bound algorithm is implemented in Python 3.5. The PH algorithm and the KDAGreedy utilize Pyomo 4.1 (Hart et al., 2011) and CPLEX 12.6. The solutions to the deterministic equivalent MSSP for each case were found using Pyomo 4.1 and CPLEX 12.6. All of the problems in this work were solved using the Auburn University Hopper Cluster.

The Pharmaceutical R&D Pipeline Management Problem

The pharmaceutical R&D pipeline management problem consists of a set of new pharmaceutical drug development projects. The goal of the problem is to determine the clinical trial schedule which yields the highest ENPV given uncertainty in the outcome of each clinical trial. In this work, we use the formulation presented by Colvin and Maravelias (2008).

The mathematical formulation of the clinical trial planning problem is characterized by a set of potential new products [$d \epsilon \mathbf{D}$]. Each potential product is required to complete a series of clinical trials [$j \epsilon \mathbf{J}$]. Completion of clinical trials is limited by a set of resources [$r \epsilon \mathbf{R}$]. Clinical trials have both an associated resource cost(s) [$\rho_{d,j,r}$] and monetary cost [$C_{d,j}$]. Resource expenditures are limited by maximum resource availability [$\rho_r^{\max}$].

The scheduling of clinical trials occurs along a discretized planning horizon of n months divided into $|\mathbf{T}|$ time steps [$t \epsilon \mathbf{T}$]. Each clinical trial has a fixed duration [$\tau_{d,j}$]. The success of the clinical trial is given as a Bernoulli random variable with a known probability of success [$p_{d,j}$]. Successful completion of all clinical trials results in commercial availability, and revenue from the market success of the product is realized [Rev_d^{max}]. Penalties for having products idle in the pipeline and for reduced active patent life due to delayed development of a product, γ_d^L (loss of patent life) and γ_d^D (loss of market share), are accessed when calculating the ENPV.

Results and Discussion

The branch and bound algorithm was first used to solve a toy-box sized two-product two-clinical-trial case. We present the first five iterations of the algorithm in Fig. 3. The initial lower bound obtained by KDA is 1097. The algorithm uses the solution from the initial lower bound to select decisions variables to branch on. In this problem, the algorithm selects the first clinical trial of both Drug 1 andDrug 2 starting at the first time period, i.e., (D1, P1, 0) and (D2, P1,0). Selecting two decision variables to branch on creates four branches marked as 1A, 1B, 1C, and 1D in Fig. 3. The LPs are generated, and their upper bound solutions – obtained by the PH algorithm – are 1141.83 (1A), 1139.53 (1B), 1138.34 (1C), and 1136.03 (1D)

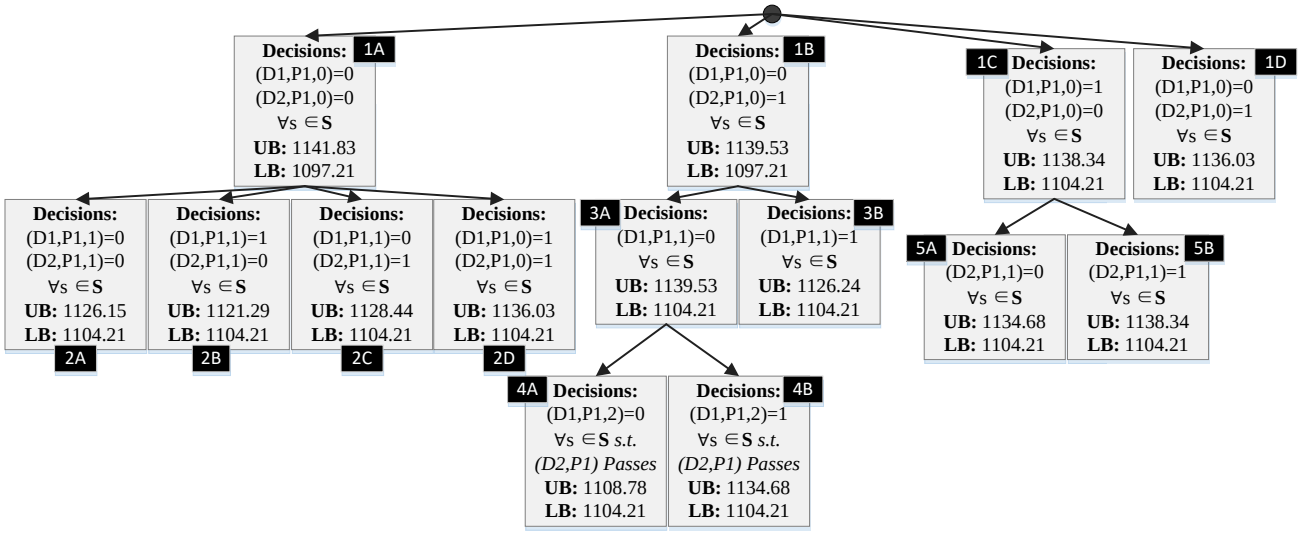

Figure 3. Five iterations of the branch and bound algorithm for the two-product two-clinical trial case study

Because the upper bound of 1A is the highest, the algorithm sets the values of the decision variables associated with (D1, P1, 0) and (D2, P1,0) equal to zero, and calls KDAGreedy. The solution obtained by the KDAGreedy suggest starting the first clinical trials of D1 and D2 at the second time period (t=1). The algorithm generates four new branches (2A-D), and finds their upper bounds using PH. The algorithm continues by selecting the end branch with the highest upper bound (e.g, 1B for the third iteration) until the stopping criteria are met.

Figure 4 plots the log (base 10) of the CPU time consumed by the algorithm versus the relative gap ($(UB_i-LB_i)/UB_i$) for the two- and three-product cases. Labels on the marker on the graph identify the number of completed iterations. In all three cases, the slopes in Fig. 4 are approximately linear indicating a logarithmic relationship between the CPU time and the relative gap. Therefore, the decision variables branched on in earlier iterations have a larger impact on the relative gap then the decisions branched on in later iterations, which initially improves the upper bound rapidly. The quality of the PH upper bound is limited due to the linear relaxation of the MSSP and use of only next stage NACs. Premature termination of the algorithm in the two-product three-trial case was also caused by the PH upper bound. To ensure that the PH-algorithm solution provides a true upper bound for the problem, all integrality constraints are relaxed. Hence, the solutions at the upper bound allows partial investments (i.e. non-integer results) on some of the clinical trials. The KDAGreedy only generates feasible solutions, in which these non-integer decision variables become zero and are never branched on. For the three-product three-trial case study, the algorithm ran for the maximum allowable wall time of ten days. At termination, the relative gap was 2.6%. The time needed to run each algorithm to completion is shown in Table 1. As expected, compared to the deterministic equivalent MSSP (also in Table 1), the branch and bound algorithm takes significantly longer to close the gap.

One of the challenges with solving real-world size MSSPs is the space complexity of the problem (i.e. the RAM required to generate the problem). As can be seen from Table 1, the branch and bound algorithm uses significantly lower random-access memory (RAM) than its deterministic equivalent counterpart for all case studies. The RAM usage of the branch and bound algorithm for the three-product case is higher than both two-product cases and the five-product case. This increase in RAM usage is caused by the number of iterations the algorithm completed. In each iteration, at least two new branches are created. The storage of these branches gradually increase memory requirements of the algorithm.

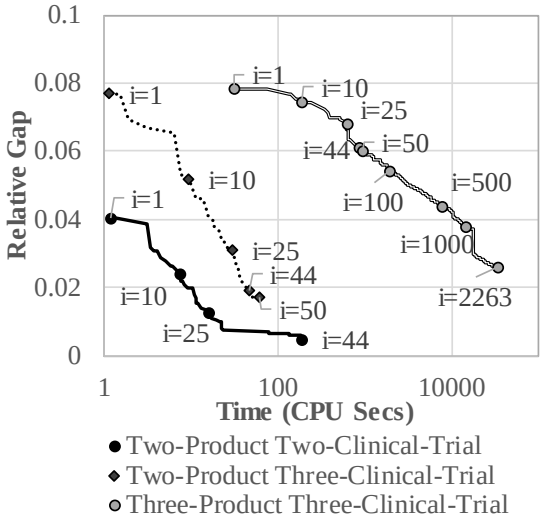

Figure 4. Plot of CPU time vs. relative gap for the two- and three-product case studies

The impact of parallelizing the PH algorithm is studied for the five-product three-clinical trial case study. The problem has a total of 1024 scenarios. Table 2 shows the number of threads used for parallelization for each instance of the problem along with the approximate number of problems solved per thread per PH iteration. Because CPLEX 12.6 recommends allocating additional processor cores for solution of the QPs when available, six cores were allocated for each thread. This also improved the efficiency of the PH algorithm.

Based on the number of iterations completed, parallelization has the greatest impact when nine threads are used. In the case where nine threads were used, 183 iterations were completed and the relative gap of the problem was reduced to 6.4%. However, our limited computational experiments showed that increasing from

Table 1. Resource usage, relative gap, and computational time results for the deterministic equivalent MSSP and the branch and bound algorithm

	Deterministic Equivalent			**Branch and Bound**		
	RAM (MB)	Relative Gap	CPU Time (HH:MM:SS)	RAM (MB)	Relative Gap	CPU Time (HH:MM:SS)
Two-Product Two-Trial	2.45	0.001	0:00:01	0.05	0.005	0:03:08
Two-Product Three-Trial	5.93	0.001	0:00:01	0.48	0.017	0:01:01
Three-Product Three-Trial	89.79	0.001	0:00:03	26.07	0.026	615:07:05
Five-Product Three-Trial	1430.15	0.001	0:00:42	2.12	0.068	643:41:52

Table 2. Parallelization results for the five-product case study

Number of Threads	Completed Iterations	Relative Gap	Problems Per Thread
3	39	0.0684	341
6	41	0.0772	170
9	183	0.064	113

three threads to six threads had an inverse effect on the solution quality. The current implementation of the branch and bound algorithm selects decisions to branch on from the KDA decision tree. In general, the algorithm selects decision variables from the KDAGreedy solution that have non-zero values starting with the variables earliest in the planning horizon. When the algorithm selects which decision variables to branch on, it selects all the decisions in one particular branch of the KDA decision tree. Because the object that holds the decision tree is not ordered, the algorithm may not always select the same branch (i.e., corresponding to the same realizations) from the KDA decision tree. The performance of the algorithm when the parallelization studies were conducted suggests that it is particularly sensitive to the order in which decision variables are selected for branching.

Conclusions and Future Directions

In this work, we successfully implemented a branch and bound algorithm that reduces the RAM requirements for solving large-scale MSSPs. The algorithm was applied to four instances of the pharmaceutical R&D pipeline management problem. Our studies reveal that, in all case studies, the CPU time for the algorithm is higher than the deterministic MSSP. However, the RAM usage of the algorithm is less than half of the amount required for solving the deterministic equivalent MSSP. Despite having a slower convergence time, the first iteration of the algorithm provides a true bound on the solution. For larger problems we expect longer computation times due to the convergence times of the PH algorithm and linear scaling in the RAM requirements.

For future work, two paths have been identified to increase the effectiveness and efficiency of the algorithm. First, the algorithm will be modified to branch on the variables with the non-integer values in the upper bound if the KDAGreedy algorithm fails to provide a branching decision variable. Second, we plan to investigate different rules for selecting the branching variables and further parallelization approaches for the algorithm.

Acknowledgements

We acknowledge the Auburn University Hopper Cluster for support of this work. Financial support for this work is provided by the US National Science Foundation through the Auburn University Integrative Graduate Research and Education Traineeship (IGERT) program (Award# 1069004), and the NSF Career Grant (Award # 1623417).

References

Apap, R., and Grossmann, I. E. (2015). Models and computational strategies for multistage stochastic programming under endogenous and exogenous uncertainties.

Birge, J. R., and Louveaux, F. (2011). *Introduction to Stochastic Programming*. Springer New York. Retrieved from https://books.google.com/books?id=Vp0Bp8kjPxUC

Christian, B., and Cremaschi, S. (2015). Heuristic solution approaches to the pharmaceutical R&D pipeline management problem. *Computers and Chemical Engineering*, *74*, 34–47.

Colvin, M., and Maravelias, C. T. (2008). A stochastic programming approach for clinical trial planning in new drug development. *Computers & Chemical Engineering*, *32*, 2626–2642.

Colvin, M., and Maravelias, C. T. (2010). Modeling methods and a branch and cut algorithm for pharmaceutical clinical trial planning using stochastic programming. *European Journal of Operational Research*, *203*(1), 205–215.

Goel, V., and Grossmann, I. E. (2004). A stochastic programming approach to planning of offshore gas field developments under uncertainty in reserves. *Computers and Chemical Engineering*, *28*(8), 1409–1429.

Gupta, V., and Grossmann, I. E. (2014). Multistage stochastic programming approach for offshore oilfield infrastructure planning under production sharing agreements and endogenous uncertainties. *Journal of Petroleum Science and Engineering*, *124*, 180–197.

Hart, William E., Watson, Jean-Paul , and Woodruff, David L.. (2011) Pyomo: modeling and solving mathematical programs in Python. Mathematical Programming Computation 3(3): 219-260.

Jiang, R., Guan, Y., & Watson, J. (2016). Cutting planes for the multistage stochastic unit commitment problem. Mathematical Programming (Vol. 157). Springer Berlin Heidelberg.

Rockafellar, R. T., & Wets, R. J.-B. (1991). Scenarios and Policy Aggregation in Optimization Under Uncertainty. *Mathematics of Operations Research*, *16*(1), 119–147.

Solak, S., Clarke, J. P. B., Johnson, E. L., and Barnes, E. R. (2010). Optimization of R&D project portfolios under endogenous uncertainty. *European Journal of Operational Research*, *207*(1), 420–433.

Tarhan, B., Grossmann, I. E., and Goel, V. (2013). Computational strategies for non-convex multistage MINLP models with decision-dependent uncertainty and gradual uncertainty resolution. *Annals of Operations Research*, *203*(1), 141–166.

Watson, J. P., & Woodruff, D. L. (2011). Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Computational Management Science*, *8*(4), 355–370.