

A Comparative Study of Decomposition Algorithms for Stochastic Combinatorial Optimization

Lewis Ntaimo

Department of Industrial and Systems Engineering, Texas A&M University, 3131 TAMU, College Station, TX 77843, USA, ntaimo@tamu.edu

Suvrajeet Sen

Department of Systems and Industrial Engineering, The University of Arizona, PO Box 210020, Tucson, Arizona 85721, USA, sen@sie.arizona.edu

This paper presents comparative computational results using three decomposition algorithms on a battery of instances drawn from two different applications. In order to preserve the commonalities among the algorithms in our experiments, we have designed a testbed which is used to study instances arising in server location under uncertainty and strategic supply chain planning under uncertainty. Insights related to alternative implementation issues leading to more efficient implementations, benchmarks for serial processing, and scalability of the methods are also presented. The computational experience demonstrates the promising potential of the disjunctive decomposition (D^2) approach towards solving several large-scale problem instances from the two application areas. Furthermore, the study shows that convergence of the D^2 methods for stochastic combinatorial optimization (SCO) is in fact attainable since the methods scale well with the number of scenarios.

Key words: Stochastic mixed-integer programming; disjunctive decomposition; stochastic server location; strategic supply chain planning.

1. Introduction

The study of computational approaches for stochastic combinatorial optimization (SCO) problems is currently in a nascent stage. While there is considerable interest in these problems, difficulties arising from both stochastic as well combinatorial optimization have made it difficult to design, implement and test algorithms for this class of problems. Over the past several years, there has been significant effort devoted to the design of algorithms for SCO problems. However, reports addressing computational implementation and testing have been slow in coming. One of the few papers related to testing algorithms for SCO problems is that by [19]. However, that paper focuses only on the subset of problems in which the first-stage decisions are discrete/binary and the second-stage consists of only continuous variables.

The latter requirement (continuous second-stage variables) yields LP value functions that are convex. Hence the algorithms tested in [19] inherit their algorithmic properties from traditional Benders' decomposition [5]. In contrast, for SCO problems in which the second-stage includes binary decision variables, new decomposition algorithms are necessary. In a recent paper, [13] have provided initial evidence that the disjunctive decomposition (D^2) algorithm [17] can provide better performance to direct methods for at least one class of SCO problems (stochastic server location problems or SSLPs).

In this paper we extend our experimental work by comparing the performance of multiple decomposition algorithms using test instances from two classes of large-scale instances. In particular, we investigate the performance of the method proposed in [10] (L^2 algorithm), the D^2 algorithm [17], and the D^2 -BAC (branch-and-cut) algorithm [18]. Such a comparative study requires the development of an algorithmic testbed in which the commonalities among the algorithms are preserved while the algorithm-specific concepts are implemented in as efficient a manner as possible. We use this resulting testbed to study the performance of the algorithms with two problem classes: server location under uncertainty [13] and strategic supply chain planning under uncertainty [3]. While the first application has binary decision variables in both stages, the second application includes binary and continuous variables in the second-stage. To date the solutions reported for the supply chain instances are, for the most part, non-optimal, and our experiments reveal that it is possible to improve upon these solutions significantly. The experiments reported here are by far the most extensive computations for optimum-seeking methods for SCO. As a byproduct of this investigation we also report on the insights related to: a) alternative implementation issues leading to more efficient implementations, b) benchmarks for serial processing, and c) scalability of the methods.

The rest of this paper is organized as follows. In the next section a general formal problem statement is given. Section 3 discusses computer implementation of the three decomposition algorithms for SCO and the issues associated with such implementation. Section 4 reports on the solution of some of the largest stochastic combinatorial optimization problems arising from the two applications under consideration. We end the paper with some concluding remarks in Section 5.

2. Problem Statement

Throughout this paper we consider the following general two-stage SCO problem:

$$\text{Min}_{x \in X \cap \mathbf{X}} c^\top x + E[f(x, \tilde{\omega})], \quad (1)$$

where c is a known vector in \Re^{n_1} , $X \subseteq \Re^{n_1}$ is a set of feasible first-stage decisions and \mathbf{X} define restrictions requiring some first-stage decision variables to be 0-1 integer. $E[\cdot]$ is the usual mathematical expectation operator with

$$E[f(x, \tilde{\omega})] = \sum_{\omega \in \Omega} p_\omega f(x, \omega),$$

$\tilde{\omega}$ is a multi-variate discrete random variable with a realization (scenario) ω with probability p_ω and sample space Ω . For any ω ,

$$f(x, \omega) = \text{Min } q(\omega)^\top y, \quad (2a)$$

$$\text{subject to } Wy \geq r(\omega) - T(\omega)x, \quad (2b)$$

$$y \geq 0, y_j \text{ binary}, j \in J_2. \quad (2c)$$

In problem formulation (2), $q(\omega)$ is the cost vector in \Re^{n_2} for scenario $\omega \in \Omega$ and J_2 is an index set that may include some or all the variables listed in $y \in \Re^{n_2}$. Although the second-stage (recourse) variable y depends on the outcome ω , this dependence is not explicitly indicated here. This is because the subproblem for each outcome ω is decoupled from all other outcomes once a vector x is given. Thus this formulation emphasizes the loosely coupled nature of two-stage SCO problems. In this paper we address instances of problem (1-2) under the following assumptions:

(A1) Ω is a finite set.

(A2) $X = \{x \in \Re_+^{n_1} \mid Ax \geq b\}$.

(A3) $f(x, \omega) < \infty$ for all $(x, \omega) \in X \times \Omega$.

Assumption (A3) requires that the subproblem (2) remain feasible for all $(x, \omega) \in X \times \Omega$ and this property is referred to as relatively complete (integer) recourse [21].

Since we assume that the problem data is governed by discrete random variables, the formulation (1-2) can also be written as the so called *deterministic equivalent problem* (DEP)

formulation or *extensive form* as follows:

$$\text{Min}_{x \in X \cap \mathbf{X}} c^\top x + \sum_{\omega \in \Omega} p_\omega q(\omega)^\top y(\omega) \quad (3a)$$

$$\text{subject to } T(\omega)x + Wy(\omega) \geq r(\omega), \quad \forall \omega \in \Omega, \quad (3b)$$

$$y(\omega) \geq 0, \quad y_j(\omega) \text{ integer}, \quad j \in J_2, \quad \forall \omega \in \Omega. \quad (3c)$$

Note that the dependence of the second-stage decision on the scenario is now explicitly made in the DEP formulation. Problem (3) is a large-scale mixed-integer programming (MIP) formulation and potentially can be solved by an MIP solver directly. However, in order to adequately capture the uncertainty in the problem, the number of scenarios $S = |\Omega|$ is generally large. Therefore, problem (3) may become intractable even for the state-of-the-art commercial MIP solvers.

3. Computer Implementation

In this section we describe a computer implementation of the three decomposition algorithms for SCO. Although our implementation is based on serial processing, our presentation can be used as the basis for future implementation on parallel/distributed computing platform. In order to achieve this goal we view each algorithm as a coordination mechanism that invokes its modules according to its design. In essence subsets of modules can be distributed using standard distributed computing middleware (e.g. CORBA). Because of this orientation we present the algorithms as modules with specific inputs and outputs. The decomposition schemes have a flavor of master-worker paradigm [8] with master programs of the following form:

$$\text{Min } c^\top x + \eta \quad (4a)$$

$$\text{subject to } Ax \geq b, \quad (4b)$$

$$(\beta^t)^\top x + \eta \geq \alpha^t, \quad t = 1, \dots, k, \quad (4c)$$

$$x \in \mathcal{B}^{n_1}. \quad (4d)$$

where the index k denotes the iteration number. Other variants of this master program (e.g. relaxations) are possible but are not provided in our implementation. The worker module will depend upon the specific algorithm. For each ω , the module will output vectors $\beta^t(\omega)$

and scalars $\alpha^t(\omega)$ which will be used to calculate

$$\beta^t = \sum_{\omega \in \Omega} p_\omega \beta^t(\omega) \quad (5)$$

and

$$\alpha^t = \sum_{\omega \in \Omega} p_\omega \alpha^t(\omega) \quad (6)$$

For each algorithm described below we provide the collection of modules as well as the inputs and outputs required for them. The algorithm is summarized using a flowchart depicting the sequence in which the modules are invoked. Note that ‘for loops’ are easy candidates for parallelization.

3.1. The L^2 Algorithm

Conceptual details regarding the L^2 algorithm are available in [10]. The L^2 algorithm solves two-stage SCO problems with purely binary first-stage decision variables and involves three main tasks per algorithmic iteration: solving a 0-1 integer master program, solving a mixed-binary subproblem for each scenario, and generating an optimality cut to add to the master program. The modules for the algorithm are summarized below with a flowchart given in Figure 1.

Modules:

Step 1. Initialization.

Inputs: Instance data, run parameters (e.g. stopping tolerance ϵ).

Output: Initialized master problem and its solution x^k , upper bound $V_0 = \infty$, $k = 1$.

Step 2. SubMIP-Solve(ω). Given x^k , evaluate $f^k(x^k, \omega)$ defined in problem (2) for each $\omega \in \Omega$.

Inputs: x^k .

Outputs: $\alpha^k(\omega), \beta^k(\omega)$, subproblem objective value $v^k(\omega)$, upper bound V^k .

Step 3. UpdateMasterMIP-Solve. Add a cut and solve problem (4).

Inputs: α^k, β^k . These quantities are the expected values of the output of

SubMIP-Solve(ω)

Outputs: x^{k+1} , lower bound v^k .

Step 4. Termination.

If $V^k - v^k \leq \epsilon$, stop. Otherwise, $k \leftarrow k + 1$, repeat from Step 2.

To provide lower bounding functions, Benders-type [5] optimality cuts are generated from the LP relaxation dual solution and added to the master program. Since the L^2 algorithm solves the subproblems as MIPs, this can be time consuming when the number of scenarios is large. This together with the generally weak optimality cut does have an adverse effect on the performance of this algorithm as our computational experiments show. To overcome this limitation, the next two disjunctive decomposition algorithms avoid solving scenario subproblem MIPs at every iteration of the algorithm. Instead, scenario subproblem LP relaxations are solved and MIP solves are done only when necessary (e.g., when computing upper bounds for the problem).

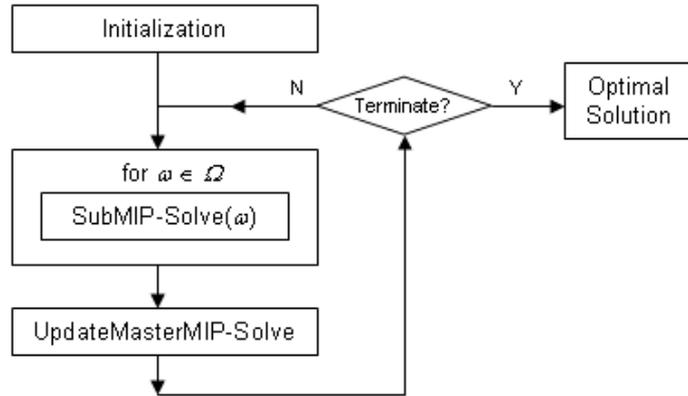


Figure 1: The L^2 Algorithm

3.2. The D^2 Algorithm

The D^2 algorithm is derived in [17] and illustrated in [16]. In the D^2 algorithm the scenario subproblem LP relaxation takes the following form:

$$f_c^k(x, \omega) = \text{Min } q(\omega)^\top y, \quad (7a)$$

$$\text{subject to } W^k y \geq \rho_c^k(x, \omega), \quad (7b)$$

$$y \in \mathfrak{R}_+^{n_2}. \quad (7c)$$

where W^k is a result of augmenting the so called “common-cut coefficients” (π^\top 's) to matrix W and $\rho_c^k(x, \omega)$ is a result of augmenting the $\pi_c(x, \omega)$'s to the right-hand side $r(\omega) - T(\omega)x$.

The function $\pi_c(x, \omega)$ is a linear function of x and ω . To derive the π 's, a simple recourse SLP (problem 18 in [17]) is solved. We shall refer to this SLP as the C^3 -SLP, which also outputs multipliers λ_0 , and λ_1 associated with the disjunctions used in cut formation. The function $\pi_c(x, \omega)$ is derived by solving another LP (problem 19 in [17]) for each scenario. We shall refer to this LP as the *RHS*-LP. The function $\pi_c(x, \omega)$ is affine for each ω , and is recorded by appending the constant term of the affine function to $r(\omega)$, and the linear part is appended to the $T(\omega)$ matrix. Hence in iteration k , these quantities may be denoted by $r^k(\omega)$ and $T^k(\omega)$.

The modules for the D^2 algorithm are summarized below with a flowchart given in Figure 2.

Modules:

Step 1. Initialization.

Inputs: Instance data, run parameters (e.g. stopping tolerance ϵ).

Output: Initialized master problem and its solution x^k , upper bound $V_0 = \infty$, $k = 1$.

Step 2. SubLP-Solve(ω). Given x^k evaluate $f_c^k(x^k, \omega)$ defined in problem (7) for each $\omega \in \Omega$.

Inputs: x^k .

Outputs: If solution $y(\omega)$ satisfy integer restrictions for all $\omega \in \Omega$, output $\alpha^k(\omega)$, $\beta^k(\omega)$, and upper bound V^k . Otherwise, output disjunction variable $j(k)$, and $E[y(\tilde{\omega})]$.

Step 3. C3SLP-Solve. Form and solve the C^3 -SLP.

Inputs: $j(k)$, and $E[y(\tilde{\omega})]$.

Outputs: Coefficients π^k and multipliers λ_0^k and λ_1^k .

Step 4. RHSLP-Solve(ω). Form and solve the *RHS*-LP for each $\omega \in \Omega$.

Inputs: x^k , λ_0^k and λ_1^k .

Outputs: $\pi_c^k(x, \omega)$.

Step 5. UpdateSubLP-Solve(ω). Update and re-solve subproblem (7) for each $\omega \in \Omega$.

Inputs: x^k , π^k and $\pi_c^k(x, \omega)$.

Outputs: $\alpha^k(\omega)$ and $\beta^k(\omega)$ for all $\omega \in \Omega$, and if solution $y(\omega)$ satisfy integer restrictions for $\omega \in \Omega$, output upper bound V^k .

Step 6. UpdateMasterMIP-Solve. Add a cut and solve problem (4).

Inputs: α^k , β^k .

Outputs: x^{k+1} , lower bound v^k .

Step 7. Termination.

If $V^k - v^k \leq \epsilon$, stop. Otherwise, $k \leftarrow k + 1$, repeat from Step 2.

We note that in order to compute an upper bound it may be necessary to solve scenario subproblem MIPs. When this is the case, the algorithm invokes the solver to perform *SubMIP-Solve*(ω) for all $\omega \in \Omega$. This task is the same as the one in the L^2 algorithm and is performed when the first-stage solution stops changing, usually as the algorithm nears termination. Subproblem MIP solves can also be initiated when the percent gap between the lower and upper bound remains constant for a preset number of iterations. At all other iterations subproblem LP relaxations (7) are solved, which are much faster and the sequential addition of the D^2 -cuts to the feasible region of the second-stage problem leads to the convexification of the region leading to potentially integral solutions. To fully close the gap between the lower and the upper bound for certain problem instances, the optimality cut of [10] discussed in the Section 3.1 is added just after solving the scenario subproblem MIPs. This usually happens in the final iteration of the algorithm.

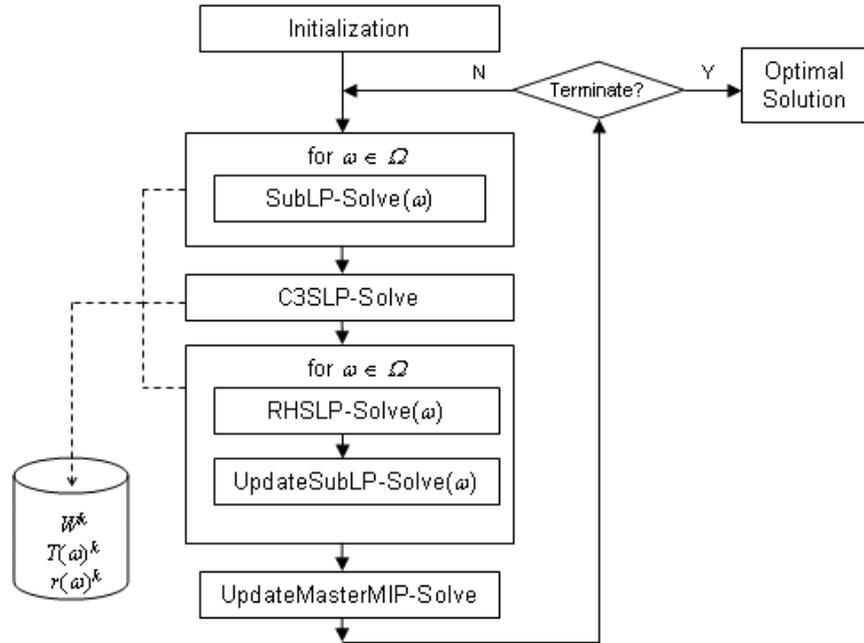


Figure 2: The D^2 Algorithm

3.3. The D^2 -BAC Algorithm

We now describe the D^2 -BAC algorithm which is derived in [18]. While the L^2 and D^2 algorithms provide two extreme ends of a spectrum, the D^2 -BAC algorithm is somewhat in between. In the D^2 -BAC algorithm the second-stage integer subproblems are solved using a “truncated” branch-and-bound (TB&B) tree, thus allowing for subproblem MIP “partial” solves. Realizing the fact that the subproblems are generally NP-hard, the decomposition method may get bogged down in attempts to solve subproblems to optimality, even while the particular first-stage solution is no where near the neighborhood of an optimal solution. Thus the essence of the D^2 -BAC approach is to allow for partial solves of the integer subproblems, so that ultimately the partial solves start to yield optimal solutions. This is done by specifying the maximum number of nodes to explore in the branch-and-bound tree for solving the subproblem MIP.

The fundamental insight in this approach is the observation that a branch-and-bound (B&B) tree together with the LP relaxations at the nodes embodies a disjunction and provides important information that can be used in approximating subproblem MIP value functions. By using the disjunctive cut principle [4], [18] obtain linear inequalities or cuts that are used to build value function approximations for the subproblem MIPs. Without providing further details on the algorithm, we now summarize the main modules of the procedure and give a flowchart in Figure 3.

Modules:

Step 1. Initialization.

Inputs: Instance data, run parameters (e.g. stopping tolerance ϵ).

Output: Initialized master problem and its solution x^k , upper bound $V_0 = \infty$, $k = 1$.

Step 2. SubLP-Solve(ω). Given x^k evaluate $f_c^k(x^k, \omega)$ defined in problem (7) for each $\omega \in \Omega$.

Inputs: x^k .

Outputs: If solution $y(\omega)$ satisfy integer restrictions for all $\omega \in \Omega$, output $\alpha^k(\omega), \beta^k(\omega)$, and upper bound V^k . Otherwise, output disjunction variable $j(k)$, and $E[y(\tilde{\omega})]$.

Step 3. C3SLP-Solve. Form and solve the C^3 -SLP.

Inputs: $j(k)$, and $E[y(\tilde{\omega})]$.

Outputs: Coefficients π^k and multipliers λ_0^k and λ_1^k .

Step 4. *RHSLP-Solve*(ω). Form and solve the *RHS*-LP for each $\omega \in \Omega$.

Inputs: x^k , λ_0^k and λ_1^k .

Outputs: $\pi_c^k(x, \omega)$.

Step 5. *TB&B-Solve*(ω). Update subproblem (7) and re-solve as an MIP using TB&B for

each $\omega \in \Omega$.

Inputs: x^k , π^k , $\pi_c^k(x, \omega)$ and number of nodes to explore in the TB&B tree.

Outputs: Dual solutions at each node of the TB&B tree, and if solution $y(w)$ satisfy integer restrictions for $\omega \in \Omega$, output upper bound V^k .

Step 6. *ERPLP-Solve*(ω). Form and solve the *ERP*-LP for each $\omega \in \Omega$.

Inputs: Dual solutions at each node of the TB&B tree for each $\omega \in \Omega$.

Outputs: $\alpha^k(\omega)$ and $\beta^k(\omega)$ for all $\omega \in \Omega$.

Step 7. *UpdateMasterMIP-Solve*. Add a cut and solve problem (4).

Inputs: α^k , β^k .

Outputs: x^{k+1} , lower bound v^k .

Step 8. *Termination*.

If $V^k - v^k \leq \epsilon$, stop. Otherwise, $k \leftarrow k + 1$, repeat from Step 2.

Figure 3 is similar to that for the D^2 algorithm (Figure 2) but now with two additional modules, *TB&B-LP-Solve* and *ERPLP-Solve*. The module *TB&B-LP-Solve* involves performing a TB&B solve of each scenario subproblem. Using the dual information from each nodal subproblem LP relaxation for a given scenario, the so called epi-reverse polar LP (Problem 19 in [17]) or *ERP*-LP is formed and solved in the *ERPLP-Solve* module. The solutions from the *ERP*-LP solves for all the scenarios are used in generating a Benders'-type optimality cut to add to the master program.

3.4. Design of the Testbed

We exploit the commonalities among the algorithms and present an object-oriented (e.g. [6]) implementation in which the algorithms share classes and data structures, thus exploiting the power of object-oriented programming. Figure 4 shows a UML class diagram for an object-oriented implementation of the decomposition algorithms. In the figure we only show classes that are relevant to the main modules of the algorithms described in the previous

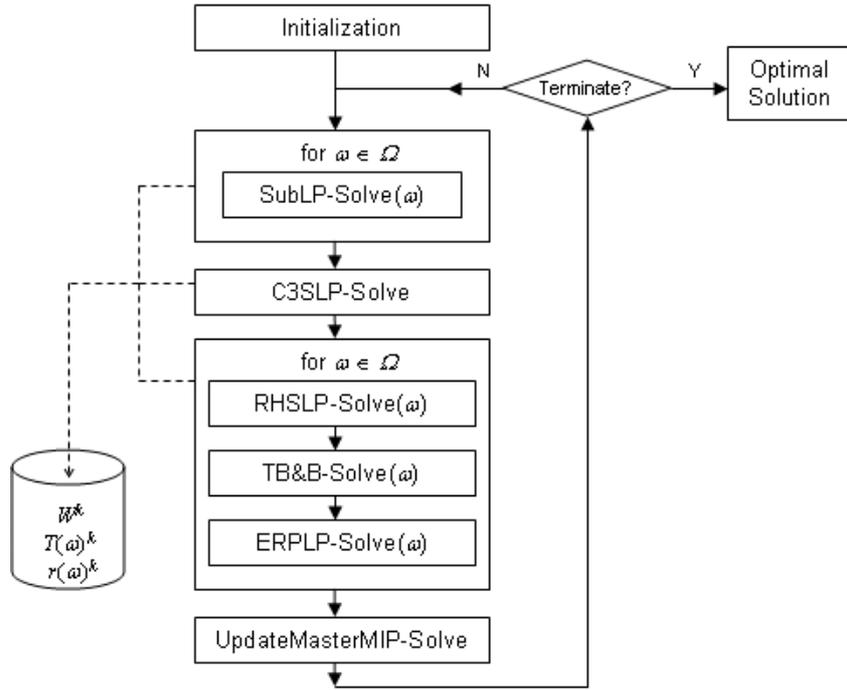


Figure 3: The D^2 -BAC Algorithm

three subsections. Classes such as those dealing with data input and solution output are omitted for ease of exposition.

As shown in Figure 4, the decomposition algorithms involve six classes corresponding to the MIP/LP optimization tasks, namely, *Master-MIP*, *Sub*, *C3-SLP*, *RHS-LP*, *TBB-LP*, and *ERP-LP*. All these classes inherit from a superclass, *LPObjectClass*, which has the program variables and methods common among all the derived classes. Two additional classes, *Sub-MIP* and *Sub-LP*, are subclasses of the superclass *Sub*, which has the program variables and methods common between the two classes. Finally, each algorithm (L^2 , D^2 and D^2 -BAC) uses only the classes that it requires. For example, the L^2 algorithm requires the *Master-MIP* class, which is derived from the *LPObjectClass*, and the *Sub-MIP* class, which is derived from the *Sub-LP* class. Next we point out some important implementation issues to consider for future implementation of decomposition algorithms for SCO.

3.5. Algorithmic Implementation Issues

[12] reports on an implementation of the three decomposition algorithms in the C programming language using the general-purpose programming system ILOG CPLEX 7.0 [9] for solving all the LP and MIP problems. Due to computational efficiency and memory considerations, all the problem matrix data such as A , W^k , $r^k(\omega)$ and $T^k(\omega)$ are stored in sparse

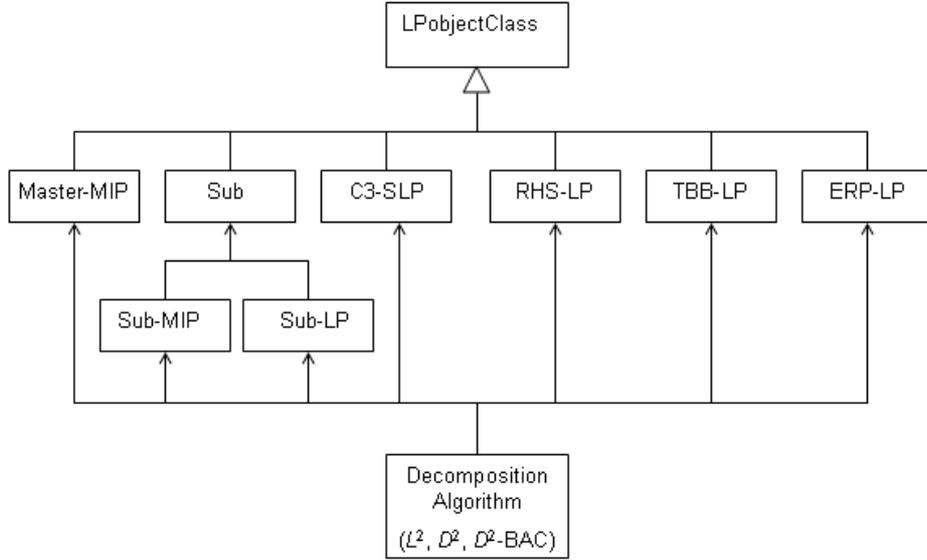


Figure 4: Class Diagram for the Three Decomposition Algorithms

matrix format. The data W^k , $r^k(\omega)$ and $T^k(\omega)$ grow at every iteration of the algorithm whenever a D^2 cut is generated. Therefore, care must be taken in how the data is stored in order to avoid program inefficiency and memory problems.

The master problem is stored as a CPLEX LP object to which optimality cuts are added at every iteration k of the algorithm. Similarly, the second-stage scenario subproblem LP and MIP are kept as CPLEX LP and MIP objects, respectively, but the right hand side ($r^k(\omega) - T^k(\omega)$) is computed and reset for each scenario $\omega \in \Omega$ before optimization. The D^2 cut coefficients π^k are appended to the subproblem MIP/LP object as well as the matrix array W^{k-1} , stored separately. This matrix is needed for forming the C^3 -SLP (problem 18 in [17]). The right hand side coefficients at iteration k are added to the arrays $r^{k-1}(\omega)$ and $T^{k-1}(\omega)$ for each scenario $\omega \in \Omega$, respectively.

A CPLEX LP object for the C^3 -SLP (problem 18 in [17]) is created only when needed and freed immediately after optimizing the SLP and getting the optimal solution. To select a disjunction variable the implementation scans the scenarios in order and chooses a variable whose solution is the most fractional for the first scenario with a fractional solution. That is, for algorithmic iteration k and the first scenario ω with a fractional solution $y(\omega)$, the disjunction variable index $j(k)$ is determined by $\{j(k) = j : \operatorname{argmin}_j \{|y_j(\omega) - 0.5|\}\}$.

In order to guarantee algorithmic convergence, the C^3 -SLP must be formed with the

constraint matrix composed of the original constraint matrix W and all the π 's that were generated by using those disjunction variables whose indices are smaller than the index for the disjunction variable chosen in this iteration (see [17] for the proof of convergence). All the other π 's are excluded from the C^3 -SLP constraint matrix. This requires keeping track of the disjunction variable at which each cut is generated by using an array to store the disjunction variable index for each π^k . Not abiding by the steps in the proof of convergence for the D^2 algorithm can lead to an implementation of the algorithm that does not converge to the optimal solution.

In forming the C^3 -SLP objective function coefficients when the second-stage LP solutions do not satisfy integer requirements for at least one scenario in problem (18) in [17], the authors suggest using the expected value of the solutions. However, to guarantee that the D^2 cut generated actually cuts off the current fractional solution, we recommend using a conditional expectation with respect to the scenarios with a fractional solution for the selected disjunction variable. This is critical to the algorithm since the D^2 cut generated and added to the scenario subproblem influences the choice of the first-stage solution via the optimality cut (4c).

Similar to the creation of the CPLEX LP object for the C^3 -SLP, the CPLEX LP object for the RHS -LP (problem 19 in [17]) is dynamically created for each scenario, optimized, and then freed at each iteration of the algorithm. Thus this avoids unnecessary use of computer memory to store potentially several CPLEX LP objects. All the appropriate solution data from each CPLEX LP are extracted before freeing the object. Some implementation issues dealing with the D^2 -BAC algorithm are pointed out in the next section. Here we note that implementation of the D^2 -BAC algorithm is much more involved and requires the implementation of a truncated branch-and-bound procedure for solving (or partially solving) the scenario subproblem MIPs.

4. Computational Experience

We now report on our computational experience in applying the decomposition algorithms to large-scale two-stage SCO problem instances from server location under uncertainty and strategic supply chain planning under uncertainty. The experimental plan was aimed at studying the three decomposition algorithms for SCO as regards to among other things, exploring alternative implementation issues which might lead to more efficient implemen-

tations in the future, establishing benchmarks for serial processing, and investigating the behavior of algorithms, especially as it relates to the scalability of the methods. Scalability deals with how change in problem instance size affects the performance of the algorithm. We note here that the performance of each algorithm is also dependent on the type of problem being solved.

The experiments were designed so that a comparison of some of our computational results with those obtained by the general-purpose programming system ILOG CPLEX 7.0 [9] applied to the DEP formulation (3) is made. All the experiments were conducted on a Sun 280R with 2 UltraSPARC-III+ CPUs running at 900 MHz. The problem instances were run to optimality or stopped when a CPU time limit of 10,800 seconds (3hrs) was reached. The large CPU times are indicative of the large-scale nature and difficulty of solving these problems. The CPLEX LP/MIP solver was used to optimize the subproblem LPs/MIPs in the decomposition algorithms. In addition, the CPLEX MIP solver was used to solve the large-scale DEP formulation for each of the two-stage problem instances if possible as a benchmark. To get the best CPU times for the DEP instances, the CPLEX parameters were set at the following values based on preliminary testing: “set mip emphasis 1” (emphasizes looking for feasible solutions), “set mip strategy start 4” (uses barrier at the root), and “branching priority order on x ” (first branches on any fractional component of x before branching on y).

4.1. Server Location Under Uncertainty

We first consider the stochastic server location problem (SSLP) introduced in [13]. SSLPs involve the placement of “servers” at potential locations in a network at some given cost to provide enough capacity to serve up to a given amount of resource to potential “clients” who pay for the resource and whose availability is random. The names *server* and *client* are used in a generic sense because of the variety of application domains for SSLP (see for example [20] and [14]). The first-stage strategic decision in the SSLP is to select where to locate the servers and requires that only one server be installed at each location, and that the total client resource demand cannot exceed the server capacity. Otherwise, penalty costs are incurred for unmet resource demand. The second-stage operational/tactical decision is a resource allocation problem that requires each client to be served by only one server to meet the demand. The goal of the SSLP is to choose locations of servers and client-server

assignments that maximize the total expected revenue is the face of uncertainty in client availability.

4.1.1. Computational Results

A computational experiment to assess the performance of the D^2 algorithm on SSLP instances with replications was conducted. Five replications for SSLP with a fixed number of server locations (m), potential clients (n) and scenarios, were randomly generated with each problem instance generated as described in [13]. To ensure independence of the problem instances, different random seeds were used for generating all the random data for all the problem instances. The problem instances are named $SSLP_{m.n.S}$, where m is the number of potential server locations, n is the number of potential clients, and S is the number of scenarios. We report the average (Avg) algorithmic iterations and the average CPU times in seconds for the five replications. We also report the minimum (Min) and maximum (Max) CPU times. Table 1 gives the deterministic equivalent problem (DEP) and subproblem dimensions.

Table 1: SSLP Instance Dimensions

Instance	DEP			SUBPROBLEM		
	Constrs	Bins	Cvars	Constrs	Bins	Cvars
SSLP5.25.50	1,501	6,255	250	30	130	5
SSLP5.25.100	3,001	12,505	500	30	130	5
SSLP5.50.50	2,751	12,505	250	55	255	5
SSLP5.50.100	5,501	25,005	500	55	255	5
SSLP10.50.50	3,001	25,010	500	60	510	10
SSLP10.50.100	6,001	50,010	1,000	60	510	10
SSLP10.50.500	30,001	250,010	5,000	60	510	10
SSLP10.50.1000	60,001	500,010	10,000	60	510	10
SSLP10.50.2000	120,001	1,000,010	20,000	60	510	10
SSLP15.45.5	301	3,390	75	60	690	15
SSLP15.45.10	601	6,765	150	60	690	15
SSLP15.45.15	901	10,140	225	60	690	15
SSLP15.45.20	1201	13,515	300	60	690	15
SSLP15.45.25	1501	16,890	375	60	690	15

4.1.2. Experiment with the L^2 Method

We applied the L^2 method to the SSLP instances and Table 2 gives the computational results. The L^2 algorithm and the CPLEX MIP solver applied to the deterministic equivalent

problem (DEP) formulation (3) fail to solve the larger instances within the time limit. In particular, the CPLEX MIP solver has smaller CPU times than the L^2 algorithm on the first four (smaller) problem instances. However, unlike CPLEX applied to the DEP, the L^2 algorithm is able to solve SSLP10.50.50 and SSLP10.50.100 to optimality. The column ‘Gap DEP’ gives the % gap reported by CPLEX upon termination of the algorithm at the time limit.

Table 2: L^2 Computational Results for SSLP Instances

Instance	Iters	CPU(secs)				Gap
	Avg	Min	Max	Avg	DEP	DEP
SSLP5.25.50	32.0	1.97	2.59	2.34	4.58	
SSLP5.25.100	32.0	3.73	4.86	4.33	14.69	
SSLP5.50.50	32.0	168.27	181.04	174.66	10.35	
SSLP5.50.100	32.0	561.67	586.74	568.87	33.25	
SSLP10.50.50	1024.0	2909.32	4823.52	3369.05	>10,000	0.44%
SSLP10.50.100	1024.0	3149.29	3955.28	3532.49	>10,000	9.02%
SSLP10.50.500	1024.0	8725.97	>10,000	9814.98	>10,000	38.17%
SSLP10.50.1000	1024.0	>10,000	>10,000	>10,000	>10,000	99.60%
SSLP10.50.2000	1024.0	>10,000	>10,000	>10,000	>10,000	46.24%
SSLP15.45.5	146.0	>10,000	>10,000	>10,000	>10,000	1.19%
SSLP15.45.10	454.0	>10,000	>10,000	>10,000	>10,000	0.27%
SSLP15.45.15	315.8	>10,000	>10,000	>10,000	>10,000	0.72%
SSLP15.45.20	483.3	>10,000	>10,000	>10,000	>10,000	2.00%
SSLP15.45.25	486.6	>10,000	>10,000	>10,000	>10,000	3.39%

L^2 algorithm stopping tolerance used is 0.001% Gap.

4.1.3. Experiment with the D^2 Method

The computational results for the experiment are reported in Table 3. As shown in the table the number of D^2 iterations, D^2 cuts and average CPU times increase with the number of scenarios. In general the D^2 algorithm performs over two orders of magnitude better the L^2 algorithm in terms of computation time. The superior performance of the D^2 method can be attributed to the C^3 theorem [17] and the fact that scenario subproblems MIP solves are not performed at every iteration of the algorithm, but only when necessary.

Table 3 provides more information than the computational study reported in [13]. Essentially, the above table is more reliable because we report results based on replicated runs, and moreover, the last four rows which lead to larger first-stage problems, were not reported in [13]. However, scalability with problem size is not as favorable. As can be seen in the table the D^2 algorithm could not solve the last problem instance within the time limit.

We also note that there is some variability in computation times among problem replications as indicated by the Min and Max CPU times.

Since the number of D^2 cuts increase with the number of algorithmic iterations, our computational experience showed that the proliferation of redundant cuts may actually have an adverse effect on the scenario subproblem computation time. Therefore, work to devise a procedure for dropping D^2 cuts that have become redundant over the course of the algorithm still remains to be done. We believe that this would potentially improve on the scenario subproblem computation times, especially when solving scenario subproblem MIPs for upper bounding. In our implementation we solved the original scenario subproblem MIPs without the D^2 cuts when it was necessary to perform upper bounding. This provided better computation times.

Table 3: D^2 Computational Results for SSLPs with Replications

Problem	D^2 Iters	D^2 Cuts	D^2 CPU Time (secs)				Gap
	Avg	Avg	Min	Max	Avg	DEP	DEP
SSLP5.25.50	20.6	10.0	0.53	0.82	0.73	4.58	
SSLP5.25.100	20.6	13.0	1.03	1.84	1.48	14.69	
SSLP5.50.50	25.0	5.8	0.68	1.64	0.98	10.35	
SSLP5.50.100	25.6	10.6	1.25	3.95	1.89	33.25	
SSLP10.50.50	233.4	229.4	138.71	295.95	228.89	>10,000	0.44%
SSLP10.50.100	243.0	240.6	228.68	480.00	318.12	>10,000	9.02%
SSLP10.50.500	298.4	297.2	1616.12	1902.20	1753.88	>10,000	38.17%
SSLP10.50.1000	298.4	297.2	3307.67	5410.10	3948.13	>10,000	99.60%
SSLP10.50.2000	308.6	307.6	8530.37	9571.04	8975.60	>10,000	46.24%
SSLP15.45.5	147.0	136.5	58.94	181.53	119.19	>10,000	1.19%
SSLP15.45.10	303.4	297.2	1306.46	2988.65	1930.00	>10,000	0.27%
SSLP15.45.15	739.3	738.3	5244.14	7210.63	6208.23	>10,000	0.72%
SSLP15.45.20	843.8	841.6	3994.73	7875.52	5883.52	>10,000	2.00%
SSLP15.45.25	1028.8	1028.8	>10,000	>10,000	>10,000	>10,000	3.39%

D^2 algorithm stopping tolerance used is 0.001% Gap.

4.1.4. Computational Experiment with the D^2 -BAC Method

The final computational experiment involves the application of the D^2 -BAC algorithm to the SSLP instances with replications. We conducted preliminary experiments to assess the performance of the algorithm by varying the number of nodes to explore in the truncated branch-and-bound (TB&B) tree. The current implementation of the D^2 -BAC algorithm uses a TB&B tree with a breadth-first strategy with node selection always favoring the node with the best objective value.

We report on a preliminary computational experiment in which the maximum number of nodes to explore in the TB&B tree was set at 3 and branch-and-bound was activated when gap between the lower and upper bounds were below 10%. Whenever there was no significant improvement in the lower bound ($< 0.001\%$) for two consecutive iterations of the algorithm, the branch-and-bound process was activated. Otherwise, no branch-and-bound was performed. Table 4 shows the results of the experiment.

Table 4: Preliminary Computational Results for SSLPs Using the D^2 -BAC Algorithm

Problem	D^2 Iters	D^2 Cuts	Total Nodes	D^2 -BAC CPU Time (secs)		
	Avg	Avg		Min	Max	Avg
SSLP5.25.50	21.4	11.6	601.6	0.68	1.08	0.81
SSLP5.25.100	21.8	14.4	1506.4	1.13	1.96	1.70
SSLP5.50.50	24.0	4.2	210.4	0.59	0.96	0.75
SSLP5.50.100	23.6	5.6	560.4	1.15	1.52	1.34
SSLP10.50.50	212.7	201.3	14053.3	169.66	413.25	327.25
SSLP10.50.100	229.5	225.3	25989.5	232.65	861.01	418.88
SSLP10.50.500	257.5	251.5	169859.5	1077.14	5960.12	2325.21
SSLP10.50.1000	236.8	234.8	235140.5	2356.96	2584.76	2473.09
SSLP10.50.2000	147.0	147.0	617898.0	> 10,000	> 10,000	> 10,000
SSLP15.45.5	233.5	197.5	2267.5	162.53	544.54	353.54
SSLP15.45.10	211.5	193.0	3343.0	172.83	2217.00	1194.92
SSLP15.45.15	325.3	316.7	5414.0	157.85	3327.16	1609.73
SSLP15.45.20	333.8	329.8	6836.5	488.88	4027.09	1871.23
SSLP15.45.25	430.8	417.5	13211.0	1652.51	>10,000	5141.48

D^2 -BAC algorithm stopping tolerance used is 0.001% Gap.

The D^2 -BAC algorithm performs better than both the L^2 algorithm and the CPLEX MIP solver applied to the DEP. However, compared to the D^2 algorithm, the results show slightly increased computation times for some problem instances and decreased computation times for others. In particular, we see decreased computation times for the last four instances, which have higher first-stage dimensions. In this case it pays off to do the TB&B because the approximations obtained from the second stage are better. We note that for one of the SSLP.15.45.25 replications the D^2 -BAC algorithm had a CPU time slightly greater than 10,000 secs as shown under the ‘‘Max’’ column. Also, note that the D^2 -BAC algorithm could not solve SSLP10.50.2000 within the specified computation time limit. The optimality gap at the point of termination was still 100%. This problem instance has the largest number of scenarios and in this case, the results seem to indicate that it does not pay off to do TB&B. But, we may also need to fine tune the rules under which TB&B is undertaken. We also note that the number of algorithmic iterations as well as the number of D^2 cuts added generally decrease with problem size compared to the D^2 algorithm. Further investigation is needed

in order to fully characterize the performance of the D^2 -BAC algorithm, especially in trying to use the number of nodes (in a TB&B tree) in an adaptive manner.

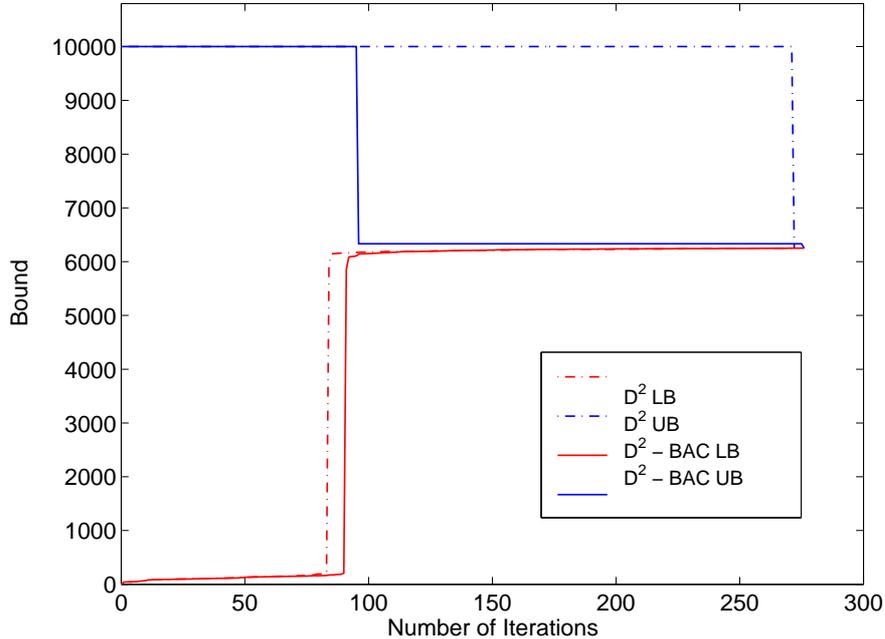


Figure 5: Convergence of the D^2 and D^2 -BAC Algorithms for SSLP10.50.100

The convergence of upper and lower bounds for the D^2 and the D^2 -BAC algorithms for a problem instance SSLP10.50.100 are given in Figure 5. We note that the bounds have been translated so that they are nonnegative. As can be seen in the figure, the lower bound increases close to the optimal value in well less than half the total number of iterations for both algorithms. This happens a little earlier for the D^2 algorithm than for the D^2 -BAC algorithm. However, good upper bounds are calculated much earlier for the D^2 -BAC algorithm due to TB&B for each scenario subproblem, which seem to generate optimality cuts that cause the first-stage solution to stabilize much faster. For the D^2 algorithm good upper bounds are calculated only after first-stage solutions stabilize, usually in the final iteration of the algorithm. After finding improved lower bounds both methods continue for the remaining iterations without changing the lower bound significantly. Nevertheless, due to early upper bounding, the D^2 -BAC method has a smaller percent gap earlier than that recorded for the D^2 algorithm.

We should caution that our conclusions regarding the D^2 -BAC method are preliminary. The implementation of this method calls for a variety of decisions many of which affect its performance. In the absence of a thorough study of the impact of these choices on the

performance of the algorithm it is premature to conclusively declare the superiority of D^2 over D^2 -BAC. Further algorithmic tests are necessary to identify fruitful ways of speeding up the D^2 -BAC implementation.

4.2. Strategic Supply Chain Planning Under Uncertainty

We now consider the two-stage stochastic programming approach for SSCh [3] and apply the algorithms towards solving this class of problems. Other related work in this area include [7], [11] and [1]. The essence of supply chain planning consists of determining the plant location, plant sizing, product selection, product allocation among plants and vendor selection for raw materials. The uncertain parameters include product net price and demand, raw material supply cost and production cost. The objective is to maximize the expected profit over a given time horizon for the investment depreciation and operations costs.

The two-stage stochastic supply chain planning problem [3] that we consider has the strategic decisions made in the first-stage while the operational or tactical decisions are made in the second-stage. The first-stage is devoted to strategic decisions (binary decisions) about plants sizing, product allocation to plants and raw materials vendor selection. The second-stage deals with making tactical decisions (mixed-binary) about the raw material volume supply from vendors, product volume to be processed in plants, and stock volume of product and raw materials to be stored in plants and warehouses. Further, tactical decisions include component volume to be shipped from plants to market sources at each time period along the time horizon. All the tactical decisions are made based on the supply chain topology decided in the first-stage. In making the strategic decisions in the first-stage it is assumed that the information on the strategic decision costs and constraints is known. However, the information on the tactical decision costs/revenue and constraints is not known *a priori*. For example there may be randomness in the cost of product/raw materials and in the demand at different markets for selling the final products.

4.2.1. Computational Results

The stochastic SSCh test set consists of seven of the ten problem instances reported in [3], where they apply a branch-and-fix coordination (BFC) [2] approach to the problem instances. This approach follows a scenario decomposition of the problem where the constraints are modelled by a splitting variables representation via the scenarios. The branch-and-fix coordination approach allows for coordinating the selection of the branching nodes and branching

variables in the scenario subproblems to be jointly optimized. The instances used in [3] have the following dimensions: 6 plant/warehouses, 3 capacity levels per plant, 12 products, 8 subassemblies, 12 raw materials, 24 vendors, 2 markets per product, 10 time periods, and 23 scenarios. We refer the reader to the given reference for further details on the problem instances. For completeness, we restate the characteristics of the deterministic model problem instances in Table 5 as reported in [3]. The columns of the table are as follows: “Constrs” is the number of constraints, “Bins” is the number of binary decision variables, “Cvars” is the number of continuous decision variables, and “Dens(%)” is constraint matrix density.

Table 5: Stochastic SSCh Deterministic Model Dimensions

Case	Constrs	Bins	Cvars	Dens(%)
c1	3,388	107	2,937	0.103
c2	3,458	108	3,068	0.100
c3	3,145	103	2,663	0.112
c4	3,405	105	3,065	0.099
c6	3,145	103	2,663	0.112
c8	3,894	114	3,634	0.087
c10	3,101	103	2,533	0.114

Table 6: Stochastic SSCh First-Stage and Second-Stage Model Dimensions

Case	FIRST-STAGE		SECOND-STAGE		
	Constrs	Bins	Constrs	Bins	Cvars
c1	73	71	3315	36	2,937
c2	73	72	3385	36	3,068
c3	70	67	3075	36	2,663
c4	70	69	3335	36	3,065
c6	70	67	3075	36	2,663
c8	79	78	3815	36	3,634
c10	66	67	3035	36	2,533

Table 7: Stochastic SSCh DEP Instance Dimensions

Case	Constrs	Bins	Cvars	Total Vars
c1	76,318	899	67,551	68,450
c2	77,928	900	70,564	71,464
c3	70,795	895	61,249	62,144
c4	76,775	897	70,495	71,392
c6	70,795	895	61,249	62,144
c8	87,824	906	83,582	84,488
c10	69,871	895	58,259	59,154

The dimensions for the first-stage and second-stage are given in Table 6. As shown in the table the SSCh model has a lot of continuous decision variables in the second-stage. The dimensions of the stochastic SSCh DEP model for the 23 scenarios are given in Table 7. As can be seen in the table, the problem instances have thousands of constraints and continuous variables and hundreds of binary variables. Continuous artificial variables were added to the instances with high penalty costs (10^{12}) in the objective function in order to induce relatively complete recourse as required by the D^2 approach. However, inducing relatively complete recourse for problem cases c5, c7 and c9 was not possible.

Due to the sheer sizes of the instances the L^2 and D^2 -BAC methods could not close the gap between the lower and upper bound to below 80% within the allowed time. Note that it was futile to even attempt to solve the DEP instances using the CPLEX MIP solver. The bottleneck with the L^2 algorithm lies in solving the large MIP subproblem instances at every iteration of the algorithm. As for the D^2 -BAC method, performing the truncated branch-and-cut on the subproblems significantly slowed the algorithm. As [3] points out, the problem instances have large percent gaps between the LP relaxation and the integer solution values and coupled with the extremely high dimensions of the problem instances, it makes it unrealistic to pretend to prove solution optimality. Nevertheless, the D^2 method was able to solve the instances to below 5% of the lower and upper bounds at termination.

4.2.2. Experiment with the D^2 Method

Table 8 shows the main results of our computational experience. The table headings “ Z_{IP} BFC” and “% Diff” give the best objective value as determined by the BFC method of [3] and the percentage difference between the best objective values determined by the D^2 algorithm and the BFC algorithm, respectively. Note that the SSCh instances are maximization problems. The D^2 algorithm was terminated when the percent gap between the lower and upper bounds went below 5% and the lower bound remained relatively constant for several consecutive iterations. This was done because there was no further improvement in the lower bound even after running the algorithm for additional iterations. As shown in the table, the D^2 algorithm solves all the problem instances to below 5% optimality gap. The algorithm obtains relatively improved solution values compared to the ones reported in [3], even up to 10% gain in the case of c8. For cases c2 and c10 our algorithm achieves optimality, which has been proven for these two cases in [3]. However, note that the computation time for case c8 is very large, probably an indication of problem instance difficulty.

Table 8: Computational Results for Strategic SSCh Problem Instances

Case	$D^2 Z_{IP}$	Z_{IP} BFC	% Diff	D^2 Iters	D^2 Cuts	CPU	Gap
c1	184439.00	178366.79	3.29	184	177	4558.29	4.139%
c2	0.00	0.00*	0.00	68	57	1342.34	0.000%
c3	230268.10	224564.20	2.48	92	85	1179.48	4.461%
c4	201454.00	197487.36	1.97	160	149	3265.06	4.070%
c6	231368.93	226578.02	2.07	114	109	1642.74	4.650%
c8	100523.00	89607.39	10.86	186	180	9650.11	3.234%
c10	139738.36	139738.36*	0.00	87	81	1083.00	0.000%

*Optimality has been proven by Alonso-Ayuso et al. (2003).

Finally, let us mention that these authors have actually justified the use of the SP model for SSCh under uncertainty for the problem instances considered. They have shown that it is always beneficial to use the SP model instead of obtaining strategic decisions based on the average scenario parameters.

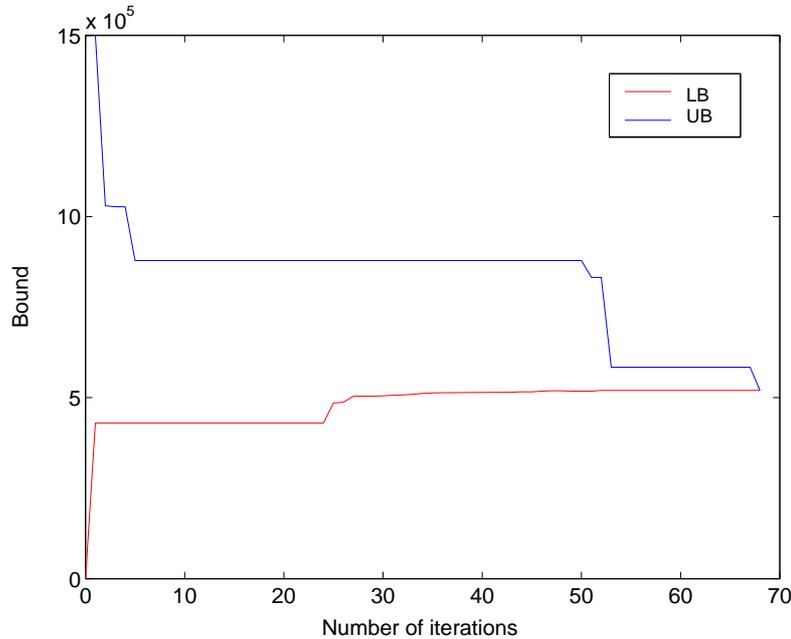


Figure 6: Convergence of the D^2 Algorithm for problem instance c2

Figures 6 and 7 show typical graphs of convergence of upper and lower bounds when applying the D^2 method to instances c2 and c3, respectively. Again the bounds have been translated so that they are nonnegative. As can be seen in Figure 6, the lower bound increases close to the optimal value in less than half the total number of iterations. However, good upper bounds are calculated only after first-stage solutions stabilize and this causes the method to continue for the remaining iterations without changing the lower bound sig-

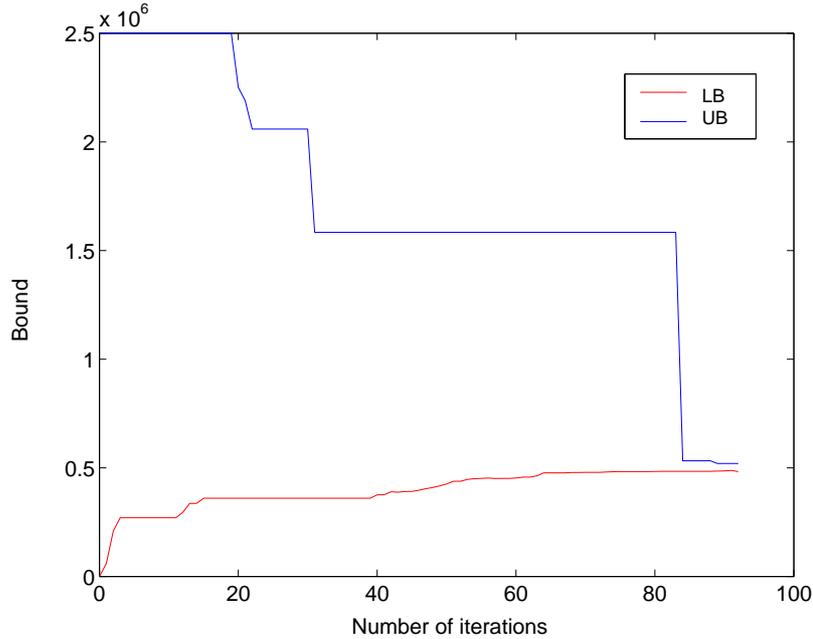


Figure 7: Convergence of the D^2 Algorithm for problem instance c3

nificantly. Once no changes are detected in the first-stage solution, a good upper bound is calculated by solving the MIP subproblems. Figure 7 shows similar results. Even though the gap could not be fully closed for c3, the generally fast convergence of upper and lower bounds for the algorithm is attractive.

5. Conclusions

In this paper, we have investigated the computational performance of three decomposition algorithms for SCO. Our experiments were conducted based on two choices: the algorithmic choice, and the problem class. An algorithmic testbed in which the commonalities among the algorithms are preserved while the algorithm-specific concepts are implemented in as efficient a manner as possible is presented. The testbed is used to study the performance of the algorithms with the two problem classes: server location under uncertainty and strategic supply chain planning under uncertainty. To date the solutions reported for the supply chain instances have been obtained by heuristic/approximation methods. The results reported in this paper provide computations for optimum-seeking methods for SCO. We have also reported on the insights related to alternative implementation issues leading to more efficient implementations, benchmarks for serial processing, and scalability of the methods. The computational experience demonstrates the promising potential of the disjunctive decomposition

approach towards solving several large-scale problem instances from the two different application areas. Furthermore, the study shows that convergence of the D^2 method for SCO is in fact attainable since the methods scale well with the number of scenarios. However, scalability with respect to the size of the first-stage problem is not clear at this point.

Acknowledgments

This research was funded by grants from the OR Program (DMII-9978780), and the Next Generation Software Program (CISE-9975050) of the National Science Foundation. The authors would like to thank Antonio Alonso-Ayuso and Laureano F. Escudero for providing the stochastic SSCh problem instances and for confirming our computational results, and the anonymous referees for their comments which helped improve upon an earlier version of the paper.

References

- [1] S. Ahmed, A.J. King, G. Parija, “A multi-stage stochastic integer programming approach for capacity expansion under uncertainty,” *J. of Global Opt.* vol. 26, pp. 3–24, 2003.
- [2] A. Alonso-Ayuso, L. F. Escudero, A. Garín, M. T. Ortuño, G. Perez, “BFC, A branch-and-fix coordination algorithmic framework for solving some types of stochastic pure and mixed 0-1 programs,” *Euro. J. of Oper. Res.* vol. 151, no. 3, pp. 503–519, 2003.
- [3] A. Alonso-Ayuso, L. F. Escudero, A. Garín, M. T. Ortuño, G. Perez, “An approach for strategic supply chain planning under uncertainty based on stochastic 0-1 programming,” *J. of Global Opt.* vol. 26, pp. 97–124, 2003.
- [4] E. Balas, “Disjunctive programming,” *Annals of Disc. Math.* vol. 5, pp. 3–51. 1979.
- [5] J. F. Benders, “Partitioning procedures for solving mixed-variable programming problems,” *Numerische Mathematik* vol. 4, pp. 238–252, 1962.
- [6] G. Booch, *Object-Oriented Analysis and Design with Applications*, 2nd ed., Benjamin/Cummings, Redwood City, CA, 1994.

- [7] L.F. Escudero, E. Galindo, E. Gómez, G. García, V. Sabau, “SCHUMAN, a modeling framework for supply chain management under uncertainty,” *Euro. J. of Oper. Res.*, vol. 119, pp. 13–34, 1996.
- [8] J.-P. Goux, J. Linderoth, and M. Yoder, “Metacomputing and the master-worker paradigm,” Preprint ANL/MCS-P792-0200, MCS Division, Argonne National Laboratories, Chicago, IL, 2000.
- [9] ILOG CPLEX. CPLEX 7.0 Reference Manual, ILOG CPLEX Division, Incline Village, NV, 2000.
- [10] G. Laporte, F. V. Louveaux, “The integer L-shaped method for stochastic integer programs with complete recourse,” *Oper. Res. Letters*, vol. 13, pp. 133–142, 1993.
- [11] S.A. MirHassani, C. Lucas, G. Mitra, C.A. Poojari, “Computational solution of capacity planning model under uncertainty,” *Parallel Comp. J.* vol. 26, no. 5, pp. 511–538, 2000.
- [12] L. Ntaimo, *Decomposition Algorithms for Stochastic Combinatorial Optimization: Computational Experiments and Extensions*, Ph.D. Dissertation, University of Arizona, Tucson, USA.
- [13] L. Ntaimo, S. Sen, “The million-variable ‘march’ for stochastic combinatorial optimization,” *J. of Global Opt.*, vol. 32, no. 3, pp. 385–400, 2005.
- [14] M. Riis, A.J.V. Skriver, J. Lodahl, “Deployment of mobile switching centers in a telecommunications network: A stochastic programming approach,” *Telecommunication Systems*, vol. 26, no. 1, pp. 93–109, 2004.
- [15] S. Sen, “Algorithms for stochastic mixed-integer programming models,” In: K. Aardal, G. Nemhauser, R. Weismantel (eds.) *Integer Programming Handbook*, Dordrecht, The Netherlands, Chapter 18, 2003
- [16] S. Sen, J. L. Hige, L. Ntaimo, “A summary and illustration of disjunctive decomposition with set convexification,” D. L. Woodruff, ed., *Stochastic Integer Programming and Network Interdiction Models*, Kluwer Academic Press, Dordrecht, The Netherlands. Chapter 6, pp. 105-123, 2002.

- [17] S. Sen, J.L. Hige, “The C3 theorem and a D2 algorithm for large scale stochastic mixed-integer programming: Set convexification,” *Math. Prog.*, vol. 104, no. 1, pp. 1–20, 2005.
- [18] S. Sen, H.D. Sherali, “Decomposition with branch-and-cut approaches for two stage stochastic mixed-integer programming,” *Math. Prog.*, vol. 106, no. 2, pp. 203–223, 2006.
- [19] B. Verweij, S. Ahmed, A.J. Kleywegt, G. Nemhauser, A. Shapiro, “The sample average approximation method applied to stochastic routing problems: a computational study,” *Comp. Opt. and Appl.* vol. 24, pp. 289–333, 2003.
- [20] Q. Wang, E. Batta, C. M. Rump, “Facility location models for immobile servers with stochastic demand,” *Naval Res. Logistics*, vol. 51, pp. 137-152, 2004.
- [21] R.J-B. Wets, “Stochastic programs with fixed recourse: the equivalent deterministic problem,” *SIAM Review*, vol. 16, pp. 309–339, 1974.