On a stochastic programming model for inventory planning*

Kai Huang and Shabbir Ahmed†
School of Industrial and Systems Engineering,
Georgia Institute of Technology, Atlanta, GA 30332, USA

July 28, 2004

Abstract

This paper considers a stochastic dynamic inventory problem involving a single item, linear cost structures, and finite distributions (but not necessarily independent) for the stochastic cost and demand parameters. We develop primal and dual algorithms for a multi-stage stochastic linear programming formulation for the problem. The complexity of the proposed algorithms is shown to be within $\mathcal{O}(N^2)$, where N is the number of nodes in the scenario tree used to model the stochastic parameters.

1 Introduction

This paper considers a stochastic extension of the finite horizon, single item, uncapacitated, dynamic inventory planning problem with linear costs:

$$\begin{array}{lll} \min & \sum_{n=1}^{T}(\alpha_n x_n + \beta_n I_n) & \\ \text{s.t.} & I_{n-1} + x_n = I_n + \delta_n & n = 1,\dots,T \\ & x_n, I_n \geq 0 & n = 1,\dots,T \\ & I_0 = 0, & \end{array} \tag{1}$$

where T is number of planning periods, x_n and I_n denote the production and ending inventory decisions for period n, respectively, and α_n, β_n and δ_n denote the per-unit production cost, holding cost and demand for the period n. It is well known that (1) can be easily solved using simple greedy schemes (cf. [6, 7]).

Beginning with the seminal work of Arrow et al. [2], stochastic inventory problems have been studied extensively (cf. [7]). Much of this work is based on specific assumptions on the underlying stochastic processes, to allow for elegant analytical solutions. Here we consider general, albeit finite, distributions for the stochastic parameters. In this situation, by using a scenario tree to model the evolution of the stochastic parameters, a stochastic extension of (1) can be formulated as a multi-stage stochastic program [3]. Such a stochastic programming formulation has been considered in [5], where the author shows that problem can be transformed to a network flow problem by introducing additional variables. In this paper, we consider a slightly different version of the stochastic inventory problem considered in [5], and develop a primal and a dual algorithm for the problem. We show that the complexity of the proposed algorithms is within $\mathcal{O}(N^2)$, where N is the number of nodes in the scenario tree used to model the stochastic parameters.

*This research has been supported by the National Science Foundation grant DMI-0133943.
†Corresponding author. E-mail: sahmed@isye.gatech.edu. 765 Ferst Drive, Atlanta, GA 30332, USA.

2 Stochastic Programming Formulation

To extend (1) to a stochastic setting, we use a scenario tree with T stages to describe the evolution of the uncertain data over the planning horizon. In this tree, the nodes in stage (or level) t of the tree constitute the states of the world that can be distinguished by information available up to time stage t. The probability associated with the state of the world represented by node n is q_n, and the time stage corresponding to node n is t_n. Each node n has a unique ancestor $a(n)$ except the root node. Each non-leaf node n is the root of a non-trivial subtree denoted by $\mathcal{T}(n)$, and $\mathcal{T} = \{1, ..., N\}$ represents the whole tree, where N is the total number of nodes in the tree. For node n, we define $\mathcal{P}(n)$ as the set of all nodes on the path from the root of $\mathcal{T}$ to node n (including node n and the root), and $\bar{\mathcal{P}}(n) = \mathcal{P}(n)\backslash\{n\}$. The stochastic problem parameters are then given by the sequence $\{\alpha_n, \beta_n, \delta_n\}_{n\in\mathcal{T}}$. With an objective of minimizing the *expected* total costs, a multi-stage stochastic programming extension of (1) is as follows:

$$\begin{array}{lll} \min & \sum_{n\in\mathcal{T}} q_n(\alpha_n x_n + \beta_n I_n) & \\ \text{s.t.} & I_{a(n)} + x_n = I_n + \delta_n & \forall\ n \in \mathcal{T} \\ & x_n, I_n \in \mathbb{R}^+ & \forall\ n \in \mathcal{T} \\ & I_0 = 0, & \end{array} \tag{2}$$

where I_0 is the initial inventory. We can reformulate (2) by introducing cumulative demand $d_n = \sum_{m\in\mathcal{P}(n)} \delta_m$ and eliminating the variables I_n using the identity $I_n = \sum_{m\in\mathcal{P}(n)} x_m - d_n$. The resulting formulation is:

$$\begin{array}{lll} \min & \sum_{n\in\mathcal{T}} c_n x_n - \bar{c} & \\ \text{s.t.} & \sum_{m\in\mathcal{P}(n)} x_m \geq d_n & \forall\ n \in \mathcal{T} \\ & x_n \in \mathbb{R}^+ & \forall\ n \in \mathcal{T}, \end{array} \tag{3}$$

where $c_n = q_n(\alpha_n + \frac{\sum_{m\in\mathcal{T}(n)} q_m\beta_m}{q_n})$ and $\bar{c} = \sum_{n\in\mathcal{T}} q_n\beta_n d_n$. Note that for each n, the computation of c_n is at most $\mathcal{O}(N)$, and so the complexity of the reformulation step is within $\mathcal{O}(N^2)$. The remainder of the paper will be concerned with formulation (3). We drop the constant term $\bar{c}$ from the objective function, and assume $c_n > 0$ and $d_n > 0$ for all $n \in \mathcal{T}$.

Although (3) is quite a simple problem, it often arises as the key substructure in more complicated planning problems, such as capacity planning under uncertainty [1]. This structure also arises in stochastic extensions of some classes of joint pricing-inventory problems, whose deterministic versions [4] involve the classical lot-sizing structure (1). The algorithms proposed next can be very effective within decomposition based methods for the problems mentioned above.

3 Algorithms

In this section, we propose a primal and a dual algorithm for the stochastic inventory problem (3). Our exposition relies on two different indexing systems for the nodes in the scenario tree $\mathcal{T}$.

Indexing scheme 1. The nodes in $\mathcal{T}$ are indexed $1, 2, \ldots, N$ in increasing order of their time stage, i.e., $t_1 \leq t_2 \leq \ldots \leq t_N$. No particular ordering is imposed on the indices of the nodes in the same time stage. Thus the root node has an index of 1.

Indexing scheme 2. The nodes in $\mathcal{T}$ are indexed $1, 2, \ldots, N$ in decreasing order of the corresponding cumulative demand, i.e., $d_1 \geq d_2 \geq \ldots \geq d_N$. If $d_m = d_n$, then $m < n$ if $t_m < t_n$.

The two indexing schemes corresponding to an example scenario tree are illustrated in Figure 1.

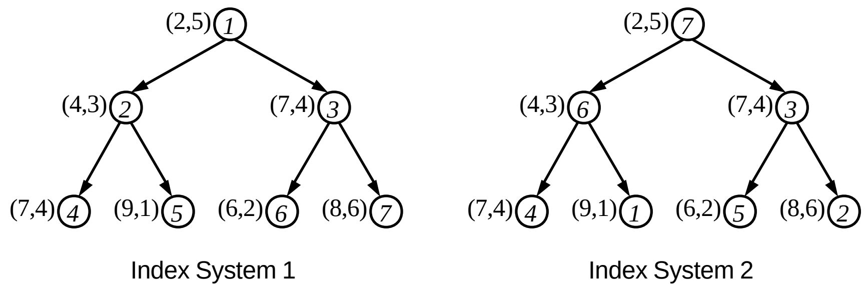

Figure 1: Indexing schemes. The numbers in parenthesis indicate (d_n, c_n).

The Primal Algorithm

We assume that the nodes are labelled according to indexing scheme 1. Note that we can construct a solution $x^0 = (x_1^0, x_2^0, ..., x_N^0)$ for (3) by setting

$$x_n^0 = \max\{0, d_n - \max_{m \in \bar{\mathcal{P}}(n)} d_m\}$$

for all $n = 1, \ldots, N$. It is easily verified that

$$\sum_{m \in \mathcal{P}(n)} x_m^0 = \max_{m \in \mathcal{P}(n)} d_m \geq d_n,$$

and therefore x^0 is a feasible solution to (3). The key idea of our primal algorithm is to start from node production levels given by the solution x^0, and then to shift some production from a group of nodes to their common ancestor, whenever the sum of the unit production costs of this node group is larger than that of the ancestor. This operation will be called "shifting-up." In each shifting-up operation, we will shift as much as possible to make at least one variable (node production level) change from positive to zero. With respect to a certain solution $x = (x_1, x_2, ..., x_N)$, we shall need the following notations to describe the algorithm:

$$\begin{aligned}
\mathcal{A}(n) &= \{m \in \mathcal{T}(n) \backslash \{n\} : x_m > 0,\ x_k = 0 \quad \forall\ k \in \bar{\mathcal{P}}(m) \backslash \mathcal{P}(n)\}, \\
s_n &= \sum_{m \in \mathcal{A}(n)} c_m, \qquad \text{and} \\
\Delta_n &= \min_{m \in \mathcal{A}(n)} x_m.
\end{aligned}$$

Note that $\mathcal{A}(n)$ is the set of closest descendants of n with positive production levels. This is the set of nodes from which production may be shifted up to node n. The primal scheme is detailed in Algorithm 1.

The algorithm first initializes the solution to x^0. Then, starting from a non-leaf node k with the largest index, the algorithm first compares the total production cost s_k of the nodes in $\mathcal{A}(k)$ with the production cost c_k of node k; then, if $s_k > c_k$, the algorithm shifts the minimum production Δ_k amongst the nodes in $\mathcal{A}(k)$ to node k. Figure 2 illustrates the primal algorithm for the example scenario tree in Figure 1. The first scenario tree in Figure 2 illustrates the initial solution. The next tree illustrates the iteration corresponding to node $k = 3$ (the non-leaf node with the largest index). Here $\mathcal{A}(3) = \{7\}$, $s_3 = 6$, $c_3 = 4$, and $\Delta_3 = 1$. Thus 1 unit of production is shifted up from node 7 to node 3. The next iteration considers node $k = 2$. Here $\mathcal{A}(2) = \{4, 5\}$, $s_2 = 5$, $c_2 = 3$, and $\Delta_2 = 3$. Thus 3 units of production are shifted from node 4 and node 5 to node 2. The remaining iterations are similar.

Algorithm 1 The Primal Algorithm

```
 1: set $x^*_n = \max\{0, d_n - \sum_{m \in \bar{\mathcal{P}}(n)} x^*_m\}$ for all $n = 1, \ldots, N$
 2: set $k = \max\{n \in \mathcal{T} : \mathcal{T}(n) \backslash \{n\} \neq \emptyset\}$.
 3: while $k \geq 1$ do
 4:   compute $\mathcal{A}(k)$, $s_k$ and $\Delta_k$.
 5:   if $c_k < s_k$ then
 6:     update the solution corresponding to the nodes in $\mathcal{A}(k) \cup \{k\}$ as follows:
        $x^*_m = \begin{cases} x^*_k + \Delta_k & \text{if } m = k \\ x^*_m - \Delta_k & \text{for all } m \in \mathcal{A}(k) \end{cases}$
 7:   else
 8:     set $k = k - 1$
 9:   end if
10: end while
11: return $x^*$
```

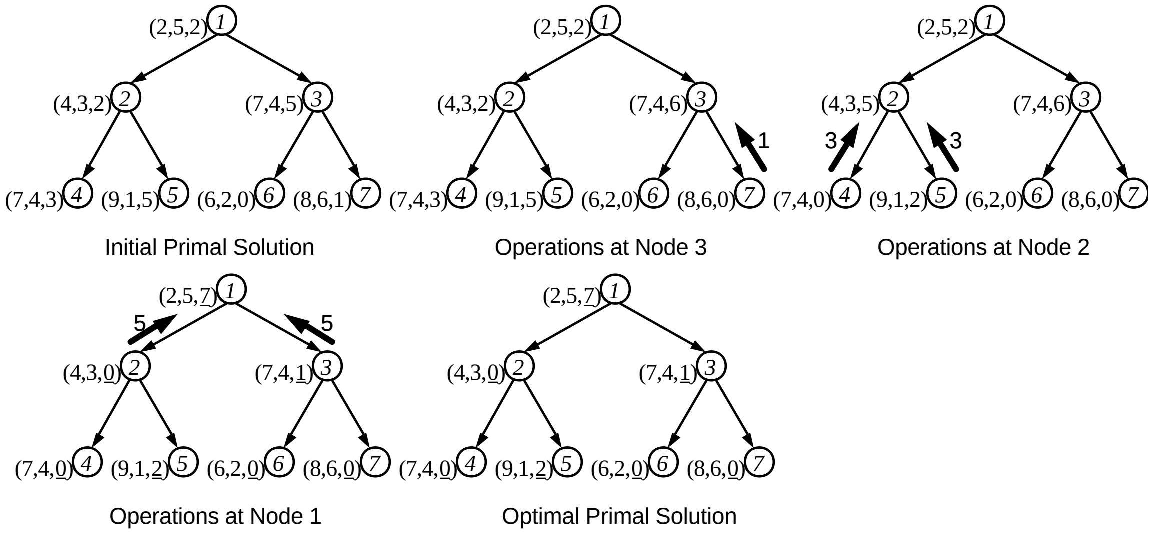

Figure 2: The Primal Algorithm. The numbers in parenthesis indicate (d_n, c_n, x^*_n).

The Dual Algorithm

Consider the dual of (3):

$$\begin{aligned} \max \quad & \sum_{n \in \mathcal{T}} d_n \pi_n \\ \text{s.t.} \quad & \sum_{m \in \mathcal{T}(n)} \pi_m \leq c_n \quad \forall\, n \in \mathcal{T} \\ & \pi_n \in \mathbb{R}^+ \qquad\quad \forall\, n \in \mathcal{T}. \end{aligned} \tag{4}$$

Here we propose an algorithm for solving (4). The algorithm is based on the following observation.

Lemma 3.1 *A solution $\pi = (\pi_1, \pi_2, ..., \pi_N)$ is feasible to (4) if and only if:*

$$0 \leq \pi_n \leq \min_{m \in \mathcal{P}(n)} \{c_m - \sum_{k \in \mathcal{T}(m) \backslash \{n\}} \pi_k\} \quad \forall\, n \in \mathcal{T}. \tag{5}$$

Furthermore, the second inequality is tight when π is optimal.

Proof. Note that for every $m \in \mathcal{P}(n)$, the corresponding row in (4) is $\sum_{k\in\mathcal{T}(m)} \pi_k \leq c_m$, i.e. $\pi_n + \sum_{k\in\mathcal{T}(m)\setminus\{n\}} \pi_k \leq c_m$. So we require that $\pi_n \leq c_m - \sum_{k\in\mathcal{T}(m)\setminus\{n\}} \pi_k$ for all $m \in \mathcal{P}(n)$. If we have a strict inequality for some n such that $\pi_n < \min_{m\in\mathcal{P}(n)}\{c_m - \sum_{k\in\mathcal{T}(m)\setminus\{n\}} \pi_k\}$, then all the constraints in which π_n appears are not tight and we can increase π_n to get a new feasible solution with greater objective value. $\square$

The dual algorithm is a greedy scheme, where we sort the dual variables according to decreasing objective function coefficients $\{d_n\}_{n\in\mathcal{T}}$, i.e., indexing scheme 2, and then set their values sequentially to make (5) tight. The scheme is detailed in Algorithm 2.

Algorithm 2 The Dual Algorithm

```
1: label the nodes of $\mathcal{T}$ according to the indexing scheme 2
2: set $\pi^*_n = 0$ for all $n = 1, \dots, N$
3: set $c^0_n = c_n$ for all $n = 1, \dots, N$
4: for $k = 1, ..., N$ do
5:     set $\pi^*_k = \min_{n\in\mathcal{P}(k)} \{c^{k-1}_n\}$
6:     set
       $c^k_n = \begin{cases} c^{k-1}_n - \pi^*_k & \text{for all } n \in \mathcal{P}(k) \\ c^{k-1}_n & \text{otherwise} \end{cases}$
7: end for
8: return $\pi^*$
```

Figure 3 illustrates the dual algorithm for the example in Figure 1. Note that the nodes are indexed according to scheme 2. The first tree illustrates the initial dual solution. The next tree illustrates the first iteration, where $k = 1$, and we set $\pi^*_1 = 1$. The costs on the nodes on $\bar{\mathcal{P}}(1) = \{6, 7\}$ are reduced to $c^1_6 = c^0_6 - \pi^*_1 = 3 - 1 = 2$ and $c^1_7 = c^0_7 - \pi^*_1 = 5 - 1 = 4$. The remaining iterations proceed similarly.

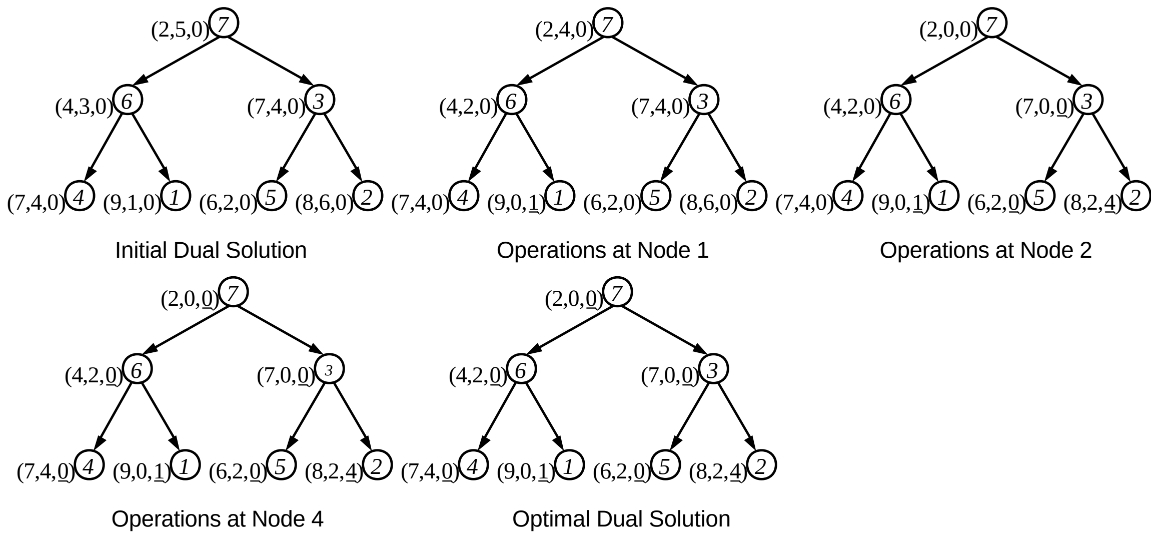

Figure 3: The Dual Algorithm.The numbers in parenthesis indicate (d_n, c^k_n, π^*_n).

4 Validity and complexity

Proof of Validity

The following results establish the correctness of the proposed algorithms. We assume that the nodes are indexed according to scheme 2.

Lemma 4.1 *At any iteration $k \in \{1, \ldots, N\}$ of the dual algorithm, the dual solution π^* satisfies*

$$\sum_{m \in \mathcal{T}^k(n)} \pi^*_m \leq c_n \qquad \forall\, n \in \mathcal{T}, \text{ and} \tag{6}$$

$$\sum_{m \in \mathcal{T}^k(n)} \pi^*_m = \sum_{m \in \mathcal{T}(n)} \pi^*_m = c_n \qquad \forall\, n \in \operatorname{argmin}_{m \in \mathcal{P}(k)} \{c^{k-1}_m\}, \tag{7}$$

where $\mathcal{T}^k(n) = \mathcal{T}(n) \cap \{1, 2, ..., k\}$.

Proof. By induction on k, it is easy to see that $c^k_n = c_n - \sum_{m \in \mathcal{T}^k(n)} \pi^*_m$ for all n. Also, since $n \in \mathcal{P}(k)$ if and only if $k \in \mathcal{T}(n)$, we have $\pi^*_k \leq c^{k-1}_n$ for all $n \in \mathcal{P}(k)$. Therefore, $c^k_n \geq 0$ for any $n \in \mathcal{T}$, and (6) follows. Furthermore, by construction, the algorithm ensures $\sum_{m \in \mathcal{T}^k(n)} \pi^*_m = c_n$ for all $n \in \operatorname{argmin}_{m \in \mathcal{P}(k)} \{c^{k-1}_m\}$. Since the feasibility constraints demand $\sum_{m \in \mathcal{T}(n)} \pi^*_m \leq c_n$, equation (7) then follows. □

In the following result we shall make use of the following notation. At any iteration k of the dual algorithm, let

$$m_k \in \operatorname{argmin}_{m \in \mathcal{P}(k)} \{c^{k-1}_m\}$$

such that

$$c^{k-1}_n > c^{k-1}_{m_k} \quad \forall\, n \in \mathcal{P}(k) \backslash \mathcal{P}(m_k).$$

That is, m_k is the closest node to k (on $\mathcal{P}(k)$) that minimizes c^{k-1}_m. Correspondingly, (7) holds for the node $n = m_k$.

Theorem 4.2 *The solution $x^* = (x^*_1, x^*_2, ...x^*_N)$ returned by the Primal algorithm and the solution $\pi^* = (\pi^*_1, \pi^*_2, ..., \pi^*_N)$ returned by the Dual algorithm are optimal for (3) and (4), respectively.*

Proof. First note that x^* is a feasible solution to (3). This is because the initial solution x^0 is feasible, and by construction, each shifting-up operation preserves feasibility by only shifting $\triangle_k = \min_{m \in \mathcal{A}(k)} x^*_m$ units. Lemma 4.1 guarantees the feasibility of the dual solution π^* (let $k = N$ in (6)). It remains to show that x^* and π^* satisfy complementary slackness, i.e., for all $n \in \mathcal{T}$:

$$\begin{array}{rcl} \pi^*_n > 0 & \Rightarrow & \sum_{m \in \mathcal{P}(n)} x^*_m = d_n \\ \sum_{m \in \mathcal{T}(n)} \pi^*_m < c_n & \Rightarrow & x^*_n = 0 \end{array} \tag{8}$$

We shall show that the above conditions hold by induction. We assume that the nodes are indexed according to scheme 2.

The base case: Consider node 1. We show that (8) holds for all nodes $n \in \mathcal{T}(m_1)$. By the definition of m_1, we have that $\pi^*_1 = c_{m_1}$. The dual constraint corresponding to node m_1 in (4) requires $\sum_{m \in \mathcal{T}(m_1)} \pi^*_m \leq c_{m_1}$. Therefore, for all $n \in \mathcal{T}(m_1) \backslash \{1\}$, we have $\pi^*_n = 0$ and $m_n = m_1$. Also, by our assumption of index system, we have $d_1 > d_n$ for all $n \in \bar{\mathcal{P}}(1)$ (otherwise n will be indexed 1). So in x^0 we must have $x^0_1 > 0$ and $\sum_{m \in \mathcal{P}(1)} x^0_m = d_1$. Furthermore, for all $n \in \mathcal{T}(1) \backslash \{1\}$, $x^0_n = 0$ since $d_n \leq d_1$. According to Primal algorithm, we have $\sum_{m \in \mathcal{P}(1)} x^0_m = \sum_{m \in \mathcal{P}(1)} x^*_m$ since we only shift quantity along the tree towards the root. If $m_1 \neq 1$ (notice $m_1 \in \mathcal{P}(1)$), observe that $c_n > c_{m_1}$ for all $n \in \mathcal{P}(1) \backslash \mathcal{P}(m_1)$. So in the Primal algorithm, when we process node m_1, all the productions of nodes $n \in \mathcal{P}(1) \backslash \mathcal{P}(m_1)$ will be shifted up, i.e., we must deplete any positive quantity along the path from 1 to m_1 (not including the latter). After we finish processing m_1, we have $x_n = 0$ for all $n \in \mathcal{P}(1)\backslash\mathcal{P}(m_1)$. Also, in the following iterations, the values of primal variables in $\mathcal{T}(m_1)\backslash\{m_1\}$ can not increase anymore (which are all zeroes). Therefore in final solution we have:

$$\begin{aligned} \textstyle\sum_{m\in\mathcal{P}(m_1)} x^*_m &= \textstyle\sum_{m\in\mathcal{P}(1)} x^0_m = d_1 \\ x^*_m &= 0 \quad \forall m \in \mathcal{T}(m_1)\backslash\{m_1\} \end{aligned}$$

The second equality comes from the first and the fact that $d_m \le d_1$ for all $m \in \mathcal{T}(m_1)\backslash\{1\}$. Now we can check condition (8). Note that the only positive dual variable in $\mathcal{T}(m_1)$ is π^*_1 and we have shown that $\sum_{m\in\mathcal{P}(1)} x^*_m = \sum_{m\in\mathcal{P}(m_1)} x^*_m = d_1$. Since $\sum_{m\in\mathcal{T}(m_1)} \pi^*_m = c_{m_1}$, we claim that $\{n \in \mathcal{T}(m_1) : \sum_{m\in\mathcal{T}(n)} \pi^*_m < c_n\} \subseteq \mathcal{T}(m_1)\backslash\{m_1\}$. The conclusion holds because $x^*_n = 0$ for all $n \in \mathcal{T}(m_1)\backslash\{m_1\}$.

The induction step: Assume that (8) holds for all nodes in $\mathcal{T}(m_1) \cup \mathcal{T}(m_2) \cup \ldots \cup \mathcal{T}(m_k)$. First, we define $\mathcal{H}(k) = \{1, 2, ..., k\}$, $\mathcal{R}(k) = \{m_n : n \in \mathcal{H}(k)\}$ and $\mathcal{F}(k) = \cup\{\mathcal{T}(m_n) : n \in \mathcal{H}(k)\}$. If $k+1 \in \mathcal{T}(m_n)$ for some $n \in \mathcal{H}(k)$, then there is no need to check $k+1$ since the corresponding conditions are already satisfied. If this is not the case, i.e., $k+1 \notin \mathcal{F}(k)$, we examine conditions (8) for nodes in $\mathcal{T}(m_{k+1})\backslash\mathcal{F}(k)$.

Note for any $\mathcal{T}(m)$ and $\mathcal{T}(n)$ ($m \ne n$), there are only three exclusive cases: $\mathcal{T}(m) \subset \mathcal{T}(n)$, or $\mathcal{T}(m) \subset \mathcal{T}(n)$, or $\mathcal{T}(m) \cap \mathcal{T}(n) = \emptyset$. In the third case, $\mathcal{T}(m)$ and $\mathcal{T}(n)$ form an independent pair. Without loss of generality, we can assume the trees $\mathcal{T}(m_1)$, $\mathcal{T}(m_2)$, ..., $\mathcal{T}(m_k)$ are pairwise independent. Notice for all $m \in \mathcal{P}(k+1)$ and $n \in \mathcal{H}(k)$, either $\mathcal{T}(m_n) \cap \mathcal{T}(m) = \emptyset$ or $\mathcal{T}(m) \supset \mathcal{T}(m_n)$ exclusively. Also, according to (6) and (7): $c_n \ge \sum_{m\in\mathcal{T}^k(n)} \pi^*_m$ for all $n \in \mathcal{P}(k+1)$, and $\sum_{m\in\mathcal{T}^k(n)} \pi^*_m = \sum_{m\in\mathcal{T}(n)} \pi^*_m = c_n$ for all $n \in \mathcal{R}(k)$. So:

$$\sum_{m\in\mathcal{T}^k(n)} \pi^*_m = \sum_{m\in\mathcal{T}(n)\cap\mathcal{F}(k)} \pi^*_m = \sum_{m\in\mathcal{T}(n)\cap\mathcal{R}(k)} c_m \le c_n \quad \forall n \in \mathcal{P}(k+1) \tag{9}$$

On the other hand, according to index system 2, we have $d_{k+1} > d_n$ for all $n \in \bar{\mathcal{P}}(k+1)$. Therefore, $x^0_{k+1} > 0$ and $\sum_{m\in\mathcal{P}(k+1)} x^0_m = d_{k+1}$. Furthermore, note for all $n \in \mathcal{H}(k)$ such that $m_n \in \mathcal{T}(k+1)$, we have $k+1 \in \bar{\mathcal{P}}(n)$, so $d_n > d_{k+1}$ (otherwise $k+1$ will be checked before n). For all $m \in \mathcal{T}(k+1)\backslash\mathcal{F}(k)\backslash\{k+1\}$, $x^0_m = 0$ since $d_m \le d_{k+1}$. Therefore, according to the Primal algorithm, we also have $\sum_{m\in\mathcal{P}(k+1)} x^*_m = d_{k+1}$. If $m_{k+1} \ne k+1$ (note $m_{k+1} \in \mathcal{P}(k+1)$), then according to the definition of m_{k+1}, for all $n \in \mathcal{P}(k+1)\backslash\mathcal{P}(m_{k+1})$ we have $\pi^*_{k+1} = c_{m_{k+1}} - \sum_{m\in\mathcal{T}(m_{k+1})\cap\mathcal{R}(k)} c_m < c_n - \sum_{m\in\mathcal{T}(n)\cap\mathcal{R}(k)} c_m$. Therefore:

$$c_{m_{k+1}} < \sum_{m\in(\mathcal{T}(m_{k+1})\backslash\mathcal{T}(n))\cap\mathcal{R}(k)} c_m + c_n \tag{10}$$

Combining (9) and (10), by the same reasoning as in the base case, we conclude that when we are processing node m_{k+1} in the Primal algorithm, we must deplete any positive quantity along the path $\mathcal{P}(k+1)\backslash\mathcal{P}(m_{k+1})$. That is, after we finish processing node m_{k+1}, we have $x_m = 0$ for all $m \in \mathcal{P}(k+1)\backslash\mathcal{P}(m_{k+1})$. In the final solution x^*, we have:

$$\begin{aligned} \textstyle\sum_{m\in\mathcal{P}(k+1)} x^*_m &= \textstyle\sum_{m\in\mathcal{P}(m_{k+1})} x^*_m = d_{k+1} \\ x^*_m &= 0 \quad \forall m \in \mathcal{T}(m_{k+1})\backslash\mathcal{F}(k)\backslash\{m_{k+1}\} \end{aligned}$$

The second equality comes from the first and the fact that $d_{k+1} \ge d_n$ for all $n \in \mathcal{T}(m_{k+1})\backslash\mathcal{F}(k)\backslash\{m_{k+1}\}$.

Note that $\mathcal{T}(m_{k+1})\backslash\mathcal{F}(k)$ is the collection of all nodes in $\mathcal{T}(m_{k+1})$ for which we need to check complementary slackness (all other nodes in it have been checked before). To verify condition (8), note that the only possible positive dual variable in this set is π^*_{k+1} and we have shown that $\sum_{m\in\mathcal{P}(k+1)} x^*_m = d_{k+1}$. Also note $\sum_{m\in\mathcal{T}(m_{k+1})} \pi^*_m = c_{m_{k+1}}$, so $\{n \in \mathcal{T}(m_{k+1})\backslash\mathcal{F}(k) : \sum_{m\in\mathcal{T}(n)} \pi^*_m < c_n\} \subseteq \mathcal{T}(m_{k+1})\backslash\mathcal{F}(k)\backslash\{m_{k+1}\}$. The conclusion then holds since $x^*_n = 0$ for all $n \in \mathcal{T}(m_{k+1})\backslash\mathcal{F}(k)\backslash\{m_{k+1}\}$. $\square$

Complexity

To compute the complexity of the proposed algorithms, we assume that the scenario tree is complete with T levels and B branches per non-leaf node.

Theorem 4.3 *The complexity of the Dual algorithm is* $\mathcal{O}(N \log N \log(\log N))$.

Proof. First, we sort the demands of N nodes, whose complexity is $N \log N$. Second, we have at most N minimization procedures. Each minimization will concern at most T numbers, whose complexity is $T \log T$. The updating will cost us at most T operations. Therefore, the total number of operations will be no more than $N \log N + N(T \log T + T)$. Notice the total number of nodes in the tree is $N = \sum_{t=0}^{T-1} B^t = \frac{B^T - 1}{B-1}$. So $T = \mathcal{O}(\log N)$ and $\mathcal{O}(N \log N) \leq \mathcal{O}(NT \log T)$. The complexity of the algorithm is no greater than $\mathcal{O}(NT \log T)$, or $\mathcal{O}(N \log N \log(\log N))$. $\square$

Before we can compute the complexity of the Primal algorithm , we need to describe in detail how to compute and update $\mathcal{A}$, s and Δ. Given any solution x^* at a particular iteration, let

$$\begin{array}{rcl} \mathcal{S}^0(n) & = & \{m \in \mathcal{T} : a(m) = n, x_m^* = 0\} \\ \mathcal{S}^+(n) & = & \{m \in \mathcal{T} : a(m) = n, x_m^* > 0\}. \end{array}$$

For the first time a node k is considered, $\mathcal{A}(k)$, s_k and Δ_k can be computed as follows:

$$\begin{array}{rcl} \mathcal{A}(k) & = & (\cup_{m \in \mathcal{S}^0(k)} \mathcal{A}(m)) \cup \mathcal{S}^+(k) \\ s_k & = & \sum_{m \in \mathcal{S}^+(k)} c_m + \sum_{m \in \mathcal{S}^0(k)} s_m \\ \Delta_k & = & \min\{\min\{x_m : m \in \mathcal{S}^+(k)\}, \min\{\Delta_m : m \in \mathcal{S}^0(k)\}\} \end{array}$$

Each time after we update the solution in node k, for each n in $\mathcal{A}(k)$ such that x_n decreases to 0, $\mathcal{A}(k)$, s_k and Δ_l for all $l \in (\bar{\mathcal{P}}(n) \backslash \mathcal{P}(k)) \cup \{k\}$ can be updated as follows:

$$\begin{array}{rcl} \mathcal{A}(k) & = & (\mathcal{A}(k) \backslash \{n\}) \cup \mathcal{A}(n) \\ s_k & = & s_k - c_n + s_n, \\ \Delta_l & = & \min\{\min\{x_m : m \in \mathcal{S}^+(l)\}, \min\{\Delta_m : m \in \mathcal{S}^0(l)\}\}. \end{array}$$

Note that Δ_l is updated according to the order of decreasing index:

Theorem 4.4 *The complexity of the Primal algorithm is* $\mathcal{O}(N^2)$.

Proof. First, to compute x^0, we need to consider N nodes, each of which needs at most T operations. So the complexity is $\mathcal{O}(NT)$. Second, in each iteration, the production in at least one node will be depleted, so the algorithm will end in N iterations. Third, for each iteration, if we compute $\mathcal{A}(k)$, s_k and Δ_k for the first time, the complexities are $\mathcal{O}(N)$, $\mathcal{O}(B)$ and $\mathcal{O}(B \log B)$, respectively. To update $\mathcal{A}(k)$ and s_k, we need $\mathcal{O}(N)$ and $\mathcal{O}(B)$ operations. To update the current solution, we need $\mathcal{O}(|\mathcal{A}(k)|)$ operations, which has an upper bound of $\mathcal{O}(N)$. Finally, we consider the effort in updating Δ. For each depleted node n in $\mathcal{A}(k)$, we need to update all Δ_l along the path from n to k, which has an upper bound of $\mathcal{O}(TBlogB)$. In each iteration, there can be several nodes in $\mathcal{A}(k)$ that are depleted, however, then the number of iterations will also decrease correspondingly. Therefore, the dominant operation is constructing and updating $\mathcal{A}(k)$, whose total complexity is $\mathcal{O}(N^2)$. $\square$

Therefore, the complexity of both algorithms is within $\mathcal{O}(N^2)$. Since (2) can be transformed to (3) in $\mathcal{O}(N^2)$ operations. So we can also solve (2) in $\mathcal{O}(N^2)$ operations.

5 Computational Results

Finally, we present some computational results using the primal and the dual algorithms. We consider a total of 18 scenario trees with the number of stages (T) varying from 5 to 13, and the number of branches (B) for each non-leaf node varying from 2 to 9. We use uniform distributions to generate the parameters (δ_n, α_n, β_n, q_n) for (2) corresponding to each node of the tree. Then we transform these parameters to the parameters (d_n, c_n) for (3). We generate 10 data sets corresponding to each of the 18 scenario trees. We compare the running times of the proposed Primal and Dual algorithms with that of the Simplex solver in CPLEX 8.1. All numerical experiments are conducted on an IBM PC with 1024 MB RAM and a PENTIUM4 1.6GHz processor.

Table 1 presents the results. The columns in the table are arranged according to the number of stages (T), the number of branches (B), total number of nodes (N), the actual running time of Simplex, the actual running time of the Primal algorithm, the running time of the Primal algorithm as a % of the running time of Simplex, the actual running time of the Dual algorithm, the running time of the Dual algorithm as a % of the running time of Simplex, respectively. The running times for a particular scenario tree are averages over 10 instances, and are in units of 10^{-3} CPU seconds. We can observe that the Dual algorithm is the fastest algorithm, and the Primal algorithm is the second. Both algorithms are much faster than the Simplex. In the worst case, the running times of Primal and Dual algorithms are 18.8 % and 1.8 % of that of the Simplex algorithm, respectively. On average, the running times of primal and dual algorithms are 10.8% and 0.4 % percent, respectively, of that of the simplex algorithm. These results are compatible with the complexity analysis.

No.	T	B	N	Simplex	Primal	%	Dual	%
1	8	2	255	17.1	1.6	9.36	0	0
2	9	2	511	35.9	6.1	16.99	0	0
3	10	2	1023	114	14.1	12.37	0	0
4	11	2	2047	420.4	57.7	13.73	4.8	1.14
5	12	2	4095	1212.5	228.2	18.82	10.8	0.89
6	13	2	8191	8499.9	790.8	9.30	28.4	0.33
7	6	3	364	28	3.2	11.43	0	0
8	7	3	1093	186	20.1	10.81	1.6	0.86
9	8	3	3280	1161	154.5	13.31	6.3	0.54
10	5	4	341	26.5	0	0	0	0
11	6	4	1365	336.1	26.4	7.85	0	0
12	7	4	5461	5290.5	479.8	9.07	12.3	0.23
13	5	5	781	89	12.5	14.04	1.6	1.80
14	6	5	3906	3395.5	243.6	7.17	7.8	0.23
15	5	6	1555	318.8	35.8	11.23	1.6	0.50
16	5	7	2801	1173.2	117.4	10.01	4.7	0.40
17	5	8	4681	3159.2	368.8	11.67	11	0.35
18	5	9	7381	13000.1	925	7.12	23.6	0.18

Table 1: Comparison of CPU times

References

[1] S. Ahmed. Semiconductor tool planning via multi-stage stochastic programming. In *Proceedings of the International Conference on Modeling and Analysis in Semiconductor Manufacturing*, pages 153–157, 2002.

[2] K.J. Arrow, T.E. Harris, and J. Marschak. Optimal inventory policy. *Econometrica*, 19:250–272, 1951.

[3] J.R. Birge and F. Louveaux. *Introduction to Stochastic Programming.* Springer, New York, 1997.

[4] L.M.A. Chan, D. Simchi-Levi, and J. Swann. Dynamic pricing strategies for manufacturing with stochastic demand and discretionary sales. 2002. Submitted for publication.

[5] M.N. EL Agizy. Dynamic inventory models and stochastic programming. *IBM Journal of Research and Development*, pages 351–356, July 1969.

[6] S.M. Johnson. Sequential production planning over time at minimum cost. *Management Science*, 3(4):435–437, 1957.

[7] P.H. Zipkin. *Foundations of inventory management.* McGraw-Hill, 2000.