A Simple Algorithm for Part Stocking to Satisfy Pooled Customer Service Requirements at Minimum Cost [*]

Kathryn E. Caggiano [†]

John A. Muckstadt [‡]

James A. Rappold [§]

Peter L. Jackson [¶]

October 2001

[*]This work was supported in part by Xelus, Inc., and the National Science Foundation (Grant DMI0075627)

[†]School of Business, University of Wisconsin, Madison, WI 53706

[‡]School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853

[§]School of Business, University of Wisconsin, Madison, WI 53706

[¶]School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853

Abstract

We consider an assignment problem in which customers must be assigned to field service locations to fulfill their anticipated demands for a particular item. The goal is to assign customers to locations so that the total inventory investment at the field service locations is minimized. The model we present can be used in a single-echelon distribution system where location replenishment lead times are identical and safety stock levels at all locations are set to a known multiple of the standard deviation of the lead time demand. We describe a branch and bound scheme that will find the optimal solution to the problem.

1 Introduction

We consider an assignment problem in which customers must be assigned to field service locations to fulfill their anticipated demands for a particular item. The goal is to assign customers to locations so that the total inventory investment at the field service locations is minimized under the established safety stock policy. This policy requires that safety stock levels at all field service locations be set to a known multiple of the standard deviation of the lead time demand.

Once modeled, the optimization problem that results is a traditional assignment problem with a concave objective function. We derive a useful property of the optimal solution to this problem and use this property to develop a simple branch and bound scheme that, when employed, will obtain an optimal solution.

The remainder of the paper is organized as follows. In Section 2, we state our notation and modeling assumptions, and formulate the problem as a concave program. In Section 3, we derive a useful property of the optimal solution and describe a method for finding the optimal solution that exploits this property. Section 4 provides an illustration of the algorithm on a sample problem. We conclude and discuss potential algorithm improvements in Section 5.

2 Problem Framework

2.1 Notation and Assumptions

We will use the following notation throughout the paper:

J	-	the set of service locations, indexed by j.
K	-	the set of customers, indexed by k.
J_k	-	the set of service locations capable of serving customer $k \in K$.
K_j	-	the set of customers that may be served by location $j \in J$.
λ_k	-	the demand rate of customer k.
y_{jk}	-	the assignment indicator for customer k to location j. That is, $y_{jk} = 1$ if customer k is assigned to service location j, and 0 otherwise.

T - the replenishment lead time for the service locations.

The system characteristics are as follows:

- Customer demands arise according to independent Poisson processes, with λ_k denoting the mean demand rate for customer $k \in K$.
- All field service locations have the same replenishment lead time T.
- At each location, the management policy is that the safety stock level for the item must be set to u standard deviations of the total lead time demand (over all customers assigned to that location). The parameter u is given and is assumed to be chosen to satisfy service level obligations. (This approach is common and appropriate for high demand rate items.)

2.2 Problem Formulation

For each customer $k \in K$, the demand over the replenishment lead time T is Poisson-distributed with mean $\lambda_k T$ and variance $\lambda_k T$. Thus, for any feasible assignment $\{y_{jk} : j \in J, k \in K\}$ of customers to locations, the total inventory requirement I (in units) is given by:

$$\begin{aligned} I &= \sum_{j \in J} \left[\sum_{k:j \in J_k} (\lambda_k T) y_{jk} + u \sqrt{\sum_{k:j \in J_k} (\lambda_k T) y_{jk}} \right] \\ &= \sum_{k \in K} (\lambda_k T) + u \sum_{j \in J} \sqrt{\sum_{k:j \in J_k} (\lambda_k T) y_{jk}}. \qquad (2.1) \end{aligned}$$

Since the first term on the right-hand side of 2.1 is a constant, minimizing I is equivalent to minimizing the second term (i.e., the safety stock requirements over all locations).

By setting customer weights $w_k = \lambda_k T$ for all $k \in K$, we can state our inventory minimization problem as a general *Square Root Assignment* problem, or (**SRA**) as follows:

$$(\textbf{SRA}) \qquad \text{minimize} \qquad \sum_{j \in J} \sqrt{\sum_{k:j \in J_k} w_k y_{jk}} \qquad (2.2)$$

subject to

$$\begin{aligned} \sum_{j \in J_k} y_{jk} &= 1 \quad \forall k \in K, & (2.3) \\ y_{jk} &\geq 0 \quad \forall j \in J, k \in K. & (2.4) \end{aligned}$$

Note that we have ignored the multiplier u in our formulation since it does not impact the assignment. Also, since the objective function is concave and the constraint set is linear, an optimal solution will lie at an extreme point of the feasible region. Hence, we do not need to specify $y_{jk} \in \{0, 1\}$ for all $j \in J, k \in K$ - the nonnegativity constraints will suffice.

3 Solution Approach

The property of the optimal solution to **SRA** that we wish to derive is a consequence of the following proposition:

Proposition 1 *For real numbers a, b, and c, such that $a \geq b \geq c > 0$, $\sqrt{a} + \sqrt{b} > \sqrt{a+c} + \sqrt{b-c}$.*

Proof: *Since $\sqrt{\cdot}$ is a strictly concave function, we have that:*

$$\begin{aligned} \sqrt{a+c} &< \sqrt{a} + \frac{1}{2\sqrt{a}}c && (3.1) \\ \textit{and } \sqrt{b-c} &< \sqrt{b} - \frac{1}{2\sqrt{b}}c. && (3.2) \end{aligned}$$

Since $a \geq b$, the result is immediate. $\square$

The link to **SRA** is that if a customer (with weight c) is being serviced by a particular location (servicing weight b), but there is a “heavier” location (servicing weight a) that can serve the customer, it is better to assign the customer to the “heavier” location. Expanding this idea, we can make the following observation:

Corollary 1 *Every optimal solution to **SRA** has the property that for some service location $j^* \in J$, $y_{j^*k} = 1$ for all k such that $j^* \in J_k$. That is, at least one service location will serve all of its potential customers.*

Proof: *By contradiction. Suppose that there existed an optimal solution $\mathbf{y}^*$ to **SRA** with the property that for every location $j \in J$, $y^*_{jk} = 0$ for some k such that $j \in J_k$. Let j' be the location with the largest assigned weight over all $j \in J$. That is:*

$$\sqrt{\sum_{k:j' \in J_k} w_k y^*_{j'k}} = \max_{j \in J} \left\{ \sqrt{\sum_{k:j \in J_k} w_k y^*_{jk}} \right\}. \qquad (3.3)$$

Let k' be a customer such that $j' \in J_{k'}$ and $y^*_{j'k'} = 0$. By the above proposition, setting $y_{j'k'} = 1$ and $y_{jk'} = 0$ for all $j \neq j'$ will lower the objective function. Hence, $\mathbf{y}^*$ cannot be optimal. □

Notice that Corollary 1 does not say *which* of the locations will serve all of its potential customers - it just guarantees that at least one location will. One might be tempted to think that the location with the *largest* customer base will always serve all of its potential customers in the optimal solution. If this were true, then a simple greedy heuristic, based on assigning customers to the "heaviest" remaining location, would provide the optimal solution to the problem. (That is, a heuristic which repeatedly chooses the location with the largest unassigned customer base and assigns to it all of its remaining customers.) Unfortunately, it is *not* always the case that the "heaviest" location will serve all of its potential customers in the optimal solution.

To see this, consider the problem depicted in Figure 1, where the service location set $J = \{A, B, C, D\}$ and the customer set $K = \{1, 2, 3, 4, 5, 6, 7, 8\}$.

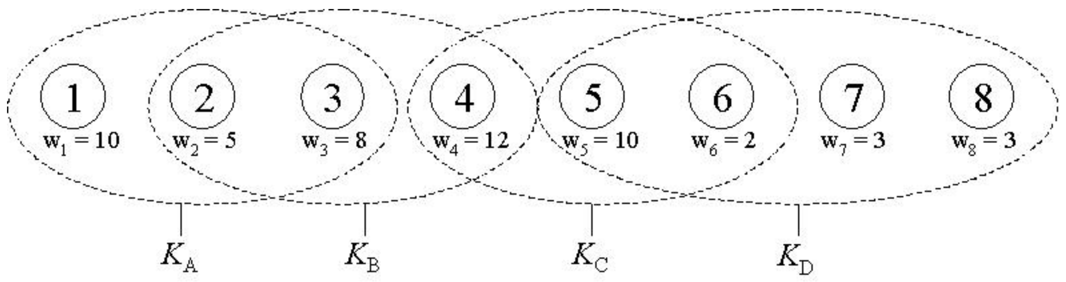

Figure 1: Example Problem Data

Applying the greedy approach to this problem results in the following assignment (shown in the order the assignments are made):

$$B : \{2, 3, 4\}, \quad D : \{5, 6, 7, 8\}, \quad A : \{1\}.$$

The corresponding objective function value is 12.405. However, the optimal solution ($A : \{1, 2, 3, \}$, $C : \{4, 5, 6\}$, $D : \{7, 8\}$) has an objective function value of 12.144 and does not assign *any* customers to location B. Thus, while the greedy approach may lead to a good solution quickly, it does not, in general, find the optimal solution.

Despite its limited implications, Corollary 1 *does* give rise to a simple branch and bound algorithm that can be used to find the optimal solution to **SRA**. The branching scheme is based on the following observation: Suppose that we *know* which location j^* services all of its potential customers in the optimal assignment. Then the optimal assignment is composed of the assignment of all customers $k \in K_{j^*}$ to j^*, *plus* the optimal assignment for the *remaining* problem with the *remaining* locations and customers; but, Corollary 1 applies to the remaining problem *as well as* to the original problem. Thus, the branching scheme we adopt works as follows: we choose each location, in turn, to be the j^* in Corollary 1, and then solve the remaining problem recursively.

The algorithm detailed below, **Find-Best-Assignment**, serves as the outer shell to the recursive procedure **Branch**, which creates a search tree to find the optimal solution. Specifically, **Branch** examines all unassigned locations sequentially and branches on those that potentially lead to the optimal solution.

```
Find-Best-Assignment($J$; $K$; $K_j, j \in J$; $w_k, k \in K$)

Input:      An instance of SRA: $J$; $K$; $K_j, j \in J$; $w_k, k \in K$;
Output:     Optimal solution to SRA: BEST-ASSIGNMENT;
Global Variables: UB, BEST-ASSIGNMENT.

  1. UB = $\infty$; BEST-ASSIGNMENT = $\emptyset$;        Initialize global variables.
  2. Branch($J$, $K$, $\{K_j : j \in J\}$, $\emptyset$);       Recursively solve SRA.
  3. return(BEST-ASSIGNMENT);


Branch($\tilde{J}$, $\tilde{K}$, $\{\tilde{K}_j : j \in \tilde{J}\}$, $A$)

Input:      Unassigned field service locations $\tilde{J} \subseteq J$;
            Unassigned customers $\tilde{K} \subseteq K$;
            Maximal unassigned customer groups $\{\tilde{K}_j \subseteq K : j \in \tilde{J}\}$;
            Assigned customer groups $A = \{A_j \subseteq K \backslash \tilde{K} : j \in J \backslash \tilde{J}\}$;
```

Output: None - solution captured in global variables;

Local Variables: current-value, current-LB.

```
1. If ($\tilde{K} = \emptyset$) {          *All customers are now assigned, so examine current solution.*
       current-value = $\sum_{j \in J \setminus \tilde{J}} \sqrt{\sum_{k \in A_j} w_k}$;
       If (current-value < UB) {
           UB = current-value;
           BEST-ASSIGNMENT = $A$;
       }
       return;
   }

2. For all $j^* \in \tilde{J}$ such that $\tilde{K}_{j^*} \neq \emptyset$ {          *Branch on all locations with potential.*
       current-LB = $\left( \sum_{j \in J \setminus \tilde{J}} \sqrt{\sum_{k \in A_j} w_k} \right) + \sqrt{\sum_{k \in \tilde{K}_{j^*}} w_k} + \sqrt{\sum_{k \in \tilde{K} \setminus \tilde{K}_{j^*}} w_k}$.
       If (current-LB < UB) {
           $\tilde{A}_{j^*} = \tilde{K}_{j^*}$;
           **Branch**($\tilde{J} \setminus j^*$, $\tilde{K} \setminus \tilde{K}_{j^*}$, $\{\tilde{K}_j \setminus \tilde{K}_{j^*} : j \in \tilde{J} \setminus j^*\}$, $A \cup A_{j^*}$);
           $\tilde{A}_{j^*} = \emptyset$;
       }
   }
   return;
```

The first step of **Branch** is only executed at the bottom of the recursion; that is, when all customers have been assigned. The corresponding solution is examined and kept if it dominates the current best solution. The second step of **Branch** creates a search tree in a depth-first fashion, exploring only those branches that have the *potential* to lead to a solution that dominates the current best solution. Note that the second step exploits the result of Corollary 1 in that each time a location j^* is examined and pursued, j^* receives *all* of the remaining customers that can be assigned to it.

4 Illustration of the Algorithm

Consider again the example problem depicted in Figure 1. The search tree created by the **Branch** algorithm for this problem is shown in Figure 2, with the solution values (or branch cutoff values) shown beneath the leaf nodes. The order in which the nodes were generated and examined by **Branch** is depth-first, from left to right. A black, horizontal line through a branch indicates that the lower bound found for the location being examined exceeded the value of the current best solution, and hence, the branch was not pursued.

Notice that locations A and C service all of their potential customers in the optimal solution, but the location with the largest potential customer base, B, does not.

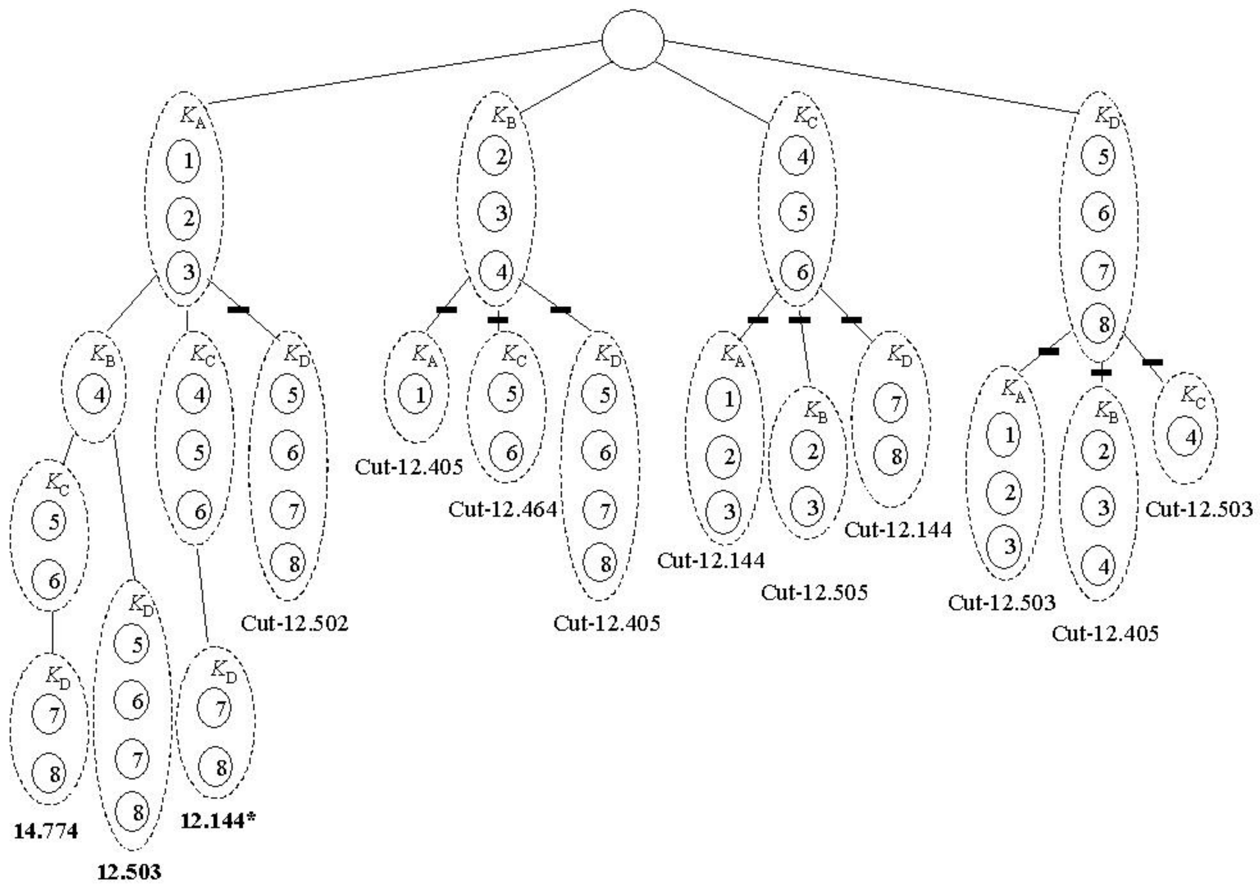

Figure 2: Branch and Bound Tree Using **Find-Best-Assignment**

5 Conclusions

In this paper we presented a model for determining optimal safety stock levels in a single-echelon distribution system where location replenishment lead times are identical and safety stock levels at all locations are set to be a known multiple of the standard deviation of the lead time demand. We showed that the model reduces to a nonlinear assignment problem whose optimal solution displays a special property, and we used this property to develop a simple branch and bound scheme that finds an optimal solution to the problem.

We conclude by mentioning several modifications to the **Branch** algorithm outlined in Section 3. Depending on the specific problem instances to be solved, any or all of these modifications may lead to improved performance:

- The order in which the locations are examined at a single node can have a significant impact on the branching efficiency of the algorithm. In our example, we simply used a static order (A, B, C, D) at each node. However, using a dynamic ordering (based, for example, on the weight of the remaining customer base for the locations) may lead to good solutions, and hence better upper bounds, more quickly.

- Alternative lower bounding techniques may be used to cut branches in lieu of, or in addition to, the one given in **Branch**. Problem-specific constraints may also be incorporated into the branching scheme to search the solution space more efficiently.

- Easy upper bounds (such as the solution value from the greedy heuristic) can be established in a pre-processing step and can be used to seed the UB value. A good initial upper bound often allows for a faster pruning of the search tree.

- Alternative branching rules may be employed in lieu of the simple depth-first rule we used in our illustration. For instance, branching on the node with the lowest lower bound may lead to good solutions more quickly.

- Finally, the algorithm can be stopped at any time. It need not be run to completion if a solution is found that is within an acceptable range of a known lower bound.