SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14853

TECHNICAL REPORT NO. 798

April 1988

LOT SIZING IN MULTI–ECHELON ASSEMBLY SYSTEMS: HEURISTICS AND WORST CASE PERFORMANCE BOUNDS

by

Dev Joneja[1]

[1]This research was supported in part by NSF Grant DMC–8451984, and by AT&T Information Systems and DuPont Corporation.

Lot Sizing in Multi-Echelon Assembly Systems: Heuristics and Worst Case Performance Bounds

by
Dev Joneja[1]

School of Operations Research
and Industrial Engineering
Cornell University
Ithaca, N.Y. 14853

April 13, 1988

Abstract

In a multi-echelon assembly system, each production stage has inputs from a number of preceding stages, and itself supplies at most one succeeding stage. The product of the final stage is used to satisfy external non-stationary demand in discrete time. There are fixed setup costs and linear inventory holding costs at each stage. The objective is to minimize the cost of operating the system over a finite time horizon. We propose a single pass approximation algorithm for this problem, and prove that in the worst case the performance of the algorithm is uniformly bounded. The algorithm extends to general production-distribution systems in which the external demand for all items follows the same seasonal pattern. The behavior of our heuristics for a randomly generated set of problems is also studied.

[1]This research was supported in part by NSF Grant DMC- 8451984, and by AT&T Information Systems and DuPont Corporation.

1 INTRODUCTION

In a multi-echelon assembly system, each production stage has inputs of sub-assemblies from a number of preceding stages, and its product in turn feeds at most one succeeding stage. The product of any stage can be considered a distint item. The product of the final stage is used to satisfy external demand. This structure can be represented by a network in which each node represents a production stage, and the arcs represent the flow of items from one stage to the next. This network is a tree where each node can have several predecessors and exactly one successor. The number of items (nodes) in the tree is N, and they are assumed to be numbered from 1 to N so that if there is an arc from item i to item j, then $i > j$ (every tree can be indexed in this manner). The final node at which external demand occurs is thus node 1. A typical assembly system with 6 nodes is shown in Figure 1.

Consider such a system in which demand occurs in discrete time, and must be satisfied without backorders. We allow the demand to vary with time, thus considering seasonality and other non-stationary demand characteristics. All demands are assumed to be deterministic and known over the entire finite planning horizon T. Delivery lead times from one stage to the next are deterministic and known, and can be assumed to be zero without loss of generality. The number of units produced at any stage which are consumed in the production of a single unit at the succeeding stage is assumed to be one. If several units are used in the production at the succeeding stage, then selecting units of measure by a simple transformation of the holding cost rates reduces it to this form.

Associated with each item is an inventory holding cost rate per unit stored per unit time, and a fixed setup cost which is incurred whenever the item is produced. The objective is to decide an ordering policy for each item in order to minimize the cost of operating the entire system over the planning horizon.

The problem has been approached in several ways in the literature. The best known algorithm for obtaining an optimal solution was proposed by Crowston and Wagner (1973). It uses a dynamic programming formulation and has computation time of $O(N2^T)$. A more general lot-sizing problem has been formulated as an integer programming problem by Steinberg and Napier (1980), and they have solved

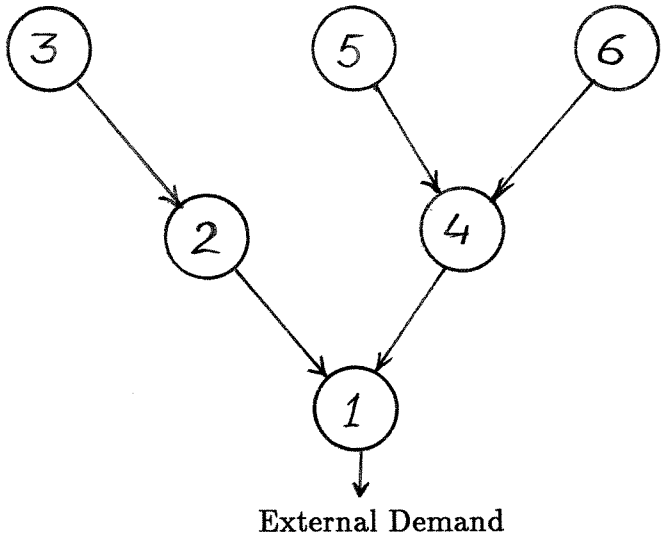

Figure 1: An assembly system with 6 nodes.

some small problems with it. A better integer programming formulation is provided by Afentakis, Gavish and Karmarkar (1984), who then solve it using branch and bound techniques. They have been successful in solving medium sized problems before the computational burden becomes very large. Besides the computational burden, the optimal policies also suffer from nervousness, where small changes in demand forecasts or the time horizon have a significant effect on the policy. This can cause severe disruptions in the implementation of the policy, especially if a small change in demand forecasts far in the future have a great impact on current order size.

A more practical approach to the problem is to employ heuristic algorithms. Several heuristics have been proposed by Graves (1981), Blackburn and Millien (1982), McLaren (1976), Roundy (1985a), and Afentakis (1987). A weakness of most such algorithms is that their performance appears to deteriorate as the number of stages in the assembly system is increased. The algorithhms for assembly system proposed by Afentakis and by Roundy overcome this, and their reported computational results indicate that both work very well in practice. Roundy also does an analysis of the worst case performance of the algorithms proposed. In the worst case, both algorithms have computational time of $O(NT^3)$.

In this paper, we present a conceptually simple heuristic for lot sizing in an assembly system. Uniform bounds on the worst case performance of the algorithm are proved, which are independent of the size of the system, the cost parameters, and the demand pattern. In practice, the algorithm performs well, and on an average the solution is within 4% of a lower bound on the optimal solution. Its performance in practice appears to be independent of the size of the network. The algorithm is extremely fast, with a computational time of $O(NT)$. Besides, it is a forward algorithm which uses the solution of the problem in the first t time periods to find the solution for time period $t+1$. At any time period t, the ordering policy for all time periods prior to t has been fixed, and only the last order of any item depends on future demands. As a result changes in the time horizon or in demand forecasts far in the future have little effect on the ordering policy in the current time period. Thus the algorithm has the advantage of controlling nervousness of the generated policy.

The rest of the paper is organized as follows. In the next section, we develop and discuss the algorithm. In section 3 we analyze the worst case performance of our algorithm, and prove that it is uniformly bounded. Section 4 discusses how the algorithm can be enhanced, and also used for more general production-distribution systems in which external demands can exist for several end products, but follows the same seasonal pattern. The behavior of our algorithms for a randomly generated set of problems is discussed in section 5. The last section presents our conclusions.

2 ALGORITHM FOR ASSEMBLY SYSTEMS

In this section we will develop a linear time algorithm for the assembly system. The following notation will be used in this paper:

N = number of items in the system,

T = the time horizon,

$s(i)$ = successor of node i in the system,

d_t = demand of final item in time period t,

K_i = setup cost of item i,

H_i = echelon holding cost rate of item i.

Vienott (1969) proved that such a system has an optimal policy in which an item orders in time period t only if its inventory at the end of time period $t-1$ is zero. He also introduced the concept of *nested policies*, in which an item orders in time period t only if its successor in the system also orders in the same time period. In general, nested policies may not be optimal. For nested policies, the inventory holding cost is easier to express in terms of *echelon inventories* and *echelon holding cost rates* (Schwarz and Schrage, 1975). The echelon inventory of item i is the total inventory of this item in the system, whether at the output of stage i or any successor stage. If H_i' is the inventory holding cost rate of item i per unit time, then the echelon holding cost rate H_i is defined as $H_i = H_i' - \sum_{n|s(n)=i} H_n'$. This thus reflects the incremental value added by the operation at stage i in the system.

For a serial system under the assumption of non-negative echelon holding costs, Love (1971) proved that there exists an optimal policy which is nested. This result was extended to assembly systems under the same assumption by Crowston and Wagner (1973). The assumption of positive echelon holding costs is in practice a weak assumption, since it implies that each stage adds some value to the items which are processed at that stage. We will make this assumption in this paper. This implies that an optimal policy for our model exists which is nested, and so we will confine our attention to nested policies. Besides, from now on the terms *inventory* and *holding cost* will refer to echelon inventory and echelon holding cost.

A further structural constraint will be placed on the types of policies which we will consider, which is motivated by recent results on lot-sizing with constant demand. For this problem, Maxwell and Muckstadt (1985) proposed a fast and effective heuristic in continuous time and with an infinite time horizon. The objective is to minimize the average cost of the policy per time period. They introduce the concept of *clusters*, which are connected subsystems of the original system. They restrict attention to policies in which all nodes in a cluster order simultaneously. For a cluster C_i we define the cluster setup cost $K^i = \sum_{n \in C_i} K_n$, and the cluster holding cost rate $H^i = \sum_{n \in C_i} H_n$, which are the aggregated setup and holding cost rates respectively over all the nodes in the cluster. Clusters have the following properties:

- If there is an arc from a node of cluster C_i to a node of cluster C_j, then there is no arc from any node of C_j to any node of C_i.

- If there is an arc from cluster C_i to cluster C_j, then $K^i/H^i \geq K^j/H^j$.

- A directed cut of a cluster C_i is a partition of its nodes into two subsets C_+ and C_- such that arcs in the network can go from nodes in C_+ to nodes in C_-, but not the other way around. Let $K^+ = \sum_{n \in C_+} K_n$, and let $H^+, K^-, and H^-$ be simiraly defined as above for sub-clusters C_+ and C_-. Then for any directed cut of cluster C_i into sub-clusters C_+ and C_-, we have $K^+/H^+ \leq K^i/H^i \leq K^-/H^-$.

In an assembly system, the clusters can be uniquely determined by an efficient algorithm in $O(N \log N)$ time (Roundy, 1985b).

In the Maxwell-Muckstadt heuristic, all items in a cluster are forced to order together. In our problem, it may not in general be optimal for all nodes in a cluster to order together, but the following theorem shows that this is true for the special case of serial systems.

Theorem 1 *In a serial system, there exists an optimal solution in which all nodes in a cluster always order together.*

Proof: For a serial system, let $\mathbf{P}$ be any optimal policy. Since there exist optimal policies that are nested, we can assume with no loss of generality that $\mathbf{P}$ is nested. Let C_i be any cluster in the system whose nodes do not all order together according to policy $\mathbf{P}$. Let τ be the first time period in which only some of the nodes in C_i order. Let the nodes in C_i that order in time period τ be denoted by C_-, while the nodes in C_i that do not order here be denoted by C_+.

Since the network is a serial network and $\mathbf{P}$ is a nested policy, clearly C_+ and C_- define a directed cut of cluster C_i. Let the following time periods be defined:

t_1 = last time period before τ in which all nodes in C_i placed an order,

t_2 = first time period after τ in which any node in C_+ placed an order.

Due to the serial network structure and nestedness of $\mathbf{P}$, note that all nodes in C_- place an order at t_2. Then the cost of removing the order at τ of all nodes in C_- is

at most $H^-(\tau - t_1)\sum_{t=\tau}^{t_2-1} d_t - K^-$. Since $\mathbf{P}$ is an optimal policy, this cost must be non-negative. Hence:

$$H^-(\tau - t_1)\sum_{t=\tau}^{t_2-1} d_t \;\geq\; K^-. \tag{1}$$

Since C_+ and C_- form a directed cut of cluster C_i we know from the third property of clusters listed above that $K^+/H^+ \leq K^-/H^-$. This, along with the inequality (1) above, gives:

$$H^+(\tau - t_1)\sum_{t=\tau}^{t_2-1} d_t \;\geq\; K^+. \tag{2}$$

Next, consider the cost of inserting an order at τ of all nodes in C_+. This is at most $K^+ - H^+(\tau - t_1)\sum_{t=\tau}^{t_2-1} d_t$. Inequality (2) above shows that this cost can not be positive. Hence the policy generated by inserting an order at τ of all nodes in C_+ is also optimal. Repeating this argument for all clusters whose nodes do not always order together completes the proof. □

Motivated by this, we will restrict ourselves to policies in which all nodes in a cluster always order together. Notice that the structure of the clusters depends on the ratio of setup costs to holding cost rates, and is independent of the actual demands. The clusters can thus be identified initially, and a *cluster network* is created by collapsing all nodes that belong to the same cluster in the original network into a single node. Clearly, the cluster network again has the structure of an assembly tree. The cluster nodes are again assumed to be indexed so that if an arc exists from cluster C_i to cluster C_j then $i > j$.

Since all nodes in a cluster C_i always order together, the total setup cost incurred by these nodes when they place an order is $\sum_{j\in C_i} K_i = K^i$. Besides, the echelon inventory of each node in the cluster is identical since they always order together. So the rate at which the total holding cost at all the nodes in cluster C_i is incurred is given by $\sum_{j\in C_i} H_i = H^i$. Therefore in the cluster network we associate with the node that represents cluster C_i a setup cost K^i and a holding cost rate H^i. Solving the assembly system problem on the cluster network with node cost K^i and H^i gives the ordering policy on the cluster network. Ordering policies on the cluster network correspond to policies on the original network in the obvious manner. The discussion above implies that the cost of a policy on the cluster network is the same as the cost of the corresponding policy on the original network. Thus all further

discussion is on the cluster network. Note that theorem 1 says that policies that are optimal for the cluster network are also optimal for the original network in the case of serial systems.

The algorithm proceeds by deciding the ordering policy of individual clusters over the entire time horizon, starting with cluster C_1 and proceeding to successively higher indexed clusters. At any stage, the ordering policy of a single cluster is being decided, and can be considered as a single item problem. However, nestedness requires that the cluster can not order in a time period unless its successor also orders in that time period. Since the ordering policy of the successor has already been decided, the reorder points of the cluster under consideration are modified to achieve nestedness.

The basic tradeoff in lot-sizing is balancing the inventory holding cost against setup or ordering cost. Consider the simple part period balancing heuristic for a single item problem (DeMatteis and Mendoza, 1968). It proceeds by placing an order in time period 1, and in each successive time period it decides whether or not an order will be placed. At any stage t, the ordering policy for each period prior to t has been decided. Then in period t it calculates the total amount of holding cost incurred since the last time an order was placed. If this total holding cost exceeds the setup cost, an order is placed in the current time period. It then proceeds to the next time period.

This idea is extended to the assembly network. We say that when the total holding cost of an item incurred since its last order exceeds its setup cost, it becomes a candidate for reorder. If this is the first cluster, at which the external demand occurs, it places an order in this time period. The first cluster thus follows the policy generated by the part period balancing heuristic. The other clusters must modify their order periods to maintain nestedness with respect to their successor cluster. Let cluster C_i becomes a candidate for reorder in time period τ. Nestedness is achieved by looking for the last time period τ' before τ in which the successor of cluster C_i placed an order. The order for cluster C_i is placed in period τ'. The next lemma shows that this can always be done.

Lemma 2 (Feasibility Lemma) *Let cluster C_i become a candidate for reorder at time τ, and let the last order of C_i before τ be at time t_1. Then there exists a time*

Cluster Algorithm for Assembly Systems

1. Find the clusters and create the cluster network. Set $n = 0$.

2. Set $n = n + 1$. If $n >$ largest cluster index, stop. Otherwise go to step 3.

3. For cluster C_n :

   (a) Set $t = 1$ (time index), $s = 1$ (last order of cluster), $H = 0$ (holding cost in periods s throught t.)

   (b) Set $t = t + 1$, $H = H^n \sum_{k=s+1}^{t}(k - s)d_k$. If $H < K^n$, go to step 3c. Otherwise (place order):

   - If $n = 1$ (final cluster), place order of cluster C_n in period t. Set $s = t$.
   - If $n > 1$, place order of cluster C_n in period τ', where τ' is the last period satisfying $\tau' \leq t$ in which cluster $C_{s(n)}$ placed an order. Set $s = \tau'$.

   (c) If $t = T$, go to step 4. Otherwise go to step 3b.

4. Go to step 2.

Figure 2:

period τ', $t_1 < \tau' \leq \tau$, at which the successor of cluster C_i places an order.

Proof: The proof is by a simple induction in increasing cluster index. Let cluster C_j be the successor of cluster C_i. By nestedness, C_j placed an order at time t_1. Since $K^j/H^j \leq K^i/H^i$, cluster C_j must become a candidate for reorder some time after t_1 and no later than τ. By induction, C_j must place an order at some time period τ', where $t_1 < \tau' \leq \tau$. □

The algorithm is stated formally in Figure 2. It will be referred to as the cluster algorithm.

As noted earlier, the clusters can be found in $O(N \log N)$ time (Roundy, 1985b). In step (3b), H can be updated in constant time. The ordering policy of each

cluster is thus determined in $O(T)$ time, and so the algorithm takes a total of at most $O(N \log N + NT)$ time. This running time can be reduced to $O(NT)$ in two ways. By adapting a sophisticated alorithm (Queyranne 1987), it is possible to compute the clusters for this system in $O(N)$ time. This would bring the total computing time to $O(NT)$. This method is based on median-finding routines, and is impractical due to its complexity. Alternately, the algorithm presented in the appendix generates exactly the same policy as the as the one we have presented above. It does not find the clusters or create the cluster network. Instead it operates on the original network. The modified algorithm uses the t time period solution to find the $t+1$ time period solution. This can be accomplished in $O(NT)$ computing time. Besides, though the number of clusters in the worst case can be N, in most cases it is expected to be much smaller. Thus the algorithm in practice is expected to run in substantially less than $O(N \log N + NT)$ time. For this reason, it is also preferable to the modified algorithm presented in the appendix.

Another point must be noted about the cluster algorithm. We compute the entire ordering policy of one cluster at a time before proceeding to the next cluster. It is straightforward to modify the algorithm so that it runs in increasing time index, in which the policy for the entire system in a t period solution is used to compute the policy in a $t+1$ period problem. Alternately, a t period solution can be extracted from the T period problem simply by looking at the policy over the This policy is then implemented in the current time period. In the next time period, new demand forecasts are obtained and the algorithm is run again to determine the policy in the next period. Using our algorithm, computations have to be done only till each item has placed an order. At this stage future demands have no effect on the current decision, and no further computations are required for its implementation. This provides a simple planning horizon rule.

3 WORST CASE PERFORMANCE BOUNDS

In this section we investigate the performance of the algorithm in the worst case. To this end, define the *relative cost* of the heuristic as the ratio of the cost of the heuristic policy to the cost of the optimal policy. The proof of the worst case performance of the heuristic is based on a lower bound on the cost of the optimal

policy.

Consider any arc in the cluster network from cluster C_i to cluster C_j. The presence of this arc imposes the constraint that the ordering policy of cluster C_i is nested with respect to that of cluster C_j (cluster C_i can not place an order in any time period in which cluster C_j does not place an order.) If this arc were removed from the network, cluster C_i would no longer be constrained by cluster C_j. Being less constrained than the original problem, this is thus a relaxation of the original problem. The solution to the relaxed problem is then a lower bound on the solution to the original problem.

Extending this idea, we first remove all arcs from the cluster network. The new network is just a collection of cluster nodes with no arcs. Thus the clusters no longer have any nestedness constraints, and can follow their indevidual optimal policies. This then decomposes the original problem by clusters into separate subproblems, one for each cluster. The cluster subproblem is then a single item lot-sizing problem with setup cost K^i, holding cost rate H^i, and demands d_t. Let $\mathbf{C_p^i}$ denote the cost of an optimal solution to this subproblem for cluster C_i. This optimal solution, if desired, can be found by the well known Wagner-Whitin algorithm. Clearly then the sum of the costs of these independent optimal policies is a lower bound on the cost of any policy *for the cluster network.* What is important is that this is also a lower bound on the cost of any policy for the original network. Note that for serial systems we have proved that any policy that is optimal for the cluster network is also optimal for the original network. Thus for serial systems, $\sum_i \mathbf{C_p^i}$ is lower bound on the cost of any policy for the original network. Moreover, Roundy (1985a) has proved that this is also true for assembly systems. Specifically:

Theorem 3 (Lower Bound) *In the cluster network for assembly systems, let* $\mathbf{C_p^i}$ *denote the cost of the the optimal policy for the single item subproblem for cluster* C_i. *Then* $\sum_i \mathbf{C_p^i}$ *is a lower bound on the cost of any policy for the assembly system.*

Let $\mathbf{C_h^i}$ denote the cost incurred by cluster C_i according to the policy generated by our heuristic. Then the cost of the heuristic policy is $\sum_i \mathbf{C_h^i}$. At the same time, $\sum_i \mathbf{C_p^i}$ is, by the last lemma, a lower bound on the cost of any optimal policy. This means that the cost of the heuristic and the lower bound can be compared cluster by cluster, using $\mathbf{C_p^i}$ and $\mathbf{C_h^i}$ for cluster C_i. To compare $\mathbf{C_p^i}$ and $\mathbf{C_h^i}$, we further

decompose these costs over subintervals of the time horizon. We will follow the convention that a setup cost is incurred in the time period in which the setup takes place. If inventory at the end of period t is carried to period $t+1$, then the cost of holding this inventory is incurred in period t. This convention is just for ease of accounting, and makes no difference to the total cost of any policy.

Lemma 4 *According to the heuristic, let cluster C_i place three successive orders at time periods t_1, t_2, and t_3. Let f denote the subinterval of the time horizon made of time periods $t_1+1, \ldots, t_3$. Let C_p^{if} be the cost incurred by cluster C_i in the subinterval f, by following the optimal policy. Let C_h^{if} be the cost incurred by this cluster in the same time periods by following the heuristic policy. Then $C_h^{if}/C_p^{if} \leq 3$.*

Proof: Consider the interval $[t_1+1\ ,\ t_3]$. Let h_1 and h_2 be the holding costs incurred by the heuristic policy for cluster C_i in the time intervals $[t_1+1\ ,\ t_2]$ and $[t_2+1\ ,\ t_3]$ respectively. First note that by step 2(b) of the algorithm, the holding cost incurred by cluster C_i between successive orders is at most K^i. Thus $h_1, h_2 \leq K^i$. The cost incurred by the heuristic in the entire interval f is then $C_h^{if} = 2K^i + h_1 + h_2$. Let $\mathbf{P}$ be an optimal policy for cluster C_i, with associated cost C_p^{if} over this interval. There are four cases to consider:

1. $\mathbf{P}$ places no order in $[t_1+1\ ,\ t_3]$. If $\bar{t}$ is the time period in which the cluster became a candidate for order, causing the order to be placed at t_2, then $\bar{t} < t_3$ and the holding cost of cluster C_i in the interval $[t_1+1\ ,\ \bar{t}]$ exceeds K^i. Thus the holding cost in the interval $[t_1+1\ ,\ t_3]$ exceeds K^i, so that $C_p^{if} \geq K^i$. Besides, $C_p^{if} \geq h_1 + h_2$. Thus $C_h^{if}/C_p^{if} = (2K^i + h_1 + h_2)/C_p^{if} \leq 2K^i/K^i + (h_1+h_2)/(h_1+h_2) = 3$.

2. $\mathbf{P}$ places a single order in $[t_1+1\ ,\ t_2]$ and none in $[t_2+1\ ,\ t_3]$, in which case $C_h^{if}/C_p^{if} \leq (2K^i + h_1 + h_2)/(K^i + h_2) \leq 1 + (K^i + h_1)/(K^i + h_2) \leq 3$.

3. $\mathbf{P}$ places a single order in $[t_2+1\ ,\ t_3]$ and none in $[t_1+1\ ,\ t_2]$, which is similar to case (2).

4. $\mathbf{P}$ places at least two orders in $[t_1+1\ ,\ t_3]$, in which case $C_p^{if} \geq 2K^i$, and $C_h^{if}/C_p^{if} \leq 2 \leq 3$.

In each case, $C_h^{if}/C_p^{if} \leq 3$. $\square$

We now combine these results to prove our main theorem.

Theorem 5 *The worst case relative cost of the heuristic is 3.*

Proof: Let the heuristic policy for cluster C_i place orders at time periods $1, t_1, t_2, \ldots, t_{n_i}$. The time horizon is divided into consecutive, non-overlapping subintervals $[2\ ,\ t_2]\ ,\ [t_2+1\ ,\ t_4], \ldots$. If n_i is odd, then the last subinterval is $[t_{n_i-1}+1\ ,\ T]$, and if n_i is even it is $[t_{n_i}+1\ ,\ T]$. Note that the setup cost K^i is always incurred in period 1, and this cost will be accounted for in the *last* subinterval. Then an argument essentially identical to Lemma 4 ensures that $C_h^{if}/C_p^{if} \leq 3$, where f denotes the last subinterval.

If f denotes any subinterval, let C_p^{if} and C_h^{if} be the optimal and heuristic costs respectively for cluster C_i in subinterval f. Let $\mathbf{C_p^i}$ and $\mathbf{C_h^i}$ be the optimal and heuristic costs respectively for cluster C_i over the entire time horizon. Then $\mathbf{C_h^i} = \sum_f C_h^{if}$, and $\mathbf{C_p^i} = \sum_f C_p^{if}$. By lemma 4, this implies $\mathbf{C_h^i}/\mathbf{C_p^i} \leq 3$. The total cost of the heuristic is $\sum_i \mathbf{C_h^i}$, while $\sum_i \mathbf{C_p^i}$ is a lower bound on the cost of any policy for the original system by the lower bound theorem. This proves the theorem. $\square$

Before ending the discussion of the worst case performance of the algorithm, a special case deserves mention. The continuous time, infinite horizon version of the assembly system with constant demand over time has been studied at length in the literature. The objective is to find policies that minimize average cost per time period of running the system. For this problem, fast algorithms exist which will always be within 6% of the optimal (Roundy, 1985b). Though our algorithm was developed for non-stationary demand patterns, it is of interest to investigate how the algorithm performs if demand actually happens to be constant over time. To facilitate comparison with existing results, the following analysis is done in continous time. It is obvious how the analogous version of the cluster algorithm works in continuous time. The following theorem establishes the worst case performance of the algorithm with constant demand.

Theorem 6 *If external demand is constant over time, then the worst case relative cost of the continuous time, infinite horizon analog of the heuristic is 1.25.*

Proof: Let the demand rate per time period be D. It is obvious that the policy generated by the heuristic is stationary, meaning that the time between successive orders of any cluster is constant. Let us define the natural reorder interval t^i of cluster C_i by $t^i = \sqrt{2K^i/H^iD}$. Simple algebra shows that cluster C_i becomes a candidate for order t^i time periods after its previous order. Since the lowes cluster C_1 orders as soon as it becomes a candidate, it will order every t^1 time periods. Thus it incurs an average cost of $\sqrt{2K^1H^1D}$ per time period.

Let cluster C_j be the successor of cluster C_i. If cluster C_i orders every t_i time periods, by nestedness, t_i is an integer multiple of t_j, the reorder interval of cluster C_j. Thus $t_i = m_i t_j$, where it is easy to verify that m_i is a positive integer satisfying $m_i t_j \leq t^i \leq (m_i + 1)t_j < 2m_i t_j$. Thus $t^i/2 < t_i \leq t^i$. The average cost per time period incurred by cluster C_i is thus $K^i/t_i + H^i D t_i/2 \leq 1.25\sqrt{2K^iH^iD}$. The total cost of the policy is thus at most $1.25\sum_i \sqrt{2K^iH^iD}$. A lower bound on the cost of any policy for the system is $\sum_i \sqrt{2K^iH^iD}$ (Roundy, 1985b). Taking the ratio, the relative cost is no larger than 1.25. Simple examples to show that this bound is tight are easily constructed. □

In the next section we discuss a variant of the algorithm which further improves on this result.

4 IMPROVEMENTS, ENHANCEMENTS AND EXTENSIONS

The algorithm presented in section 2 is a basic structure which can be enhanced in various ways to improve its performance. Clearly the policy generated by the algorithm can be locally modified by moving an order for an item forward or backward in time slightly to achieve any cost benefits. This should take care of cases where high demand in one period causes a high holding cost, and then a low demand in a succeeding period triggers an order. Similarly, it is possible to remove orders of items near the end of the time horizon and check if the cost of the policy decreases. Clearly a variety of such enhancements can be incorporated with little increase in the computational burden.

Another significant point is that the algorithm utilizes the part-period balancing heuristic to decide when a cluster becomes a candidate for reorder. This decision

can be made using any other heuristic for the single item problem, such as the Silver-Meal heuristic, least unit cost heuristic, etc. The Silver-Meal heuristic is of particular interest since it is probably the best known and has been found to outperform most other heuristics in computational tests. Unfortunately, its worst case performance can be arbitrarily bad. The part-period balancing heuristic on the other hand has bounded worst case performance, and our algorithm as described above also has this desirable property.

A further possibility of enhancements derives from the fact that in the cluster algorithm, if a cluster C_i becomes a candidate for reorder at time τ, and the last order before τ of its successor was at τ', then the cluster places its order at τ' (step (3*b*) of the algorithm.) Thus the order of the cluster is *pulled* back from τ to τ'. Let the first order of the successor cluster after τ be at time τ''. Then we can alternately *push* the order or cluster C_i to τ''. Let H' be the holding cost of cluster C_i in periods s through τ', and H'' be the holding cost in periods s through τ''. Then $H' \leq K^i \leq H''$, and the order is placed at τ or τ' depending on whether H' or H'' is closer to K^i. Specifically, in this *push-pull* version of the heuristic, step (3*b*) of the algorithm is replaced by the following rule:

> If $n > 1$, then let $H' = H^n \sum_{k=s+1}^{\tau'}(k-s)d_k$, $\quad H'' = H^n \sum_{k=s+1}^{\tau''}(k-s)d_k$.
> If $K^n - H' \leq H'' - K^n$, order C_i in period τ' and set $s = \tau'$; otherwise order C_i in τ'' and set $s = \tau''$.

In general, this will have an adverse affect on the worst case behavior of the algorithm. However the behavior of the algorithm under constant demands over time improves by using the push-pull enhancement. By an argument very similar to Theorem 6, we can show that the worst case relative cost with constant demands will be 1.10. The algorithm can be further refined along the same lines to get a worst case relative cost of 1.06 with constant demands. Besides, this indicates how various other variations can be used to improve the performance of the algorithm.

Though the algorithm has been developed for assembly systems, it can be used to obtain poicies for general production/distribution systems in which demands for all items follows the same seasonal pattern. This is largely along the lines of Roundy (1985). In a general production/distribution system in discrete time over a finite horizon, external demand can exist for several items. Let the demand for item i

in period t be given by the product $v_i d_t$. Here v_i is a "volume parameter" for the item, and d_t is the seasonal demand pattern which is the same for each item. Under this demand pattern, Roundy shows that clusters can be obtained and the cluster network created. The cluster network itself is a general acyclic directed graph. He further shows that this can be converted to an assembly structure by the addition of certain arcs and deletion of redundant arcs, and policies for the new tree correspond to policies for the original system. The reader is referred to his original paper for the details. At this stage, our heuristic can be used to obtain policies for the assembly structure, which correspond to policies for the original network. Besides, the worst case performance bounds carry through, and in the worst case the relative cost of the policy is no greater than 3.

5 COMPUTATIONAL RESULTS

The algorithm was coded in the C programming language and tested on a wide variety of randomly generated problems. The algorithm yields the cost of the heuristic policy, and the lower bound to the optimal solution. The lower bound is found by solving the individual cluster subproblems using the Wagner-Whitin algorithm. The figures reported in this section are the percentage deviation of the heuristic cost from the lower bound. The true relative cost (relative to the optimal cost) is thus better than the numbers reported here.

The algorithm was tested to investigate its average behavior over a wide spectrum of parameters. Besides, we also investigated the sensitivity of the relative cost to the size and the structure of the network, variability in demand, length of the time horizon, and variability in the cost parameters. We considered networks where the number of items $N = 10, 25, 50, 100$, and 500. For each value of N, three types of networks were created: (a) the serial network, whose depth equals N, (b) a binary tree structure, which is a very dense tree whose depth is $O(\log N)$, and (c) a tree of intermediate depth of approximately $N/5$. Except when the effects of the time horizon were being considered, a time horizon of 18 time periods was used. The other data was generated randomly as follows:

- Demands d_t were generated from a uniform distribution with mean 1 and standard deviation 0.5 (high), 0.25 (intermediate) and 0 (low).

- K_i/H_i for item i was generated from a uniform distribution with mean 4 and standard deviation 2, 1.5, 1.0, or 0.5. This number reflects the natural reorder interval of the item. We took $H_i \equiv 1/K_i$.

A total of 750 problems were solved on an AT&T 6300 personal computer.

The overall results are shown in Table 1, which gives the percent deviation of the heuristic cost from the lower bound for various values of N and various tree structures. Overall, the heuristic solution is about 4% larger than the lower bound. The largest deviation observed in the 750 test problems was 10.9%, and it appears that this is due to small demands triggering an order in certain time periods. Local modification of the ordering policies, as discussed in the last section, should alleviate this problem. The table also gives in seconds the average computation time of a problem on the personal computer.

No dependence of the relative cost on the size of the network is observed. The reason behind this appears to be as follows. Notice that the algorithm tries to schedule orders of any cluster according to a single-item heuristic (the part-period balancing method in this case). It then modifies the order points to attain nestedness relative to the schedule of the successor cluster. Thus each cluster follows its single-item schedule with small modifications for nestedness. As a result, the performance of each cluster is about as "good" as that of any other cluster. Increasing the size of the network increases the number of clusters, but the performance of the clusters does not deteriorate. As a result the performance of the heuristic is fairly independent of the size of the network.

Figure 5 gives the sensitivity of the heuristic to various problem parameters. The average performance appears to improve somewhat with constant demands. As discussed above, this appears to be caused by small demands triggering an order if demand is highly variable. As expected, the performance improves as the variability of the cost parameters K_i and H_i decreases. In other words, the performance is worst when the item costs can be very different from each other. It should be noted that all the other experiments were done with this worst case scenario of very high variability of the distribution from which K_i/H_i were drawn.

As the time horizon is increased from 6 to 36 time periods, the performance of the heuristic deteriorates initially. This happens until $T = 18$, which is approximately

Depth of Tree	Number of items 10	25	50	100	500	Average Deviation
serial	4.0%	4.2	3.8	3.2	3.1	3.7%
intermediate	3.7	4.3	4.4	4.2	3.8	4.1
binary tree	3.6	3.9	4.8	4.8	4.4	4.3
Average devn.	3.8%	4.1	4.3	4.1	3.8	4.0%
Computing time (sec.)	.3	.7	1.3	1.8	3.8	

Table 1: Average deviation of heuristic from lower bound, and computing time per problem.

3 times the largest natural reorder interval of any item. For time horizons larger than 18 time periods, the performance is constant. Again, note that in all other experiments we used this worst case time horizon of 18.

In conclusion, the average performance of the heuristic (4% larger than the lower bound) is worse than that reported by either Afentakis (1987) or Roundy (1985a). However, our algorithm has the advantage of bounded worst case performance, simplicity, speed and control of nervousness of the system. Besides, the performance appears to be independent of the size of the assembly system, and even very large systems can be solved very fast on slow personal computers.

6 CONCLUSIONS

In this paper we discussed the lot-sizing problem for assembly systems with non-stationary demands, and proposed a heuristic with uniformly bounded worst case performance. The algorithm also provides us with a lower bound on the cost of the optimal solution. Besides, the algorithm has the advantages of simplicity and speed. In practice, the algorithm provided policies which were within 4% of the optimal policy on an average. Besides, Theorem 6 for constant demands indicates that for "well behaved" demand structures the algorithm is guaranteed to be very close to optimal.

The concept of clusters can actually be linked to Lagrangean dual multipliers in

	Std. deviation of demand process, σ		
	0.5	0.25	0
Average deviation from lower bound	4.4%	4.5%	3.2%
Demand $\sim$ Uniform (mean = 1, std. deviation $=\sigma$.) $K_i/H_i \sim$ Uniform (mean = 4, std. deviation = 2) $T = 18, \quad N = 100$, various tree structures			

	Time horizon, T				
	6	12	18	24	36
Average deviation from lower bound	3.8%	3.8%	4.6%	4.7%	4.7%
Demand $\sim$ Uniform (mean = 1, std. deviation = 1.25) $K_i/H_i \sim$ Uniform (mean = 4, std. deviation = 2) $N = 100$, various tree structures					

	Std. deviation of K_i/H_i, σ			
	2.0	1.5	1.0	0.5
Average deviation from lower bound	4.6%	4.1%	3.0%	2.5%
Demand $\sim$ Uniform (mean = 1, std. deviation = 1.25) $K_i/H_i \sim$ Uniform (mean = 4, std. deviation = σ) $T = 18, \quad N = 100$, various tree structures				

Figure 3: Sensitivity of heuristic solution to various problem parameters.

an integer programming formulation of the problem. This formulation is along the lines of a similar formulation for the joint replenishment problem (Joneja, 1987). The heuristic solution can then be used for effective solution of the Lagrangean dual problem, and for providing bounds for use in branch and bound implementations to solve the integer programming formulation.

References

[1] Afentakis, P. 1987. A Parallel Heuristic Algorithm for Lot-Sizing in Multi-Stage Production Systems. *IIE Transactions,* March 1987, 34-42.

[2] Afentakis, P., B.Gavish and U.Karmarkar. 1984. Computationally Efficient optimal Solutions to the Lot Sizing Problem in Multi-Stage Assembly Systems. *Management Science,* **30,** 2, 222-239.

[3] Blackburn, J.D. and R.A.Millien. 1982. Improved Heuristics for Multi-Stage Requirement Planning Systems. *Management Science,* **28,** 1, 44-56.

[4] Crowston, W.B. and M.H.Wagner. 1973. Dynamic Lot Size Models for Multi-Stage Assembly Systems. *Managemant Science,* **20,** 14-21.

[5] Dematteis, J.J. and A.G.Mendoza. 1968. An Economic Lot-Sizing Technique. *IBM Systems journal,* **7,** 30-46.

[6] Graves, S.C. 1981. Multi-Stage Lot Sizing: An Iterative Approach. *In L.B.Schwarz, ed., "Multi-Level Production/Inventory Control Systems: Theory and Practice."* North-Holland: Amsterdam.

[7] Love, S.F. 1972. A Facilities in Series Inventory Model with Nested Schedules. *Managemant Science,* **18,** 327-338.

[8] Maxwell, W.L. and J.A.Muckstadt. 1985. Establishing Consistent and Realistic Reorder Intervals in Production Deistribution Systems. *Operations Research,* **33,** 6, 1316-1341.

[9] Queyranne, M. 1987. Finding 94% Effective Policies in Linear Time for Some Production/Inventory Systems. *Unpublished Report, University of British Columbia, Vancouver.*

[10] Roundy, R.O. 1985a. Efficient Effective Lot-Sizing for Multi-Product Multi-Stage Production-Distribution Systems with Correlated Demands. *Technical Report 671, School of OR&IE, Cornell University.*

[11] Roundy, R.O. 1985b. 94% Effective Lot Sizing in Multi-Stage Assembly Systems. *Technical Report 674, School of OR&IE, Cornell University.*

[12] Schwarz, L.B. and L.Schrage. 1975. Optimal and System Myopic Policies for Multi-Echelon Production Inventory Assembly Systems. *Management Science,* **21,** 11, 1285-1294.

[13] Steinberg, E. and H.A.Napier. 1980. Optimal Multi-Level Lot Sizing for Requirement Planning Systems. *Management Science,* **26,** 12, 1258-1271.

[14] Veinott, A.F. 1969. Minimum Concave Cost Solutions of Leontief Substitution Models of Multi-Facility Inventory Systems. *Operations Research,* **17**, 262-291.

[15] Wagner, H.M. and T.M.Whitin. 1958. Dynamic Version of the Economic Lot Size Model. *Management Sci.,* **5** , 1, 89-96.

APPENDIX

We describe here an $O(NT)$ implementation of the algorithm which operates directly on the original network and does not first find the cluster network. The algorithm dynamically creates over time groups of items which order together. These groups are actually unions of the clusters which form the cluster network. If in the cluster network some clusters actually always place an order together, then they will belong to the same group. The algorithm starts by placing an order of every item at time period 1. It proceeds from one time period to the next, and uses the ordering policy up to time $t-1$ to decide at time t whether or not any item will place an order there.

Let s_{nt} be the last time period before time t when item n placed an order, and let H_{nt} be the holding cost of item n in time periods s_{nt} through t. As before, item n becomes a candidate for order if $H_{nt} \geq K_n$. If this is item 1 which satisfies external demand, it reorders at once in period t. Otherwise, the order period must be modified to attain nestedness. Let $s_{s(n)t}$ be the last period before time t at which the successor $s(n)$ of node n placed an order. If $s_{s(n)t} > s_{nt}$, then item n is ordered at $s_{s(n)t}$ too. Otherwise, items n and $s(n)$ are merged into one group, and their setup costs and holding costs are aggregated. The group will then be considered a single node, accumulating holding cost at the aggregated rate, until it places an order. The items in the group will remain in it untill the group orders, and after that they will be considered separately again.

The algorithm can be stated formally as follows:

Modified Assembly Algorithm

1. Set $t = 1$, $s_{nt} = 1 \quad \forall\, n$, $H_{nt} = 0 \quad \forall\, n$, $trig(n) = 1 \quad \forall\, n$. Each item belongs to a separate group initially.

2. Set $n = 0$.

   (a) Set $n = n + 1$.

   (b) If $n > N$, go to step 3. If $n \leq N$, and $trig(n) \geq s_{s(n)t} > s_{nt}$, then the last order of group n had been adjusted too far back. Adjust it forward by removing the order of n at s_{nt} and placing it at $s_{s(n)t}$. Update H_{nt}. If $H_{nt} \geq K_n$:

   - If $n = 1$ order group 1 at time t; separate the items in the group and for each such item m, set $s_{mt} = t$, $H_{mt} = 0$, $trig(m) = t$.
   - If $n > 1$ and $s_{s(n)t} > s_{nt}$, order group n at time $s_{s(n)t}$. Separate the items in the group, and for each such item m, set $s_{mt} = s_{s(n)t}$, $trig(m) = t$, update H_{mt}.
   - If $n > 1$ and $s_{s(n)t} = s_{nt}$, merge group n and its successor node into one group. Update its H_{nt} and go to step 2b.

   (c) Go to step 2a.

3. Set $t = t + 1$. If $t > T$, stop. Otherwise go to step 2.

This algorithm can be implemented so that at each time t the inner loop (step 2) can be executed in $O(N)$ time. Then the running time of the algorithm will be $O(NT)$. It can also be shown that the algorithm produces exactly the same ordering policy as the original assembly heuristic in this paper. Hence all the results which are true for the original heuristic also hold for this algorithm.