Lee, Heungsoon F. (Heungsoon Felix)

A LINE BALANCING STRATEGY FOR DESIGNING FLEXIBLE ASSEMBLY SYSTEMS

Heungsoon F. Lee
Department of Industrial and Operations Engineering

Roger V. Johnson
School of Business Administration

The University of Michigan
Ann Arbor, MI 48109-2117

Technical Report 88-15

October 1988

Abstract

We consider some of the critical design problems for flexible assembly systems where flexible assembly robots perform versatile assembly operations. The design problems to be addressed here determine three types of resource capacities: processing, material handling, and buffer capacities. We want to minimize the total cost which consists of work-in-process (WIP) inventory cost and total resource cost, subject to a constraint of meeting demand. We propose a methodology which is based on a strategy of balancing average workload per machine. This strategy is commonly used when designing manufacturing systems. The methodology presented decomposes the entire decision process into two types of analyses: deterministic and stochastic. The deterministic analysis determines processing capacities by assigning operations to machines such that the number of machines is minimized, the demand is met, and average workload per machine is balanced. Given the decisions from the deterministic analysis, the stochastic analysis takes into account variance of operation times to determine material handling and buffer capacities. The methodology iteratively performs these two analyses in a systematic way and attempts to minimize the total cost by searching for a good trade-off between the machine cost and cost for the other resources and WIP inventories.

1. INTRODUCTION

We define a flexible assembly system (FAS) as a mechanized system that can perform versatile assembly operations to assemble more than one product and which requires minimal time to either change from the assembly of one product to another, or to other product options.

In this paper, we provide a method to determine two dimensions of the design of an FAS which meet conditions stated later. The first is the configuration of flexible assembly machines and the assignment of initially planned operations to them, which together determine the processing capability of each assembly component of the system. The second dimension is the selection of the number of automatic guided vehicles (AGVs) and pallets, which determine the material handling equipment capacity of the system. Further, it is shown how the method can be extended to add a third dimension, which is the determination of the size and allocation of buffers, which determine the capacity for work-in-process (WIP). The throughput or effective capacity of the FAS depends on the above three capacities, and their interaction on the mix of products planned for the system.

The goal in making these design decisions is to minimize the total resource costs (assembly machines, material handling equipment and buffers), and WIP costs, subject to the requirements of meeting aggregate demand.

The solution approach searches a necessary set of feasible undominated machine configurations. For each configuration there are three stages of analysis. The first stage, the main focus of this paper, applies deterministic analysis to find a good balanced work load for each machine using optimal or near-optimal branch-and-bound methods. For each solution that is found using the deterministic analysis, the second stage applies closed queueing network methods to calculate the optimal number of AGVs and pallets and the resulting throughput. This analysis recognizes that multiple products lead to usage fluctuations on the various machines which cannot always be offset by buffer capacities. The third stage applies simulation to test a sensible range of buffer capacity for WIP.

There are few papers on FAS design problems in the literature. Liu and Sanders (1986) attempt to find the optimal buffer sizes and the optimal number of pallets for an FAS, maximizing the throughput rate. They use a discrete simulation and a gradient-based search technique. Seliger, et al.(1984, 1987a,b) developed the MOSYS interactive support system for planning FASs which incorporates a queueing network method and simulation.

FASs can be evaluated with a wide range of parameters by either of these two techniques at different design stages. Therefore, the approach involves trial-and-error by the designer who interactively works with the support system. Many researchers use a Markovian single-class closed queueing network model to solve flexible manufacturing system (FMS) design-related problems. Vinod and Solberg (1985) and Dallery and Frein (1986) study the optimal configuration problem of FMSs which determines the number of pallets, AGVs, and flexible machines at each station when the number of stations and their workloads are known. This problem minimizes the system/operating cost, meeting product requirements. Yao and Shanthikumar (1986, 1987) study the optimal server allocation problem of FMSs which allocates a given number of flexible machines to stations in order to maximize the throughput rate. This problem assumes that the number of stations and pallets, and workloads of the stations are known.

This research was motivated by the following observations on FASs. Many companies producing automobiles, electronic components, computers, and electric consumer products are in the process of designing or implementing FASs because an increasing number of product variants, shorter product life cycles, the need to react flexibly to short-term variations of demand, and a highly competitive market necessitate a more flexible means of production [Owen (1984) and Spur, et al. (1987)]. In the near future, more FASs can be expected to be adopted because a) they will become more feasible technically, b) they are much cheaper than flexible machining systems, and c) the cost of manual labor involved in assembly is high [Boothroyd (1982), Riley and Yarrow (1983), Owen (1985), Hitz(1987)]. However, we lack aids for performing design and planning of FASs. Flexible and powerful design methods using analytical models need to be developed to retain an overview of complex interdependecies among system elements of FASs and to provide a small number of good design alternatives quickly to which the more detailed models such as simulation can be applied. This is because the variety of possible solutions increases with the number of functions integrated in FASs, which leads to a high workload for the designers. [Spur, et al. (1985), Suri and Diehl (1985)]

The remainder of the paper is organized in the following way. Section 2 describes characteristics for the class of FASs we are considering in this paper. Section 3 states design problems under investigation for the FASs. In Section 4, the framework of the solution approach is presented as a methodology which incorporates the analyses in a systematic way. Section 5 is devoted to the deterministic analysis. A mathematical formulation and both optimal and heuristic algorithms are presented. Computational results are reported for these algorithms. Section 6 is devoted to the stochastic analysis which employs a queueing network method and simulation. In Section 7, computational experience with the methodology is presented and, finally, Section 8 presents our conclusions.

2. CHARACTERISTICS OF FASs

The FAS under investigation has three characteristics. The first characteristic is that the FAS is a flow system where a base part enters the system and is processed by a cluster of stations containing flexible machines of type 1, followed by a cluster of stations containing flexible machines of type 2, continuing in this manner to completion. The machines of each type are typically of a particular technology with different capabilities from the other machine types in the system. Each task must be performed by a particular type of machine. For example, in the assembly of circuit boards, machines of type 1 might be Panasonic surface mount technology machines. All tasks requiring this technology must be performed by these machines. Further, the sequence of the machine types is assumed to be uniquely defined, an assumption which is true or not unreasonably restrictive for many products planned for specific technologies. Often, there will be just one type of machine (robot). We further keep the configured system manageable by permitting paralleling of an entire machine type, but not of only selected machines within a machine type. A flow system is common for FASs since the large volume and short operation times of FASs necessitates efficiency of a flow system.

The second characteristic is that flexible assembly machines, usually robots, have finite work space due to the limited physical configurations. Because a subpart feeding mechanism associated with each assembly operation uses some of the finite work space, we can assign only a finite number of operations to a robot. We assume that each operation uses the same size of the work space. This assumption is realistic when subparts are all of relatively similar sizes (Ammons et al., 1985) or when subparts are carried on a part magazine by an AGV to standardized docking stations installed around robots (Ranky, 1986). Under this assumption, the finite work space of a robot is rephrased as a staging capacity, R, where R specifies the maximum number of operations that can be assigned to the robot.

The third characteristic is that the FAS operates in mixed-model lines on which different product types are assembled simultaneously with a known mix ratio with very small batch sizes, or, in the case of a standard product with options, units with different option combinations are placed in an interspersed sequence. We use an aggregate product type to represent individual product types. Precedence diagrams of all the product types are merged and represented as one super precedence diagram for the aggregate product type. We assume that this super precedence diagram is *acyclic*. For example, we do not allow operation 1 to precede operation 2 in one product type and operation 2 to precede operation 1 in another product type. This assumption makes sense for assemblies such as automobiles or PC boards where the product types have similar operation sequences. However, we allow operation times to differ among the product types. Demand and operation times for the aggregate product type are specified as the sum of demands and the weighted average operation times among the product types, respectively. The variance of the operation times between models is an important input to the design of the material handling and buffer capacities, and is considered explicitly in the stochastic analysis described in Section 6.

3. PROBLEM STATEMENT

For the FAS characterized in the previous section, we are interested in the following design decisions for the three types of critical resource capacities. They are a) processing capacity (the number of robots or other automated machines to be installed and the assignment of operations to these machines), b) material handling capacities (the number of pallets and AGVs), and c) buffer capacities for WIP parts (buffer sizes and allocation). The goal of these design decisions is to minimize the total cost which consists of total resource cost and WIP inventory cost, subject to a constraint of meeting the aggregate demand. The solution approach we propose for these decisions is based upon a strategy of balancing average workload per machine which is commonly used when designing traditional manufacturing systems as well as FMSs (Berrada and Stecke, 1986). The solution approach decomposes the entire decision process into the steps outlined in Section 4 and detailed in Sections 5 to 6.

4. THE LINE BALANCING APPROACH TO FAS DESIGN

The overall strategy adopted to design a FAS consists of three main stages which are repeated with decreasing cycle times as needed, generating a small number of FAS design alternatives which have the small total cost and meet the aggregate demand. The reason that the cycle time reduction is sometimes needed follows.

Since we deal with only a finite number of buffer spaces, variable operation times can result in buffer blocking, which forces robots to be idle and causes loss of some processing capacities. As the variance increases, the blocking effects become more significant. Consequently, the FAS may not satisfy demand if it sticks with the decisions obtained from the deterministic analysis which assumes no variance of operation times. To resolve this problem, we need to add some slack processing capacity to the FAS. One way to do this in the solution approach we propose is to repeat the entire analyses with a smaller cycle time. This means that the deterministic analysis is reapplied to the FAS with higher demand level than the original one, d.

These concepts lead to the following methodology:

A Line Balancing Methodology for the FAS design

Step 1. Apply the deterministic analysis for each machine type by solving the balancing-workload procedure with an initial cycle time, C.

Step 2. Apply queueing network analysis (QNA) to find material handling capacities, given the decisions from the deterministic analysis.

Step 3. Apply simulation analysis to determine buffer capacities given the decisions from the deterministic analysis and QNA.

Step 4. If the current cycle time is small enough, terminate. Otherwise, reapply the deterministic analysis with a smaller cycle time and go to step 2.

The above methodology is depicted in Figure 1. It generates a set of FAS design alternatives as the cycle time reduces from C by trading off machine cost versus cost for other resources and WIP inventories.

The deterministic analysis treats the weighted average operation times as constant to determine the processing capacities. It assigns operations to machines such that the number of assembly machines is minimized, the aggregate demand is met, and average workload per machine is balanced. Given the decisions on the processing capacities from the deterministic analysis, the stochastic analysis takes into account the variance of operation times in order to determine the material handling and buffer capacities. The objective of the stochastic analysis is to minimize the associated resource costs plus the WIP inventory cost, subject to a constraint of meeting the demand. The following sections elaborate this solution approach, starting with the deterministic analysis.

5. DETERMINISTIC ANALYSIS

5.1 Mathematical Formulation

We first focus on a special case of the deterministic analysis by making two assumptions. The first assumption is that all machines in the FAS are of the same type which is flexible enough to perform all operations. The second assumption is that an FAS consists of a single line but not of parallel lines. Later in Section 5.3, we will relax the assumptions to study more general cases. In that section, we will also address the issue of balancing average workload per machine.

The deterministic analysis for FASs assigns operations to stations, subject to cycle time and staging capacity constraints. The cycle time constraint recognizes that in order to meet demand d, the sum of average operation times at each station should not exceed 1/d, the average interdeparture time by a completed part from the system. It is noted that 1/d is called a cycle time in the literature of the assembly line balancing (ALB) problem. The staging capacity constraint limits the number of operations that may be assigned to a particular station (machine) to that of its staging capacity. Operation assignments are also constrained by precedent requirements between tasks, and also by the nature of a flow system, which does not allow a part to revisit any station. The objective is to minimize the number of the flexible machines (robot assembly stations) along the line.

The notation used in this paper is presented below:

d	=	period demand,
C	=	cycle time, set to $1/d$,
R	=	staging capacity of the flexible assembly machine,
n	=	the number of operations (tasks),
t_j	=	weighted average operation time of operation j for $j=1$ to n,
a_j	=	staging space of operation j, set to 1 for $j=1$ to n from the assumption of equal staging space,
M	=	the number of robot assembly stations on a line.

Thus, the special case of the deterministic analysis can be mathematically stated as

P.1:

Min M

subject to

$$\sum_{j=1}^{n} t_j X_{ij} \leq C, \qquad i=1,\ldots,M, \qquad (1)$$

$$\sum_{j=1}^{n} a_j X_{ij} \leq R, \qquad i=1,\ldots,M, \qquad (2)$$

$$\sum_{i=1}^{M} X_{ij} = 1, \qquad j=1,\ldots,n, \qquad (3)$$

$$X_{ij} = 0 \text{ or } 1, \quad \text{for all } i \text{ and } j, \qquad (4)$$

a set of between-task precedence requirements must be met, (5)

a part does not revisit any station (dictated by the flow line), (6)

where X_{ij} is an assignment decision variable which is set to 1 when operation j is assigned to the ith station; otherwise, it is set to 0. Constraints (1) and (2) are the cycle time and staging capacity constraints, respectively. Constraints (3) and (4) are assignment constraints which force each operation to be assigned to only one station. See Talbot and Patterson (1984) for more formal presentation of constraints (5) and (6).

5.2 Solution Procedure for Problem P.1

We develop an optimal algorithm for Problem P.1 by generalizing an optimal algorithm for the traditional ALB for two reasons. Problem P.1 without constraint (2), that is, P.1 with $R=\infty$ becomes exactly a traditional ALB. Constraint (2) itself is well-structured due to $a_j=1$, which makes the generalization easy. We will name the optimal algorithm for Problem P.1 as ALB for FASs, to contrast with the traditional ALB throughout this paper.

We generalize Johnson's (1988) algorithm for the traditional ALB for the following four reasons. First, experiments with 64 ALBs used in the literature (Talbot and et al., 1986) showed that it is the fastest optimal algorithm among ones known. Second, it is designed to find a good feasible solution very quickly; hence, it can be easily modified to form a good heuristic algorithm. Third, it has low memory requirements. Fourth, the necessary modifications are not unduly complicated.

Johnson's algorithm enumerates, either explicitly or implicitly, a tree of feasible solutions using a 'laser' type, depth-first, branch-and-bound algorithm. The algorithm exploits the precedence structure, dominance rules, and bounds to give eight ways to improve its performance. We describe below the necessary modifications of the algorithm, starting with the tree generation procedure.

Modified Tree Generating Procedure

```
Step 1. Obtain problem data: number of tasks, n; task duration, $t_j$; cycle time C; staging
    space $a_j = 1$; staging capacity, R and precedence relations among tasks

    Set the current station number, k = 1.
    Set unused_Station_k_time to C.
    Set unused_Station_k_space to R
    Set the newly_selected_task = 1.
    Set the number of tasks currently assigned, p = 0.

Step 2. Add a task to the current station.

    Assign the newly_selected_task to Station k.
    Subtract $t_{(newly\_selected\_task)}$ from unused_Station_k_time.
    Subtract $a_{(newly\_selected\_task)}$ from unused_Station_k_space.
    Add 1 to p.
    Record newly_selected_task as Task_assignment_p.
    Set backtrack_task = 0.
    If p equals n, go to Step 6.

Step 3. Add another task at Station k.

    Task j is selected as the newly_selected_task, where:
        a) Task j is not already assigned;
        b) j > previous newly_selected_task;
        c) j > backtrack_task;
        d) $t_j \leq$ unused_Station_k_time;
        e) $a_j \leq$ unused_Station_k_space ; and
        f) All required preceding tasks of Task j are already assigned to a station.

    If no such task exists and newly_selected_task > 0, then {apply all the eight fathoming
    rules which will be described below; if fathomed, go to step 7. Otherwise} go to Step 4.
    If no such task exists and backtrack_task > 0, then go to step 7.
    { Apply fathoming rules 4 to 8; if fathomed, repeat step 3 to look for another task. }
    Otherwise, Task j becomes the newly_selected_task. Go to Step 2.

Step 4. Start a new station.
```

```
    Add 1 to k.
    Set unused_Station_k_time to C.
    Set unused_Station_k_space to R.

Step 5. Task j is the newly_selected_task, where Task j is the lowest numbered Task j
    which satisfies:
        a) Task j is not already assigned;
        b) All required preceding tasks of Task j are already assigned to a station; and
        c) j > backtrack_task.

    If no task exists and p = 0, enumeration is complete. Stop.
    If no task exists and p > 0, go to step 7.
    {Apply fathoming rules 5 to 8; if fathomed, repeat step 5 to look for another task.}
    Otherwise, go to Step 2.

Step 6. A complete solution has been completed.

    Save it as the Incumbent_Solution if it is the first solution, or if it is better than the
    previous Incumbent_Solution.
    {If the number of stations of this Incumbent_Solution equals the lower bound of the
    number of stations, stop with the optimum solution of ALB.}

Step 7. Backtrack.

    If unused_Station_k_time = C, reduce k by 1.
    Set backtrack_task = Task_assignment_p.
    Remove backtrack_task from Station k.
    Increase unused_Station_k_time by $t_{(backtrack\_task)}$.
    Increase unused_Station_k_space by $a_{(backtrack\_task)}$.
    Decrease p by 1.
    Set newly_selected_task = 0.
    If unused_Station_k_time < C and unused_Station_k_space < R, then go to Step 3
    Otherwise, go to Step 5.
```

The task re-numbering procedure and task duration incrementing rule of Johnson's algorithm are not affected by the addition of the staging capacity constraint (2). This is because they exploit only precedence relations and the differences of average task times.

Modified Node Fathoming Methods

In Johnson's algorithm, a node is fathomed in executing the tree generation procedure above, by explicitly constructing the subtree which emanates from the node, or by successful application of eight methods, each of which identify certain situations where a node cannot contain a solution which is superior to a solution already found. Four of the methods employ dominance rules and the other four employ bound arguments. We present the modifications of the eight methods which are necessary due to the additional constraint

(2). The spots in which these methods are placed in the tree generating procedure are already specified with mark { } in the above procedure.

No modifications are necessary for the first three dominance rules. They are Jackson Rule-1 dominance, Jackson Rule-2 dominance, and the first station dominance. This is true since swapping two tasks which are assigned to different stations does not violate the staging capacity constraint due to $a_j=1$ for all j.

The fourth dominance rule, which is the labeling dominance rule, is affected. This rule employs Schrage and Baker's (1978) labeling scheme, which allocates a numerical label to each task in such a way that the sum of labels of any feasible set of tasks is unique. Instead of a one-dimensional array in the original algorithm, a two-dimensional array is maintained to store two numbers of cumulative fractional stations found for a particular feasible set of tasks: one which is occupied by the staging space and the other by the task durations. The array is addressed by the sum of labels. If each of the two numbers of cumulative fractional stations associated with a newly formed set is greater than or equal to the corresponding number of a previous different grouping of the same set of tasks, then the node is fathomed.

A limitation of the labeling dominance rule is that the sum of labels of feasible sets of tasks frequently exceeds the available array space. Therefore, testing is performed only for sets of tasks which have a sum of labels less than the selected array size. Throughout the experimentation, an array size of 32,600 is used as in Johnson's algorithm. Limiting the array size in this way is not perceived to be a major drawback, as Johnson (1988) pointed out, since the greatest power of dynamic programming probably is elimination of nodes at the early portion of the tree, which is exactly where the sum of labels is sufficiently small.

In order to make the labeling dominance rule more effective, we use 2 pairs of the 2-dimensional array. Let P_1, P_2, P_3 and P_4 be the different nodes (i.e., the different groupings) of the same set of tasks such that P_k is generated by the tree generation procedure before P_i if $k < i$. Clearly, the nodes have the same label. Denote $M_C(P_k)$ and $M_R(P_k)$ as the number of cumulated fractional stations occupied by node P_k for k=1 to 3 in terms of the task durations and staging space, respectively. Suppose that P_1 does not dominate P_2, nor vice versa by this dominance rule; hence, $M_C(P_1) < M_C(P_2)$, $M_R(P_1) > M_R(P_2)$ or $M_C(P_1) > M_C(P_2)$, $M_R(P_1) < M_R(P_2)$. Then, these values are stored in the 2 pairs of the 2-dimensional array addressed by the label of the nodes. When the tree generation procedure generates node P_3, the labeling dominance rule undergoes the following logic. If $M_C(P_k) \leq M_C(P_3)$ and $M_R(P_k) \leq M_R(P_3)$ for k=1 or 2, then node P_3 is fathomed and the tree generation procedure backtracks. If $M_C(P_k) \geq M_C(P_3)$ and $M_R(P_k) \geq M_R(P_3)$ for *either* k=1 or 2, then $M_C(P_3)$ and $M_R(P_3)$ replace $M_C(P_k)$ and $M_R(P_k)$, respectively. If $M_C(P_k) \geq M_C(P_3)$ and $M_R(P_k) \geq M_R(P_3)$ for *both* k=1 and 2, then $M_C(P_3)$ and $M_R(P_3)$ replace $M_C(P_1)$ and $M_R(P_1)$, respectively, and memory spaces for $M_C(P_2)$ and $M_R(P_2)$ are left available for P_4 if P_4 is not fathomed.

All four bound methods are slightly modified. Given node P with partial assignments, let $B_C(j,P)$ and $B_R(j,P)$ for j=1 to 4 be the j^{th} bound method of Johnson's algorithm, specifying the number of stations needed at least to accommodate the task durations and staging space, respectively, of the remaining tasks. $\text{Max}_j \{ \max \{ M_C(P) + B_C(j,P), M_R(P) + B_R(j,P) \} \}$ is computed at each node and, if it is greater than or equal to the number of stations in the incumbent solution, the node is fathomed.

Heuristic Version

The optimal procedure explained above is modified to form a heuristic that eliminates search of portions of the enumeration tree which are considered to have a low probability of containing an optimal solution. The heuristic extends one given by Johnson (1988) and abandons nodes that use more than their ration of the available idle time or of the available staging space. Total permitted idle time {total permitted unused staging space} in a solution that is better than the incumbent solution is computed by determining (the number of stations in the incumbent solution - 1) x cycle time C {staging capacity R} - sum of task durations {sum of required staging space}. Each station's ration of idle time {unused staging space} is the total permitted idle time {unused staging space} divided by one less than the number of stations in the incumbent solution.

5.3 Extensions

We studied in the previous section ALB for FASs with a single line and one machine type. In this section,we consider more general ALB for FASs, that is, ALB for FASs with parallel lines or/and more than one flexible machine type.

ALB for FASs with Parallel lines

By allowing FASs with parallel lines which are identical to the single line, a smaller number of flexible machines can sometimes be achieved by finding a balance with lower balance delay. This may happen for an FAS with a single line in which staging capacity R is sometimes large compared to cycle time C, so that a large portion of R is not used up before a new station starts to be filled. Suppose an FAS has n_p parallel lines. Since each of the parallel lines is identical, each line is supposed to produce d/n_p in order to meet demand. This means that the cycle time for each line becomes n_p/d which is n_p times that of a single line which produces d. Thus, when we let M_p^* be the minimum number of the machines obtained by solving ALB with a single line and cycle time n_p/d, the minimum number of the machines for FASs with n_p parallel lines is M_p^* times n_p.

Algorithm 1 below finds the minimum number of the machines when parallel lines are permitted for an FAS. It searches for the optimum number of parallel lines by sequentially increasing the number of parallel lines and applying each time the optimal ALB algorithm for FASs with a single line. The algorithm terminates when increasing the number of parallel lines cannot lead to a better solution, that is, when the minimum number of machines found up to n_p parallel lines $\leq (n_p+1) \times T_M$ where T_M is the theoretical minimum number of machines for each line by staging space, which is the rounded up integer of the sum of staging space of tasks divided by staging capacity R.

Algorithm 1.

```
begin

 minimum_no <- a large positive number
 $n_p$ <- 1
 done <- false

 while not (done or $n_p$ > the number of tasks) do

  begin
  C <- $n_p / d$
  solve ALB for FASs with a single line and cycle time C

  if $M_p \times n_p$ < minimum_no then
   begin
   minimum_no <- $M_p \times n_p$
   save task assignments
   end

  if minimum_no $\leq (n_p + 1) \times T_M$ then
   done <- true
  else
   $n_p$ <- $n_p + 1$
  end

  end {while}

end.
```

Since there are usually multiple optimal operation assignments for problem P.1, we introduce a balancing-workload procedure which finds among them one that balances average workload per machine. (This is akin to the type II assembly line balancing problem as defined by Baybars (1986).) The procedure first finds the smallest cycle time for which the optimal solution for P.1 gives the same minimum number of machines. Given the optimal number of parallel lines, n_p^*, from Algorithm 1, this procedure sequentially increases the cycle time from perfectly balanced workload, TW/M_p^*, where TW is the sum of all average operation times. The procedure checks if there exists a feasible solution for problem P.1 for a given cycle time and M_p^*. This can be easily done by a variant of the optimal algorithm for Problem P.1. The procedure finds the smallest cycle time when it finds a feasible solution for the first time. Then the procedure, using the similar variant of the optimal algorithm, generates all feasible solutions at the smallest cycle time and finds one which best suits a balancing measure such as $\mathrm{Min} \sum_{i=1}^{M_p^*} | TW/M_p^* - W_i(C_s) |$ where C_s is the smallest cycle time and $W_i(C_s)$ is the average workload at station i for a feasible solution at C_s, which is the sum of average operation times assigned to station i.

ALB for FASs with Tasks Decomposable into Multiple Machine types

ALB for FASs can be applied where there are more than one machine type, in those cases where the problem can be decomposed into a series of independent problems, as discussed in Section 2. This is done simply by treating independently the group of tasks processed by each machine type according to the visited sequence and applying the balancing-workload procedure to each group of tasks.

5.4 Experimental Results

Experiments were performed on the 64 ALB problems assembled by Talbot, et al.(1986) which have been used in the literature of ALB as a benchmark of comparing different ALB algorithms. The number of tasks of these problems ranges from 7 to 111. The algorithm was coded in FORTRAN and run on IBM3600-600, using the VS-opt3 compiler. Each problem was run with a 3-second time trap. When the run time exceeded 3 seconds, the program stopped executing and printed out the incumbent solution. Twelve different staging capacities were experimented for each of the 64 problems. They were R = 2 to 10, 15, 20, and 30. For each R, we collected the following statistics: total CPU time taken to solve the 64 problems, the number of the verified optimal solutions found among 64, and the number of problems among 64 for which the last new incumbent solution was found after .1 and .5, respectively. The last two statistics were collected to see how quickly the algorithm found a (near) optimal solution. These statistics were summarized in Table 1. Table 1 also includes the statistics for Johnson's algorithm, which was shown to be successful for the traditional ALB (Johnson, 1988).

Table 1.

The total CPU time monotonically increased and then decreased with respect to staging capacity, R. The number of the verified optimal solutions monotonically decreased and then increased with respect to R. The interpretation of this monotonic behavior is as follows. For small R such as R=2 and 3, the staging capacity constraint was more constrained than the cycle time constraint, which made operation assignments easy. For $4 \leq R \leq 10$, it appeared that one constraint did not dominate the other and both constraints played an active role in assigning operations to stations, which resulted in more computation time. For large R such as $R > 10$, the cycle time constraint was more constrained than the staging capacity constraint. At R=7, the algorithm had the longest CPU time (32.607 seconds) and the smallest number of the verified optimal solutions (55 out of 64). This means that for nine problems, the tree generation procedure did not enumerate the entire tree of feasible operation sequences with the 3-second time trap.

The last two columns of Table 1 show that there were at most five (two) problems for which the last new incumbent solution was found after .1 (.5) second. This means that the algorithm found an optimal solution very quickly and most computation time was spent to verify its optimality by enumerating the tree and showing that there was no better solution. Thus, it is possible that the optimal algorithm with a small time trap can be used as a good heuristic .

We also conducted experiments for the heuristic algorithm given in Section 4.2. We collected the following statistics for each R: total CPU time taken to solve the 64 problems and the number of the optimal solutions found by the heuristic algorithm among ones verified by the optimal algorithm (column 4 of Table 1). These statistics were summarized in Table 2.

Table 2.

No significant difference in CPU time was observed between the heuristic and optimal algorithms. This is explained by the following two reasonings. First, the optimal algorithm found a (near) optimal solution very quickly as discussed above. Second, the new incumbent solution found by the optimal algorithm gave the tighter bounds, which subsequently helped fathom more nodes. However, the heuristic algorithm might miss the solution by fathoming a node containing the better solution. Therefore, the effects of fathoming a node by bounds might be diminished for the heuristic. As a result, there were several problems for which the heuristic algorithm did not terminate even after the 3-second time trap. The conclusion drawn is that the heuristic algorithm offers no detectable advantage over the optimal algorithm with a small time trap.

We also solved one example to demonstrate the ALB for FASs with parallel lines and multiple machine types. In the example, a part is processed first by machine type 1 and then by machine type 2. We used Sawyer's 30-task ALB (1970) and Kilbridge and Wester's 45-task ALB (1961) to characterize the group of tasks processed by machine type 1 and 2, respectively.[1] The staging capacities of the two machine types were specified as R_1=20 and R_2=15, and cycle time C as 54 seconds. The theoretical minimum number of machines per line was derived as 2 for machine type 1 and 3 for machine type 2. Algorithm 1 was applied to each group of tasks. The results were summarized in Table 3.

Table 3.

Table 3 shows that the smallest number of machines required for the FAS is 6 type-1 machines which are configured as 2 or 3 parallel lines, and 10 type-2 machines which are configured as 1 or 2 parallel lines. In order to find a solution which balances average workload among solutions giving the minimum number of machines, the balancing-workload procedure was applied to each optimal number of parallel lines, and the smallest cycle time was recorded in the last column of Table 3. Note that one second and two seconds were reduced for n_p=1 and 2 of machine type 2, respectively. When reliability of the system is an important design factor, the FAS can be configured as 3 parallel lines of machine type 1 (each with 2 machines) which are followed by 2 parallel lines of machine type 2 (each with 5 machines).

[1] We changed one task duration among Kilbridge and Wester's 45 tasks from 55 seconds to 30 since no task duration can be longer than the cycle time, 54 seconds.

6. STOCHASTIC ANALYSIS

If there is no variation of operation times, that is, there are no product type changes, we can use paced lines for each machine type for the FAS as in the traditional ALB (See Figure 2). However, as variance of operation times increases, the FAS using paced lines becomes less productive for the following reasons. If the speed of the paced lines is set to cycle time C, many parts are not completed and need a rework. On the other hand, if the speed of the lines is set to the longest possible service time to avoid any rework, machine utilization is very low. To cope with this drawback of the pace lines, the FAS can have asynchronous transfer lines like one using AGVs and buffers between stations in order to absorb fluctuating operation times as product types change (See Figure 3).

Given the decisions of the assignment of operations and the number of machines from the deterministic analysis, the stochastic analysis takes into account the variance of operation times in order to determine material handling and buffer capacities. The objective is to minimize the WIP inventory cost plus the associated resource costs subject to a constraint of meeting the demand.

We decompose the stochastic analysis into two subanalyses to make an entire decision process more tractable. They are queueing network analysis followed by simulation analysis.

6.1 Queueing Network Analysis (QNA)

QNA is used to determine material handling resource capacities, the number of pallets and AGVs, which are denoted as N and S_0 respectively, assuming that the FAS has sufficient buffer spaces available. The objective of QNA is to minimize cost for WIP inventories and the material handling resource subject to a constraint of meeting the demand. Input to QNA consists of the number of nodes, **M** (i.e., the number of robot stations the aggregate product needs to visit), average workload at each node, $\overline{W}=(W_1,W_2,...,W_M)$ (i.e., the sum of average operation times assigned to the station), the number of servers at each node, $\overline{S}=(S_1,S_2,...,S_M)$ (i.e., the number of parallel lines at the station), and average workload for the AGVs, W_0 (i.e., the number of nodes multiplied by the average transit time between stations). For example, for the FAS with two machine types in Figure 3, we have M=7 and $S_i=3$ for i=1 to 2 for the first two stations of machine type 1, and $S_i=2$ for i=3 to 7 for the following five stations of machine type 2.

When unloading a completed part and loading a new base part take place simultaneously at a loading/unloading station by a human worker or automated pallet changer, the FAS can be modeled as a single class closed queueing network (CQN) since the number of WIP parts (pallets) in the FAS remains unchanged (Stecke and Solberg, 1985). This is the case when a base part fixed on a pallet outside the system waits for a pallet containing a completed part to arrive at the loading/unloading station in order to enter the system. With assumptions of exponential service time at each node and sufficient buffer spaces, we have a product-form CQN model to which the algorithms by Vinod and Solberg (1985) or Dallery and Frein (1986) are applied. The algorithms determine the number of pallets and AGVs, minimizing cost for the associated material handling resources and WIP inventories such that the throughput of the CQN model is greater than or equal to demand. The mathematical statement for QNA can be given as follows:

P.2:

Min $z(N,S_0)$

subject to

$$TH(\mathbf{M},\overline{W},\overline{s},W_0;\ N,\ S_0) \geq d \tag{7}$$

where z is any increasing cost function with respect to N and S_0, and TH is the throughput function of the CQN model. Suri (1983) provides a theoretical explanation of the robustness of this closed queueing model.

6.2 Simulation Analysis

With the decisions from the deterministic analysis and QNA specified, simulation analysis allows more detailed representation for the FAS. Simulation analysis can include perturbation analysis (Ho, et al., 1984) or a search procedure (Liu and Sanders, 1986) to find the minimum number of buffers required to meet demand for the aggregate product type or demand for each product type. See also Yano, et al. (1988) for simulation study for subpart delivery policies and buffer size selection for FASs.

6.3 Monotonic Behavior of the Methodology

The line balancing methodology generates a set of FAS design alternatives as the cycle time reduces from C by trading off machine cost versus cost for other resources and WIP inventories. There is some evidence that this trade-off behaves nicely. First, the following lemma states the monotonic behavior of the number of machines with respect to the cycle time.

Lemma 1. The number of machines obtained by the deterministic analysis is nondecreasing for each machine type as the cycle time decreases.

Proof: Without any loss of generality, we fix a machine type arbitrarily and show that the lemma holds true. Let C_1 and C_2 be two cycle times such that $C_2 > C_1$, and $TM^*(C_1)$ and $TM^*(C_2)$ be the optimum total number of machines for the machine type obtained by the deterministic analysis with C_1 and C_2, respectively. Let $n_p^*(C_1)$ be the optimal number of parallel lines associated with C_1 for the machine type. Clearly, the optimal operation assignments associated with cycle time C_1 is feasible to problem P.1 with cycle time specified as $n_p^*(C_1) \times C_2$ since $n_p^*(C_1) \times C_2 > n_p^*(C_1) \times C_1$ and other constraints of P.1 are unaffected. Therefore, $TM^*(C_1) \geq TM^*(C_2)$ by the definition of $M^*(C_2)$.

Second, it appears that cost for buffers and WIP inventories also has a monotonic behavior with respect to the cycle time. When the cycle time is very small, the FAS is provided ample processing capacities and the deterministic analysis assigns many machines to the FAS. As a result, an arriving part seldom waits to be served and cost for buffers or WIP inventories can be minimal. In the other extreme case, when the cycle time is set to C, utilization must be near 1 to meet demand for each station whose average service time (workload) is close to C. This will cause a long queue of parts in front of the station, and the associated cost for buffers and WIP inventory will be maximal.

7. EXPERIMENT WITH THE LINE BALANCING METHODOLOGY

We conducted a number of experiments for the line balancing methodology which was implemented for only the first two stages, that is, the deterministic analysis and QNA based on the CQN model. Simulation analysis was not included and decisions on buffer capacities was out of scope in the experiments. The 2-second time-trap was used each time problem P.1 was solved in the deterministic analysis. We set the size of cycle time reduction, δ, to 1 time unit since we used integer-valued operation times.

We have a clear termination test for the implemented methodology. From Lemma 1, we know that resource cost for machines from the deterministic analysis is nondecreasing as the cycle time decreases. We also obtain lower bounds of the number of pallets and AGVs for QNA using the asymptotic bound analysis (Muntz and Wong, 1974). Subsequently, these bounds provide the lower bound of cost for those resources and WIP inventories. At a particular cycle time, if the cost of the incumbent solution is less than or equal to the sum of resource cost for machines at the cycle time and the lower bound of the cost for QNA, then the methodology terminates since it cannot find a better solution for any smaller cycle time.

We used the following parameter values for the aggregate product. We experimented two demand levels for a period: d=100 and 300. The number of operations was set to n=100. Operation times were randomly generated from a discrete uniform distribution ranging from 1 to 10. The precedences between operations was also randomly generated using a density parameter which was defined as the ratio of a number of present precedent arcs to the total number of possible precedent arcs, i.e., $\binom{n}{2}$. Each arc was equally likely. We experimented two density values, 0.1 and 0.5. Redundant precedent arcs are counted in the density computation.

We used the following parameter values for the resources. We considered an FAS with one machine type. Its staging capacity was specified as R=30 which was used by Ammons, et al. (1985) for a large computer manufacturing system. The available processing time of each machine was given as 10,000/period and average transfer time between two stations was given as 5. We assumed that an AGV was dedicated to each pallet from start of the assembly to finish. This is realistic because in an FAS with high demand and short operation times, an AGV is unlikely to drop the pallet off and then go and service other material handling requirements (Hall and Stecke, 1986). We had the total system cost specified as the following linear function: $TC = C_S \times \sum_{i=1}^{M} S_i + C_N \times N$ where C_S is resource cost per machine and C_N is the sum of cost for one WIP inventory and resource cost for one AGV and one pallet. We experimented two sets of (C_S, C_N): (2000,1200) and (2000,400).

Experimental results were summarized in Table 4.1 and 4.2. For each problem, the following statistics were recorded: the best solution $(\mathbf{M}, N, \bar{s})$ found and its cost, the number of times that the two analyses were solved, and total CPU time taken. The assignment of operations to stations were not shown for simplicity.

Table 4.1 and 4.2

8. CONCLUSIONS

In this paper, we presented a methodology which determines the three types of critical resource capacities for flexible assembly flow systems. Those three types are processing, material handling, and buffer capacities. The methodology which is based on a strategy of balancing average workload per machine takes the form of an iterative procedure which repeatedly solves deterministic and stochastic analyses as the cycle time decreases from C. The deterministic analysis balances average workload among machines while the stochastic analysis takes into account the variance of processing times. The methodology generates a set of FAS design alternatives with respect to the cycle time by trading off machine cost versus cost for other resources and WIP inventories. We also showed some evidence that this trade-off behaves nicely.

For the deterministic analysis, we presented an optimal algorithm for solving the ALB for FASs which may have parallel lines and more than one machine type. The optimal algorithm generalized Johnson's algorithm for the traditional ALB, based on the justifications given earlier. The computational results showed that the algorithm with 3 second time trap found from 55 to 64 verified optimal solutions among 64 ALBs, depending on the staging capacity, and that computation time is sensitive to the value of the staging capacity. The results also showed that the optimal algorithm found a (near) optimal solution very quickly; hence, the optimal algorithm with a small time trap such as .1 CPU second for about 100 tasks can be used as a good heuristic.

One way to improve the performance of the optimal algorithm is to find tighter bounds. From the observations that the four bound methods treat the cycle time and staging capacity constraints independently, tighter bounds can be achieved by considering both constraints simultaneously. In fact, Problem P.1 modified to exclude constraint (5) becomes the 2-dimensional bin-packing problem. Thus, developing more effective bound methods leads to the following two issues: a) how to find the tight lower bound of the 2-dimensional bin-packing problem and b) how to update the tight bound efficiently as one task is added to or subtracted from the current node. We leave these issues as further research.

For the stochastic analysis, we proposed two hierarchical subanalyses in order to make an entire decision process more tractable: QNA followed by simulation analysis. QNA with a single class product-form closed queueing network model was used to provide design decisions for material handling resources, assuming sufficient buffer spaces. Then, simulation analysis determines buffer sizes with a more detailed model for FASs, given decisions from the deterministic analysis and QNA.

ACKNOWLEDGEMENT

We would like to thank Professors M.M. Srinivasan and C.A. Yano for their helpful comments and suggestions on this paper.

REFERENCES

Ammons, J.C., Lofgren, C.B. and McGinnis, L.F., 1985, A large scale machine loading problem in flexible assembly. *Annals of Operations Research*, **3**, 319-332.

Baybars, I., 1986, A survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, **32**, 909-932.

Berrada, M. and Stecke, K.E., 1986, A branch and bound approach for machine load balancing in flexible manufacturing systems. *Management Science*, **32**, 1316-1335.

Boothroyd, G., Poli, C. and Murch, L., 1982, *Automatic Assembly* (New York: Marcel Dekker).

Dallery, Y. and Frein, Y., 1986, An efficient method to determine the optimal configuration of a flexible manufacturing system. *Proc. of the 2nd ORSA/TIMS Conf. on Flexible Manufacturing Systems, Ann Arbor, MI,* pp.269-282.

Hall, D.N. and Stecke, K.E., 1986, Design problems of flexible assembly systems. *Proc. of the 2nd ORSA/TIMS Conf. on Flexible Manufacturing Systems, Ann Arbor, MI,* pp.145-156.

Hitz, K., 1987, Flexible integrated computer-aided manufacturing systems increase productivity. *Robotics and Computer-Integrated Manufacturing*, **3**, 123-128.

Ho, Y., Suri, R., Cao, X., Diehl, G., Dille, J. and Zazanis, M., 1984, Optimization of large multiclass (non-product-form) queueing networks using perturbation analysis. *Large Scale Systems*, **7**, 1-16.

Johnson, R.V., 1988, Optimally balancing large assembly lines with 'FABLE'. *Management Science*, **34**, 240-253.

Kilbridge, M. and Wester, L., 1961, A heuristic method of assembly line balancing. *Journal of Industrial Engineering*, **12**, 292-298.

Liu, C. and Sanders, J., 1986, Stochastic design optimization of asynchronous flexible assembly systems. *Proc. of the 2nd ORSA/TIMS Conf. on Flexible Manufacturing Systems, Ann Arbor, MI,* pp.191-202.

Muntz, R.R. and Wong, J.W., 1974, Asymptotic properties of closed queueing network models. *Proc. of the 8th annual Princeton Conference on Information Sciences and Systems,* Princeton University.

Owen, T., 1984, *Flexible Assembly Systems* (New York: Plenum Press).

Owen, T., 1985, *Assembly with Robots* (New Jersey: Prentice Hall).

Ranky, P.G., 1986, The design of an end of arm tool management system for flexible assembly systems utilizing industrial robots. Report No. RSD-TR-21-86, Center for Research on Integrated Manufacturing, College of Engineering, The University of Michigan, Michigan.

Riley, F. and Yarrow, E., 1983, A new approach to assembly machine justification. *Proc. 2nd European Conf. Automated Manufacturing*, Birmingham, U.K.

Sawyer, J.F., 1970, *Line Balancing*. Machinery and Allied Products Institute, Washington, D.C.

Schrage, L. and Baker, K., 1978, Dynamic programming solution of sequencing problems with precedence constraints. *Operations Research*, **26**, 444-459.

Seliger, G. and Wieneke, B., 1984, Analytical approach for function oriented production system design. *Robotics and Computer-Integrated Manufacturing*, **1**, 307-313.

Seliger, G., Wiehweger, B. and Wieneke, B., 1987a, Descriptive methods for computer-integrated manufacturing and assembly. *Robotics and Computer-Integrated Manufacturing*, **3**, 15-21.

Seliger, G., Wiehweger, B. and Wieneke, B., 1987b, Decision support in design and optimization of flexible automated manufacturing and assembly. *Robotics and Computer-Integrated Manufacturing*, **3**, 221-227.

Spur, G., Furgac, I., Deutschlander, A., Browne, J. and O'Gorman, P., 1985, Robot planning system. *Robotics and Computer-Integrated Manufacturing*, **2**, 115-123.

Spur, G., Furgac, I. and Kirchhoff, U., 1987, Robot system integration into computer-integrated manufacturing. *Robotics and Computer-Integrated Manufacturing*, **3**, 1-10.

Stecke, K.E., 1983, Formulation and solution of nonlinear integer production problems for flexible manufacturing systems. *Management Science*, **29**, 273-288.

Stecke, K.E. and Solberg, J.J., 1985, The optimality of unbalancing both workloads and machine group sizes in closed queueing networks of multi-sever queues. *Operations Research*, **33**, 882-910.

Suri, R., 1983, Robustness of queueing network formulae. *J. of ACM*, **30**, 564-594.

Suri, R. and Diehl, G., 1985, Manuplan - a precursor to simulation for complex manufacturing systems. *Proc. of the Winter Simulation Conference.*

Talbot, F.B. and Patterson, J.H., 1984, An integer programming algorithm with network cuts solving the assembly line balancing problem. *Management Science*, **30**, 85-99.

Talbot, F.B., Patterson, J.H. and Gehrlein, W.V., 1986, A comparative evaluation of heuristic line balancing techniques. *Management Science*, **32**, 430-454.

Vinod, B. and Solberg, J., 1985, The optimal design of flexible manufacturing systems. *International Journal of Production Research*, **23**, 1141-1151.

Yano, C.A., Lee, H.F. and Srinivasan, M.M., 1988, Issues in the design and operation of flexible assembly systems for large products: a simulation study. Report No. 88-10, Dept. of Industrial and Operations Engineering, The University of Michigan, Michigan.

Yao, D.D. and Shanthikumar, J.G., 1986, On server allocation in multiple center manufacturing systems. Dept. of Industrial Engineering and Operations Research, Columbia University, New York.

Yao, D.D. and Shanthikumar, J.G., 1987, Optimal server allocation in a system of multi-server stations. *Management Science*, **33**, 1173-1180.

Problem type	R	Total CPU time*	No of V. Opt#	LNS > .1§	LNS > .5&
Traditional ALB	∞	6.309	64	3	1
ALB for FASs	2	1.811	64	2	0
	3	5.342	63	1	0
	4	11.931	61	0	0
	5	20.212	59	2	1
	6	20.625	58	3	0
	7	32.607	55	4	2
	8	28.694	56	5	2
	9	25.537	57	5	2
	10	15.820	60	3	1
	15	7.077	63	4	1
	20	6.907	63	3	1
	30	6.902	63	3	1

* Total CPU time: CPU time in seconds to solve the 64 ALB problems on IBM 3600-600, with 3 seconds time trap for each problem, using the FORTRAN VS-opt3 compiler

# No of V. Opt: the number of problems among 64 for which the verified optimal solutions were found by the optimal algorithm with 3 second time trap

§ LNS >.1: the number of problems among 64 for which the last new incumbent solution was found after .1 CPU second

& LNS > .5: the same as 'LNS > .1' except .5 CPU second instead of .1

Table 1. Experiment with the Optimal ALB algorithm for FASs

R	Total CPU time*	No of Opt#
2	1.909	62
3	5.364	60
4	11.782	56
5	14.198	55
6	15.853	55
7	24.996	51
8	24.891	49
9	24.330	50
10	18.673	54
15	9.933	58
20	9.873	58
30	9.882	58

* the same as in Table 1

# No of Opt: the number of the optimal solutions found by the heuristic among ones verified by the optimal algorithm (see column 4 of Table 1)

Table 2. Experiment with the Heuristic ALB Algorithm for FASs

Machine type	number of parallel lines n_p	minimum no. of machines per line M_p^*	total no. of machines $n_p \times M_p^*$	the smallest cycle time at M_p^* C_s **
1	1	7	7	***
	2	3	6	108
	3	2	6	162
2	1	10	10	53
	2	5	10	106
	3	4	12	***

** The required cycle time is 54 for a single line.
*** Not computed since the total number of machines is not optimal.

Table 3. Deterministic Analysis for the FAS with Parallel Lines & 2 Machine Types

Problem (d, density, C_S, C_N)	Solution **M**, N, $\overline{S}$	Total Cost	No. of analyses	CPU time (sec.)	Proven optimality at each stage ?
(100, 0.1, 2000, 400)	4, 11, (2)	20,400	4	1.58	Yes
(100, 0.1, 2000, 1200)	4, 11, (2)	29,200	6	2.15	Yes
(100, 0.5, 2000, 400)	4, 11, (2)	20,400	4	3.63	Yes
(100, 0.5, 2000, 1200)	4, 11, (2)	29,200	6	5.36	Yes
(300, 0.1, 2000, 400)	4, 31, (5)	52,400	6	2.29	Yes
(300, 0.1, 2000, 1200)	4, 21, (6)	73,200	9	3.06	Yes
(300, 0.5, 2000, 400)	4, 31, (5)	52,400	5	4.25	Yes
(300, 0.5, 2000, 1200)	4, 21, (6)	73,200	8	6.78	Yes

Table 4-1. Experiment with the Line Balancing Methodology

Problem (d, density, C_S, C_N)	Solution M, N, $\overline{S}$	Total Cost	No. of analyses	CPU time (sec.)	Proven optimality at each stage ?
(100, 0.1, 2000, 400)	4, 9, (2)	19,600	4	1.57	Yes
(100, 0.1, 2000, 1200)	4, 9, (2)	26,800	5	1.96	Yes
(100, 0.5, 2000, 400)	4, 9, (2)	19,600	4	3.46	Yes
(100, 0.5, 2000, 1200)	4, 9, (2)	26,800	5	4.27	Yes
(300, 0.1, 2000, 400)	4, 25, (5)	50,000	6	2.14	Yes
(300, 0.1, 2000, 1200)	4, 25, (5)	70,000	9	2.87	Yes
(300, 0.5, 2000, 400)	4, 25, (5)	50,000	6	4.50	Yes
(300, 0.5, 2000, 1200)	4, 25, (5)	70,000	9	6.49	Yes

Table 4-2. Experiment with the Line Balancing Methodology

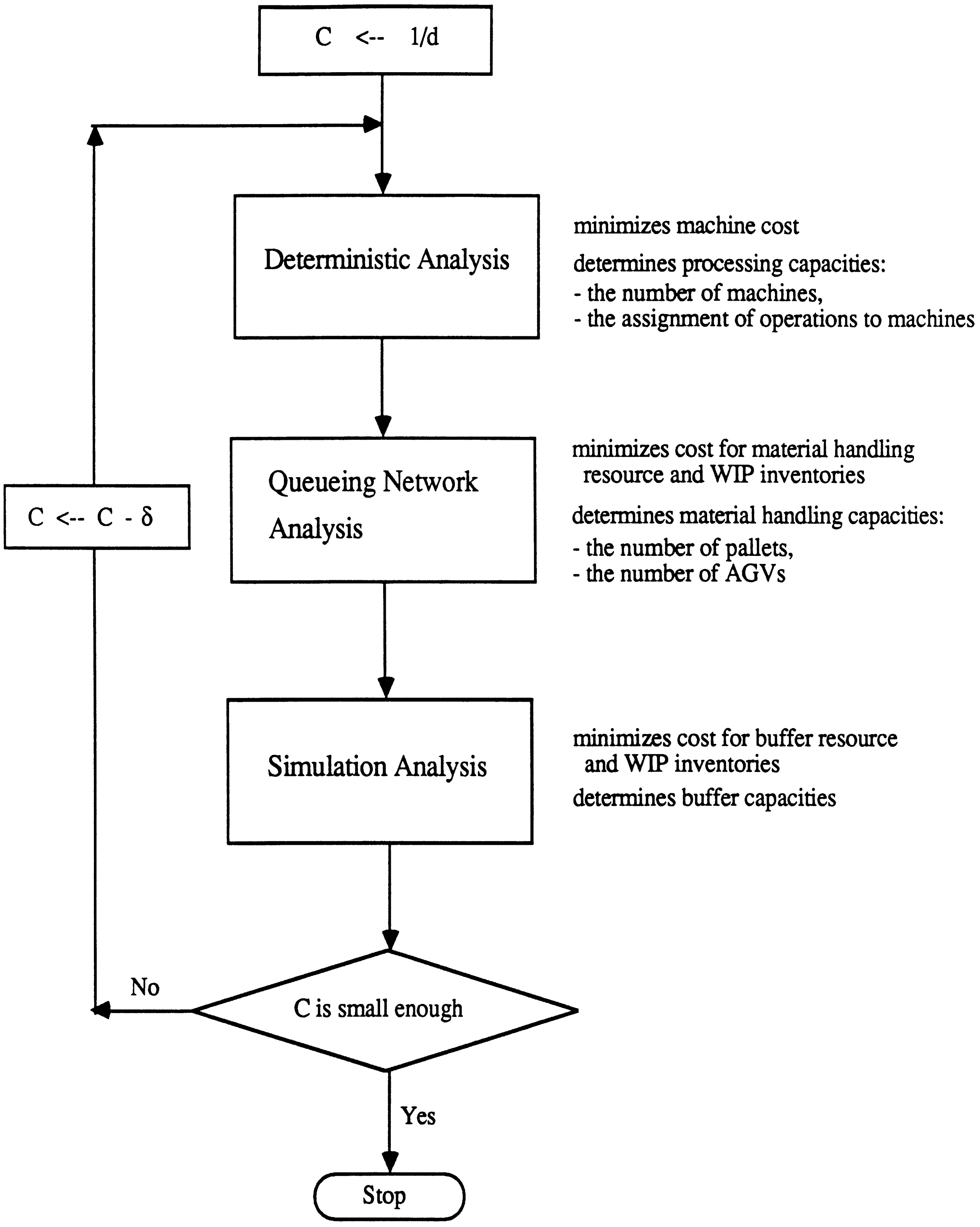
C <-- 1/d
Deterministic Analysis
minimizes machine cost
determines processing capacities:
- the number of machines,
- the assignment of operations to machines
Queueing Network Analysis
minimizes cost for material handling resource and WIP inventories
determines material handling capacities:
- the number of pallets,
- the number of AGVs
C <-- C - δ
Simulation Analysis
minimizes cost for buffer resource and WIP inventories
determines buffer capacities
No
C is small enough
Yes
Stop

Figure 1. A Line Balancing Methodology for the FAS Design

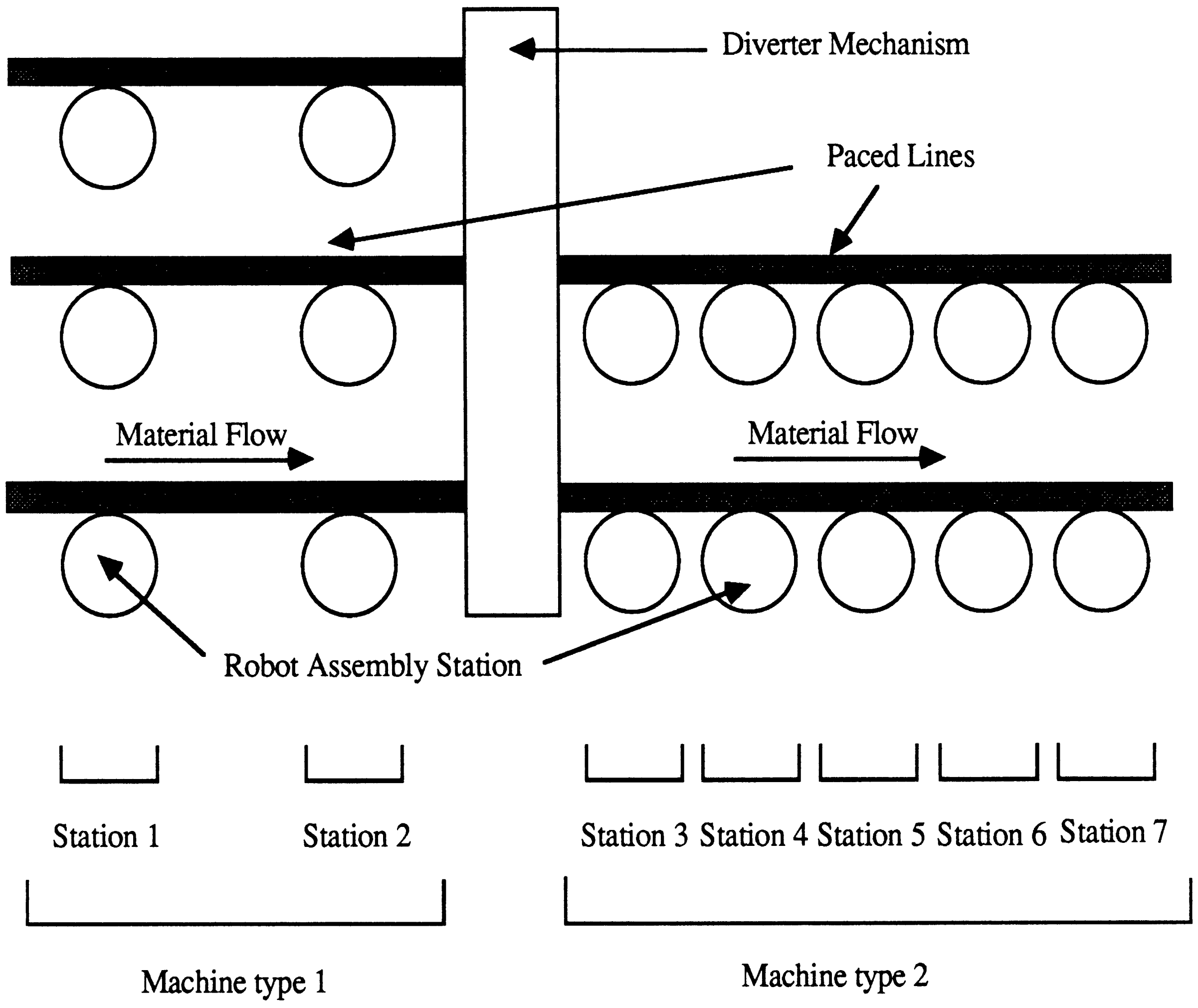

Figure 2. Constant Operation Time: an FAS with Two Machine Types

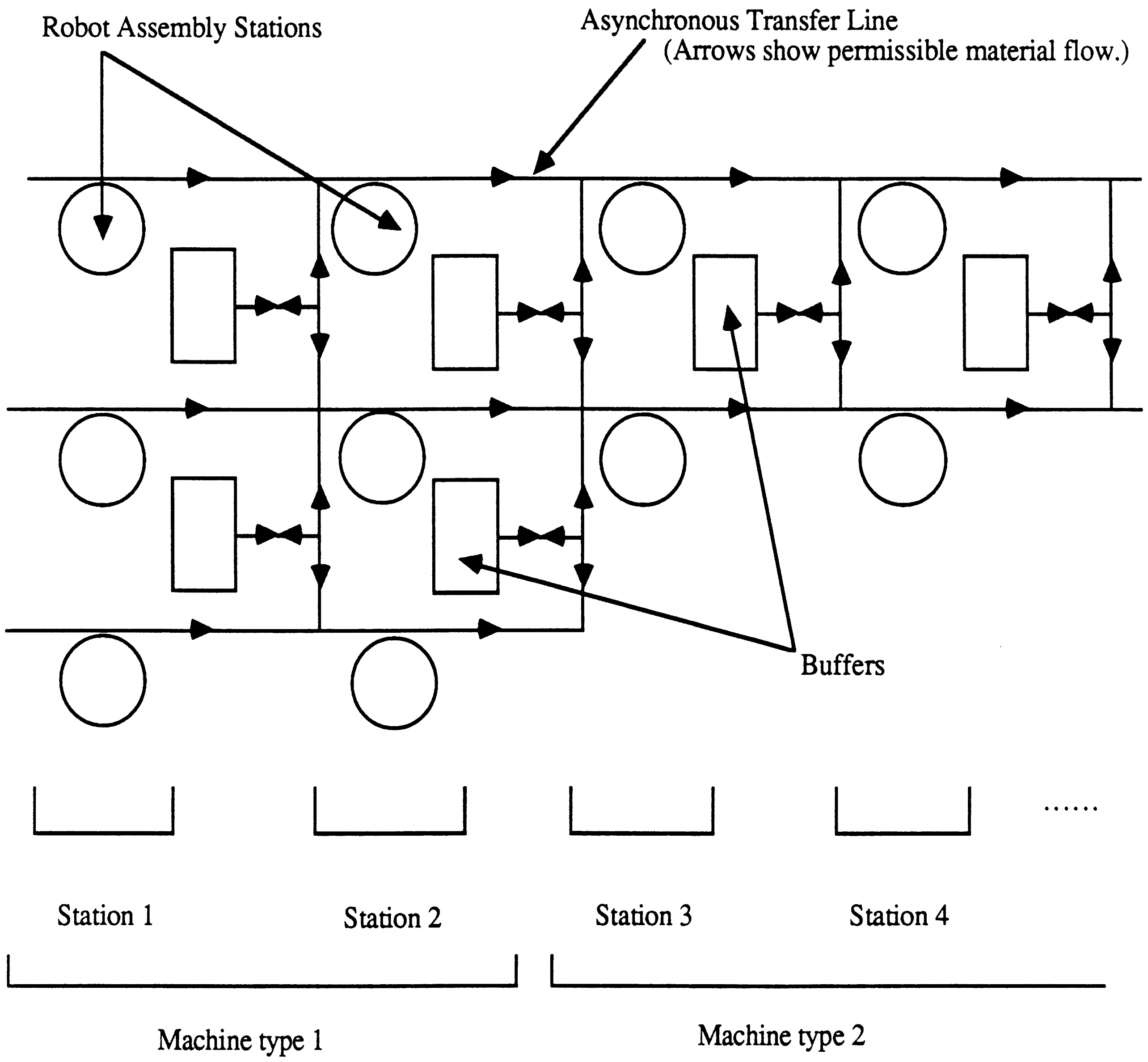

Figure 3. Variable Operation Time: an FAS with Two Machine Types