TECHNICAL RESEARCH REPORT

Sequencing Wafer Handler Moves to Improve the Performance of Hybrid Cluster Tools

by Manh-Quan T. Nguyen, Jeffrey W. Herrmann

T.R. 2000-31

ISR develops, applies and teaches advanced methodologies of design and analysis to solve complex, hierarchical, heterogeneous and dynamic problems of engineering technology and systems for industry and government.

ISR is a permanent institute of the University of Maryland, within the Glenn L. Martin Institute of Technology/A. James Clark School of Engineering. It is a National Science Foundation Engineering Research Center.

Web site http://www.isr.umd.edu

Sequencing wafer handler moves to improve the performance of hybrid cluster tools

Manh-Quan T. Nguyen and Jeffrey W. Herrmann
Department of Mechanical Engineering and Institute for Systems Research
University of Maryland
College Park, MD 20742
July 24, 2000

ABSTRACT

Cluster tools are highly integrated machines that can perform a sequence of semiconductor manufacturing processes. The sequence of wafer handler moves affects the total time needed to process a set of wafers. Reducing this time can reduce cycle time, reduce tool utilization, and increase tool capacity. This paper introduces the cluster tool scheduling problem for hybrid cluster tools, which are multiple-stage tools that have at least one stage with two or more parallel chambers. This paper presents algorithms that can find superior sequences of wafer handler moves. Experimental results show that the tool performance can be improved significantly if the wafer handler follows a cyclic sequence instead of using a dispatching rule.

1. INTRODUCTION

Manufacturing semiconductor devices involves three main steps: formation of p and n-type regions of the required conductivity within the semiconductor chip by doping; formation of reliable metal-semiconductor contacts on the surface of the chip; and encapsulation and packaging of the chip to provide protection and a convenient method of making electrical connection. In the first and second steps, the chips are processed together as wafers.

Most operations process each wafer individually. However, identical wafers move together from one process to the next. Each set of wafers is a lot. The container used to move and store the wafers in a lot is called a cassette.

A cluster tool is a manufacturing system with integrated processing modules linked mechanically. Typical cluster tools include load locks that store cassettes of wafers (cassette modules), process modules that modify the properties of the wafers, and single or multiple wafer handler(s) that transport the wafers (transport modules). These modules are linked together by an evacuated transfer space. Because it has multiple chambers, a cluster tool can process multiple wafers simultaneously.

After a lot enters the cluster tool, each wafer must undergo a series of activities. Each activity is performed in a different chamber. The wafer handler transports each wafer from one chamber to another. For example, the cluster tool shown in Figure 1 has one load lock (LL), which stores a cassette of wafers, and three process stages. Each wafer starts in the LL and must visit the first-stage chamber, one of the two second-stage chambers, and one of the two third-stage chambers before returning to LL.

A hybrid cluster tool has multiple stages, and at least one stage has two or more parallel, identical chambers. A wafer must visit exactly one chamber in each stage, so no wafer visits every chamber. A cluster tool can improve yield and device performance since wafers are exposed to fewer contaminates between process steps. The tool can include an in-situ metrology step that provides real-time feedback on process performance. A cluster tool with multiple parallel chambers can increase capacity and reduce cycle times by reducing the total time needed to process a lot of wafers. Moreover, it may be more reliable, since a single chamber's failure does not necessarily stop production. Semiconductor manufacturers are increasingly using cluster tools. Annual sales of cluster tools is projected to increase from \$11.2 billion in 1997 to \$21.9 billion in 2000 [1].

The sequence of wafers leaving the LL is not important, since the wafers are identical, and an activity's time is the same for every wafer. But the sequence of wafer handler moves, which determines when each activity starts, will change the total time needed to process a lot of wafers. We will call this the lot makespan. This paper addresses the problem of sequencing the wafer handler moves to minimize the lot makespan. Reducing the lot makespan can reduce cycle time, reduce tool utilization, and increase tool

capacity. Moreover, the lot makespan is a necessary component for calculating overall equipment effectiveness (OEE) and cost-of-ownership (COO), which are usually used to evaluate cluster tool performance [11, 12].

Like machine tools, cluster tools use controllers that supervise the tool operations, monitor the tool status, and handle exceptions that abnormal events cause. Under normal operation, sequencing wafer handler moves is an important responsibility. In practice, cluster tools use a push dispatching rule or a pull dispatching rule to sequence the wafer handler moves. After completing one move, the wafer handler will wait where it is (if no more wafers are ready to move) or start another move (if at least one wafer is ready). If multiple wafers are ready to be moved, the cluster tool must decide which move the wafer handler will perform.

In this case, the dispatching rule selects the next move. The pull rule gives priority to the wafer that has fewer remaining process steps. The push rule gives priority to the wafer that has more remaining process steps. Consider the cluster tool in Figure 1. Suppose there are unprocessed wafers in the LL, the first stage chamber is empty, and one second-stage and both third stage chambers each have a wafer that has finished processing and needs to move to the next stage. The pull rule will give priority to the wafer in a third-stage chamber. The push rule will give priority to the next unprocessed wafer in LL that needs to visit the first-stage chamber. Note that the wafer in the second-stage chamber cannot be moved because both third-stage chambers are full.

Although these rules help the cluster tool sequence the wafer handler moves, the push and pull dispatching rules do not guarantee that the resulting sequence has the optimal lot makespan for the given lot size, tool configuration, and activity processing times. For instance, consider a two-stage cluster tool that has two chambers in each stage. The first-stage activity requires 10 seconds. The second-stage activity requires 40 seconds. A wafer handler move requires 5 seconds. The lot has eight wafers. Figure 2 presents the Gantt charts of two sequences. The push dispatching rule generates first sequence, which has a lot makespan the of 285. The second sequence is an optimal sequence, which has a lot makespan the of 275. Each Gantt chart has five rows. The bottom row displays the wafer handler activities. The two rows above that displays the activities in the first-stage chambers. The top two rows display the activities in the second-stage chambers.

Cluster tool performance can be improved by determining a good sequence of wafer handler moves and providing it to the cluster tool controller, which can then use this sequence to direct normal operations. We will treat the problem as a deterministic machine scheduling problem, since the processing and move times have little variation, and small variations do not invalidate a given sequence.

This paper presents a branch-and-bound algorithm that can find an optimal sequence of wafer handler moves. It also presents a heuristic that can find good cyclic sequences quickly. The remainder of this paper is organized as follows. Section 2 reviews the related literature. Section 3 formulates the problem. Section 4 describes the branch-and-bound algorithm. Section 5 presents the heuristic. Section 6 presents experimental results that compare the performance of the algorithms. Section 7 summarizes our results and concludes the paper.

2. RELATED LITERATURE

Wood [2] derives formulas that relate the total lot processing time to the number of wafers in the lot for ideal sequential and parallel tools. Considering the transitions at the beginning and the end of the lot, Perkinson *et al.* [3] derive a model that relates the total lot processing time to the number of wafers. Both papers present linear models and identify two operating regions: in one region, the total lot processing time is constrained by the wafer handling time; in the other region, by the module process time. Venkatesh *et al.* [4] analyze the throughput of a sequential cluster tool with a dual-blade wafer handler. They also identify conditions when the tool operation is constrained by the wafer handler. Srinivasan [5] presents more detailed Petri net models for sequential and parallel tools and uses these to determine the steady state behavior of the tool. Herrmann *et al.* [6] study the impact of process changes on cluster tool performance. They propose using a network model for a prespecified sequence of wafer moves and cluster tool simulation software when the controller uses a dispatching rule or scheduling algorithm to sequence the wafer moves. They choose the cluster tool performance measure of interest is the lot makespan. None of the previous work addresses the problem of reducing the total lot processing time (lot makespan) by sequencing the wafer handler moves.

Jeng *et al.* [7] study the problem of sequencing robot activities for a robot-centered parallel-processor workcell where n jobs and m identical processors exist in the cell. They provide a branch and bound algorithm to find an optimal sequence of robot activities, which minimizes the total completion times. Hall *et al.* [8] discuss the problem of scheduling activities in a serial two or three machine manufacturing cell that is served by a robot. For multiple part-type problems in a two-machine cell, they provide an algorithm that simultaneously finds sequences of parts and robot moves to minimize the steady state cycle time. They also address a conjecture about the optimality of repeating one-unit cycles for a three-machine cell with general data and identical parts. Restricted to a special problem where the number of machines is arbitrary, but all parts are of the same type, Crama and van de Klundert [9], relying on the concept of pyramidal permutation, present a dynamic programming approach that finds an minimum one-unit cycle time in $O(m^3)$ time. Herrmann and Nguyen [10] analyze the problem of finding an optimal sequence of wafer hander moves for sequential cluster tools, which have just one chamber per stage.

3. PROBLEM STATEMENT

This paper focuses on single load lock, single wafer handler cluster tools. The following information about the cluster tool scheduling problem is given. The cluster tool has one load lock (LL) and S stages ($S > 1$). The number of chambers in one stage can be one or more. Let m_i be the number of chambers in stage S_i ($i = 1,\ldots, S$). Let $M = m_1 + \ldots + m_S$ be the total number of chambers. The chambers are numbered 1 to M. Let $\lambda = \min \{m_1, \ldots, m_S\}$. Each stage S_i has a wafer processing time p_i. The wafer handler move time is p_r. The lot has L identical wafers. Since each wafer must visit each stage and return to LL, the total number of moves is $L(S+1)$.

The sequence of wafers leaving LL is not important, since the wafers are identical. However, the sequence of moves affects the lot makespan C_{max}, the total time needed to complete all moves. The scheduling objective is to minimize the lot makespan. By convention, scheduling problems are described by triplets of the form $\alpha \mid \beta \mid \gamma$. The α field describes the machine environment. We use $\alpha = CTm_1\text{-}m_2$ to describe a two-stage hybrid cluster tool that has m_1 chambers in the first stage and m_2 chambers in the second stage. We use $\alpha = CTm_1\text{-}m_2\text{-}m_3$ to describe a three-stage hybrid cluster tool that also has m_3 chambers in the third stage. For our problem, the objective function $\gamma = C_{max}$.

When processing begins, the wafer handler is at LL, and all of the wafers are unprocessed and in LL. For convenience, we will number the wafers in the order they leave LL. Let $R_{0,j}$ denote the move that takes wafer j from LL to a chamber in S_1. Let $R_{i,j}$ denote the move that takes wafer j from a chamber in S_i to a chamber in S_{i+1} ($i = 1, ..., S-1$). Let $R_{S,j}$ denote the move that takes wafer j from a chamber in S_S to LL.

A feasible sequence of moves must satisfy the following constraints. For all $j = 1, \ldots, L-1$, $R_{0,j}$ must precede $R_{0,j+1}$. All wafers must follow the fixed sequence of processing steps. Therefore, for all $j = 1, \ldots, L$, and $i = 0, ..., S-1$, $R_{i,j}$ must precede $R_{i+1,j}$. Since there are no buffers (besides LL) to store wafers, S_{i+1} must have at least one free chamber before $R_{i,j}$ begins. That is, the wafer handler must have moved the previous wafer to the next stage.

The following facts describe the operation of the cluster tool. Each and every move requires the wafer handler. Since there is just one wafer handler, then, at any time, there is at most one move in process. The wafer handler cannot unload an empty or busy chamber and cannot load a busy or full chamber. (A full chamber has a wafer that has completed processing and is waiting to be moved.)

A chamber at stage S_i begins processing wafer j when move $R_{i-1,j}$ ends ($i = 1, ..., S$). This activity cannot be interrupted until the chamber is finished processing the wafer. For example, if a chamber in S_i starts processing at time t, then the chamber is busy during the interval $[t, t + p_i]$, and the wafer cannot be unloaded during that time.

Move $R_{i,j}$ starts when the chamber finishes processing wafer j and the wafer handler completes any previous move. $R_{i,j}$ requires p_r time units if the wafer handler is already at the chamber that processed wafer j (at LL if the move is $R_{0,j}$). $R_{i,j}$ requires $2p_r$ time units otherwise, for the wafer handler must move to the correct chamber at S_i before moving the wafer to a chamber at stage S_{i+1} (to LL if the move is $R_{S,j}$). The wafer handler cannot make anticipatory moves. That is, the wafer handler cannot move to the chamber before processing ends.

That is, $R_{0,1}$ requires p_r time units. For $j \geq 2$, $R_{0,j}$ requires p_r time units if and only if the previous move is $R_{S,k}$ for some $k < j$. For $i \geq 1$ and $j \geq 1$, $R_{i,j}$ requires p_r time units if and only if the previous move is $R_{i-1,j}$.

Special cases. We can identify two special cases. If $p_r = 0$, there is no scheduling problem since moves require no time, and all wafers move as soon as they are ready. If all $p_i = 0$, then an optimal solution is $R_{01}, R_{11}, \ldots, R_{S1}, R_{02}, R_{12}, \ldots, R_{S2}, \ldots, R_{0L}, R_{1L}, \ldots, R_{SL}$. This sequence has a lot makespan of $L(S+1)p_r$.

Cyclic sequences. Unless the lot size L is very small, a typical sequence has three phases, which we label filling-up, steady state (or cyclic), and completion. The chambers are empty when processing begins. Until the first wafer is completed, the tool is filling up with wafers. Then the tool is in a steady-state phase as it completes wafers and loads new wafers. When there are no more wafers to start, the tool enters the completion phase and completes wafers until the last wafer is unloaded from the last stage. Then processing ends.

We assume that $\lambda < L$. Let us define a λ-unit cycle as a subsequence that loads and unloads each stage λ times and thus completes λ wafers. Complete sequences formed by repeating a cycle in the steady state and completion phase we will call λ-unit cyclic sequences. Consider the sequences presented in Figure 2. The push sequence has a cyclic phase that starts at time 65 and follows the 1-unit cycle $R_{2,q} - R_{1,q+2} - R_{0,q+4}$, for q = 1, 2, 3, 4. The optimal sequence has a cyclic phase that starts at time 65 and follows the 2-unit cycle $R_{2,q} - R_{1,q+2} - R_{2,q+1} - R_{1,q+3} - R_{0,q+4} - R_{0,q+5}$, for q = 1 and 3.

The order of events in the completion phase resembles that in the cyclic phase, but the cycles are incomplete because there are no unprocessed wafers and some moves are no longer needed. The six moves in the completion phase of the push sequence form four incomplete cycles.

Note that the cycle does not define the filling-up phase, which ends with the first wafer being completed. There may exist more than one feasible filling-up phase for a given cycle. Unfortunately, there may be no optimal sequences in the set of λ-unit cyclic sequences.

Theorem 1: The best λ-unit cyclic sequence is not necessarily optimal for the problem $CTm_1\text{-}\ldots\text{-}m_S \mid\mid C_{max}$.

For the proof (by counterexample), please see Section 5.

4. THE BRANCH-AND-BOUND ALGORITHM

This section discusses Algorithm BB. The algorithm begins by using the push dispatching rule to construct a feasible sequence and then using the pull dispatching rule to construct another feasible sequence. The smaller lot makespan becomes the initial upper bound on the optimal lot makespan. For each partial solution constructed, the algorithm creates a lower bound by calculating the completion time of the last scheduled activity.

4.1. Algorithm BB

Given S, $m_1, \ldots, m_S, p_r, p_1, \ldots, p_S$, and L, Algorithm BB proceeds as follows. Note that this algorithm calls Algorithm P to generate sequences using the push and pull dispatching rules. Section 4.2 describes Algorithm P.

Step 0

Use the push dispatching rule (Algorithm P) to generate a feasible sequence. Use the pull dispatching rule (Algorithm P) to generate a second feasible sequence. Save the sequence with the smaller lot makespan as the current best sequence, and let the upper bound UB equal the lot makespan.

Step 1

Initialize the cluster tool. All L unprocessed wafers are in LL, and the wafer handler is at LL. All of the chambers are free. The current sequence is a sequence with no moves. Set C = 0, n = L, and $t_k = 0$ for all chambers k (k = 1, …, M).

Step 2

Based on the tool state, let t = C, the last move completion time, and identify all feasible moves. $R_{0,j}$ is feasible if $n > 0$, $j = L+1-n$, and there is a free chamber in S_1. This can begin at time t. $R_{i,j}$ $(0 < i < S)$ is feasible if wafer j is at chamber k in S_i and there is a free chamber in S_{i+1}. This can begin at $\max\{t, t_k\}$. $R_{S,j}$ is feasible if wafer j is at chamber k in S_S. This can begin at $\max\{t, t_k\}$.

Use the permutation condition and Theorem 3 to remove any dominated moves from further consideration.

Step 3

For each feasible move, form a new sequence with that move and perform one of the following steps (which calculates the lower bound and updates the tool state). Then go to Step 4.

- If the feasible move was $R_{0,j}$, then go to Step 3a.
- If the feasible move was $R_{i,j}$, $0 < i < S$, then go to Step 3b.
- If the feasible move was $R_{S,j}$, then go to Step 3c.

Step 3a

Reduce n by one. If the wafer handler was at LL, then the move completion time $C = t + p_r$. Otherwise, the move completion time $C = t + 2p_r$. Let q be the lowest-numbered free chamber in S_1. The wafer handler is now at chamber q, which now has wafer j, and $t_q = C + p_1$. Let $LB = t_q$. Go to Step 3d.

Step 3b

Let k be the chamber in S_i that was processing wafer j. If the wafer handler was at chamber k, then the move completion time $C = \max\{t, t_k\} + p_r$. Otherwise, the move completion time $C = \max\{t, t_k\} + 2p_r$. Chamber k is now free. Let q be the lowest-numbered free chamber in S_{i+1}. The wafer handler is now at chamber q, which now has wafer j, and $t_q = C + p_{i+1}$. Let $LB = t_q$. Go to Step 3d.

Step 3c

Let k be the chamber in S_S that was processing wafer j. If the wafer handler was at chamber k, then the move completion time $C = \max\{t, t_k\} + p_r$. Otherwise, the move completion time $C = \max\{t, t_k\} + 2p_r$. Chamber k is now free. $LB = C$. Go to Step 3d.

Step 3d

If $LB \geq UB$, then discard this new sequence. If this new sequence includes all $L(S+1)$ moves, the lot makespan equals C. Consequently, if $C < UB$, save this new sequence as the current best sequence and set $UB = C$.

Step 4

If any incomplete new sequences remain, select one, identify the corresponding tool state, and go to Step 2. Otherwise, stop and return the current best sequence and UB, its lot makespan.

4.2. Algorithm P

Given S, $m_1, \ldots, m_S$, p_r, $p_1, \ldots, p_S$, and L, Algorithm P proceeds as follows. Note that Algorithm P generates only non-delay schedules.

Step 1

Initialize the cluster tool. All L unprocessed wafers are in LL, and the wafer handler is at LL. All of the chambers are free. Set $t = 0$ and $n = L$. $t_k = 0$ for all chambers k ($k = 1, \ldots, M$).

Step 2

Based on the tool state, identify any feasible moves that could begin at time t. $R_{0,j}$ can begin at time t if $n > 0$, $j = L+1-n$, and there is a free chamber in S_1. $R_{i,j}$ ($0 < i < S$) can begin at time t if wafer j is at S_i, $t_k \leq t$, and there is a free chamber in S_{i+1}. $R_{S,j}$ can begin at time t if wafer j is at S_S and $t_k \leq t$. If there is exactly one feasible move, then perform that move and go to Step 4. If there is more than one feasible move and the dispatching rule is push, select the feasible move $R_{i,j}$ with the smallest value of i and go to Step 4. If there is more than one feasible move and the dispatching rule is pull, select the feasible move $R_{i,j}$ with the largest value of i and go to Step 4. Otherwise, go to Step 3.

Step 3

Let $t = \min\{ t_k : t_k > 0, k = 1, \ldots, M\}$. Go to Step 2.

Step 4

Update the tool state.

- If the selected move was $R_{0,j}$, then reduce n by 1. If the wafer handler was at LL, then the move completion time $C = t + p_r$. Otherwise, the move completion time $C = t + 2p_r$. Let q be the lowest-numbered free chamber in S_1. The wafer handler is now at chamber q, which now has wafer j, and $t_q = C + p_1$.
- If the selected move was $R_{i,j}$, $0 < i < S$, let k be the chamber in S_i that was processing wafer j. If the wafer handler was at chamber k, then the move completion time $C = t + p_r$. Otherwise, the move completion time $C = t + 2p_r$. Chamber k is now free. $t_k = 0$. Let q be the lowest-numbered free chamber in S_{i+1}. The wafer handler is now at chamber q, which now has wafer j, and $t_q = C + p_{i+1}$.
- If the selected move was $R_{S,j}$ let k be the chamber in S_S that was processing wafer j. If the wafer handler was at chamber k, then the move completion time $C = t + p_r$. Otherwise, the move completion time $C = t + 2p_r$. Chamber k is now free. $t_k = 0$. The wafer handler is now at LL.

Step 5

If all L(S+1) moves are complete, then stop. The lot makespan equals C. Otherwise, let t = C and go to Step 2.

4.3. Dominance Criteria

Algorithm BB uses some dominance criteria to avoid unnecessary searching. First, since all chambers in a stage are identical, move $R_{i,j}$ loads the lowest-numbered free chamber in S_{i+1} (i = 0, ..., S-1).

In addition, each wafer must be moved in turn. That is, $R_{i,j}$ must precede $R_{i,j+1}$ for all i = 0, ..., S, and j = 1, ..., L-1. If all stages have exactly one chamber (all $m_i = 1$, i = 1, ..., S), then all feasible sequences satisfy this condition, which we call the *permutation condition*. In a hybrid tool, there may exist feasible sequences that violate this condition. We will show however, that, in some cases, there exists an optimal sequence that does satisfy this condition.

Theorem 2. If, for each i = 1, ..., S, either $m_i = 1$ or $p_i \geq p_r$, then there exists an optimal sequence that satisfies the permutation condition.

Proof. Consider an optimal feasible sequence Q that violates the permutation condition. Then, find i such that $R_{k,j}$ precedes $R_{k,j+1}$ for k = 0, ..., i-1, but $R_{i,j+1}$ precedes $R_{i,j}$. Since $R_{0,j}$ must precede $R_{0,j+1}$, then i is at least 1.

If $m_i = 1$, then $R_{i-1,j+1}$ is infeasible until $R_{i,j}$ empties the chamber in S_i. Thus, $R_{i,j}$ must precede $R_{i-1,j+1}$, which precedes $R_{i,j+1}$. This is a contradiction, so S_i must have multiple chambers ($m_i > 1$). By the given, $p_i \geq p_r$.

Now, form a new sequence Q' by interchanging $R_{k,j}$ and $R_{k,j+1}$ for k = i, ..., S. We will show that Q' is a feasible sequence and that, since $p_i \geq p_r$, it does not increase the lot makespan. If Q' is not a permutation sequence yet, then we can repeat this construction until we have a feasible permutation sequence that does not increase the lot makespan of Q. Thus, this forms a feasible permutation sequence that is also optimal.

Q' is a feasible sequence because creating it only interchanges wafer j moves with wafer j+1 moves. If there was a chamber free to accept wafer j+1, then it is still free to accept wafer j (and vice versa).

Now consider two cases. In the first case, there is, in Q, a move between $R_{i-1,j+1}$ and $R_{i,j+1}$. Thus, in Q, $R_{i,j+1}$ requires $2p_r$ time units (since $R_{i-1,j+1}$ does not immediately precede it). Also, $R_{i,j}$ requires $2p_r$ time units (since $R_{i-1,j}$ does not immediately precede it). After the interchange, in Q', both moves still require $2p_r$ time units. For k = i+1,..., S, move $R_{k,j}$ in Q' requires the amount of time that $R_{k,j+1}$ required in

Q (and vice versa). Thus, all moves still require the same amount of time. Because $R_{i-1,j}$ precedes $R_{i-1,j+1}$, wafer j is complete at S_i before wafer j+1. Thus, in Q', $R_{i,j}$ can start at the time that $R_{i,j+1}$ started in Q. $R_{i,j+1}$ is delayed after the interchange and can certainly start in Q' when $R_{i,j}$ started in Q. Thus Q' delays no moves other than those interchanged and they can start at the same time, so the lot makespan is not increased.

In the second case, there is, in Q, no move between $R_{i-1,j+1}$ and $R_{i,j+1}$. Thus, in Q', there is no move between $R_{i-1,j+1}$ and $R_{i,j}$. Let Q1 be the subsequence in Q that occurs between $R_{i-1,j}$ and $R_{i-1,j+1}$. Now we need to consider the following sub-cases: (A) Q1 is empty, or Q1 is not empty and doesn't end with $R_{i-2,j+1}$; (B) Q1 ends with $R_{i-2,j+1}$.

Consider subcase A. Let t_d denote the time that $R_{i-1,j+1}$ becomes feasible (because S_{i-1} finishes processing wafer j+1 and there is a free chamber in S_i). Consider the move that precedes $R_{i-1,j+1}$. Let t_c denote the time that this move finishes. Thus, $R_{i-1,j+1}$ begins at $\max\{t_c, t_d\}$. Let t_a be the time that $R_{i-1,j}$ finishes. If Q1 is empty, $t_a = t_c$. Otherwise, because the first move in Q1 is not $R_{i,j}$, $t_c \geq t_a+2p_r$. Note that $R_{i,j}$ in Q and $R_{i,j+1}$ in Q' both require $2p_r$ time units. We need to show that, in Q', $R_{i,j}$ finishes no later than $R_{i,j+1}$ finishes in Q. Then, Q' does not increase the lot makespan because no remaining moves are delayed. One of the following conditions will hold:

- If $p_r \leq p_i \leq t_c-t_a+2p_r$ and $t_d \leq t_c$, then, in Q, $R_{i,j+1}$ completes at $t_c+3p_r+p_i$. In Q', $R_{i,j}$ completes at t_c+4p_r. Since $p_i \geq p_r$, $R_{i,j}$ finishes earlier.
- If $p_i \geq t_c-t_a+2p_r$ and $t_d \leq t_c$, then, in Q, $R_{i,j+1}$ completes at $t_c+3p_r+p_i$. In Q', $R_{i,j}$ completes at $t_a+2p_r+p_i$. Since $t_c > t_a$, $R_{i,j}$ finishes earlier.
- If $p_i \geq t_c-t_a+2p_r$ and $t_c \leq t_d \leq t_a+p_i-2p_r$, then, in Q, $R_{i,j+1}$ completes at $t_d+3p_r+p_i$. In Q', $R_{i,j}$ completes at $t_a+2p_r+p_i$. Since $t_d \geq t_c > t_a$, $R_{i,j}$ finishes earlier.
- If $p_i \geq p_r$ and $t_c \leq t_d$ and $t_d \geq t_a+p_i-2p_r$, then, in Q, $R_{i,j+1}$ completes at $t_d+3p_r+p_i$. In Q', $R_{i,j}$ completes at t_d+4p_r. Since $p_i \geq p_r$, $R_{i,j}$ finishes earlier.

Consider subcase B. Let t_d denote the time that $R_{i-1,j+1}$ becomes feasible (because S_{i-1} finishes processing wafer j+1 and there is a free chamber in S_i). Consider the move that precedes $R_{i-1,j+1}$. Let t_c denote the time that this move finishes. Because the last move in Q1 is $R_{i-2,j+1}$, $t_d = t_c+p_{i-1}$. Thus, $R_{i-1,j+1}$ begins at t_d and ends at t_d+p_r. Let t_a be the time that $R_{i-1,j}$ finishes. Because the first move in Q1 is not $R_{i,j}$, $t_c \geq t_a+2p_r$. Note that $R_{i,j}$ in Q and $R_{i,j+1}$ in Q' both require $2p_r$ time units. We need to show that, in Q', $R_{i,j}$ finishes no later than $R_{i,j+1}$ finishes in Q. Then, Q' does not increase the lot makespan because no remaining moves are delayed. One of the following conditions will hold:

- If $p_i \geq t_d-t_a+p_r$, then, in Q, $R_{i,j+1}$ completes at $t_d+2p_r+p_i$. In Q', $R_{i,j}$ completes at $t_a+2p_r+p_i$. Since $t_d > t_c > t_a$, $R_{i,j}$ finishes earlier.
- If $p_r \leq p_i \leq t_d-t_a+p_r$, then, in Q, $R_{i,j+1}$ completes at $t_d+2p_r+p_i$. In Q', $R_{i,j}$ completes at t_d+3p_r. Since $p_i \geq p_r$, $R_{i,j}$ finishes earlier.

This completes the proof. ♦

Finally, Algorithm BB uses Theorem 3 as a dominance property, since it prohibits a move $R_{i,j}$ if there exists another move $R_{p,q}$ that can be done first without delaying the completion of $R_{i,j}$. Note that using this condition limits Algorithm BB to the set of active schedules.

Theorem 3. Given Q1, a feasible partial sequence that satisfies the permutation constraint, move $R_{p,q}$ dominates $R_{i,j}$ if both are feasible and the following conditions hold: The last move in Q1 ends at time t. The wafer handler is at chamber k after this move (k may be LL). $R_{i,j}$ can begin at time $t_a \geq t$ and wafer j is at chamber c_a, which is not chamber k. $R_{p,q}$ can begin at time $t_b \geq t$ and wafer q is at chamber c_q. Either $c_q = k$ and $t_b+p_r \leq t_a$ or c_q is not k and $t_b+2p_r \leq t_a$.

Proof. Consider a complete feasible permutation sequence Q that begins with Q1 and $R_{i,j}$. Since c_a is not k, $R_{i,j}$ requires $2p_r$ time units. Form a new sequence Q' by moving $R_{p,q}$ before $R_{i,j}$. Because $R_{p,q}$ remained feasible from the end of Q1 to its position in Q, Q' is also a feasible permutation sequence. If $c_q = k$ and $t_b+p_r \leq t_a$, the wafer handler can complete $R_{p,q}$ at t_b+p_r and still begin $R_{i,j}$ at t_a. Otherwise, c_q is not k and

$t_b+2p_r \le t_a$. Still, the wafer handler can complete $R_{p,q}$ at t_b+2p_r and still begin $R_{i,j}$ at t_a. Thus, no move must be delayed, and the lot makespan of Q' is not worse than the lot makespan of Q. This completes the proof.

♦

In summary, Algorithm BB reduces its search space by applying the following three dominance criteria:

1. The wafer handler should always load the lowest-numbered free chamber in a stage.
2. The permutation condition, which forces the wafer handler to unload wafer j before wafer j+1 in the same stage ($R_{i,j}$ should precede $R_{i,j+1}$).
3. Theorem 2, which prohibits a move $R_{i,j}$ if there exists another move $R_{p,q}$ that can be done first without delaying the completion of $R_{i,j}$.

Note that if an instance of the scheduling problem does not satisfy the given of Theorem 2, then Algorithm BB may be unable to find an optimal solution.

5. THE TRUNCATED BRANCH-AND-BOUND ALGORITHM

This section presents an algorithm (Algorithm TBB) to find the best λ-unit cyclic schedule. (Recall that $\lambda = \min\{m_i\}$.) This algorithm is called "truncated" because, unlike Algorithm BB, it stops adding moves to a sequence when the sequence has $\lambda + 1$ wafer completions. Algorithm TBB first generates two feasible sequences using the push and pull dispatching rules and uses the smaller lot makespan as the upper bound on the best lot makespan. For a partial solution, the completion time of the last scheduled activity is the lower bound.

Algorithm TBB

Given $m_1, ..., m_S, p_r, p_1,..., p_S$, and L, Algorithm TBB proceeds as follows. Recall that $\lambda = \min\{m_1, \ldots, m_S\}$.

Step 0

Use the push dispatching rule (Algorithm P) to generate a feasible sequence. Use the pull dispatching rule (Algorithm P) to generate a second feasible sequence. Save the sequence with the smaller lot makespan as the current best sequence, and let the upper bound UB equal the lot makespan.

Step 1

Initialize the cluster tool. All L unprocessed wafers are in LL, and the wafer handler is at LL. All of the chambers are free. The current sequence is a sequence with no moves. Set C = 0, n = L, F = 0, d = 0, e = 0, and $t_k = 0$ for all chambers k.

Step 2

Based on the tool state, let t = C, the last move completion time, and identify all feasible moves. $R_{0,j}$ is feasible if n > 0, j = L+1-n, and there is a free chamber in S_1. This can begin at time t. $R_{i,j}$ $(0 < i < S)$ is feasible if wafer j is at chamber k in S_i and there is a free chamber in S_{i+1}. This can begin at $\max\{t, t_k\}$. $R_{S,j}$ is feasible if wafer j is at chamber k in S_S. This can begin at $\max\{t, t_k\}$.

Step 3

For each feasible move, form a new sequence and perform one of the following steps (which calculates the lower bound and updates the tool state). Then go to Step 4.

- If the feasible move is $R_{0,j}$, then go to Step 3a.
- If the feasible move is $R_{i,j}$, $0 < i < S$, then go to Step 3b.
- If the feasible move is $R_{S,j}$, then go to Step 3c.

Step 3a

Add one to e. Reduce n by one. If the wafer handler was at LL, then the move completion time $C = t + p_r$. Otherwise, the move completion time $C = t + 2p_r$. Let q be the lowest-numbered free chamber in S_1. The wafer handler is now at chamber q, which now has wafer j, and $t_q = C + p_1$. Let $LB = t_q$. If LB is greater than or equal to UB, then discard this new sequence.

Step 3b

Add one to e. Let k be the chamber in S_i that was processing wafer j. If the wafer handler was at chamber k, then the move completion time $C = \max\{t, t_k\} + p_r$. Otherwise, the move completion

time $C = \max\{t, t_k\} + 2p_r$. Chamber k is now free. Let q be the lowest-numbered free chamber in S_{i+1}. The wafer handler is now at chamber q, which now has wafer j, and $t_q = C + p_{i+1}$. Let LB = t_q. If LB is greater than or equal to UB, then discard this new sequence.

Step 3c

Add one to e. Let k be the chamber in S_S that was processing wafer j. If the wafer handler was at chamber k, then the move completion time $C = \max\{t, t_k\} + p_r$. Otherwise, $C = \max\{t, t_k\} + 2p_r$. Chamber k is now free. LB = C. Add one to F. If F = 1, let d = e. (This marks the first wafer completion.)

If $F = \lambda + 1$, consider the last e-d moves. If they form a λ-unit cycle, then repeat this cycle until all wafers are completed. Let C be the lot makespan of this complete cyclic sequence. If C < UB, then save the sequence as the current best sequence and let UB = C. Otherwise, discard this complete sequence. If the last e-d moves do not form a λ-unit cycle, then discard this new sequence.

Step 4

If any incomplete new sequences remain, select one, identify the corresponding tool state, and go to Step 2. Otherwise, stop. The current best sequence is an optimal sequence.

Figure 3 presents Algorithm TBB's search tree for an instance of CT1-2. Each node describes the last move in the partial sequence and the tool state $\langle x_1, x_2, x_3 \rangle$ after the move completes, where $x_i = 1$ if chamber i has a wafer and 0 otherwise. Four branches yield 1-unit cycles (listed below and denoted σ1, σ2, σ3, and σ4). The other branches (marked with a "$") are discarded because the cycle after the filling-up phase was not a valid 1-unit cycle.

$$\sigma 1 = R_{2,j} - R_{1,j+1} - R_{0,j+2},$$
$$\sigma 2 = R_{2,j} - R_{1,j+2} - R_{0,j+3},$$
$$\sigma 3 = R_{2,j} - R_{0,j+2} - R_{1,j+2}, \text{ and}$$
$$\sigma 4 = R_{2,j} - R_{0,j+1} - R_{1,j+1}.$$

Note that σ1 is similar to σ2, and σ3 is similar to σ4, but the resulting cyclic sequences have different filling-up phases. The cyclic sequence that uses σ1 and the cyclic sequence that uses σ4 do not use the third chamber.

Proof of Theorem 1: We provide a counterexample. Consider CT1-2 || C_{max}, with $p_r = 1$, $p_1 = p_2 = 5$, L = 9. $\lambda = \min\{1, 2\} = 1$. As discussed above, there are four feasible 1-unit cyclic sequences. The 1-unit cyclic sequences that use σ1, σ2, and σ3 all have a lot makespan of 85. The 1-unit cyclic sequence that uses σ4 has a lot makespan of 108. However, these are not optimal, as there exists a sequence with a lot makespan of 81. See Figure 4, which shows the 1-unit cyclic sequence that uses σ2 and this better sequence. ♦

6. RESULTS

We conducted experiments to determine the performance of Algorithms BB and TBB. Specifically, we wanted to know how much computational effort each required and whether either algorithm produced superior sequences. We wanted to determine how tool configuration, lot size, and the ratio of move time to process time affected performance.

We created 72 problem sets of instances, nine for each of eight tool configurations. Each problem set included ten randomly generated instances. Each problem set used different parameter values to generate its instances. The parameters were chosen so that 24 problem sets contained instances with short move times and long processing times, another 24 problem sets contained instances with approximately equal move times and processing times, and the remaining 24 problem sets contained instances with a long move time and short processing times. We will refer to these three classes as short, equal, and long moves, respectively.

Table 6 lists the parameter values for each move class, and Table 7 identifies the problem set for each combination of tool configuration, lot size, and move class. Given S, L, and a, b, c, d, we used the following generation scheme to create the instances. Note that all data are integers, so X ~ U[a, b] implies

that the random variable X has a discrete probability distribution and P{X = x} = 1/(b-a+1) for x = a, a+1, …, b.

$$p_r \sim U[a, b].$$
$$p_i \sim U[c, d], \text{ for } i = 1, \ldots, S.$$

The push and pull dispatching rules were the benchmarks. (Note that both algorithms begin with these sequences.) For each instance, we used Algorithm BB, Algorithm TBB, the push dispatching rule (Algorithm P), and the pull dispatching rule (Algorithm P) to find solutions. However, we halted Algorithm BB if it generated 100,000 nodes for an instance and Algorithm TBB if generated 50,000 nodes for an instance.

Tables 8 and 9 present the results for each problem set. The third and fourth columns present the average CPU times that Algorithm BB and Algorithm TBB required. The fifth column is the number of instances that the Algorithm BB solved within 100,000 nodes. Similarly, the sixth column is the number of instances that the Algorithm TBB solved within 50,000 nodes. The seventh, eighth, ninth, and tenth columns show the average lot makespan of the sequences that each algorithm constructed. The last four columns show the average percent improvement from the push and pull sequences.

From Table 8, we can see that, when L = 5 or the tool configuration is CT1-1, Algorithm BB was able to find optimal sequences. When the moves are short and the tool configuration is CT1-2 or CT2-1, Algorithm BB constructs sequences that are slightly better than the sequences that Algorithm TBB constructs. When the moves are equal or long and the problem is too hard for Algorithm BB to solve optimally, Algorithm TBB constructs much better sequences. For all problem sets, Algorithm TBB can find the best cyclic sequence with very little computational effort.

From Table 9, we can see that, when the moves are short and the tool configuration is CT1-1-1, neither Algorithm BB nor Algorithm TBB perform better than the push dispatching rule. When the tool configuration is CT1-2-2, however, Algorithm BB is slightly better than Algorithm TBB, and both are better than the dispatching rules. In most of the other problem sets, Algorithm TBB is better than Algorithm BB. When the tool configuration is CT1-1-1, CT1-2-2, or CT2-2-1, Algorithm TBB can find the best cyclic sequence with very little computational effort. For the CT2-2-2 instances, Algorithm TBB is searching for 2-unit cycles, so that requires additional computational effort.

Figure 5 presents the computing times those Algorithms BB and TBB required for CT1-2-2. Figure 6 illustrates the percent improvement from the push sequence to the best sequence found by Algorithm BB and TBB for CT1-2-2. Note that the lot does not affect the performance of Algorithm TBB as much as it affects the performance of Algorithm BB.

Table 6. Parameters for Move Classes.

Move Class	Parameters			
	a	b	c	d
Short	1	10	20	40
Equal	10	20	10	20
Long	20	40	1	10

Table 7. Problem sets.

Tool Configuration	S	L	Problem Set for Move Class		
			Short Moves	Equal Moves	Long Moves
CT1-1	2	5	1	13	25
CT1-1	2	10	2	14	26
CT1-1	2	15	3	15	27
CT1-2	2	5	4	16	28
CT1-2	2	10	5	17	29
CT1-2	2	15	6	18	30
CT2-1	2	5	7	19	31
CT2-1	2	10	8	20	32
CT2-1	2	15	9	21	33
CT2-2	2	5	10	22	34
CT2-2	2	10	11	23	35
CT2-2	2	15	12	24	36
CT1-1-1	3	5	37	49	61
CT1-1-1	3	10	38	50	62
CT1-1-1	3	15	39	51	63
CT1-2-2	3	5	40	52	64
CT1-2-2	3	10	41	53	65
CT1-2-2	3	15	42	54	66
CT2-2-1	3	5	43	55	67
CT2-2-1	3	10	44	56	68
CT2-2-1	3	15	45	57	69
CT2-2-2	3	5	46	58	70
CT2-2-2	3	10	47	59	71
CT2-2-2	3	15	48	60	72

Table 8. The performance of Algorithms BB and TBB on two-stage cluster tool configurations.

Tool Configuration	Problem Set	CPU time (mm:ss)		Instances solved		Average makespan				% improvement over push		% improvement over pull	
		BB	TBB	BB	TBB	BB	TBB	Push	Pull	BB	TBB	BB	TBB
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)
CT1-1	1	0:01	0:01	10	10	303.1	303.1	303.1	303.1	0	0	0	0
CT1-1	2	0:02	0:01	10	10	585.6	585.6	585.6	585.6	0	0	0	0
CT1-1	3	0:28	0:01	10	10	868.1	868.1	868.1	868.1	0	0	0	0
CT1-2	4	0:01	0:01	10	10	280.9	285.3	294.1	294.1	4.49	2.99	4.49	2.99
CT1-2	5	0:46	0:01	7	10	536.2	546.6	564.6	564.6	5.03	3.19	5.03	3.19
CT1-2	6	1:23	0:01	0	10	789.3	807.1	835.1	835.1	5.48	3.35	5.48	3.35
CT2-1	7	0:01	0:01	10	10	298.3	302.4	306.1	306.1	2.55	1.21	2.55	1.21
CT2-1	8	0:57	0:01	7	10	575.9	585.7	589.6	589.6	2.32	0.66	2.32	0.66
CT2-1	9	1:32	0:01	0	10	859.4	869.2	873.1	873.1	1.57	0.45	1.57	0.45
CT2-2	10	0:01	0:01	10	10	216.0	216.3	221.7	231.7	2.57	2.44	6.78	6.65
CT2-2	11	0:51	0:01	4	10	378.2	377.1	401.5	406.9	5.80	6.08	7.05	7.32
CT2-2	12	1:31	0:01	0	10	559.7	543.3	588.7	609.4	4.93	7.71	8.16	10.85
CT1-1	13	0:01	0:01	10	10	372.7	372.7	420.3	420.3	11.33	11.33	11.33	11.33
CT1-1	14	0:02	0:01	10	10	745.2	745.2	852.3	852.3	12.57	12.57	12.57	12.57
CT1-1	15	1:13	0:01	0	10	1117.7	1131.7	1284.3	1284.3	12.97	11.88	12.97	11.88
CT1-2	16	0:01	0:01	10	10	401.3	401.3	468.0	468.0	14.25	14.25	14.25	14.25
CT1-2	17	1:16	0:01	0	10	811.0	798.3	951.0	951.0	14.72	16.06	14.72	16.06
CT1-2	18	1:36	0:01	0	10	1294.0	1194.8	1434.0	1434.0	9.76	16.68	9.76	16.68
CT2-1	19	0:01	0:01	10	10	383.0	383.0	451.2	451.2	15.12	15.12	15.12	15.12
CT2-1	20	1:16	0:01	0	10	775.6	761.2	919.2	919.2	15.62	17.19	15.62	17.19
CT2-1	21	1:27	0:01	0	10	1134.4	1138.2	1387.2	1387.2	18.22	17.95	18.22	17.95
CT2-2	22	0:03	0:02	10	10	382.0	382.0	446.6	419.8	14.46	14.46	9.00	9.00
CT2-2	23	1:20	0:02	0	10	818.9	753.7	908.6	847.0	9.87	17.05	3.32	11.02
CT2-2	24	1:39	0:03	0	10	1263.8	1126.6	1370.6	1263.8	7.79	17.80	0.00	10.86
CT1-1	25	0:01	0:01	10	10	510.5	510.5	812.5	812.5	37.17	37.17	37.17	37.17
CT1-1	26	0:02	0:01	10	10	1021.0	1021.0	1700.5	1700.5	39.96	39.96	39.96	39.96
CT1-1	27	0:22	0:01	10	10	1531.5	1531.5	2588.5	2588.5	40.83	40.83	40.83	40.83
CT1-2	28	0:01	0:01	10	10	478.5	478.5	767.7	767.7	37.67	37.67	37.67	37.67
CT1-2	29	0:44	0:01	10	10	957.0	957.0	1607.7	1607.7	40.47	40.47	40.47	40.47
CT1-2	30	1:34	0:01	0	10	1803.7	1386.0	2447.7	2447.7	26.31	43.38	26.31	43.38
CT2-1	31	0:01	0:01	10	10	547.0	547.0	897.7	897.7	39.07	39.07	39.07	39.07
CT2-1	32	0:55	0:01	9	10	1098.9	1094.0	1884.7	1884.7	41.69	41.95	41.69	41.95
CT2-1	33	1:33	0:01	0	10	2083.6	1641.0	2871.7	2871.7	27.44	42.86	27.44	42.86
CT2-2	34	0:02	0:02	10	10	471.5	471.5	794.6	697.1	40.66	40.66	32.36	32.36
CT2-2	35	1:15	0:03	0	10	1210.0	943.0	1616.6	1507.0	25.15	41.67	19.71	37.43
CT2-2	36	1:35	0:04	0	10	2032.0	1414.5	2438.6	2204.1	16.67	42.00	7.81	35.82

Table 9. The performance of Algorithms BB and TBB on three-stage cluster tool configurations.

Tool configuration	Problem Set	CPU time (mm:ss)		# instances solved		Average makespan				% improvement over push		% improvement over pull	
		BB	TBB	BB	TBB	BB	TBB	Push	Pull	BB	TBB	BB	TBB
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)
CT1-1-1	37	0:01	0:01	10	10	369.0	369.0	369.0	371.1	0	0	0.57	0.57
	38	1:04	0:01	5	10	677.5	677.5	677.5	683.1	0	0	0.82	0.82
	39	1:44	0:01	0	10	986.0	986.0	986.0	995.1	0	0	0.91	0.91
CT1-2-2	40	0:17	0:02	10	10	334.9	337.8	349.9	350.8	4.29	3.46	4.53	3.71
	41	1:34	0:03	0	10	613.5	616.8	638.4	641.1	3.90	3.38	4.31	3.79
	42	2:00	0:04	0	10	889.4	895.4	926.9	930.5	4.05	3.40	4.42	3.77
CT2-2-1	43	0:34	0:03	7	10	349.2	350.3	359.6	356.0	2.89	2.59	1.91	1.60
	44	1:38	0:05	0	10	638.8	634.1	644.6	641.6	0.90	1.63	0.44	1.17
	45	2:05	0:07	0	10	923.8	918.1	929.6	926.6	0.62	1.24	0.30	0.92
CT2-2-2	46	0:49	0:46	6	5	290.1	294.0	296.1	318.1	2.03	0.71	8.80	7.58
	47	1:42	1:27	0	0	549.6	524.9	563.9	578.6	2.54	6.92	5.01	9.28
	48	2:10	1:44	0	0	817.3	760.3	831.9	839.5	1.76	8.61	2.64	9.43
CT1-1-1	49	0:03	0:01	10	10	521.7	521.7	591.8	591.8	11.85	11.85	11.85	11.85
	50	1:37	0:01	0	10	1083.0	1032.2	1195.8	1195.8	9.43	13.68	9.43	13.68
	51	2:10	0:01	0	10	1687.0	1541.7	1799.8	1799.8	6.27	14.34	6.27	14.34
CT1-2-2	52	1:13	0:04	0	10	515.6	509.2	573.7	577.5	10.13	11.24	10.72	11.83
	53	1:43	0:06	0	10	1107.9	1005.2	1161.7	1164.5	4.63	13.47	4.86	13.68
	54	2:15	0:08	0	10	1695.9	1501.2	1749.7	1753.5	3.07	14.20	3.28	14.39
CT2-2-1	55	1:14	0:06	0	10	539.3	525.2	600.6	600.6	10.21	12.55	10.21	12.55
	56	1:50	0:09	0	10	1158.0	1044.5	1216.6	1216.0	4.82	14.15	4.77	14.10
	57	2:19	0:11	0	10	1774.0	1446.5	1832.6	1832.6	3.20	21.07	3.20	21.07
CT2-2-2	58	1:12	1:17	0	0	534.4	537.9	577.2	549.9	7.42	6.81	2.82	2.18
	59	1:51	1:25	0	0	1110.0	1077.8	1169.2	1110.0	5.06	7.82	0.00	2.90
	60	2:10	1:42	0	0	1659.9	1595.8	1761.2	1659.9	5.75	9.39	0.00	3.86
CT1-1-1	61	0:01	0:01	10	10	670.0	670.0	1095.4	1071.6	38.84	38.84	37.48	37.48
	62	1:28	0:01	0	10	1585.6	1340.0	2267.4	2267.4	30.07	40.90	30.07	40.90
	63	1:54	0:01	0	10	2757.6	2010.0	3439.4	3415.6	19.82	41.56	19.26	41.15
CT1-2-2	64	0:35	0:04	10	10	694.5	694.5	1152.4	1105.3	39.73	39.73	37.17	37.17
	65	1:40	0:06	0	10	1984.7	1389.0	2360.4	2337.1	15.92	41.15	15.08	40.57
	66	2:06	0:08	0	10	3192.7	2083.5	3568.4	3521.3	10.53	41.61	9.33	40.83
CT2-2-1	67	0:50	0:06	10	10	650.0	650.0	1047.2	1069.6	37.93	37.93	39.23	39.23
	68	1:40	0:10	0	10	1836.3	1300.0	2167.2	2167.4	15.27	40.01	15.28	40.02
	69	2:09	0:12	0	10	2961.7	1950.0	3287.2	3309.6	9.90	40.68	10.51	41.08
CT2-2-2	70	1:09	1:15	0	0	780.0	883.3	1092.0	968.2	28.57	19.11	19.44	8.77
	71	1:44	1:25	0	0	1946.2	1903.2	2212.0	2100.0	12.02	13.96	7.32	9.37
	72	2:11	1:42	0	0	3049.8	2819.3	3332.0	3066.4	8.47	15.39	0.54	8.06

7. SUMMARY AND CONCLUSIONS

This paper studied the hybrid cluster tool scheduling problem. The goal is to improve tool performance by reducing the total lot processing time (the lot makespan). We have developed and tested two enumeration algorithms that can find sequences of wafer handler moves. Algorithm BB searches the entire set of sequences. Algorithm TBB focuses on cyclic sequences. This paper focused on cluster tools that have a single wafer handler and a single load lock. The scheduling objective is to minimize the lot makespan.

Algorithm TBB requires significantly less computational effort than Algorithm BB does. Compared to the sequences that current push and pull dispatching rules generate, we found that Algorithm TBB constructs sequences that reduce the lot makespan significantly. This is especially true when the move time and processing times are approximately equal and when the move time is longer than the processing times. Algorithm BB, however, requires excessive computational effort and cannot find better sequences (except for the smaller problem instances). Of course, these statements about performance apply to the specific problem instances that we created. Still, we believe that focusing on cyclic sequences reduces the search effort and yet yields very good sequences that improve cluster tool performance significantly. (Both the filling-up phase and the cycles must be considered when constructing a cyclic sequence.)

Although our experiments have focused on two- and three-stage cluster tools, the algorithms presented here can be applied to configurations with more stages.

Future work on cluster tool scheduling should consider more complex tool configurations and sequencing anticipatory moves, which position the wafer handler before the wafer is ready to be moved. Such anticipatory moves should further improve the cluster tool performance.

Acknowledgements

This research was supported by the Semiconductor Research Corporation and the National Science Foundation under grant DMI 97−13720. We would like to thank Professor Gary W. Rubloff, Mr. Brian F. Conaghan, Mr. Praveen Mellacheruvu, and Mr. Rock Z. Shi of the Institute for Systems Research, University of Maryland, for their useful insights. The research was performed using the facilities of the Computer-Integrated Manufacturing Laboratory at the University of Maryland.

Bibliography

1. Semiconductor Business News. "Applied Materials, Novellus, LAM Research lead cluster tool market" CMP Media Inc. 26 March 1998. 14 Jan. 2000. <http://www.semibiznews.com/stories/8c26tools.htm>
2. Wood, Samuel C, "Simple performance models for integrated processing tools," *IEEE Transactions on Semiconductor Manufacturing*, vol. 9, n. 3, pp. 320-328, 1996.
3. Perkinson, Terry L., Peter K. McLary, Ronald S. Gyucsik, and Ralph K. Cavin, "Single-wafer cluster tool performance: an analysis of throughput," *IEEE Transactions on Semiconductor Manufacturing*, vol. 7, n. 3, pp. 369-373, 1994.
4. Venkatesh, Srilakshmi, Rob Davenport, Pattie Foxhoven, and Jaim Nulman, "A steady-state throughput analysis of cluster tools: dual-blade versus single-blade robots," *IEEE Transactions on Semiconductor Manufacturing*, vol. 10, n. 4, pp. 418-424, 1997.
5. Srinivasan, R. S, "Modeling and performance analysis of cluster tools using Petri nets," *IEEE Transactions on Semiconductor Manufacturing*, vol. 11, n. 3, pp. 394-403, 1998.
6. Herrmann, Jeffrey W., Niranjan Chandrasekaran, Brian F. Conaghan, Manh-Quan T. Nguyen, Gary W. Rubloff, and Rock Z. Shi, "Evaluating the impact of process changes on cluster tool performance," to appear on *IEEE Transactions on Semiconductor Manufacturing*.
7. Jeng, Wu-De, James T. Lin, and Ue-Pyng Wen, "Algorithms for sequencing robot activities in a robot-centered parallel-processor workcell", *Computers Ops. Res*. v20, n2, pp. 185-197, 1993
8. Hall, Nicholas G., Hichem Kamoun, and Chelliah Sriskandarajah, "Scheduling in robotic cells: classification, two and three machine cells," *Operations Research*, vol. 45, n.3, pp. 421-439, 1997.

9. Crama, Yves and Joris van de Klundert, “Cyclic scheduling of identical parts in a robotic cell”, Operations Research, vol. 45, n. 6, Nov. – Dec. 1997.
10. Herrmann, Jeffrey W., and Manh-Quan T. Nguyen, “Sequencing wafer handler moves to improve the performance of sequential cluster tools,” Technical Report 2000-03, Institute for Systems Research, University of Maryland College Park, 2000.
11. Murphy, Robert, Puneet Saxena, William Levinson, “Use OEE; don’t let OEE use you,” *Semiconductor International*, pp. 125-132, Sep. 1996.
12. Dance, Daren L., Devid W. Jimenez, and Alan L. Levine, “Understanding equipment cost-of-ownership,” *Semiconductor International*, pp. 117-122, July 1998.
13. Nguyen, Manh-Quan T., “Improving Cluster Tool Performance By Finding The Optimal Sequence And Cyclic Sequence Of Wafer Handler Moves,” M.S. thesis, Institute for Systems Research, University of Maryland, 2000.

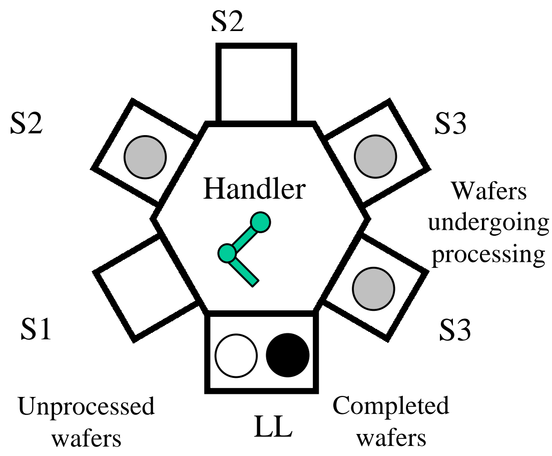
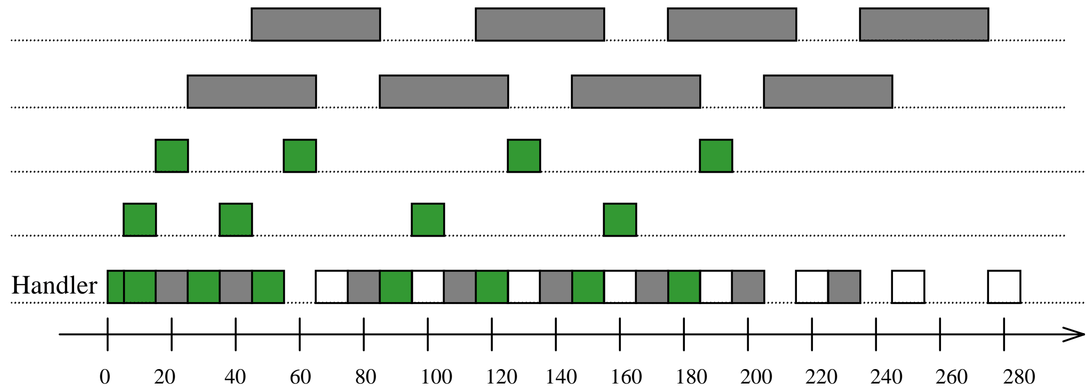


Figure 1. A CT1-2-2 Cluster Tool Configuration.

PUSH SEQUENCE: MS = 285



OPTIMAL SEQUENCE: MS = 275

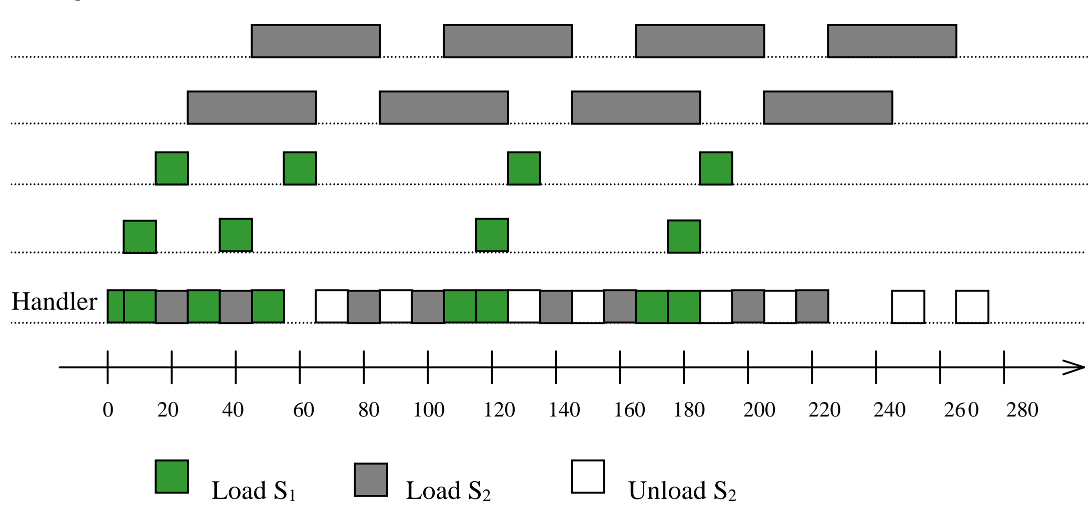


Figure 2. Sequences for a CT2-2 instance.

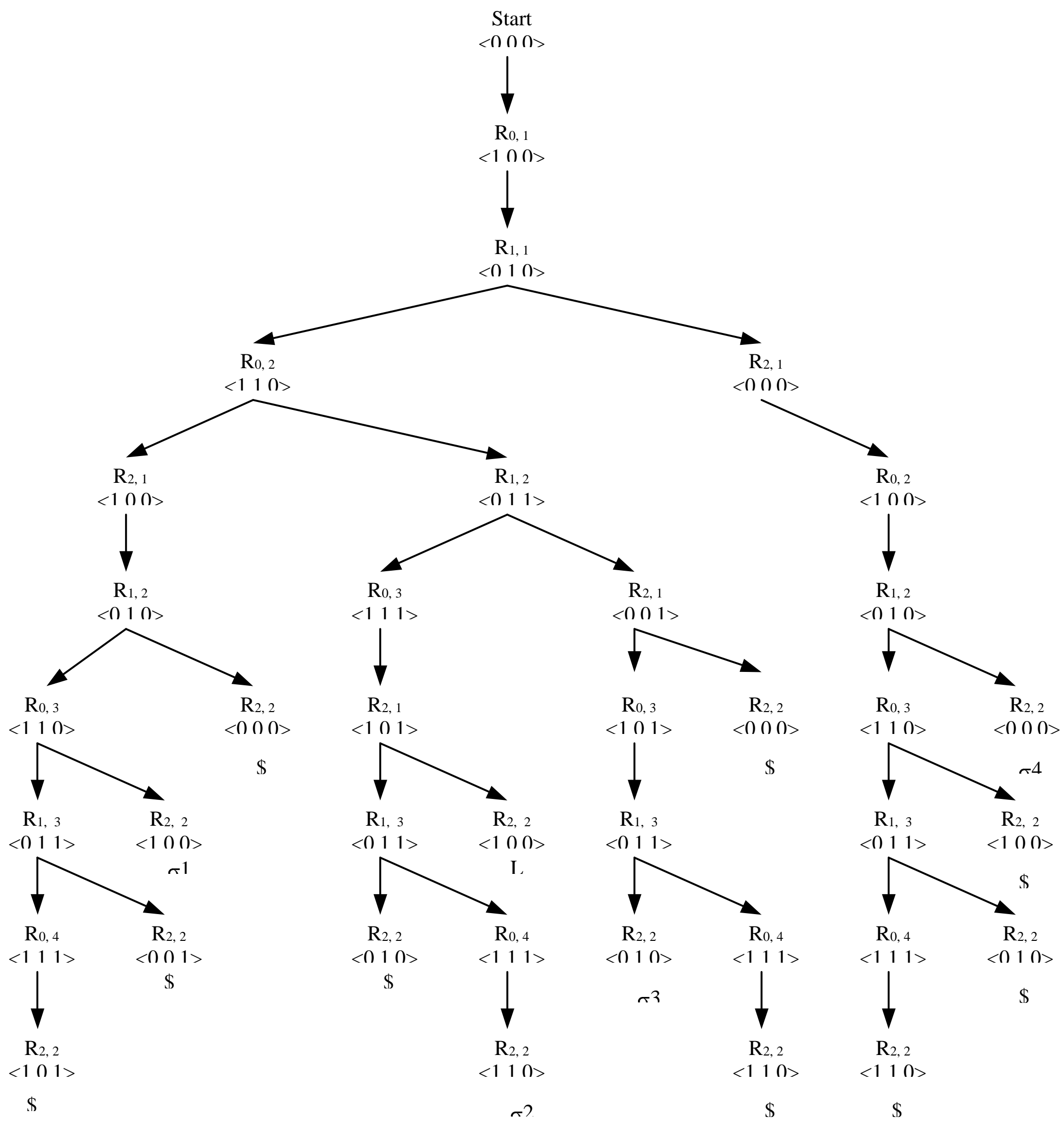
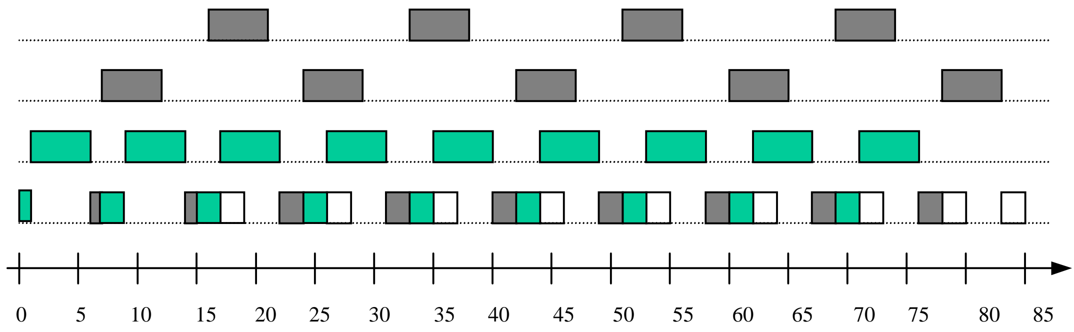


Figure 3. Algorithm TBB search tree for CT1-2.

1-unit cyclic sequence: MS = 85.



Better sequence: MS = 81.

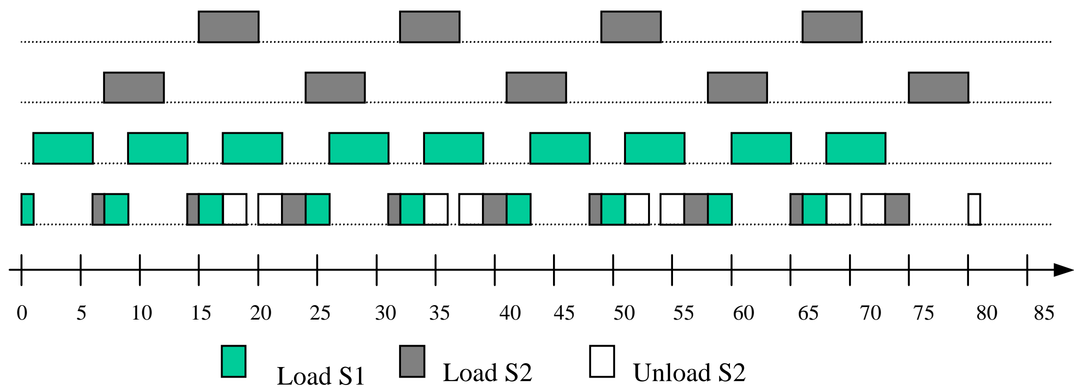


Figure 4. Sequences for a CT1-2 instance.

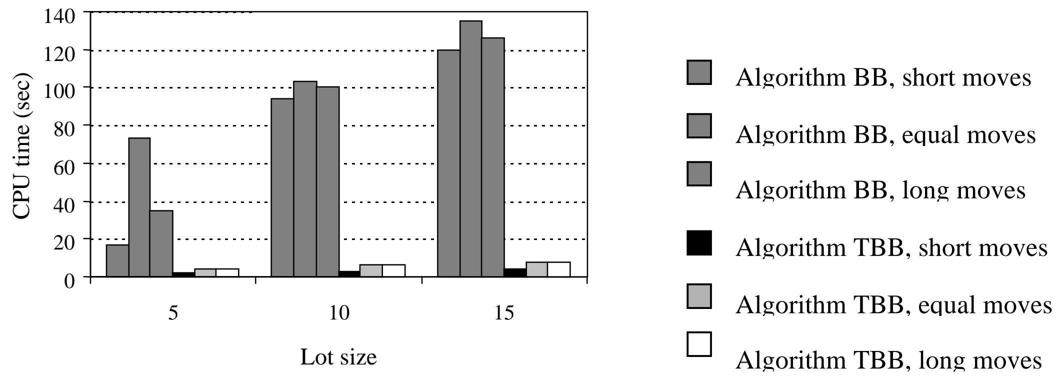


Figure 5.

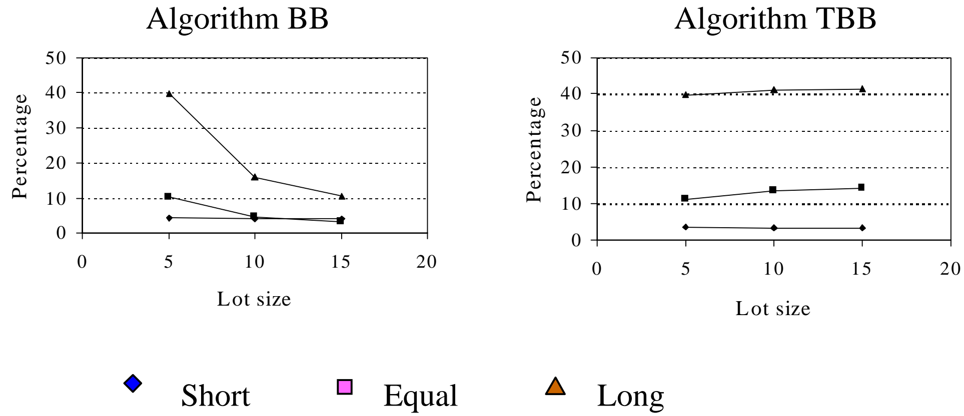


Figure 6.