Master's Thesis

Improving Cluster Tool Performance by Finding the Optimal Sequence and Cyclic Sequence of Wafer Handler Moves

by Manh-Quan Tam Nguyen
Advisor: Jeffrey Herrmann

M.S. 2000-3

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

IMPROVING CLUSTER TOOL PERFORMANCE BY FINDING THE OPTIMAL SEQUENCE AND CYCLIC SEQUENCE OF WAFER HANDLER MOVES.

by

MANH-QUAN TAM NGUYEN

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2000

Advisory Committee:

Assistant Professor Jeffrey W. Herrmann, Chair
Associate Professor Shapour Azarm
Professor Gary W. Rubloff

ABSTRACT

Title of Thesis:	IMPROVING CLUSTER TOOL PERFORMANCE BY FINDING THE OPTIMAL SEQUENCE AND CYCLIC SEQUENCE OF WAFER HANDLER MOVES.
Degree candidate:	Manh-Quan Tam Nguyen
Degree and year:	Master of Science, 2000
Thesis directed by:	Assistant Professor Jeffrey W. Herrmann Department of Mechanical Engineering

The research aims to develop algorithms that can minimize the total lot processing time (makespan) of cluster tools used for semiconductor manufacturing. Previous research focuses on finding an optimal sequence of wafer handler moves in a cluster tool that has one process chamber in each stage. In practice, if the number of chambers in a stage is more than one, either a pre-specified sequence of moves is given in advance or a dispatching rule is applied. No previous work has addressed the problem of finding an optimal sequence of wafer handler moves to improve performance of cluster tools with more than one chamber in a stage.

Cluster tools are highly integrated machines that can perform a sequence of semiconductor manufacturing processes. The performance of cluster tools becomes increasingly important as the semiconductor industry produces larger wafers with smaller device geometry. Some factors that motivate the use of cluster tools, instead of stand-alone tools, include increased yield and throughput, less contamination, and less human intervention.

In this research, the cluster tool is modeled as a manufacturing system with a material handling system (wafer handler). The model specifies all constraints that a feasible sequence of wafer handler moves must satisfy. The thesis develops two cluster tool scheduling algorithms. Given the lot size, the wafer handler move time, the in-chamber processing times, and the tool configuration the first algorithm, based on a complete forward branch-and-bound algorithm, searches for an optimal solution from the set of all feasible sequences of wafer handler moves. The second algorithm, a truncated branch-and-bound algorithm, quickly searches for the best solution from the set of feasible cyclic sequences of wafer handler moves. For simple tool configurations, analytical makespan models are also derived.

The results show that, in many cases, the search algorithms can significantly reduce the total lot processing time. This reduces tool utilization, reduces manufacturing cycle times, and increases tool capacity.

IMPROVING CLUSTER TOOL PERFORMANCE BY FINDING THE OPTIMAL SEQUENCE AND CYCLIC SEQUENCE OF WAFER HANDLER MOVES.

by

MANH-QUAN TAM NGUYEN

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2000

Advisory Committee:

Assistant Professor Jeffrey W. Herrmann, Chair
Associate Professor Shapour Azarm
Professor Gary W. Rubloff

©Copyright by

Manh-Quan Tam Nguyen

2000

DEDICATION

TO MY PARENTS, BROTHERS, AND SISTERS

ACKNOWLEDGMENTS

I wish to thank my advisor, Dr. Jeffrey W. Herrmann, who helped me understand the theory of scheduling. It was under his guidance that I was able to develop the algorithms, which form such an important part of this thesis. I am grateful to him for his patience, encouragement and cornucopia of suggestions throughout the proceeds of this project work.

Thanks are equally due to Dr. Gary W. Rubloff, and my colleagues Mr. Brian F. Conaghan, Mr. Niranjan Chandrasekaran, Mr. Praveen Mellacheruvu and Mr. Rock Shi who helped me gain invaluable insights into the area of Semiconductor Manufacturing. This acknowledgement would be incomplete without the mention of Dr. Edward Lin, whose help and suggestions while coding the algorithms proved to be invaluable.

TABLE OF CONTENTS

Dedication ……..…………………………..…………..……………………………… ii
Acknowledgement ……………………………………………………………….. iii
List of Tables ………………………………………………………………………vii
List of Figures……………………………………………………………………..ix

CHAPTER 1: INTRODUCTION..1

1.1 Motivation..1
1.2 Research Objective 2
1.3 Models and Insights 3
1.4 Outline of Thesis Report..5

CHAPTER 2: BACKGROUND AND LITERATURE REVIEW..........................7

2.1 Semiconductor Manufacturing Overview..7
2.2 Cluster tools 9
2.3 Current Push and Pull Dispatching Rules..11
2.4 Related Literature...15
2.4.1 Cluster tool scheduling 15
2.4.2 Robot scheduling 17
2.5 Summary..19

CHAPTER 3: PROBLEM FORMULATION...20

3.1 Notation..20

3.2 Problem statement 21
3.2.1 Assumptions 21
3.2.2 Objective 21
3.2.3 Constraints 22
3.2.4 Permutation versus non-permutation 24
3.2.5 Problem notation 24
3.3 Number of feasible sequences 25
3.4 Determining the lot makespan 30
3.5 Summary 32

CHAPTER 4: BRANCH-AND-BOUND ALGORITHM 33

4.1 The forward branch-and-bound algorithm 33
4.2 Dominance criteria 40
4.3 Number of feasible sequence under the first and second dominance criteria 47
4.4 Better lower bounds 48
4.5 Summary 50

CHAPTER 5: THE CYCLIC BEHAVIOR OF THE CLUSTER TOOLS AND TRUNCATED BRANCH-AND-BOUND ALGORITHM 51

5.1 Performance of a cluster tool processing a finite lot size 51
5.2 The CT1-1 Problems 54
5.3 The CT1-1-1 problems 55
5.4 THE CTm_1-...-m_S | | C_{max} PROBLEMS 65
5.5 Summary 73

CHAPTER 6: IMPLEMENTATION 75

6.1 Introduction 75
6.2 Implementation as executables 77

6.2.1 Cluster tool models using Algorithm BB 80
6.2.2 Cluster tool models using Algorithm TBB 83
6.2.3 Cluster tool models using the push and pull dispatching rules 90
6.2.4 Cluster tool models using the prespecified sequence of wafer handler moves 95
6.3 The graphic user interface for the two- and three-stage cluster tool models 97
6.4. Integration into subfactory model 101
6.5 Summary 102

CHAPTER 7: RESULTS AND DISCUSSION 104

7.1 Performance of the search using Algorithm BB 104
7.2 Performance of the search using Algorithm TBB 110
7.3 Summary 115

CHAPTER 8: SUMMARY AND CONCLUSIONS 116

APPENDIX A 119
APPENDIX B 123
APPENDIX C 125
APPENDIX D 135
APPENDIX E 139
APPENDIX F 144

REFERENCES 159

List of Tables

1. Numbers of wafers in each stage at each level τ associated with the selected move 30
2. Cycle time and lot makespan of the 1-unit cyclic sequences that use cycle σx (x = 1, 3, 4, and 5) 57
3. Cycle time and makespan of $\sigma 2$-sequence, using $f1 = R_{0,1} - R_{1,1} - R_{0,2} - R_{2,1} - R_{1,2}$, for some instances 60
4. Cycle time and makespan of $\sigma 2$-sequence, using $f2 = R_{0,1} - R_{1,1} - R_{2,1} - R_{0,2} - R_{1,2}$, for some instances 61
5. Cycle time and makespan of the 1-unit cyclic sequences when $L = 15$, $p_r = 13$, $p_1 = 18$, $p_2 = 14$, $p_3 = 7$ 62
6. MS_1 and MS_{22} as functions of L 63
7. TBB, push, and pull sequences for CT1-2 69
8. Variables used to build cluster tool models 76
9. Status of a two-stage tool using the push dispatching rule 91
10. Status of a two-stage tool using the pull dispatching rule 91
11. Status of a three-stage tool using the push dispatching rule 92
12. Status of a three-stage tool using the pull dispatching rule 93
13. Two-stage problem sets 105
14. Three-stage problem sets 105
15. Computing times and percentage improvement of Algorithm BB over the push and pull sequences for some two-stage cluster tool configurations (OH = 0) 106
16. Computing times and percentage improvement of Algorithm BB over the push and pull sequences for some three-stage cluster tool configurations (OH = 0) 107

17. Computing time and lot makespan of a CT1-2-2 when increasing number of nodes and applying better lower bound 109

18. Computing times and percentage improvement of Algorithm TBB over the push and pull sequences for some two-stage cluster tool configurations (OH = 0) 111
19. Computing times and percentage improvement of Algorithm TBB over the push and pull sequences for some three-stage cluster tool configurations (OH = 0) 113

LIST OF FIGURES

1. Configuration of a 1-2-2 cluster tool……………………………………………… 10
2. Sequences constructed following the push and pull dispatching rules and the optimal sequence……………………… 14
3. Partial outtree of the 1-1-2 cluster tool……………………………………………29
4. Outtree graph of the CT1-1 (3 wafers per lot)…………………………………..37
5. A scenario demonstrating the third dominance criterion………………………..45
6. Gantt charts of push and optimal sequences for CT2-2 (L = 8)………………...53
7. Two-unit cycle versus one-unit cycle……………………………………………...64
8. Search tree of a CT1-2 using Algorithm TBB……………………….. 70
9. The best 1-unit cyclic sequence and the optimal sequence for CT1-2………....72
10. An arbitrary outtree graph…………………………………………………………79
11. Flow chart for program BB……………………………….............................81
12. Flow chart for program TBB……………………… 84
13. Flow chart for program GCCS...87
14. Flow chart for program CheckCycle...89
15. Flow chart for program P 94
16. Flow chart for program PS 96
17. Interaction of GUI and cluster tool models………………………………….97
18. GUI for two-stage cluster tools……………………………..........................98
19. GUI for three-stage cluster tools……………………………….........................98
20. GUI for two-stage cluster tools after 3 runs…………………..........................100
21. GUI for three-stage cluster tools after 4 runs………………….........................100
22. Information flow in the integrated subfactory model..102

A-1. Complete outtree graph of a CT1-1-2 (L = 3) when the first and second dominance criteria are not active………………………………………………120
A-2. Complete outtree graph of a CT1-1-2 (L = 3) when the first and second dominance criteria are active…………………………………………………….122

B-1. Gantt chart of a complete σ1-sequence (L = 3)………………………………. .. 123

B-2. Gantt chart of a complete σ2-sequence (L = 4)………………………………. .. 124

C-1. Gantt chart for a complete σ1-sequence (L =3)……………………………….. 125

C-2a. Gantt chart of a complete σ3-sequence with filling-up f1 (L = 4)…………….. 126

C-2b. Gantt chart of a complete σ3-sequence with filling-up f2 (L = 4)…………….. 127

C-3a. Gantt chart of a complete σ4-sequence with filling-up f1 (L = 4)…………….. 129

C-3b. Gantt chart of a complete σ4-sequence with filling-up f2 (L = 4)…………..... 130

C-4a Gantt chart of a complete σ5-sequence with filling-up f1 (L = 4)……………. 132

C-4b Gantt chart of a complete σ5-sequence with filling-up f2 (L = 4)…………….. 133

D-1 Gantt chart of a complete σ2-sequence with the filling-up phase f1 (L = 5)….. 135

D-2 Gantt chart of a complete σ2-sequence with the filling-up phase f2 (L = 5)….. 137

E-1 Gantt chart of a complete σ6-sequence with filling-up f1 (L = 5)…………….. 139

E-2 Gantt chart of a complete σ6-sequence with filling-up f2 (L = 5)…………….. 142

CHAPTER 1

INTRODUCTION

1.1 Motivation

This research aims to develop algorithms that can minimize the total lot processing time (makespan) of cluster tools used for semiconductor manufacturing. Previous research focuses on finding an optimal sequence of wafer handler moves in a cluster tool that has one process chamber in each stage. In practice, if the number of chambers in a stage is more than one, either a prespecified sequence of wafer handler moves is given in advance or a dispatching rule (push/pull) is applied to find the sequence of wafer handler moves. Then the requisite performance metrics of the cluster tools may be measured and evaluated. No previous work has addresses the problem of finding an optimal sequence of wafer handler moves to improve performance of cluster tools with more than one chamber in a stage.

Cluster tools are highly integrated machines that can perform a sequence of semiconductor manufacturing processes. The performance of cluster tools becomes increasingly important as the semiconductor industry produces larger wafers with smaller device geometry. Some factors that motivate the use of cluster tools, instead of stand-alone tools, include increased yield and throughput, less contamination, and less human intervention.

In this research, the cluster tool is modeled as a manufacturing system with a material handling system. The success of the work will play a vital role in modeling the current wafer fabrication facilities. The model also serves as a decision support tool for the management to understand and be able to make decision on selecting the right tools to achieve the goal equipment productivity, which is one of the keys to increasing the overall factory productivity.

1.2 Research Objective

The objective of this research is to develop scheduling algorithms that can find the optimal sequence of wafer handler moves for a given lot size, wafer handler move time, and chamber processing times. This improves cluster tool performance by reducing the total time needed to process the lot. This can reduce cycle time, reduce tool utilization, and increase tool capacity. Note that in cluster tools with two load locks, the tool can be in steady state for long period of time by having different load locks be alternated ready for processing. This can be achieved by readying one of the load lock to send wafers into the cluster tool, while wafers in the other load lock is currently being processed. A cluster tool may have dual end effector handler to move wafers. This implies dependent rotation; however, only one end effector can extend or retract to load or unload a wafer at any given instant. The research will consider only the single-load-lock and single-wafer-handler tools.

To model the cluster tool, the wafer handler sequencing problem will be formulated in standard scheduling form by providing framework and identifying all constraints that a feasible sequence of wafer handler moves must follow. One algorithm that guarantees to find an optimal sequence of wafer handler moves will be developed. Based on the careful study on the cyclic behavior of the cluster tool in steady state, another algorithm will be developed to find near optimal solutions.

1.3 Models and Insights

In this thesis, two scheduling algorithms are introduced. These procedures give the relationship between handler lot size, move time, and chamber processing times to the lot makespan for a given tool configuration.

1. The first algorithm is a complete branch-and-bound (BB) algorithm. This procedure, when run completely, is guaranteed to find an optimal solution from all possibly feasible sequences of wafer handler moves.

2. The second algorithm is a truncated branch-and-bound (TBB) algorithm. This procedure will search for the best solution from feasible cyclic sequences of wafer handler moves.

For two simple tool configurations, the two- and three-stage cluster tools that have one chamber in each stage, analytical models for the makespan and average cycle time will be derived.

The models allow one to perform the sensitivity analysis of lot makespan with respect to process times, handler move time, and tool configuration. Given the relationship between semiconductor process parameters (such as pressure, temperature, and so on) to the processing times, one can also perform a sensitivity analysis of the tool performance with respect to these process parameters. Insights gained from our results included:

- The cluster tool performance can be greatly improved using Algorithm BB and Algorithm TBB to schedule the wafer handler moves instead of using the current dispatching rules. This improvement was greater when the processing times are approximately the same as or smaller than the move times.
- For the models that implement Algorithm BB, the computing effort increases as the lot size, the number of chambers in each stage, and the number of stages increase. Also conducting longer searches or using more complicated lower bounds did not improve the search performance significantly.
- The computing effort is significantly reduced and becomes less sensitive with respect to the lot size, when Algorithm TBB is used. Although Algorithm TBB may not be able to find optimal sequences of wafer handler moves in some instances, the resulting sequences are still much better than the sequences found by the dispatching rules.

1.4 Outline of Thesis Report

The remainder of the thesis is organized as follows. Chapter 2 gives a brief overview on semiconductor manufacturing, cluster tool structure, current dispatching rules and background literature concerning both cluster tool and robot scheduling.

Chapter 3 formulates the problem, identifies all constraints that a feasible sequence of wafer handler moves must follow. Also, the methodologies to determine the problem size and lot makespan of a given sequence of wafer handler move are presented.

Chapter 4 presents a forward branch-and-bound algorithm that can be implemented to any tool configuration to find an optimal sequence of wafer handler moves. The algorithms for the push and pull dispatching rules are also presented. The attempt to reduce the solution space by introducing three dominance criteria and sophisticated lower bounds are also proposed.

The steady state behavior of the cluster tools processing finite lot sizes is studied in Chapter 5. Analytical models for CT1-1 and CT1-1-1 are derived to determine the cycle time and lot makespan of the 1-unit cyclic sequence of wafer handler moves, given lot size, handler move time, and chamber processing time. This Chapter presents an efficient search algorithm, the truncated branch-and-bound algorithm.

Chapter 6 describes the basic requirements to implement the BB and TBB algorithms to two- and three-stage cluster tool models. The Graphic User Interfaces are also

constructed. Chapter 7 tests the performance of the simulation models using the branch-and-bound and truncated branch-and-bound algorithms by comparing two main performance criteria, the lot makespan and CPU time, to that of the simulation models using the dispatching rules. Chapter 8 summarizes the work and gives suggestions for future research extension from this work.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

2.1 Semiconductor Manufacturing Overview

Semiconductors contain numerous electrical pathways, which connect thousands or even millions of transistors and other electronic components. These transistors store information on the semiconductors, either by holding an electrical charge or by holding little or no charge. Almost all of today's computer chips are built on silicon wafers that are made of highly purified sand.

The manufacturing of 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. The two wafer fabrication steps can be decomposed into nine small steps (International Sematech, 2000):

- Crystallize melted silicon to form ingots that are sliced into wafers.
- Polish one side of the wafer to remove the damage caused by slicing; chips are built on this side.

- Deposit a layer of silicon dioxide on the polished side of the wafer by subjecting the wafers to oxygen or water vapor at high temperature. This layer is called dielectric.
- Use photolithography process to create images of multiple layers of circuit patterns on a chip. First the wafer is coated with a light-sensitive chemical called photo-resist. Then light is shone through a patterned plate to expose the resist.
- Etch away the non-hardened resist and materials below it, then strip off the hardened resist to form three-dimensional patterns on wafers.
- Repeat several photolithography and etch steps to build multiple layers of circuit patterns on a single chip.
- Diffuse or force dopant atoms into certain areas of the chips through chemical exposure and heating or ion implantation to form p and n-type regions.
- Form microscopically thin lines of metal interconnects by first depositing a layer of conducting metal on the entire wafer surface and then removing unwanted metal using photolithography and etch processes. This includes vertical interconnects between layers and horizontal interconnects across each layer of the chip.
- Each chip on the completed wafer is tested for electrical performance. Any failing chips are marked so that they can be discarded when they are separated with wafer saws.

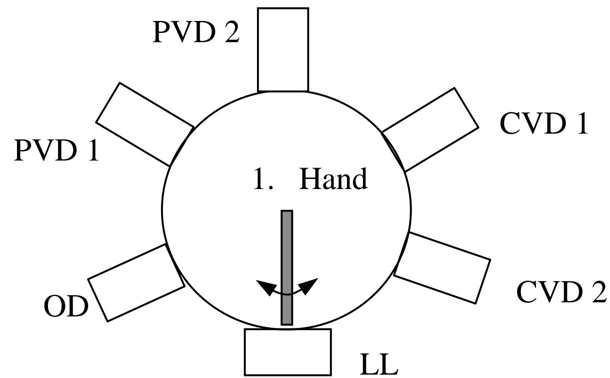
Most operations process each wafer individually. However, identical wafers move together from one process to the next. Each set of wafers is a lot, and a typical lot has 20 wafers. The container used to move and store the wafers in a lot is called a cassette.

2.2 Cluster tools

The term cluster tools describes a specific class of capital equipment used in semiconductor manufacturing. The SEMI E21-96 standard defines a cluster tool as "An integrated, environmentally isolated manufacturing system consisting of process, transport, and cassette modules mechanically linked together." 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, it may undergo additional operation such as pump down or metrology. Each wafer must undergo a series of activities such as orientation/degassing (OD), titanium physical vapor deposition (Ti PVD), or tungsten chemical vapor deposition (W CVD). Such activities are performed in different chambers. After processing wafers, chambers may become temporarily unavailable during automated module cleaning operation.

The wafer handler transports each wafer from one chamber to another. For example, the cluster tool shown in Figure1 has one load lock (LL), which stores a cassette of wafers, and three process stages. The first stage has one OD chamber, the second has two Ti PVD chambers, and the third has two W CVD chambers. Each wafer, starting



Note: OD = Orientation/Degassing chamber

PVD 1, PVD 2 = first and second physical vapor deposition chambers

CVD 1, CVD 2 = first and second chemical vapor deposition chambers

Figure 1: Configuration of a 1-2-2cluster tool

from LL, must visit the OD chamber, one of the two PVD chambers, one of the two CVD chambers, and then return to LL.

Sequential cluster tools integrate a sequence of processes, while other tools have two or more identical chambers that are used in parallel. A sequential 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, such tool may be more reliable, since a failure of a single chamber does not necessarily stop production. The cluster tool in Figure 1 is a combination of sequential and parallel.

Semiconductor manufacturers are increasingly using cluster tools. Annual sales of cluster tools are projected to increase from $11.2 billion in 1997 to $21.9 billion in 2000 (Semiconductor Business News, 1998).

2.3 Current Push and Pull Dispatching Rules

The sequence of wafers going to the cluster tool is not important, since the wafers are identical, and an activity's processing time is the same for every wafer. However, the sequence of wafer handler moves will change the lot makespan. The lot makespan is the total time needed to process a lot of wafers. An activity is either the handler moving

a wafer from one chamber to another or a chamber processing a wafer (for example, chemical vapor deposition in the CVD chamber). A different sequence of wafer handler moves yields a different sequence of activities, and this sequence may have a different lot makespan. Typically, the wafer handler waits until a wafer is ready to move from one chamber to another. Then, the wafer handler moves from its current location to the chamber that has the wafer. Then the wafer handler moves the wafer to the next chamber. After completing the move, the wafer handler will move another wafer if one is ready or will wait where it is until another wafer is ready. If multiple wafers are ready to be moved at the same time, the cluster tool must decide which move the wafer handler will perform. A dispatching rule is often used.

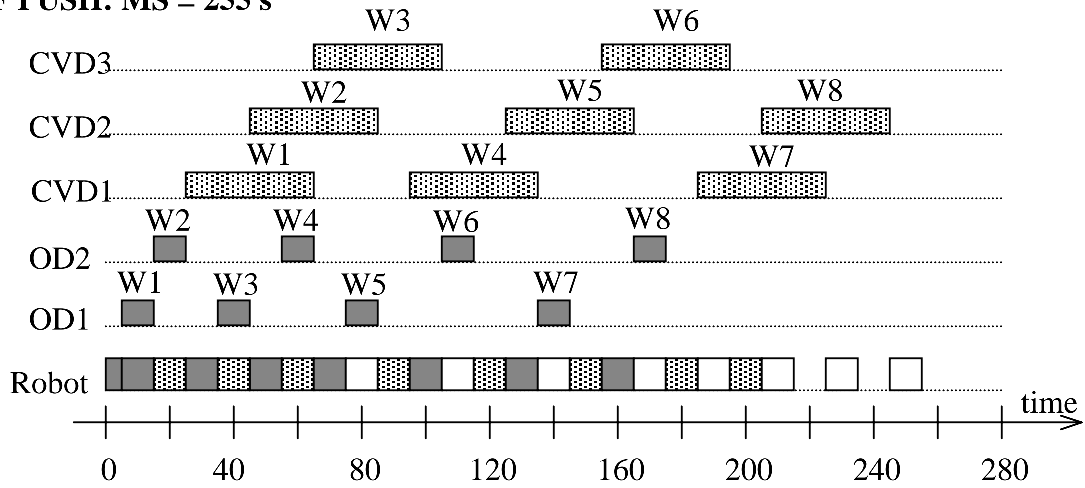
Typical cluster tools use a push dispatching rule or a pull dispatching rule. The dispatching rule selects the next move when two or more different wafers are waiting for the wafer handler. The pull rule gives priority to the wafer that has fewer remaining process steps. The push rule gives priority 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, one of the second stage chambers is holding a processed wafer, and the third stage chambers are empty. The pull rule will give priority to the wafer in the second stage chamber. The push rule will give priority to the next unprocessed wafer in LL that needs to visit the first stage chamber.

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

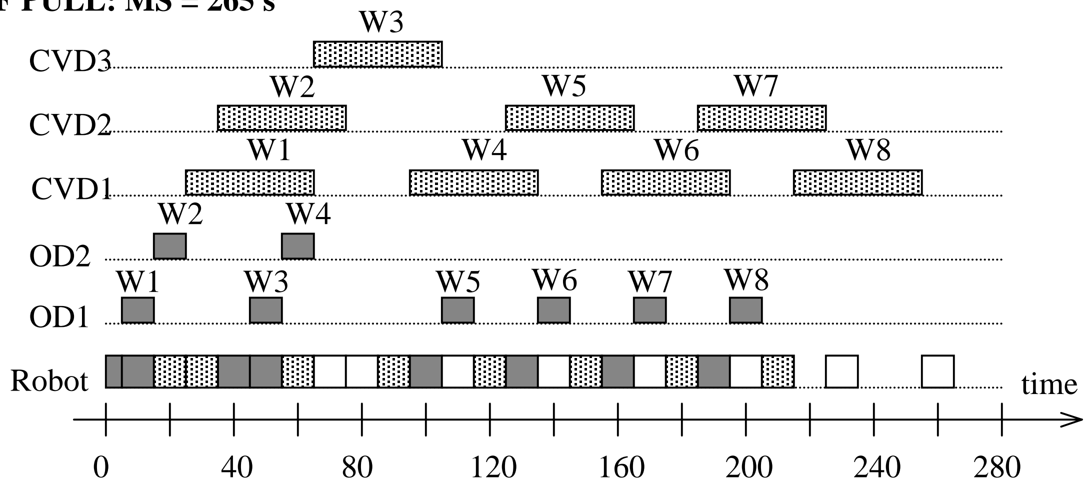
lot makespan for the given lot size, tool configuration, and activity processing times. For instance, consider a two-stage cluster tool that has two first-stage chambers and three second-stage chambers. Each first-stage activity requires 10 seconds, and each second-stage activity requires 40 seconds. A wafer handler move requires 5 seconds. The lot has eight wafers. (We will ignore the time needed to pump down the central chamber after the wafers enter the LL.) Figure 2 presents the Gantt charts of the activities under the push and pull dispatching rules and under an optimal sequence. Notice that the Gantt charts present activities that occur in the same chamber in the same row.

The sequence under the pull rule is inferior in this case, since it abandons the third second-stage chamber and the second first-stage chamber after 65 seconds. The push rule sequence and the optimal sequence repeat different patterns after 65 seconds. The pattern of the optimal sequence, unload second stage → load first stage → load second stage, eliminates one unloaded move for loading first-stage chambers and is better than the pattern of the push sequence, load first stage → unload second stage → load second stage. The optimal sequence, in this case, follows neither dispatching rule throughout the entire lot.

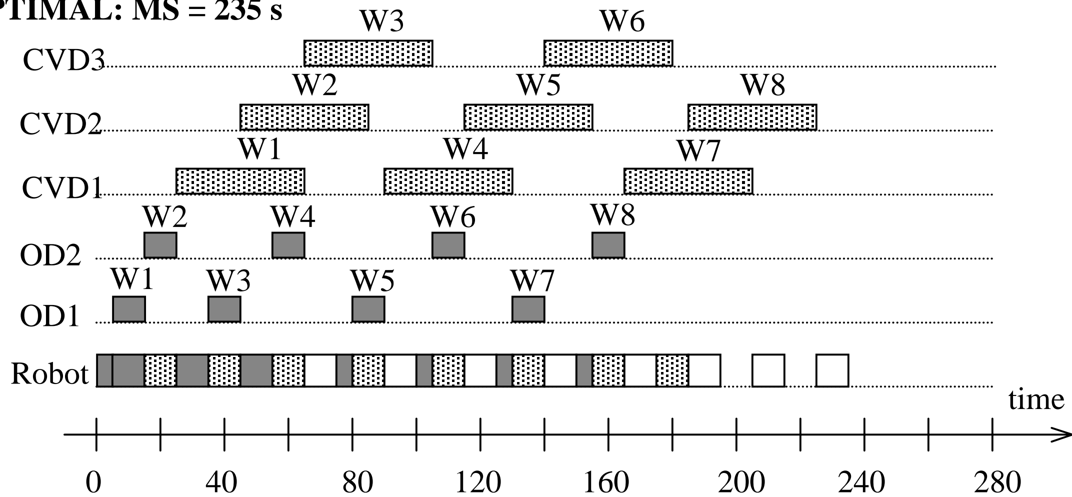
IF PUSH: MS = 255 s



IF PULL: MS = 265 s



OPTIMAL: MS = 235 s



Note: ▇ Load OD and OD activity   □ Unload CVD

▒ Load CVD and CVD activity   Wj = the jth wafer (j = 1,…, L)

2. Figure 2. Sequences constructed following the push and pull dispatching rules and
3. the optimal sequence.

2.4 Related Literature

Usually the purpose of operation research applied to cluster tools is to define metrics for cluster tool performance, then develop models or methods for tracking cluster tool performance. Such performance metrics include throughput, which is the average number of wafers that the tool processes per unit time, and total lot processing time (or lot makespan), which is the elapsed time from when a new lot of wafer is loaded into a tool until the completed lot is unloaded.

2.4.1 Cluster tool scheduling

Kise *et al.* (1991) discuss new flow shop scheduling problems related to automated manufacturing systems in which n jobs are processed on two machines. Jobs are transported by a robot between an input/output station and a machine, or between two machines. They provide an algorithm that find optimal sequences of n jobs with the objective of minimizing the makespan under a specified move cycle in $O(n^3)$ time.

Wood (1996) derives formulas that relate the total lot processing time to the number of wafers in the lot for ideal sequential and parallel tools. Note that in Wood's paper, cycle time is defined the same as what we defined total lot processing time. The models use two measurable parameters that aggregate tool operations: the incremental cycle time, which is the average increase in total lot processing time resulting from a lot size increment of one wafer, and the fixed cycle time, which is the independent-of-lot-size portion of total lot processing time. Wood suggests using the empirically determined

increment and fixed cycle times to predict the improvement in a cluster tool's maximum throughput, or using analytical models of increment and fixed cycle times to predict the impact of tool configuration on cycle time and throughput for a hypothetical integrated tool.

Considering the transitions at the beginning and the end of the lot, Perkinson *et al.* (1996) derive a model that relates the total processing time to the number of wafers for cluster tools that have single-wafer process chambers organized around a wafer transport mechanism. Using timing diagrams, they derive the so-called fundamental period, then determine throughput as inverse of the fundamental period. Perkinson *et al.* also suggest three ways to improve throughput. The look-ahead algorithms and multi-speed transporters tempt to increase the net wafer handler speed by modifying the action of the handler when it is moving a wafer; the incorporate dual load locks tempts to minimize the lengths of the beginning and ending phases of processing a lot. They, as well as Wood (1996), 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.

Following Perkinson's work, Venkatesh *et al.* (1997) analyze the steady state throughput of a sequential cluster tool with a dual-blade robot. Their analysis shows that, under the process-bound condition, a cluster tool with single-blade robot would need to double the speed of the robot to achieve the similar throughput as the dual-blade

cluster tool. In the transport-bound condition, the throughput is the same for both dual-blade and single-blade cluster tools.

Srinivasan (1998) presents more detailed Petri net models for sequential and parallel tools and uses these to determine the steady state throughput of the tool. His models can be applied to a tool with either single or dual-blade wafer handler, and with either anticipatory or non-anticipatory handler moves. However, the sequence of wafer handler moves must be dictated in advance.

Herrmann *et al.* (1999) 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 by sequencing the wafer handler moves.

2.4.2 Robot scheduling

Jeng *et al.* (1993) 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. This branch-and-bound algorithm can find solutions for small and medium sized problems (refer to values of n

and m) within reasonable times. For large sized problems, they proposed a heuristic for finding a near optimal solution.

A state space approach is used in Sethi *et al.* (1992) to address the problem of sequencing parts and robot moves in a robotic cell, which is defined as a flow-line manufacturing system. Their objective is to maximize the long-run average throughput of the system subject to the constraint that the parts are to be produced in proportion of their demand. For the cell that has M machines producing a single part type, they show in a constructive manner that the number of one-part cycles is exactly M!.

Extending the results from Sethi's paper, Hall *et al.* (1997) provide an algorithm that simultaneously finds sequences of parts and robot moves to minimize the steady state cycle time, for multiple part-type problems in a two-machine cell. 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 (1997), 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. Both Crama and Hall address that many interesting related problems are still open, such as the conjecture that one-unit cycles are optimal among all possible robot move sequences.

Kamoun *et al.* (1999), revisiting the problem dictated in Hall (1997), develop a heuristic procedure that aims to maximize the long-run average throughput for part sequencing problem under different robot move cycles in three-machine cell. They also provide a methodology for extending this heuristic to four-machine cell and even larger cell.

2.5 Summary

Semiconductor manufacturing is a complicated process involving many steps, which require processing on highly automated and expensive equipment. This attracts research into the performance of the cluster tools with respect to throughput and lot makespan. Both analytical and simulation models for some simple tool configurations have been developed. As the tool configuration becomes more complex, understanding and improving cluster tool performance become more important.

However, due to the complexity of the problem, there is currently no available model for a general tool configuration that can obtain an optimal performance of a cluster tool, given any set of processing parameters such as in-chamber process times and wafer handler move time.

CHAPTER 3

PROBLEM FORMULATION

3.1 Notation

The following notation is used in the thesis:

L = lot size.

S = number of stages in the cluster tool.

i = stage index, $i = 1,\ldots,S$.

j = wafer index, $j = 1,\ldots, L$.

S_i = stage i.

$R_{0,j}$ = move wafer j from LL to a chamber in S_1.

$R_{i,j}$ = move wafer j from a chamber in S_i to a chamber in S_{i+1} ($i = 1, ..., S-1$).

$R_{S,j}$ = move wafer j from a chamber in S_S to LL.

m_i = number of chambers in stage S_i.

m_1-m_2-...-m_S = tool configuration, denoting that the tool has S stages and each stage S_i has m_i chambers.

h = number of wafer handlers.

p_r = time that the wafer handler needs to travel from one chamber to another, from LL to a chamber, or from a chamber to LL.

p_i = time required for a wafer to be processed in a chamber in S_i.

T = the total number of wafer moves. $T = L(S+1)$.

3.2 Problem statement

3.2.1 Assumptions

In this thesis, we make the following assumptions to simplify the problem:

- In a stage, the processing time includes chamber's overhead time and is a deterministic constant.
- The move time includes times for picking up, moving, and loading a wafer.
- The move time from a chamber to another, or from load lock to a chamber, or from a chamber to load lock is a deterministic constant.
- Tool overhead time includes initial pumpdown and vent times of the lot.
- Breakdowns or failures of the components are rare and not considered.
- We will only consider the single load lock, single wafer handler cluster tools.

3.2.2 Objective

This research focuses on an S stage, single load lock cluster tool. The number of chambers in any stage can be any positive integer. Let m_i be the number of chambers in stage S_i (i = 1,..., S). Let $M = m_1 + ... + m_S$ be the total number of chambers. The chambers are numbered 1 to M.

Each stage has a wafer processing time p_i, and the wafer handler requires p_r time units to move from one chamber to another. The lot has L wafers. The sequence of wafers leaving LL is not important, since the wafers are identical, and the processing time at a stage is the same for every wafer. We will number the wafers in the order they leave LL. However, the sequence of wafer handler moves will change the lot makespan.

The research studies the problem of minimizing the lot makespan C_{max}, the total time required to process a lot of wafers. Reducing the total time needed to process the lot can reduce cycle time, reduce tool utilization, and increase tool capacity. Moreover, C_{max} is a necessary component for calculating overall equipment effectiveness (OEE) and cost-of-ownership (COO), which are usually used to evaluate cluster tool performance (Murphy, 1996; Dance, 1998).

The problem is to find a feasible sequence of wafer handler moves that minimizes the total time required to process all the wafers and return them to LL.

3.2.3 Constraints

A chamber at stage S_i begins processing wafer j as soon as move $R_{i-1,j}$ ends (i = 1, ... S). Move $R_{i,j}$ can begin after this chamber finishes processing wafer j and after the wafer handler completes the 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 $R_{S,j}$). In this work we assume that the wafer handler cannot move to the chamber before processing ends. The lot makespan is the time that the last move ends.

A feasible sequence must satisfy the following constraints.

- Constraints caused by nature of semiconductor manufacturing

  a) Precedence constraints: All wafers must follow the fixed sequence of processing steps. $R_{i,j}$ must precede $R_{i+1,j}$ for i = 0, ..., S-1.

b) No Preemption: An activity in a chamber cannot be interrupted until the wafer is finished its processing in that chamber. 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.

- Constraints caused by the tool architecture

  c) Feasible move: 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.

  d) Tool configuration: Since a chamber can process only one wafer at a time, the total number of wafers in a stage must be smaller than or equal to the number of chambers in that stage.

  e) Wafer handling: Since the cluster tool has a single wafer handler, then, at any time, there is at most one move occurring.

  f) Non-anticipation: The wafer handler cannot anticipate the next move. That is, if the next move is $R_{i,j}$ and the wafer handler is idle, the wafer handler must wait where it is until the busy chamber at S_i finishes processing wafer j. Only then can the wafer handler move from its current location to the (now full) chamber to unload the wafer and move it.

- Constraints caused by lot size

  g) Lot size L: the wafer handler cannot move a wafer from LL to a chamber in S_1 if there are no unprocessed wafers in LL.

3.2.4 Permutation versus non-permutation

Permutation implies that the order in which the wafers enter the first stage is maintained throughout the system. In the cluster tool environment, permutation means that wafer j is always loaded to or unloaded from a stage before wafer j+1, i.e. $R_{i, j}$ precedes $R_{i, j+1}$. Obviously, this constraint must be satisfied if there is only one chamber at each stage. However, it may be violated if a tool has multiple chamber stages. For example, consider a cluster tool that has two chambers, A and B, in the last stage S_S. Assume that at time t, wafer j and j+1 are finished their processes in chamber A and B respectively. Then move $R_{S, j}$ is prior to move $R_{S, j+1}$ if the permutation constraint is active. But, move $R_{S, j+1}$ may be prior to move $R_{S, j}$ since blocking at chamber A is allowed.

The introduction of non-permutation constraint complicates the problem since it increases number of feasible sequences. In Chapter 4, we will prove that, for some cases, violating the permutation constraint will not improve performance of the cluster tool.

3.2.5 Problem notation

A scheduling problem is described by a triplet $\alpha \mid \beta \mid \gamma$. The α field describes the machine environment and contains a single entry. We let $\alpha = CTm_1\text{-}m_2\text{-}...\text{-}m_S$.

The β field provides details of special processing characteristics and constraints and may contain no entries, a single entry, or multiple entries. All seven constraints depicted in Section 3.2.2 are always active; hence they do not need to be introduced in

the β field except for some special cases presented below. Notice that constraint d has already been introduced in the α field and constraint g is an input parameter. The following entries may appear in the β field.

- "permu.", if "permu." appears in the β field, all sequences must follow the permutation condition. If "permu." is not in the β field, the permutation condition may be violated.
- h, if there are more than one wafer handler in the tool, then the number of wafer handler must be introduced. Otherwise, h equals to 1.
- anticipation, if anticipation appears in the β field, the wafer handler must anticipates the next move. Otherwise, all considering sequences are non-anticipatory.

The γ field contains the objective to be minimized and usually contains a single entry. For our problem, $\gamma = C_{max}$.

Thus, for example, CT1-2 | | C_{max} denotes the problem of finding the sequence of wafer handler moves that minimizes the total lot makespan of a two-stage, single wafer handler cluster tool. The first stage has one chamber and the second stage has two chambers. The permutation condition may be violated.

3.3 Number of feasible sequences

Consider a general problem CTm_1-…-m_S | | C_{max}. According to the precedence constraint, a wafer must follow a fixed order of processing steps, i.e., the wafer must visit every single stage. This implies that each wafer requires (S+1) wafer handler

moves. Hence, there are a total of L(S+1) times that the scheduler must select a feasible wafer handler move. Except for the move $R_{S,j}$, follow each wafer handler move $R_{i,j}$ there is a process activity in a chamber, i.e. there are L process activities occurring in each stage's chambers. As a result, there are a total of L(2S+1) activities that need to be scheduled.

Excluding the activities in process chambers, we can construct a directed graph representing all feasible sequences of wafer handler moves. The graph will have a form of an outtree, a precedence graph wherein, the number of immediate predecessors of a node is either zero or one. The number of levels of the directed graph equals to the number of times the scheduler must make selection L(S+1). The state of the tool changes according to the move. For instance, after move $R_{0,\,j}$, the number of unprocessed wafers in LL decreases by one, and one free chamber in the first stage becomes busy. Hence, the number of nodes at level τ depends on the type and number of nodes at previous level τ-1. Starting with an empty tool and the wafer handler is at LL, the move $R_{0,1}$ is located at level 1. Denote number of nodes at level τ as $N(\tau)$. We can find the total number of nodes in a level of the outtree graph using the recursive formulas.

$$\begin{cases} N(1) = 1 \\ N(\tau) = \sum_{k=1}^{N(\tau-1)} z_k \,,\ \forall\, \tau = 2,\ldots,L(S+1). \end{cases} \tag{1}$$

Where z_k is the total number of feasible choices resulting from the state of the cluster tool associated with the node k at level τ-1. Using the feasible move constraint, we can

easily determine z_k. Define $l_{i,k}$ as the number of wafers in stage S_i ($i = 1,\ldots, S$) and $l_{0,k}$ as the number of unprocessed wafers in LL after the move, representing by node k, finishes. We can determine the number of choices in each move resulting from tool state of node k as follow.

◊ $R_{0,j}$, move a wafer j from LL to the first stage S_1: the number of choices for this move is strictly depended on number of unprocessed wafers in LL and status of S_1. Since every unprocessed wafer is the same (we actually index a wafer when loading it to S_1), the number of departure will be either one, if there is at least one unprocessed wafer in LL, or zero. The number of destination is the number of free chamber in the first stage (m_1-$l_{1,k}$). We can mathematically write the number of choices for the move R_0 as

$$\zeta_{0,k} = \begin{cases} m_1 - l_{1,k} & \text{if } 1 \le l_{o,k} \le L \\ 0 & \text{if } l_{0,k} = 0 \end{cases} \tag{2}$$

◊ $R_{i,j}$, for $i = 1,\ldots, S-1$, load a stage other than S_1: the number of choices for these moves depends on status of the departure stage S_{i-1} and destination stage S_i. The number of departure is the number of wafers in the departure stage $l_{i-1,k}$ and the number of destination is the number of free chamber in the destination stage m_i-$l_{i,k}$. Number of choices for loading these stage as

$$\zeta_{i,k} = l_{i-1,k}(m_i - l_{i,k}) \qquad \text{for } i = 1,\ldots, S-1 \tag{3}$$

◊ $R_{S,j}$, unload the last stage: since the LL can store the whole lot of wafer, the number of choice for this move only depends on status of the last stage. In fact, the number of choice equals to the number departure, i.e. number of wafers in the last stage $l_{S,k}$.

Thus, we can write

$$z_k = \sum_{i=0}^{S-1} \zeta_{i,k} + l_{S,k} \tag{4}$$

Using equations (1), (2), (3), and (4), we can determine the number of nodes in each level τ. Starting at the single node in level 1 and going to one of the nodes in the last level, the collection of all nodes in this path form a feasible sequence of wafer handler moves. The number of feasible sequences equals to the number of nodes in the last level L(S+1). Since the sequences under the push and pull dispatching rules are feasible, they are including in the outtree graph.

For example, consider problem CT1-1-2 | | C_{max}. Let L equals 3. Denote S1 and S2 as the names of chamber in the first and second stages respectively. Denote S31 and S32 as the names of the first and second chambers in the third stage. The number of times that the scheduler must make selection is L(S+1) = 3(3+1) = 12. Table 1 presents details on the numbers of wafers in each stage at each level τ associated with the selected move. Figure 3 presents a branch of the directed outtree that forms by collecting selected feasible moves in Table 1. In Figure 3, each node is represented by a circle. Departures and destinations of the moves are also given next to each node. Wafer index at each node is also given inside the parentheses. For example, at level 3, S2-S31(1) means moving wafer 1 from stage 2 to the first chamber in stage 3. Noticing that the number of dashed arrows outgoing the nodes is not exactly determined in Figure 3. A complete exploration of all possible nodes in Appendix A shows that there are 552 feasible sequences of wafer handler moves for a 1-1-2 cluster tool, processing 3 wafers per lot.

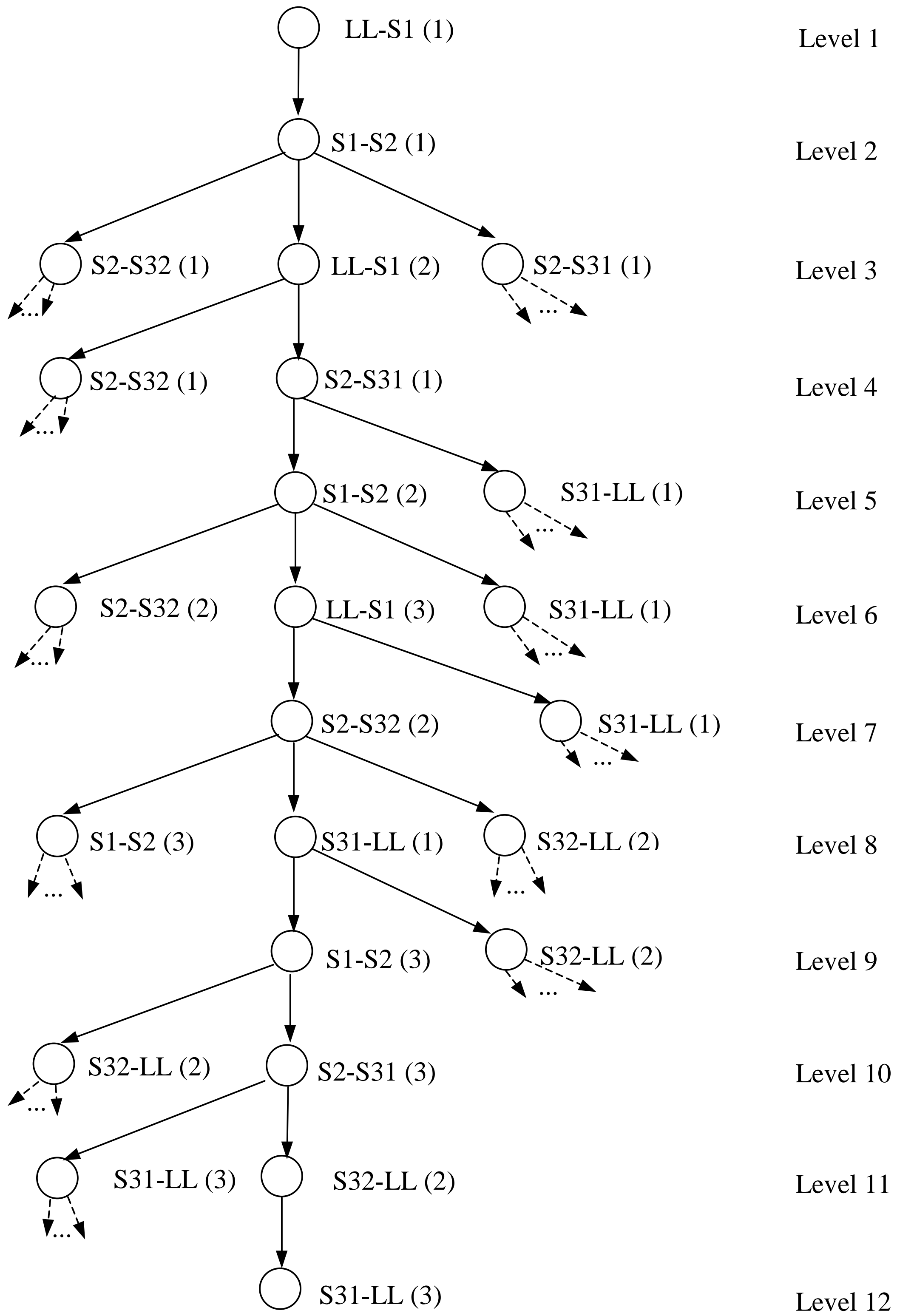


Figure 3. Partial outtree of the 1-1-2 cluster tool.

Table 1. Numbers of wafers in each stage at each level τ associated with selected moves.
$m_1 = 1, m_2 = 1, m_3 = 2$.

τ	selected move	l_0	l_1	l_2	l_3	ζ_0	ζ_1	ζ_2	$z = \zeta_0 + \zeta_1 + \zeta_2 + l_3$	Next feasible moves
0		3	0	0	0	1	0	0	1	LL-S1
1	LL-S1	2	1	0	0	0	1	0	1	S1-S2
2	S1-S2	2	0	1	0	1	0	2	3	LL-S1, S2-S31, S2-S32
3	LL-S1	1	1	1	0	0	0	2	2	S2-S31, S2-S32
4	S2-S31	1	1	0	1	0	1	0	2	S1-S2, S31-LL
5	S1-S2	1	0	1	1	1	0	1	3	LL-S1, S2-S32, S31-LL
6	LL-S1	0	1	1	1	0	0	1	2	S2-S32, S31-LL
7	S2-S32	0	1	0	2	0	1	0	3	S1-S2, S31-LL, S32-LL
8	S31-LL	0	1	0	1	0	1	0	2	S1-S2, S32-LL
9	S1-S2	0	0	1	1	0	0	1	2	S2-S31, S32-LL
10	S2-S31	0	0	0	2	0	0	0	2	S31-LL, S32-LL
11	S32-LL	0	0	0	1	0	0	0	1	S31-LL
12	S31-LL	0	0	0	0	0	0	0		

3.4 Determining the lot makespan

The first task to solve this scheduling problem is to find the lot makespan for a given cluster tool under a sequence of wafer handler moves. For a very simple tool configuration and small lot size, one can easily construct a Gantt chart of all activities, then determine the makespan as in Figure 2.

The lot makespan under a fixed sequence of wafer handler moves can also be computed by determining the critical path in a network model, a collection of nodes and directed arcs (Herrmann *et al.* ,1999 and Chandrasekaran, 1999). Associated with an activity, either handler move activity or process activity in a chamber, there is a node with a weight that is equal to the processing time of the activity. The directed arcs, connecting the nodes, describe the precedence constraint between activities. A path will start at the first node, representing the move to load the first wafer from load lock to a chamber in the first stage, and end at the last node, representing the move to unload the last wafer from the last stage. The total weight of the maximum weight path, or the length of the longest path, corresponds to the makespan under the fixed sequence of wafer handler moves.

When the tool configuration is more complex, i.e. more than two processing stages or more than two chambers in a stage, and the lot size is a large number, constructing the Gantt chart and network model for such tools are time-consuming tasks. In such cases, discrete-event simulation models should be developed. One example is the Cluster Tool Performance Simulator (CTPS) software that Lee Schruben (1999) developed at Cornell University. The input of CTPS includes the tool configuration, the lot size, the sequence of processes that each wafer should undergo, the duration time of each operation (both wafer handler move and process activities), and a rule (push or pull) for moving the wafer within the tool. Limitations of the CTPS include:

- The tool configuration is not flexible; in fact, every stage cannot have more than two chambers.
- Either push or pull dispatching rule must be used to generate the sequences.

We can easily develop simulation model that uses the dispatching rule (push or pull), to generate a sequence of activities then determine its makespan for any given tool configuration. Also, we can easily develop simulation model that determines the makespan of a cluster tool under a pre-specified sequence of wafer handler moves.

3.5 Summary

This chapter has identified the main objective of the problem. That is finding the optimal sequence of wafer handler moves to minimize the lot makespan, given lot size, handler move time, and chamber processing time, for single wafer hander and single load lock cluster tool. Using the introduced framework, the problem can be formulated as a standard scheduling problem. This chapter describes seven constraints for constructing feasible sequences of handler moves and an additional permutation constraint. It presented a methodology to construct an outtree graph for a given tool configuration based on the constraints. Finally, it described methodologies to determine the lot makespan from a given sequence of handler moves and to generate a schedule and determine the lot makespan using current push and pull dispatching rules.

CHAPTER 4

BRANCH-AND-BOUND ALGORITHM

Currently, there is no available algorithm that generates an optimal sequence of wafer handler moves for a multi-chamber, multi-stage cluster tool. In this chapter, we will develop a forward branch-and-bound algorithm for finding a sequence that minimizes the lot makespan.

4.1 The forward branch-and-bound algorithm

A branch-and-bound procedure is an enumeration scheme that can discard a partial solution by showing that the objective value obtained with the partial solution is not optimal. This involves computing a lower bound on the value of any solution that uses the partial solution and an upper bound on the value of the optimal solution. If the partial solution's lower bound is greater than optimal value's upper bound, then the partial solution cannot lead to an optimal solution, so it can be discarded.

The algorithm first generates two feasible sequences using the push and pull dispatching rules and uses the better sequence's lot makespan as the upper bound on the optimal lot makespan. The better sequence is also used as an optimal sequence candidate. For a partial solution, we use the completion time of the last scheduled activity as its lower bound.

Given $m_1,...,m_S$, p_r, $p_1,...,p_S$, and L, the algorithm proceeds as follows.

Algorithm BB:

Step 0:

Use the push and pull dispatching rules to generate two feasible sequences (see Algorithm P). Take the better sequence as an optimal sequence candidate, and use its makespan as an upper bound on the optimal value.

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 $t = 0$, $n = L$, and $t_k = 0$ for all chambers k.

Step 2:

Based on the tool state, identify all feasible moves.

- $R_{0,j}$ is feasible if $j = L+1-n$ 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 calculate LB as follows: add the feasible move to the current sequence, compute the move's completion time, and update the tool state.

- 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.
- Otherwise, 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 $C = \max \{t, t_k\}+ p_r$. Chamber k is now free. Go to Step 3d.

Step 3d:

Let t = C. If LB is greater than or equal to the upper bound on the optimal value, then discard this new sequence. If the new sequence includes all L(S+1) moves, the lot makespan equals C. If the lot makespan is less than the upper bound, save the new sequence as the current best sequence and set the upper bound equal to the lot makespan.

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.

For example, consider the problem CT1-1 | | C_{max}. The following information is given: $p_r = 5$, $p_1 = 10$, $p_2 = 40$, and L = 3. The push sequence is $R_{0,1}$, $R_{1,1}$, $R_{0,2}$, $R_{2,1}$, $R_{1,2}$, $R_{0,3}$, $R_{2,2}$, $R_{1,3}$, $R_{2,3}$. The lot makespan is 185. Figure 4 shows a graph of all feasible sequences. Any path from the top node (the first feasible move) to a lower node corresponds to a feasible partial sequence. Branch A corresponds to the push sequence. Because the upper bound is 185, this is the only branch fully explored. The other branches are discarded when the partial sequences have eight moves because the lower bounds are greater than or equal to 185 ($C \geq 145$. $LB = C + p_2 \geq 185$).

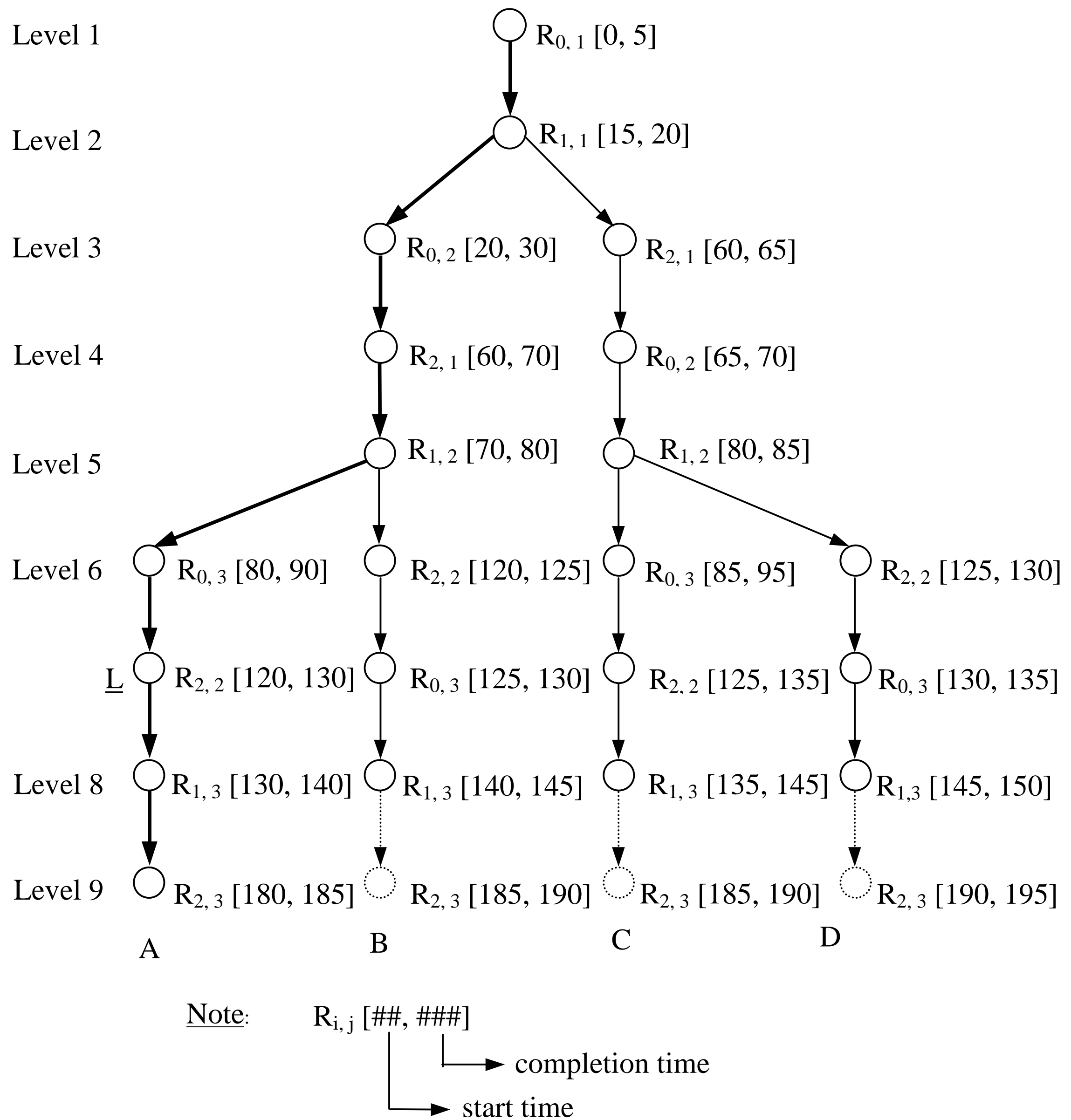


Figure 4. Outtree graph of the CT1-1 (3 wafers per lot).

We now present an algorithm that uses the push (or pull) dispatching rule to construct a feasible schedule.

Algorithm P:

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 $n = L$.

Step 2:

Based on the tool state, identify any feasible moves that could begin at time t. Move $R_{0,j}$ can begin at time t if $j = L+1-n$ and there is a free chamber in S_1. Move $R_{i,j}$ $(0 < i < S)$ can begin at time t if wafer j is at chamber k in S_i, $t_k \leq t$, and there is a free chamber in S_{i+1}. Move $R_{S,j}$ can begin at time t if wafer j is at chamber k in S_S and $t_k \leq t$.

- If there is exactly one feasible move, then perform that move. 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. 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. Go to Step 4.
- Otherwise, go to Step 3.

Step 3:

Among the busy chambers k in S_1, ..., S_S, identify the minimum value of t_k. Let $t = t_k$. 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 starts processing wafer j, and $t_q = C + p_1$.
- If the selected move was $R_{i,j}$, $0 < i < S$, then 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. Let q be the lowest-numbered free chamber in S_{i+1}. The wafer handler is now at chamber q, which now starts processing wafer j, and $t_q = C + p_{i+1}$.
- Otherwise, 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. The wafer handler is now at LL.

Step 5:

Let $t = C$. If $n = 0$ and all chambers are free, then stop. The lot makespan equals C. Otherwise, go to Step 2.

Note that Algorithm P generates only non-delay schedules.

4.2 Dominance criteria

Since all chambers in a stage are identical, move $R_{i,j}$ loads the lowest-numbered free chamber in S_{i+1} ($i = 0, ..., S-1$). Doing this will eliminate the number of search branches, thus reduce the solution space when the number of chambers in a stage is greater than one.

The permutation constraint states that 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 constraint. Otherwise, there may exist feasible sequences that violate this constraint. We will show however, that, for some tool configurations, there exists an optimal sequence that does satisfy this constraint. Thus, we can limit the branch-and-bound search to those sequences. This will improve our search performance.

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

Proof. Consider an optimal feasible sequence Q that violates the permutation constraint. 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 $m_i = 1$, $R_{i-1,j+1}$ is infeasible until $R_{i,j}$ empties the chamber in S_i. $R_{i,j}$ must precede $R_{i-1,j+1}$ and $R_{i,j+1}$, so Q is infeasible. Thus, S_i must have multiple chambers ($m_i > 1$) and $p_i \geq p_r$.

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:

B1: Q1 empty or Q1 not empty and doesn't end with $R_{i-2,j+1}$.

B2: Q1 ends with $R_{i-2,j+1}$.

Consider case B1. 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.

- 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 Case B2. 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.

- 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.

By limiting the branch-and-bound algorithm to permutation sequences, we limit the number of sequences that need to be considered. We can abbreviate the search more with the following dominance property.

Since wafer processing can happen concurrently in different chambers, a move can be discarded if there exists another move at the same level that can be proceeded before the first move without delaying the first move's completion time and changing its processing time. For instance, consider a scenario shown in Figure 5.

Assume that there are three moves (a, b, and c) available at time t, and the wafer handler is at chamber B. Both move a and move c take $2p_r$ while move b takes p_r. Both moves b and c dominate move a, since they can finish before wafer W_A is available and move a will always take $2p_r$. Even though move c can finish before wafer W_B is available, it does not dominate move b because the time required for move b will change to $2p_r$ if move c precedes move b.

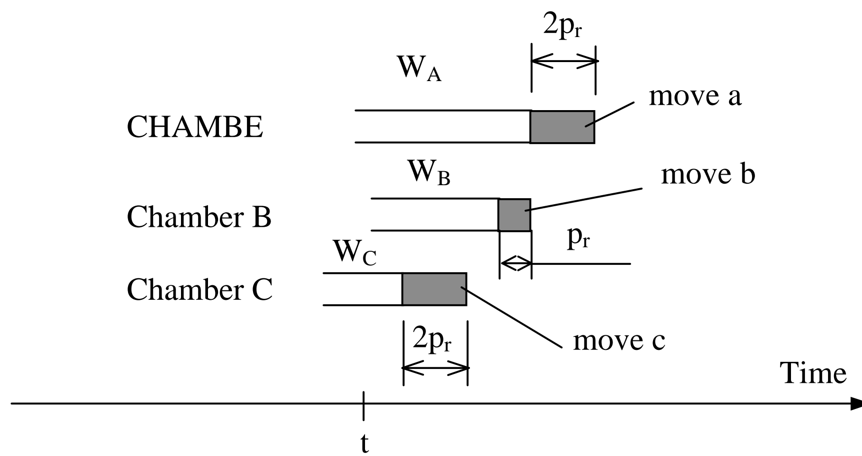


Figure 5. A scenario demonstrating the third dominance criterion.

Let Q1 be a feasible partial sequence. $R_{p,q}$ dominates $R_{i,j}$ if, for any complete, feasible permutation sequence Q that starts with Q1 and $R_{i,j}$, there is a complete, feasible permutation sequence Q' that starts with Q1 and $R_{p,q}$ and the makespan of Q' is not greater than the makespan of Q. Thus, the branch-and-bound algorithm will not find a better sequence by searching the sequences that start with Q1 and $R_{i,j}$.

Theorem 2. 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 $t_b + 2p_r \leq t_a$. 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.

Note that using this criterion limits the branch-and-bound algorithm to the set of active schedules. In summary, the solution space of the problem can be reduced by applying three dominance criteria.

1) The wafer handler should always load the lowest-numbered free chamber in a stage.
2) The permutation constraint, 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) The active constraint, 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 $R_{i,j}$.

4.3 Number of feasible sequence under the first and second dominance criteria

Applying dominance criteria 1 and 2 and defining the function $\delta(x)$ as

$$\delta(x) = \begin{cases} 1 & \text{if } x \geq 1 \\ 0 & \text{if } x = 0, \end{cases} \tag{5}$$

we can rewrite the number of choices in each wafer handler move for the permutation problem as follows.

- R_i, load S_i: number of choices

$$\zeta_{i,k} = \delta(l_{i-1,k})\delta(m_i - l_{i,k}), \forall i = 1,\ldots,S \tag{6}$$

- R_S, unload S_S, unload the last stage: number of choices will be either one, if there is at least a wafer in the last stage, or zero, which mathematically expresses as $\delta(l_{S,k})$.

Thus, we can write the total number of feasible choices resulting from the state of the cluster tool associated with the node k at level τ-1.

$$z_k = \sum_{i=1}^{S} \delta(l_{i-1,k})\delta(m_i - l_{i,k}) + \delta(l_{S,k}). \tag{7}$$

For instance, revisit the 1-1-2 cluster tool, processing 3 wafers per lot. If applying the first and second criteria, the number of feasible sequences reduces to 69 from 552 sequences. (See Appendix A for details on determining the number of feasible sequences).

The third criterion involves processing times of the activities, hence, it may be active for some instants and inactive for others. The dominance criteria can be applied to the algorithm BB while finding all feasible moves in Step 2.

4.4 Better lower bounds

In section 4.1, the completion time of the last scheduled activity is used as lower bound LB for a partial solution. Let σ be the partial sequence. Denote $LB(\sigma)$ as lower bound of σ. Let C_a be the completion time of the last activity in σ. Then,

$$LB1(\sigma) = C_a \tag{8}$$

Although this lower bound is very easy to determine, it will not detect a bad sequence (one that has partial makespan greater than or equal to the upper bound) until the search is almost at the end of a searching branch. Three better lower bounds proposed below can be applied to improve the search performance of the Algorithm BB.

1) Let C be the completion time of the last move in σ. Let n_σ be the total number of moves that have been performed in σ. The remaining move is $L(S-1) - n_\sigma$. Neglect all activities in the process chambers, and assume that all remaining moves require p_r. Then,

$$LB2(\sigma) = C + (L(S-1) - n_\sigma)p_r \quad (9)$$

Thus, LB2(σ) will likely help the algorithm discard poor solutions when move time p_r is long compared to processing time p_i.

2) From σ, determine number of wafers has been started W1(σ), which equals to the number of moves $R_{0,j}$ in σ. Then to complete a branch starting with σ, there will be (L-W1(σ)) move(s) needed to load the unprocessed wafer(s) to S_1. Assume that all moves require pr. Total time needed to completed these moves is $(L-W1(\sigma))p_r$. Neglect all activities in the process chambers that happen between $R_{0, L-W1(s)}$ and $R_{0,L}$. Hence, if W1(σ) < L,

$$LB3(\sigma) = C + (L - W1(\sigma))p_r + \sum_{i=1}^{S}(p_i + p_r) \quad (10)$$

3) Let Wi(σ) be numbers of wafer(s) that have started processing in S_i (i = 1, …, S). Notice that Wi(σ) equals to the number of moves $R_{i-1,j}$ in σ. Each chamber in Si will be loaded $(L - Wi(\sigma))/m_i$ times. Assume that all moves require p_r. Then,

$$LB4_i(\sigma) \geq C + \frac{(L - W_i(\sigma))}{m_i}(p_i + p_r), \ \forall i = 1,...,S. \quad (11)$$

Thus, LB3(σ) and $LB4_i(\sigma)$ will likely help the algorithm discards poor sequences when move time p_r is short compared to processing times p_i.

For a given σ, we can check whether σ is a bad sequence by first determining the maximum value of all quantities in the right hand side in the Equations 9, 10, 11, and

12. Then, if this value is greater than or equal to the upper bound, σ is a bad sequence, hence, σ can be discarded.

4.5 Summary

This chapter presents a forward branch-and-bound algorithm that can be implemented to any tool configuration to find an optimal sequence of wafer handler moves with the objective of minimizing the lot makespan. The better sequence between the push and pull sequence is initially used as candidate of the solution and its makespan is used as an upper bound, which is updated whenever a better solution is found. For a partial solution, we use the completion time of the last scheduled activity as its lower bound. The algorithm searches all possibly feasible sequences of wafer handler moves for single wafer handler and single load lock cluster tools; hence it guarantees the optimality of the solution. The algorithms for the push and pull dispatching rules are also presented.

Three dominance criteria are introduced to greatly reduce the solution space to improve the search performance. A numerical example shows that the solution space can be greatly reduced by using the first and second criteria. We have proved that, in cases when the handler move time is not greater than the chamber processing times, there exists an optimal solution that satisfy three dominance criteria. Sophisticated lower bounds are proposed and will be numerically evaluated in Chapter 7.

CHAPTER 5

THE CYCLIC BEHAVIOR OF THE CLUSTER TOOLS AND TRUNCATED BRANCH-AND-BOUND ALGORITHM

5.1 Performance of a cluster tool processing a finite lot size

Normally, if the lot size L is large enough, the performance of a cluster tool might be separated into three phases: filling-up, steady state (or cyclic), and completion. The tool is 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. Then when there are no more wafers to start, the tool enters the completion phase. Processing ends when the last wafer is unloaded from the last stage. Starting with an empty tool and the wafer handler is at load lock ready to pick up a wafer, after some number of events (wafer handler moves) the tool will converge to a steady state. Just after the last unprocessed wafer is loaded to a chamber of the first stage, the tool orderly begins to flush wafers out until the last wafer is unloaded from the last stage.

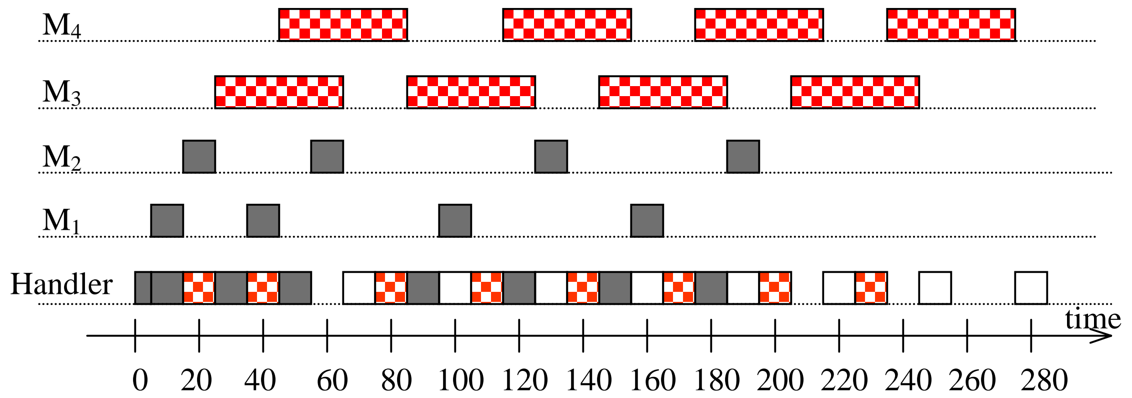
Let us define a λ-unit cycle as a subsequence during which each stage is loaded and unloaded λ times (λ wafers are completed). The resulting sequence formed by repeating a cycle in the steady state and completion phase is called the λ-unit cyclic sequence. Note that the cycle does not define the filling-up phase, which ends with the first wafer being completed.

This chapter presents an efficient search algorithm that can quickly find the best λ-unit cyclic sequence of wafer handler moves. The algorithm requires less computational effort than a complete branch-and-bound algorithm. The problem is to find the λ-unit cyclic sequence of wafer handler moves that minimizes the total time required to process all the wafers and return them to LL.

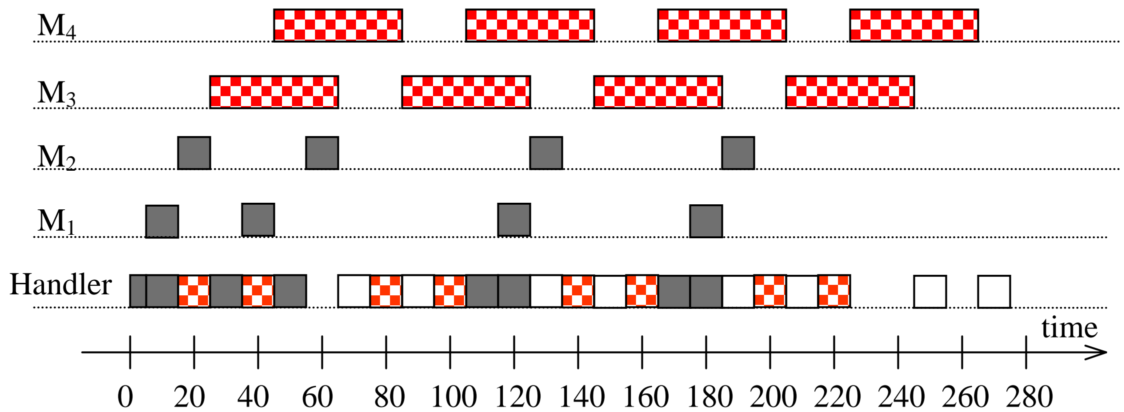
Srinivasan (1998) observes that "during steady state operation in a typical cluster tool, there is a certain sequence of events that occurs recurrently, forming a cycle." Figure 6 presents the Gantt charts of the push sequence and optimal sequence for a tool that has two chambers in the first stage and two chambers in the second stage. Based on Srinivasan's observation, we can say that the cyclic phases in the push and optimal sequences start at time 65. The cyclic phase of the push sequence consists of four 1-unit cycles σ1, while the optimal sequence consists of two 2-unit cycles σ2, where $\sigma 1 = R_{2, p} – R_{1, p+2} – R_{0, p+4}$ (p = 1,...,L-4), and $\sigma 2 = R_{2, q} – R_{1, q+2} – R_{2, q+1} – R_{1, q+3} – R_{0, q+4} – R_{0, q+5}$ (q = 1,..., L-6).

Actually, there are many ways to define the starting point of a cyclic cycle. For example, we can say that the first cyclic cycle of the push sequence in Figure 6 starts at time 75 and σ1 will be $R_{1, p+1} – R_{0, p+3} – R_{2, p}$ (p = 2,..., L-3). And so on. Hence, for convenience, we may sometimes consider different starting events for different sequences.

PUSH: MS = 285 S



Optimal: **MS = 275 s**



Note: Load S_1 Load S_2 Unload S_2

Figure 6. Gantt charts of push and optimal sequences for CT2-2 (L = 8).

Another important observation is that the order of events in the completion phase resembles that in the cyclic phase. In Figure 6, the six events in the completion phase of the optimal sequence can be divided into two incomplete cycles: the first includes $R_{2, L-3} – R_{1, L-1} – R_{2, L-2} – R_{1, L}$; the second includes $R_{2, L-1} – R_{2, L}$. The reason for lacking an event in an incomplete cycle is such event has become infeasible. For example, the first incomplete cycle lacks the load S_1 events, since there is no wafer in load lock after time 185; hence, load S_1 is infeasible.

There may exist more than one feasible filling-up sequence, we will begin by studying two special cases: CT1-1 and CT1-1-1.

5.2 The CT1-1 Problems

In this section, we analyze cycle time and makespan of the two 1-unit sequences for a two-stage cluster tool following two 1-unit cycle that Sethi *et al.* (1992) provide. For a tool that has M machines, S = M stages, each 1-unit cycle can be exactly described by M+1 wafer handler moves: M_i^- move a wafer to machine M_i (i = 1,…, S) and M_S^+ move a wafer from M_S. For the CT1-1 environment, M = 2, and the corresponding wafer handler moves of the two 1-unit cycles are:

σ1: $R_{2,j-1} – R_{0,j} – R_{1,j} – R_{2,j}$ (j = 2,..., L),

σ2: $R_{2,j-1} – R_{1, j} – R_{0,j+1} – R_{2,j}$ (j = 2,..., L).

Each cycle has only one feasible filling-up phase: for σ1 that is $R_{0, 1} – R_{1, 1}$ and for σ2 that is $R_{0, 1} – R_{1, 1} – R_{0, 2}$.

Theorem 3. For CT1-1, the 1-unit cycle time P_x and makespan MS_x ($x = 1, 2$) of the 1-unit cyclic sequences σx is given by:

$$P_1 = 3p_r + p_1 + p_2, \tag{12a}$$

$$MS_1 = LP_1, \tag{12b}$$

$$P_2 = 4p_r + \max\{2p_r, p_1, p_2\}, \tag{13a}$$

$$MS_2 = 3p_r + p_1 + p_2 + (L-1)P_2. \tag{13b}$$

Proof. The cycle time P_x and makespan MS_x are evaluated in Appendix B.

Theorem 4. For CT1-1, if $p_r \geq p_1$ and $p_r \geq p_2$, then $P_2 > P_1$ and $MS_2 > MS_1$.

Proof. Assume that $p_r \geq p_1$ and $p_r \geq p_2 \Rightarrow P_2 = 4p_r + \max\{2p_r, p_1, p_2\} = 6p_r$, and $P_1 = 3p_r + p_1 + p_2 \leq 5pr \Rightarrow P_2 > P_1$. Clearly that when $P_2 > P_1$, then $MS_2 > MS_1$ since $MS_2 - MS_1 = (L-1)(P_2 - P_1) > 0$.

5.3 The CT1-1-1 problems

In this section, we derive cycle times and makespans of the 1-unit sequences for a three-stage cluster tool. There are six 1-unit cycles that Sethi *et al.* (1992) provide. For the CT1-1-1 problems, $M = 3$, the corresponding wafer handler moves of the six 1-unit cycles are:

$\sigma 1$: $R_{3,j-1} - R_{0,j} - R_{1,j} - R_{2,j} - R_{3,j}$.

$\sigma 2$: $R_{3,j-1} - R_{0,j+1} - R_{2,j} - R_{1,j+1} - R_{3,j}$.

$\sigma 3$: $R_{3,j-1} - R_{2,j} - R_{0,j+1} - R_{1,j+1} - R_{3,j}$.

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

$\sigma 5$: $R_{3,j-1} - R_{1,j} - R_{0,j+1} - R_{2,j} - R_{3,j}$.

$\sigma 6$: $R_{3,j-1} - R_{2,j} - R_{1,j+1} - R_{0,j+2} - R_{3,j}$.

For $\sigma 1$, there is just one feasible filling-up phase. For the other five cycles, there are two feasible filling-up phases. Thus, there are eleven feasible cyclic sequences for CT1-1-1.

Theorem 5. For CT1-1-1, the cycle times and lot makespans of the 1-unit cyclic sequences that use cycle σx (x = 1, 3, 4, and 5) are given in Table 2.

Proof. The cycle times and makespans are evaluated in Appendix C.

Table 2. Cycle time and lot makespan of the 1-unit cyclic sequences that use cycle σx (x = 1, 3, 4, and 5).

Cycle	Cycle time	Filling-up phase †	Lot makespan	Equation number
σ1	$P_1 = 4p_r + p_1 + p_2 + p_3$	∅	$MS_1 = LP_1 = 4p_r+p_1+p_2+p_3+(L-1)P_1$	(14a,b)
σ3	$P_3 = 4p_r + \max\{3p_r + p_1, p_3, p_r + p_1 + p_2\}$	$f1 = R_{0,1} – R_{1,1} – R_{2,1} – R_{0,2} – R_{1,2}$	$MS_{31} = 4p_r + p_1 + p_2 + p_3 + (L-1)P_3$	(15a,b)
		$f2 = R_{0,1} – R_{1,1} – R_{0,2} – R_{2,1} – R_{1,2}$	$MS_{32} = 9p_r + p_1 + p_3 + \max\{p_2, 2p_r\} + \max\{p_1, p_2, p_3, 2p_r\} + (L-2)P_3$	(16a,b)
σ4	$P_4 = 5p_r + p_2 + \max\{2p_r, p_3, p_1\}$	$f1 = R_{0,1} – R_{1,1} – R_{2,1} – R_{0,2}$	$MS_{41} = 4p_r + p_1 + p_2 + p_3 + (L-1)P_4$	(17a,b)
		$f2 = R_{0,1} – R_{1,1} – R_{0,2} – R_{2,1}$	$MS_{42} = 8p_r + p_1 + p_2 + p_3 + \max\{p_1, p_r + p_3 + \max(p_2, 2p_r)\} + (L - 2)P_4$	(18a,b)
σ5	$P_5 = 4p_r + \max\{p_r + p_2 + p_3, 3p_r + p_3, p_1\}$	$f1 = R_{0,1} – R_{1,1} – R_{0,2} – R_{2,1}$	$MS_{51} = 4p_r + p_1 + p_2 + p_3 + (L-1)P_5$	(19a,b)
		$f2 = R_{0,1} – R_{1,1} – R_{2,1} – R_{0,2}$	$MS_{52} = 9p_r + p_1 + 2p_2 + p_3 + \max\{p_1, p_3, 2p_r\} + (L - 2)P_5$	(20a,b)

† Note filling-up phase has been adjusted in Appendix C.

Theorem 6. For CT1-1-1, the cycle time and lot makespan of the 1-unit cyclic sequences that use cycle σ2 are given by:

- Filling-up phase $f1 = R_{0,1} – R_{1,1} – R_{0,2} – R_{2,1} – R_{1,2}$:

$$P_{21} = \frac{1}{L-2}\sum_{j=2}^{L-1} P_{21j}. \quad (21a)$$

$$MS_{21} = 13p_r + p_1 + p_3 + a_1 + b_1 + c_1 + (L – 2) P_{21} + \max(p_2 - 2p_r – c_{L-1}, 0) \quad (21b)$$

Where: for all j = 2,…, (L-1)

$$P_{21j} = 7p_r + a_j + b_j + c_j, \quad (21c)$$

$$b_j = \max(p_2 - 3p_r - c_{j-1}, 0) \quad (21d)$$

$$a_j = \max(p_1 - 2p_r - b_j, 0) \quad (21e)$$

$$c_j = \max(p_3 - 2p_r - a_j, 0) \quad (21f)$$

$$b_1 = \max(p_2 - 2p_r, 0) = \max(p_2, 2p_r) - 2p_r \quad (21g)$$

$$a_1 = \max\{p_1 - \max(p_2, 2p_r), 0\} \quad (21h)$$

$$c_1 = \max(p_3 - 2p_r - a_1, 0) \quad (21i)$$

- Filling-up phase f2 = $R_{0,1} - R_{1,1} - R_{2,1} - R_{0,2} - R_{1,2}$:

$$P_{22} = \frac{1}{L-2}\sum_{j=1}^{L-2} P_{22j}. \quad (22a)$$

$$MS_{22} = 8p_r + p_1 + p_2 + p_3 + \max\{p_3, 3p_r + p_1\} + \max\{p_2 - 2p_r - g_{L-2}, 0\} + + (L-2)P_{22}. \quad (22b)$$

Where:

$$P_{22j} = 7p_r + d_j + e_j + g_j, \text{ for all } j = 1, ..., (L-2) \quad (22c)$$

- j = 1

$$e_1 = \max\{p_1 + p_2 - \max(p_3, p_1 + 3p_r), 0\} \quad (22d)$$

$$d_1 = \max\{p_1 - 2p_r - e_1, 0\} \quad (22e)$$

$$g_1 = \max\{p_3 - 2p_r - d_1, 0\} \quad (22f)$$

- $1 < j \leq (L\text{-}2)$

$e_j = \max\{p_2 - 3p_r - g_{j-1}, 0\}$ (22g)

$d_j = \max\{p_1 - 2p_r - e_j, 0\}$ (22h)

$g_j = \max\{p_3 - 2p_r - d_j, 0\}$ (22i)

Proof. The cycle times and makespans are evaluated in Appendix D.

Theorem 7. For CT1-1-1, the cycle time and lot makespan of the 1-unit cyclic sequences that use cycle σ6 are given by:

- Filling-up phase f1 = $R_{0,1} - R_{1,1} - R_{0,2} - R_{2,1} - R_{1,2} - R_{0,3}$:

$$P_{61} = \frac{1}{L-3}\sum_{j=2}^{L-2} P_{61j}. \quad (23a)$$

$$MS_{61} = 13p_r + p_1 + p_3 + w_{L-2} + \max(2p_r, p_1, p_2) + \max\{4p_r, p_2, p_3 - \max[0, p_1 - \max(2p_r, p_2)]\} + \max\{2p_r, p_2, p_3 - w_{L-2}\} + (L-3)P_{61} \quad (23b)$$

Where:

$P_{61j} = 4p_r + w_j + \max\{p_2, p_3 - w_j, 4p_r\}$, for all $j = 2, ..., (L\text{-}2)$ (23c)

$w_2 = \max\{p_1 - \max\{4p_r, p_2, p_3 - \max[0, p_1 - \max(2p_r, p_2)]\}, 0\}$ (23d)

$w_{j+1} = \max\{p_1 - \max(p_2, p_3 - w_j, 4p_r), 0\}$, for all $j = 3,..., (L\text{-}2)$. (23e)

- Filling-up phase f2 = $R_{0,1} - R_{1,1} - R_{2,1} - R_{0,2} - R_{1,2} - R_{0,3}$:

$$P_{62} = \frac{1}{L-3}\sum_{j=2}^{L-2} P_{62j}. \quad (24a)$$

$$MS_{62} = 12p_r + p_1 + p_2 + p_3 + v_{L-2} + \max\{p_r + p_1 + p_2, p_3, 5p_r + p_1\} + \max\{2p_r, p_2, p_3 - v_{L-2}\} + (L-3)P_{62}. \quad (24b)$$

Where: $P_{62j} = 4p_r + v_j + \max\{p_2, p_3\text{-}v_j, 4p_r\}$, for all j = 2, ..., (L-2) (24c)

$$v_2 = \max\{p_1 - \max[p_2, 4p_r, p_3 - p_1 - p_r], 0\} \quad (24d)$$

$$v_{j+1} = \max\{p_1\text{-}\max(p_2, p_3 - v_j, 4p_r), 0\}, \text{ for all } j = 3, ..., (L\text{-}2). \quad (24e)$$

Proof. The cycle times and makespans are evaluated in Appendix E.

Hall *et al.* (1997) show that, for a mobile-robot cell, which has 3 machines and 3 stages, processing single part-type, there is a unique formula for cycle time if the tool follows σ2. However, this is not applicable to our tool structure. For example, consider the problem CT1-1-1 | | C_{max}. Table 3 and 4 present cycle times and makespans of σ2-sequence, using filling-up f1 and f2 respectively, for some instances. Let $p_r = 1$, $p_1 = 5$, and $p_3 = 10$.

Table 3. Cycle time and makespan of σ2-sequence, using f1 = $R_{0,1} - R_{1,1} - R_{0,2} - R_{2,1} - R_{1,2}$, for some instances.

Additional data	P_{21j}								P_{21}	MS_{21}
	j = 2	j = 3	j = 4	j = 5	j = 6	j = 7	j = 8	j = 9		
$p_2 = 20$, L = 9	24	24	24	24	24	24	24	-	24	232
$p_2 = 20$, L = 10	24	24	24	24	24	24	24	24	24	256
$p_2 = 10$, L = 9	15	17	15	17	15	17	15	-	15.86	158
$p_2 = 10$, L = 10	15	17	15	17	15	17	15	17	16	173

Table 4. Cycle time and makespan of σ2-sequence, using f2 = $R_{0,1} - R_{1,1} - R_{2,1} - R_{0,2} - R_{1,2}$, for some instances.

Additional data	P_{22j}								P_{22}	MS_{22}
	j = 1	j = 2	j = 3	j = 4	j = 5	j = 6	j = 7	j = 8		
$p_2 = 20$, L = 9	30	24	24	24	24	24	24	-	24.85	237
$p_2 = 20$, L = 10	30	24	24	24	24	24	24	24	27.75	261
$p_2 = 10$, L = 9	20	15	17	15	17	15	17	-	16.57	160
$p_2 = 10$, L = 10	20	15	17	15	17	15	17	15	16.38	177

The following Corollaries are direct results from Theorem 5.

Corollary 1. For a CT1-1-1, if $P_x \geq P_y$, then $MS_{x1} \geq MS_{y1}$ for all x, y = 1, 3, 4, and 5.

Proof. By comparing formulas of MS_1, MS_{31}, MS_{41}, and MS_{5f1}.

Corollary 2. For a CT1-1-1, there is no guarantee that $MS_{x1} = MS_{x2}$ for all x = 2, 3, 4, 5, and 6.

Proof. By considering the following instance. Let L = 15, $p_r = 13$, $p_1 = 18$, $p_2 = 14$, $p_3 = 37$. Table 5 presents the cycle times and lot makespans of the eleven sequences dictating in Theorems 5, 6, and 7.

Table 5. Cycle time and makespan of the 1-unit cyclic sequences when $L = 15$, $p_r = 13$, $p_1 = 18$, $p_2 = 14$, $p_3 = 37$.

<table>
<tr><td>x</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr>
<tr><td>P_{x1}</td><td rowspan="2">121</td><td>102</td><td rowspan="2">109</td><td rowspan="2">116</td><td rowspan="2">128</td><td>104</td></tr>
<tr><td>P_{x2}</td><td>102</td><td>104</td></tr>
<tr><td>MS_{x1}</td><td rowspan="2">1815</td><td>1561</td><td>1647</td><td>1745</td><td>1913</td><td>1587</td></tr>
<tr><td>MS_{x2}</td><td>1556</td><td>1652</td><td>1757</td><td>1901</td><td>1593</td></tr>
</table>

Note that in this instance, MS_{22} is the makespan of the best 1-unit cyclic sequence.

Lemma 1. Repetition of the best 1-unit wafer hander moves cycle does not guarantee optimality for the problem CT1-1-1 | | C_{max}.

Proof. We will prove this lemma by providing a counterexample. Let $L = 10$, $p_r = 16$, $p_1 = 20$, $p_2 = 11$, and $p_3 = 18$. Then $\sigma 2$ is the optimal steady state cycle whose $P_{21} = P_{22} = 112$. The makespans of the $\sigma 2$-sequences with different filling-up phase f1 and f2 are $MS_{21} = 1142$ and $MS_{22} = 1141$. However, the optimal makespan is of the sequence that forms by repeating $\sigma 1$ whose $P_1 = 113$, and $MS_1 = 1130$.

However, we expect that if L is large enough then repeating of the optimal 1-unit cycle will lead to an optimal 1-unit cyclic sequence. For instance, revisit the counterexample in Lemma 1, if L greater than 21, then MS_{22} is optimal. Table 6 shows MS_1 and MS_{22} as functions of lot size L. In this example, MS_1 equals MS_{22} when L equals 21.

Table 6. MS_1 and MS_{22} as functions of L.

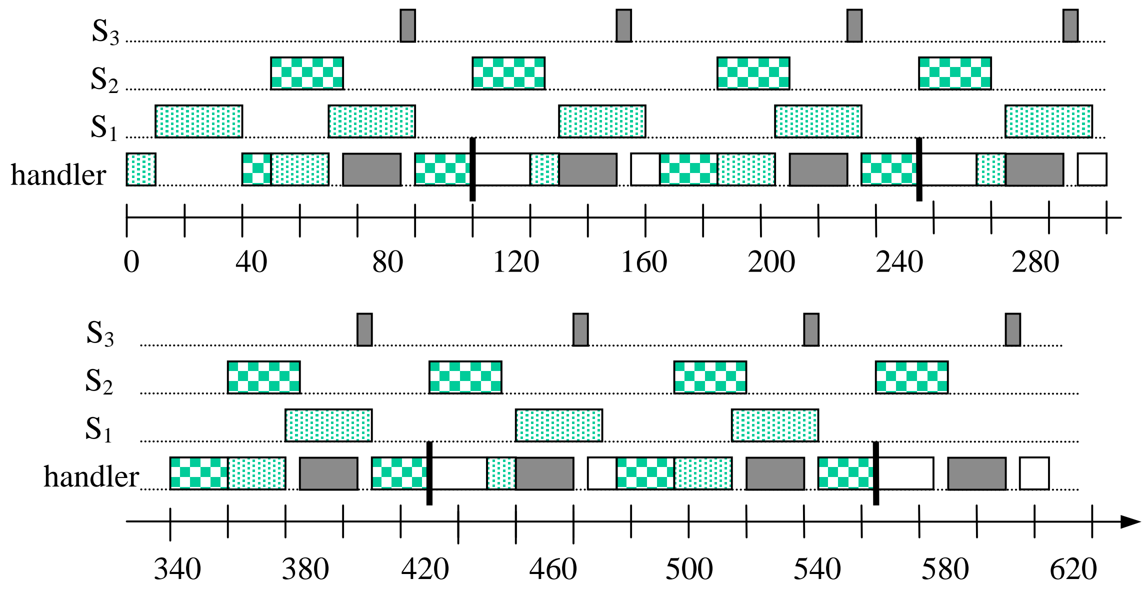
L (wafers)	19	20	**21**	22	23	24	25
MS_1 (unit times)	2147	2260	**2373**	2486	2599	2712	2825
MS_{22} (unit times)	2149	2261	**2373**	2485	2597	2709	2821

Hall *et al.* (1997) prove that, for a mobile-robot cell, which has M machines and M stages, processing single part-type, the repetition of 1-unit cycles dominates more complicated policies that produce 2-unit cycles. However, this is not applicable to the problem CT1-1-1 | | C_t.

Theorem 8. The 1-unit cycles do not dominate 2-unit cycle for the problems CT1-1-1 | | C_t and CT1-1-1 | | C_{max}.

Proof. We will prove this theorem by providing a counterexample. Let L = 8, $p_r = 10$, $p_1 = 30$, $p_2 = 25$, $p_3 = 5$. Then the optimal 1-unit cycle is σ5 whose $P_5 = 80$. The best 1-unit cyclic sequence is formed by repeating σ5 with filling-up phase f1 = $R_{0,1} – R_{1,1} – R_{0,2} – R_{2,1}$. $MS_{5f1} = 660$ time units. However, the optimal sequence forms by repeating 2-unit cycle, $R_{3,j} – R_{0,j+2} – R_{2,j+1} – R_{3,j+1} – R_{1,j+2} – R_{0,j+3} – R_{2,j+2} – R_{1,j+3}$, which has an average cycle time of 77.5 time units. The optimal makespan is 645. Figure 7 presents the Gantt charts of the optimal sequence and the sequence formed by repeating σ5 with filling-up phase f1.

OPTIMAL: MS = 645



REPEATING CYCLE 5 WITH FILLING-UP F1: MS = 660

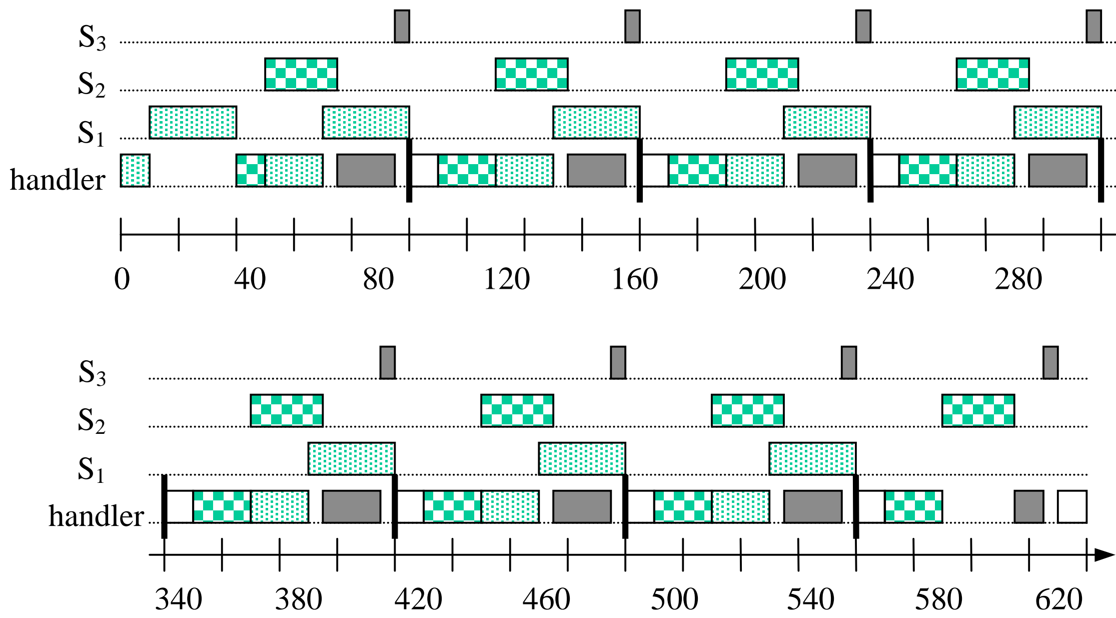


Figure 7. Two-unit cycle versus one-unit cycle.

5.4 THE CTm_1-...-m_S | | C_{max} PROBLEMS

In general a stage may have more than one chamber; the number of chambers in a stage can be any integer. Although the branch-and-bound algorithm can find an optimal sequence of wafer handler moves, its main limitation is the computing effort increases as the lot size, the number of chambers in each stage, and the number of stages increases. In such complicated problems, we propose an algorithm that can find the best solution among the λ-unit cyclic schedules, $\lambda = \min\{m_1, \ldots, m_S\}$. As a result from Corollary 2, this best schedule must have the best filling-up phase associated with the best cyclic phase. The following assumptions are used to identify the λ-unit cycle:

- The cyclic phase starts with the first move that unloads a finished wafer from the last stage (and returns it to the load lock).
- The cyclic phase of a cluster tool can be divided into complete cycles. In each complete cycle, the number of unprocessed wafer(s) leaving the load lock must equal the number of finished wafer(s) returning to the load lock. Furthermore, this number must be greater than or equal to λ.
- The occurring order of events in an incomplete cycle of the completion phase resembles the occurring order of events in a complete cycle.

The first assumption guarantee that all possibly feasible filling-up phase will be considered. We now present a truncated branch-and-bound algorithm to find the best λ-unit cyclic schedule. This algorithm is a modified version of the branch-and-bound

algorithm proposed in Chapter 4. The algorithm first generates two feasible sequences using the push and pull dispatching rules and uses the better sequence's lot makespan as the upper bound on the optimal lot makespan. For a partial solution, we use the completion time of the last scheduled activity as its lower bound.

Given $m_1,...,m_S$, p_r, $p_1,...,p_S$, and L, the algorithm proceeds as follows.

Algorithm TBB:

Step 0:

Use the push and pull dispatching rules to generate two feasible sequences. Take the better sequence as an optimal sequence candidate, and use its makespan as an upper bound on the optimal value.

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 t = 0, n = L, and $t_k = 0$ for all chambers k.

Step 2:

Based on the tool state, identify all feasible moves. $R_{0,j}$ is feasible if 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 calculate LB as follows: add the feasible move to the current sequence, compute the move's completion time, and update the tool state.

- 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.
- Otherwise, 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} + p_r. Chamber k is now free. Go to Step 3d.

Step 3d:

Let t = C.

- If LB is greater than or equal to the upper bound on the optimal value, then discard this new sequence. Go to Step 4.
- If the new sequence includes all L(S+1) moves, the lot makespan equals C. If the lot makespan is less than the upper bound, save the new sequence as the current best sequence and set the upper bound equal to the lot makespan. Go to Step 4.
- If the number of finished wafers is greater than λ, go to Step 3e.
- Otherwise, go to Step 4.

Step 3e:

If a λ-unit cycle exist in the steady state phase, then repeat this λ-unit cycle until n = 0 and all chambers are free. The lot makespan equals C. If the lot makespan is less than the upper bound, save the new sequence as the current best sequence and set the upper bound equal to the lot makespan. Go to Step 4.
Otherwise, discard this new sequence. Go to Step 4.

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.

Note that Algorithm TBB also applies to the CT1-1 | | C_{max} and CT1-1-1 | | C_{max} problems. Figure 8 presents the search tree of a CT1-2 using the Algorithm TBB. The search found four 1-unit cyclic sequences.

Q1 repeats $\sigma 1 = R_{2,j} - R_{1,j+1} - R_{0,j+2}$,

Q2 repeats $\sigma 2 = R_{2,j} - R_{1,j+2} - R_{0,j+3}$,

Q3 repeats $\sigma 3 = R_{2,j} - R_{0,j+2} - R_{1,j+2}$, and

Q4 repeats $\sigma 4 = R_{2,j} - R_{0,j+1} - R_{1,j+1}$.

Note that σ1 is similar to σ2 and σ3 is similar to σ4. Neither Q1 nor Q4 uses the second chamber of S_2. Table 7 compares results of the search with the push/pull dispatching rule for some instances of a CT1-2 when L = 10.

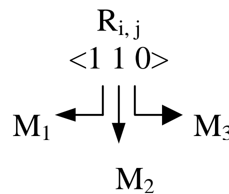
Table 7. TBB, push, and pull sequences for CT1-2.

p_r	p_1	p_2	TBB		push		pull	
			MS	cycle	MS	cycle	MS	cycle
10	5	5	400	σ4	580	σ1	580	σ1
5	10	20	315	σ2	315	σ2	405	σ1
3	5	10	177	σ3	222	σ1	222	σ1

Start <0 0 0>

- $R_{0,1}$ <1 0 0>
  - $R_{1,1}$ <0 1 0>
    - $R_{0,2}$ <1 1 0>
      - $R_{2,1}$ <1 0 0>
        - $R_{1,2}$ <0 1 0>
          - $R_{0,3}$ <1 1 0>
            - $R_{1,3}$ <0 1 1>
              - $R_{0,4}$ <1 1 1>
                - $R_{2,2}$ <1 0 1> $
              - $R_{2,2}$ <0 0 1> $
            - $R_{2,2}$ <1 0 0> σ1
          - $R_{2,2}$ <0 0 0> $
      - $R_{1,2}$ <0 1 1>
        - $R_{0,3}$ <1 1 1>
          - $R_{2,1}$ <1 0 1>
            - $R_{1,3}$ <0 1 1>
              - $R_{2,2}$ <0 1 0> $
              - $R_{0,4}$ <1 1 1>
                - $R_{2,2}$ <1 1 0> σ2
            - $R_{2,2}$ <1 0 0> $
        - $R_{2,1}$ <0 0 1>
          - $R_{0,3}$ <1 0 1>
            - $R_{1,3}$ <0 1 1>
              - $R_{2,2}$ <0 1 0> σ3
              - $R_{0,4}$ <1 1 1>
                - $R_{2,2}$ <1 1 0> $
          - $R_{2,2}$ <0 0 0> $
    - $R_{2,1}$ <0 0 0>
      - $R_{0,2}$ <1 0 0>
        - $R_{1,2}$ <0 1 0>
          - $R_{0,3}$ <1 1 0>
            - $R_{1,3}$ <0 1 1>
              - $R_{0,4}$ <1 1 1>
                - $R_{2,2}$ <1 1 0> $
              - $R_{2,2}$ <0 1 0> $
            - $R_{2,2}$ <1 0 0> $
          - $R_{2,2}$ <0 0 0> σ4

Termination criteria:
- Pattern found:
  - $\sigma 1 = R_{2,j} - R_{1,j+1} - R_{0,j+2}$
  - $\sigma 2 = R_{2,j} - R_{1,j+2} - R_{0,j+3}$
  - $\sigma 3 = R_{2,j} - R_{0,j+2} - R_{1,j+2}$
  - $\sigma 4 = R_{2,j} - R_{0,j+1} - R_{1,j+1}$
- Pattern not found: $



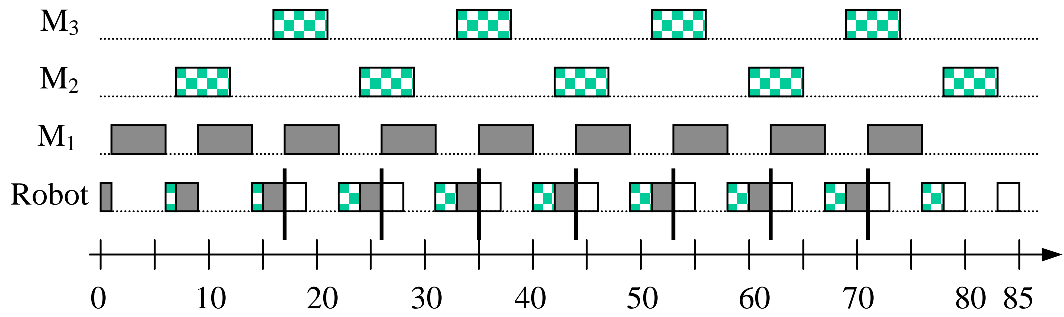
where: 1 = busy
0 = free

Figure 8. Search tree of a CT1-2 using Algorithm TBB.

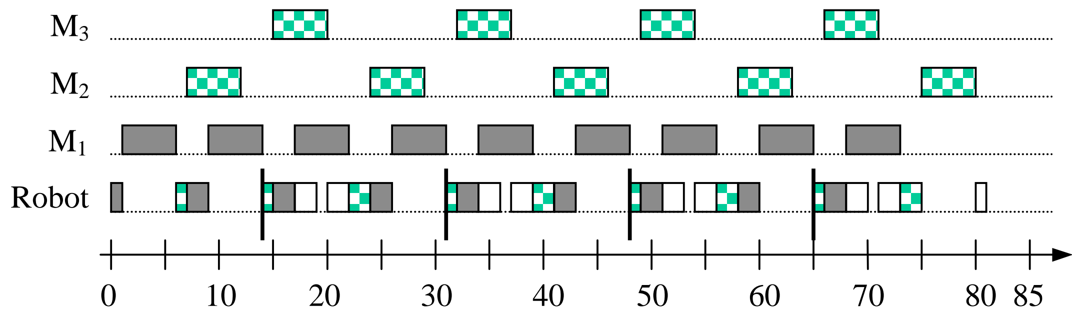
Theorem 9. The λ-unit cyclic schedule is not necessarily optimal for the problem CTm_1-…-m_S | | C_{max}.

Proof: We will again prove this theorem by providing a counterexample. Consider CT1-2 | | C_{max}, with $p_r = 1$, $p_1 = p_2 = 5$, $L = 9$. Here, $\lambda = \min\{1, 2\} = 1$. The best 1-unit cyclic sequence is Q2. This has a lot makespan of 85. The optimal schedule repeats 2-unit cycle in its steady state phase. The optimal makespan is 81. Figure 9 presents Gantt charts of the sequence Q2 and the optimal sequence.

Best 1-unit cyclic sequence Q2: MS = 85.



Optimal sequence: MS = 81.



Note $R_{0,j}$ $R_{1,j}$ $R_{2,j}$

Figure 9. The best 1-unit cyclic sequence and the optimal sequence for CT1-2.

5.5 Summary

The steady state behavior of the cluster tools processing finite lot sizes is studied in this chapter. Normally, if the lot size L is large enough, the performance of a cluster tool might be separated into three phases: filling-up, steady state (or cyclic), and completion. Until the first wafer is completed, the tool is filling up with wafers. During steady state operation in a typical cluster tool, there is a certain sequence of events that occurs recurrently, forming a cycle. The order of events in the completion phase resembles that in the cyclic phase.

The analytical models for CT1-1 and CT1-1-1 are derived to determine the cycle time and lot makespan of the 1-unit cyclic sequence of wafer handler moves, given lot size, handler move time, and chamber processing time. We have found counterexamples showing that the 1-unit cyclic sequence may not be optimal even when there is only one chamber in each stage. Also, counterexamples have been found to show the need of searching both filling-up and cyclic phases.

This chapter presents an efficient search algorithm, Algorithm TBB, that can quickly find the best λ-unit cyclic sequence of wafer handler moves ($\lambda = \min\{m_i,..., m_S\}$). The algorithm requires less computational effort than a complete branch-and-bound algorithm; however, it will not find the optimal solutions. Algorithm TBB, which is a modified version of Algorithm BB, searches all possibly feasible filling-up phases and cyclic phases to find the best λ-unit cyclic sequence. A partial branch is truncated

either when a cycle is found or when the branch is long enough. Algorithm BB can be implemented to build models of single wafer handler and single load lock cluster tools for any tool configuration.

CHAPTER 6

IMPLEMENTATION

This chapter describes the implementation of the cluster tool models as software that can generate, for any given tool configuration and lot size, a sequence of wafer handler moves and then determine the makespan of the resulting sequence of all activities. Although the cluster tool models can be applied to any cluster tool configuration, we have implemented them for two- and three-stage cluster tools.

6.1 Introduction

For implementing the cluster tools models, the software was written in Java, since it provides many advantages over C/C++ such as automatic garbage collection, pure object orientation with multiple inheritance, and multi-threading. One of the advantages is the concept of a vector, a changeably sized array of objects, which will be used very frequently in the programs.

Before introducing the Java code to implement Algorithm BB and Algorithm TBB for the two and three stage cluster tools, it is necessary to introduce some main variables and their construction in the programs. There are two main objects that will be used, the Job and Machine, which allow us to store some information. Each basic object can store several information such as a String, an Integer, a Double, or a Vector of other Objects; that is a Job can store a Vector of Machines and vice versa (see Appendix F for

construction of a Job and a Machine). Table 8 presents the variables used to build cluster tool models.

Table 8. Variables used to build cluster tool models.

Variables	Storing information	Construction
l	number of unprocessed wafers in LL	integer
Robot position	name of stage and index of chamber in stage, or LL	Job
Global timer	completion time of the latest move	Machine
Chamber	index and completion time of wafer being processed	Machine
Stage	list of the chambers in the stage	Vector of Machines
Handler move activity	▪ type of move ▪ index of wafer being moved ▪ index of activity	Job
In-chamber processing activity	▪ type of processing (stage name) ▪ index of chamber that processed the activity ▪ index of wafer being processed ▪ index of activity	Job
Sequence	list of activities	Vector of Jobs

Hence, the *l*, Stages, Robot position, and Global timer variables provide information regarding to the current cluster tool status, while the Sequence variable provides information regarding to the history of activities. For instance, we need to determine whether a move $R_{a,j*}$ which moves a wafer from stage S_a to stage S_{a+1}, takes p_r or $2p_r$. $R_{a, j*}$ is feasible only if S_a contains at least one wafer and S_{a+1} has at least one free chamber. We assume that the wafer handler should always load the lowest-numbered free chamber in a stage, and the permutation constraint is active. Let S_{a_k} (k = 1,..., m_a) be the name of the chamber in S_a, which processes the wafer j*, that has the smallest completion time. Let $S_{a+1_k'}$ be the name of the lowest-numbered free chamber in S_{a+1}. If robot position is different from S_{a_k}, then $R_{a, j*}$ takes $2p_r$ (otherwise, p_r). Two activities $R_{a, j*}$ and $S_{a+1_k', j*}$ will be added to the Sequence. The wafer handler is now at $S_{a+1_k'}$. The global time is reset to completion time of $R_{a, j*}$.

6.2 Implementation as executables

The input to the Java executable is a string of (2S + 3) numbers, separated by white spaces. The first number is the lot size L. Next S numbers are the tool configuration (m_1, ..., m_S). Then, S+1 numbers are p_r, p_1,..., p_S. The last number is the overhead time. For example, the string 25 1 2 5 10 40 400 is input for a CT1-2, processing 25 wafers per lot. $p_r = 5$, $p_1 = 10$, $p_2 = 40$, and OH = 400.

The output of a cluster tool model is a list of activities. From this list, we can find the lot makespan and the utilization of each chamber in the tool and of the wafer handler

per lot, which is the ratio of production time compared to total available time. For each chamber, the utilization is the total time that the wafers are actually processed in the chamber divided by the makespan. Note that the waiting or blocking time in the chamber will not be included when determining the chamber utilization. For the wafer handler, the utilization is the sum of unloaded move times and loaded move times divided by the makespan.

While writing the computer codes to implement Algorithm BB, we will make use the automatic garbage collection ability of Java to simplify the codes. In fact, while generating a trial list, the tool's status changes whenever a move is added to the trial list. Therefore, information of tool's status must be recovered when generating the next trial list. Using Java language, we can easily overcome this difficulty by creating temporary tool environment whenever a move is added to the trial list.

For instance, consider an arbitrary outtree graph in Figure 10. There are four branches in this graph. After searching the first branch A-B-C1-D1-E1-F1, the status of the tool associates with the last node F1. To search the next branch A-B-C1-D1-E2-F2, we need to delete nodes E1 and F1 from the trial list and to recover the tool's status to the one associated with node D1, then add nodes E2 and F2. And so on. This means information of the tool's status associated with every single node in the graph must be permanently stored. If we create temporary trial list and tool environment any time a node is added, then when jumping to the next branch, the first trial list and tool's status associated with the last nodes become garbage and will be automatically destroyed.

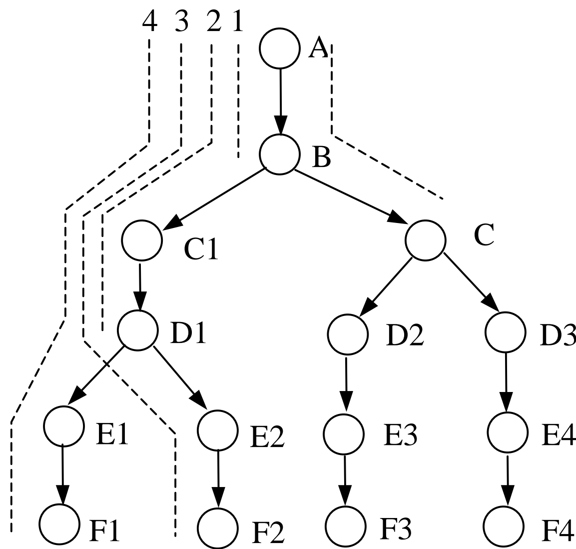


Figure 10. An arbitrary outtree graph.

In details, this procedure is follows. Start at the first node A, when adding node B, a list A-B and associated tool's environments have created (the dashed line 1 in Figure 10). And so on. The first branch A-B-C1-D1-E1-F1 representing by the dashed line 4, after being evaluated its length, becomes garbage and will be destroyed. The second branch A-B-C1-D1-E2-F2, representing by the dashed line 3, will start from the end of the dashed line 2. And so on. Note that the best list and upper bound will be updated whenever a better branch is found.

6.2.1 Cluster tool models using Algorithm BB

Figure 11 shows the flow chart for Algorithm BB to find an optimal sequence of wafer handler moves for given lot size, tool configuration, processing times required at each stage, and wafer handler move time. Example 1 in Appendix F presents the main portion of the Java codes, the "search ( )" method, for a three-stage cluster tool model using Algorithm BB. Note that in this model, all three dominance criteria are active. The first and second criteria are applied when making the move; in fact, the wafer handler will move the earliest completed wafer from S_i to the lowest-numbered free chamber in S_{i+1}. The third dominance criterion is applied when finding the feasible moves.

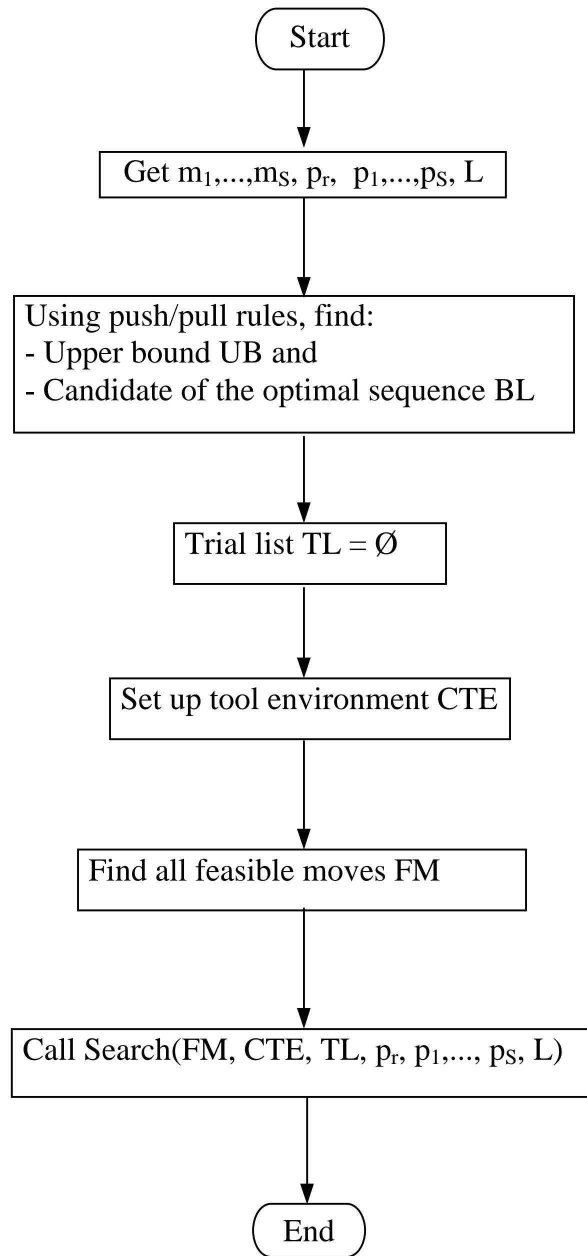


Figure 11. Flow chart for program BB (to be continued).

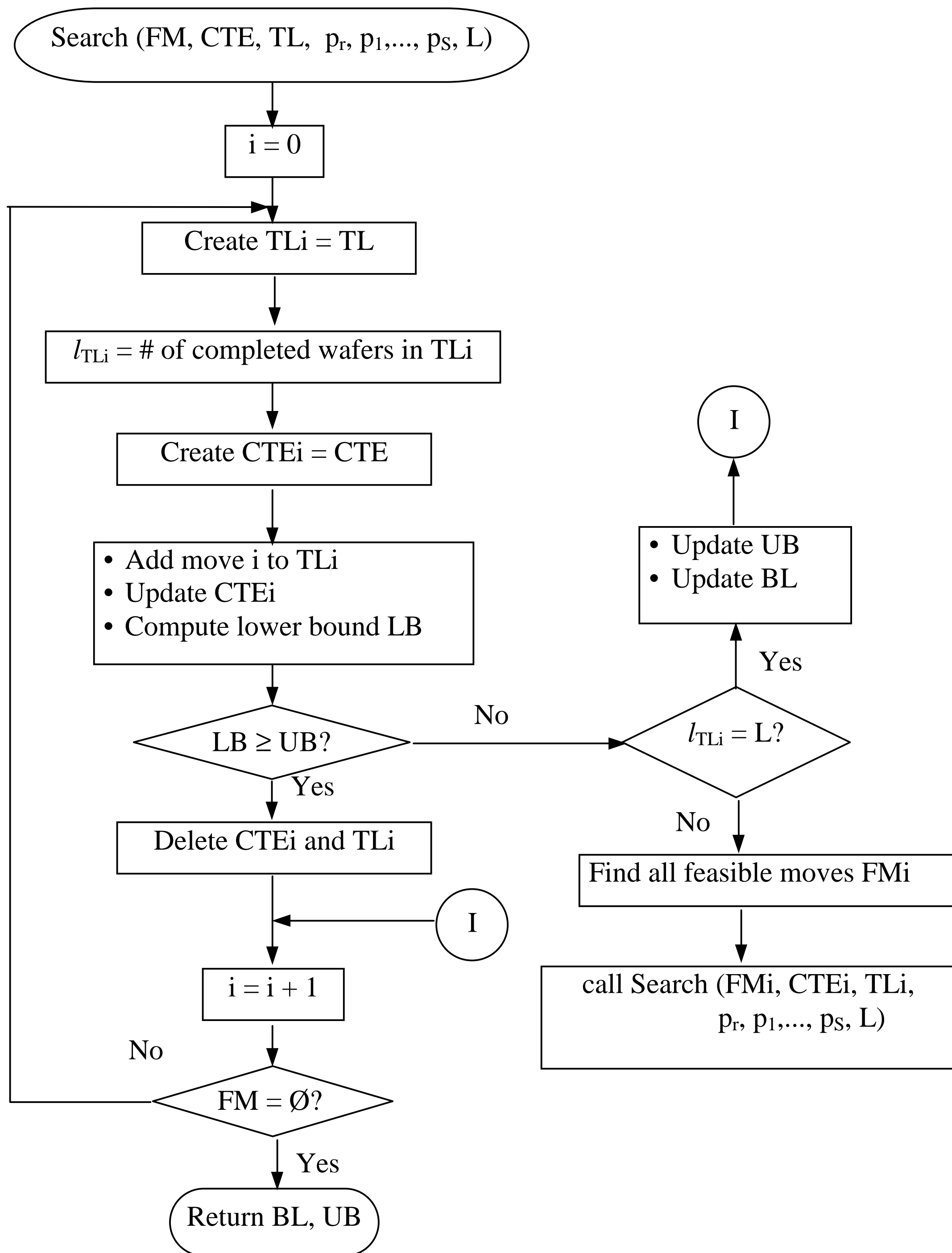


Figure 11 continued.

6.2.2 Cluster tool models using Algorithm TBB

The flow chart for Algorithm TBB to find an optimal sequence of wafer handler moves for given lot size, tool configuration, processing times required at each stage, and wafer handler move time is presented in Figure 12. Since Algorithm TBB is a modified version of Algorithm BB, we will only discus on how to modify the BB model. Example 2 in Appendix F presents the main portion of the Java code, the smart_search() method, for a three-stage cluster tool model using Algorithm TBB.

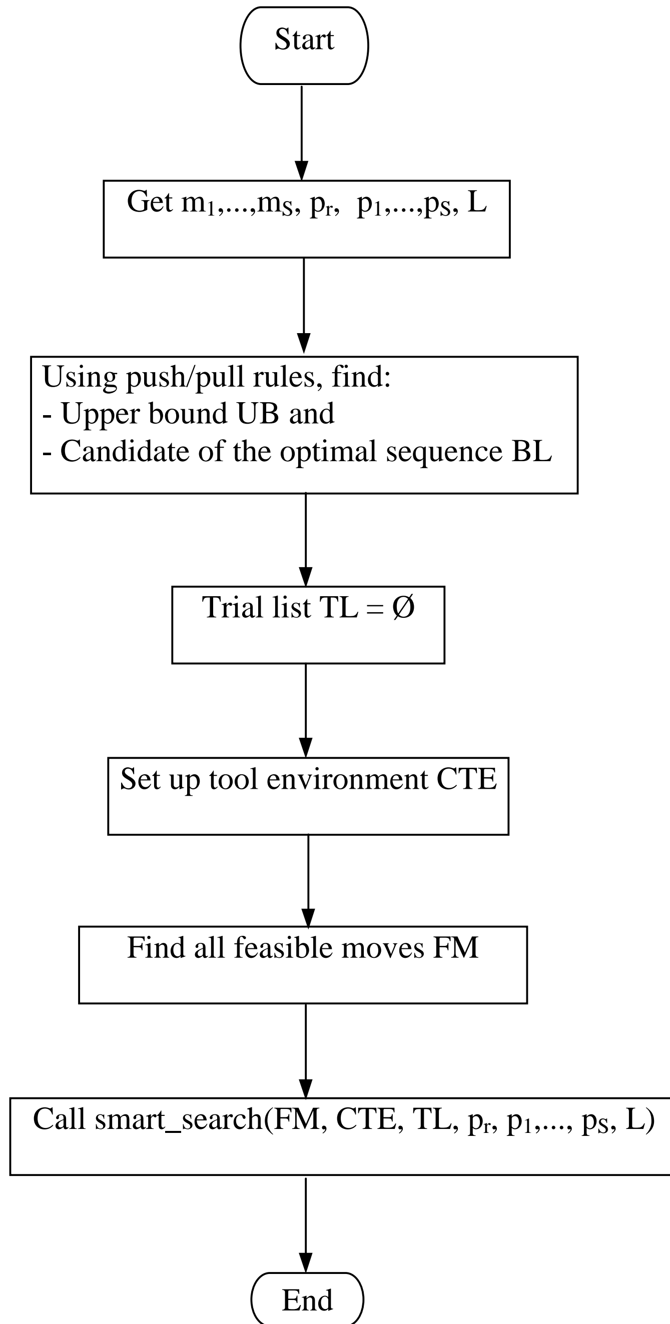


Figure 12. Flow chart for program TBB (to be continued).

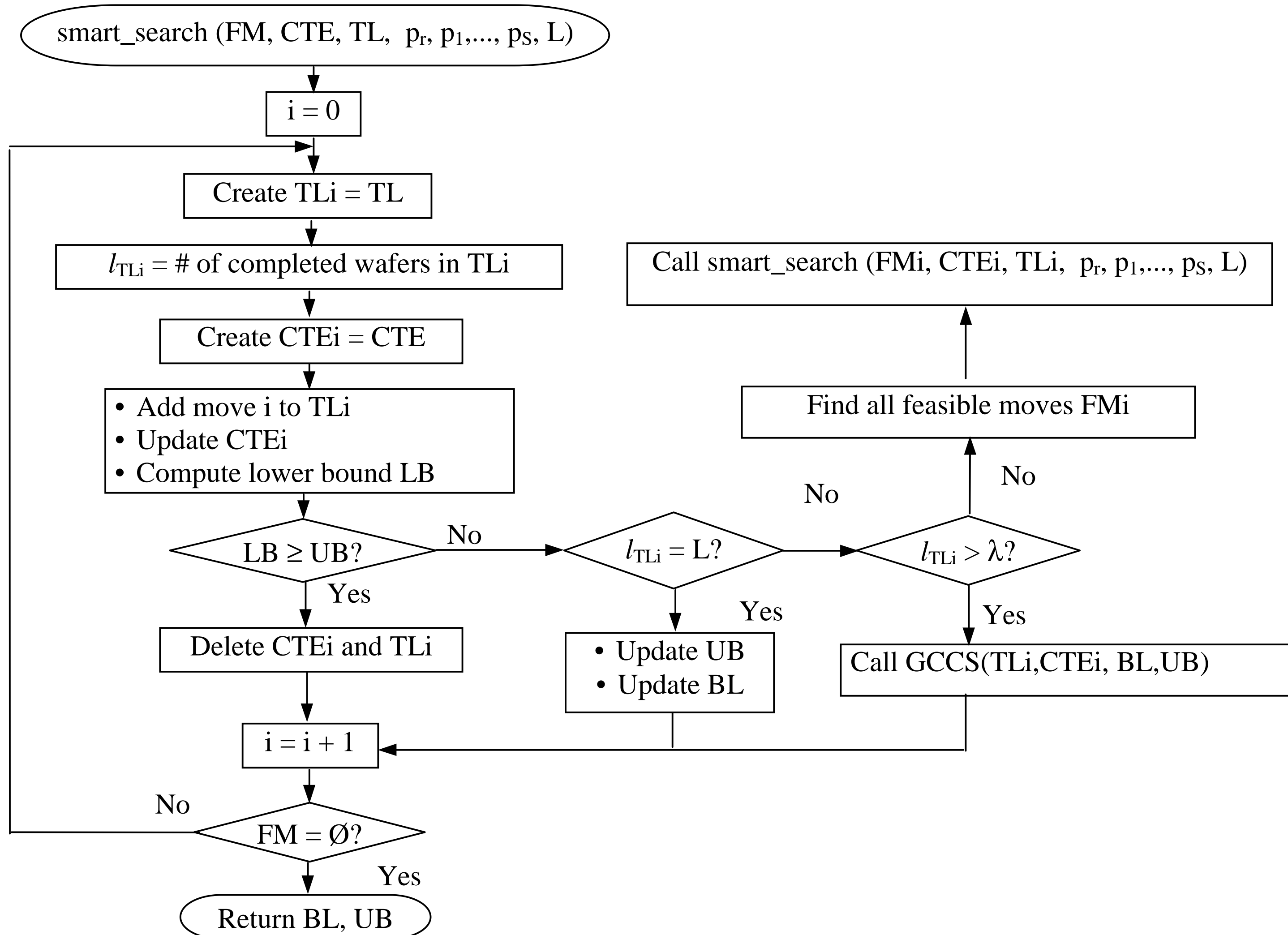


Figure 12 continued.

Given a trial sequence σ in which its number of completed wafers is smaller than L but greater than $\lambda = \min(m_i)$, Algorithm GCCS can generate a complete cyclic sequence if there exists a complete cycle in σ. Figure 13 shows the flow chart for Algorithm GCCS.

Algorithm GCCS

- Get departure list DL, which includes the departure stages of all moves in the trial sequence. For example, the departure stage of $R_{0,j}$ is LL, the departure stage of $R_{i,j}$ is S_i ($i = 1, \ldots, S$).
- From DL, check whether a λ-unit cycle exists and to determine the starting and ending position of the cycle in DL (Algorithm CheckCycle). Then,
- If the λ-unit cycle does not exist, discard the trial sequence σ. Go to the next trial sequence.
- If the λ-unit cycle exists, collect the departures of moves in the filling-up phase and the first cycle from DL. The collected departures of moves as well as other input information (lot size, move time and so on) are used as input to the fixed sequence cluster tool model to generate a complete sequence of all activities, Q. Then determine the makespan of Q and compare it to the current upper bound value. Update candidate upper bound and candidate best sequence if necessary. Discard both Q and σ and go to the next trial sequence.

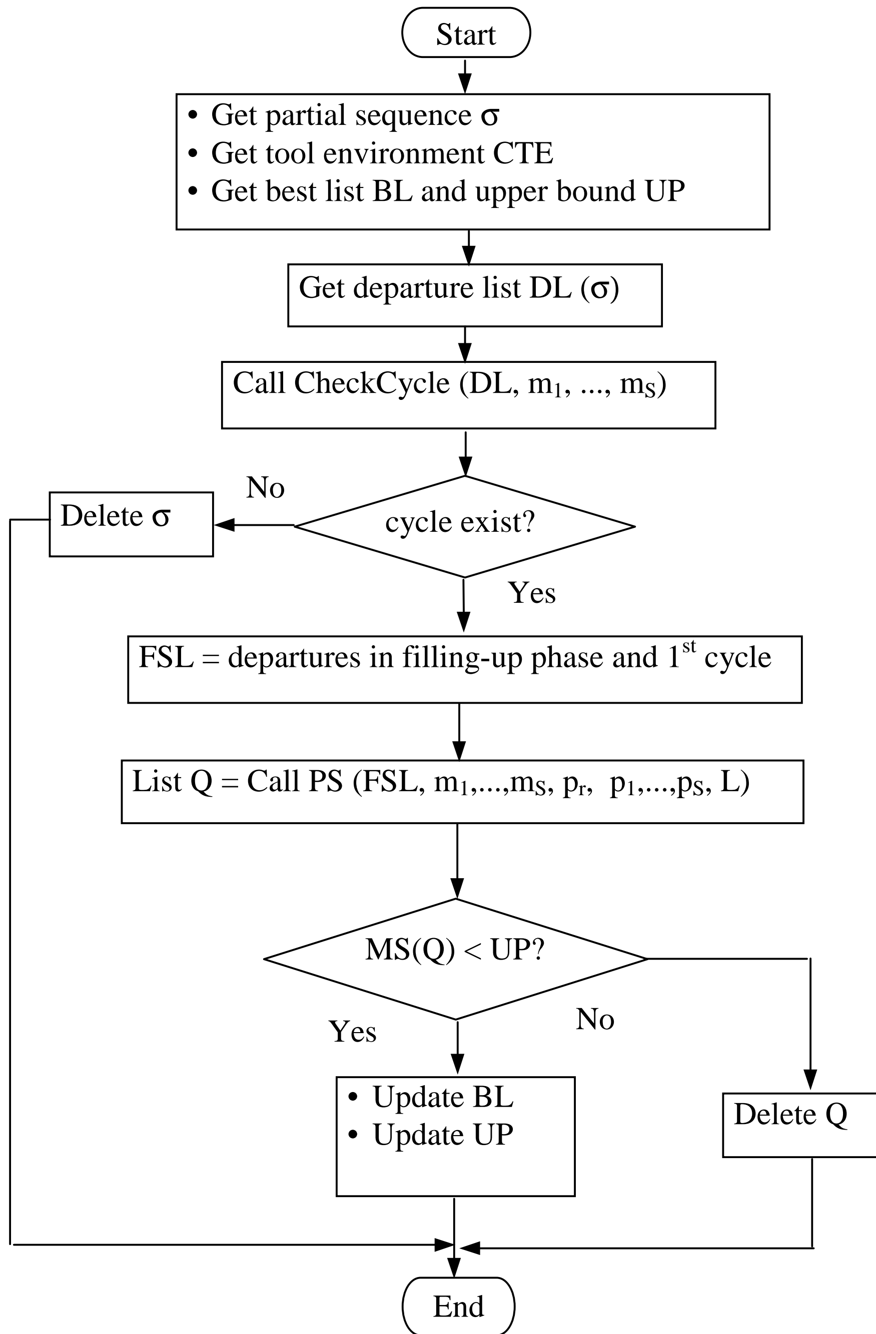


Figure 13. Flow chart for program GCCS.

We now present the algorithm CheckCycle, given the departure list DL. Let Wi be the number of wafer(s) that have started processing in S_i (i = 1, …, S). Let OUT be the number of completed wafers in DL. Let n_{DL} be total numbers of departures in DL. Denote START and END as starting and ending points of the cyclic cycle. Figure 14 shows the flow chart for Algorithm CheckCycle.

Algorithm CheckCycle

Let x = 0, Wi(DL) = 0, OUT = 0, START = 0, and END = 0.

Step 1:

Find the position x of the first S_S in DL (the first wafer is completed). Set START to (x-1).

Step 2:

For x = (START+1) to (n_{DL}-1), denote DL(x) as the xth departure stage in DL.

- If DL(x) = LL, add one to W1. If DL(x) = Si (i = 1,…, S-1), add one to W(i+1). Otherwise, add one to OUT.
- Let END = x.
- If Wi $\geq \lambda$ and Wi = Wj = OUT, for all i $\neq$ j, (i, j = 1, …, S), then
  - If DL(x+1) = DL(START+1), then STOP, the cyclic cycle is found as well as its starting and ending points in DL.
  - Otherwise, increase START by one and reset Wi = OUT = 0.
- If W1 $\geq \lambda$ but Wi $\neq$ Wj for at least one pair i and j or Wi $\neq$ OUT for at least one i, then increase START by one and reset Wi = OUT = 0
- If x = (n_{DL}-1), then STOP, the cyclic cycle is not found. Otherwise, next x.

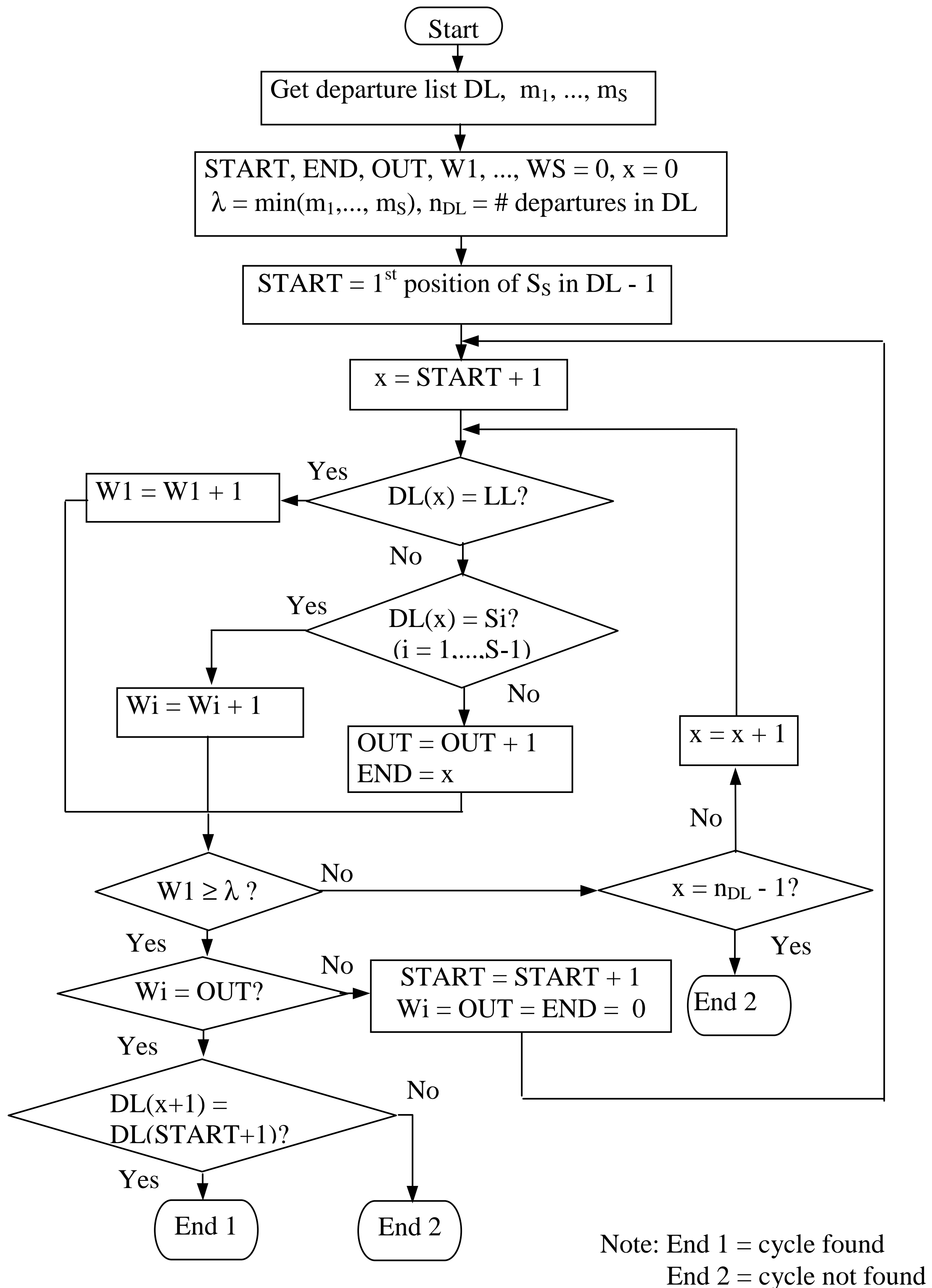


Figure 14. Flow chart for program CheckCycle.

6.2.3 Cluster tool models using the push and pull dispatching rules

Algorithm P in Chapter 4 can be implemented to any tool configuration. In this section, we will introduce the cluster tool models using Algorithm P for a two- and three-stage cluster tool. Table 9 and Table 10 present the status of two-stage cluster tools using the push and pull dispatching rules to sequence the wafer handler moves respectively. Table 11 and Table 12 present the status of three-stage cluster tools using the push and pull dispatching rules to sequence the wafer handler moves respectively In Table 9 and Table 10, the status of a chamber is defined as follows.

Free, F: there is no wafer in the chamber

Done, D: wafer is finished processing, ready to unload

Busy, B: wafer is still in process at the chamber

For a stage, we also define its status as follows.

F: there is at least one free chamber.

D: there is at least one done chamber.

B: all chambers are busy.

Figure 15 shows the flow chart for Algorithm P. Example 3 and Example 4 in Appendix F present the main portion of the Java codes for a three-stage cluster tool model using the push and pull rules respectively.

Table 9. Status of a two-stage tool using the push dispatching rule

Case	Status of stage		Priority	Resulting Action
	S_1	S_2		
1	F	F	1	Load S_1
2	F	D		
3	F	B		
4	D	F	2	Load S_2
5	D	D	3	Unload S_2
6	D	B	otherwise	Unload S_2
7	B	F	4	Check S_1 and S_2 • if one of S_2 is done first: unload S_2 • if one of S_1 is done first or at the same time as one of the S_2: load S_2
8	B	D	3	Unload S_2
9	B	B	otherwise	Unload S_2

Table 10. Status of a two-stage tool using the pull dispatching rule

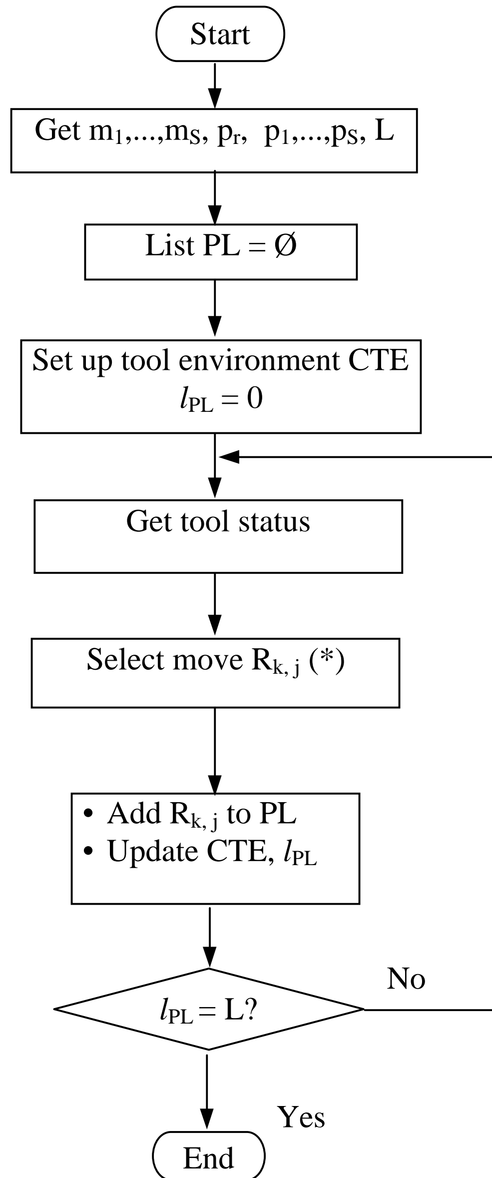
Case	Status of stage		Priority	Resulting Action
	S_1	S_2		
1	F	D	1	Unload S_2
2	D	D		
3	B	D		
4	F	F	3	Load S_1
5	D	F	2	Load S_2
6	B	F	4	Check S_1 and S_2 • if one of S_2 done first or at the same time as one of S_1: unload S_2 • if one of S_1 done first: load S_2
7	F	B	3	Load S_1
8	D	B	otherwise	Unload S_2
9	B	B		

Table 11. Status of a three-stage tool using the push dispatching rule.

Case	Status of stage			Priority	Resulting Action and Priority
	S_1	S_2	S_3		
1	F	F	F	1	Load S_1
2	F	F	D		
3	F	F	B		
4	F	D	F		
5	F	D	D		
6	F	D	B		
7	F	B	F		
8	F	B	D		
9	F	B	B		
10	D	F	F	2	Load S_2
11	D	F	D		
12	D	F	B		
13	D	D	F	3	Load S_3
14	D	D	D	4	Unload S_3
15	D	D	B	otherwise	Unload S_3
16	D	B	F	5	Check S_2 and S_3 - if S_2 done first, load S_3 - if S_3 done first, unload S_3
17	D	B	D	4	Unload S_3
18	D	B	B	otherwise	Unload S_3
19	B	F	F	6	Check S_1, S_2, and S_3 - if S_1 done first, load S_2 - if S_2 done first, load S_3 - if S_3 done first, unload S_3
20	B	F	D	4	Unload S_3
21	B	F	B	7	Check S_1 and S_3 - if S_1 done first, load S_2 - if S_3 done first, unload S_3
22	B	D	F	3	Load S_3
23	B	D	D	4	Unload S_3
24	B	D	B	otherwise	Unload S_3
25	B	B	F	5	Check S_2 and S_3 - if S_2 done first, load S_3 - if S_3 done first, unload S_3
26	B	B	D	4	Unload S_3
27	B	B	B	otherwise	Unload S_3

Table 12. Status of a three-stage tool using the pull dispatching rule.

Case	Status of stage			Priority	Resulting Action
	S_1	S_2	S_3		
1	F	F	F	4	Load S_1
2	D	F	F	3	Load S_2
3	B	F	F	6	Check S_1, S_2, and S_3 - if S_3 done first, unload S_3 - if S_2 done first, load S_3 - if S_1 done first, load S_2
4	F	D	F	2	Load S_3
5	D	D	F		
6	B	D	F		
7	F	B	F	4	Load S_1
8	D	B	F	5	Check S_2 and S_3 - if S_3 done first, unload S_3 - if S_2 done first, load S_3
9	B	B	F		
10	F	F	D	1	Unload S_3
11	D	F	D		
12	B	F	D		
13	F	D	D		
14	D	D	D		
15	B	D	D		
16	F	B	D		
17	D	B	D		
18	B	B	D		
19	F	F	B	4	Load S_1
20	D	F	B	3	Load S_2
21	B	F	B	7	Check S_1, and S_3 - if S_3 done first, unload S_3 - if S_1 done first, load S_2
22	F	D	B	4	Load S_1
23	D	D	B	otherwise	Unload S_3
24	B	D	B		
25	F	B	B	4	Load S_1
26	D	B	B	otherwise	Unload S_3
27	B	B	B		



Note: (*) Use Table 9, 10, 11, or 12 to select $R_{k,j}$

Figure 15. Flow chart for program P.

6.2.4 Cluster tool models using the prespecified sequence of wafer handler moves

The cluster tool model using the prespecified sequence of wafer handler moves is the simplest model. The prespecified sequence includes of two parts: the first part is the list of starting and destination of moves in the filling-up phase, the second is the list of that in a steady state cycle. These two parts are separated by a signal phrase “begin loop”. For example, the text file containing the prespecifed sequence for a CT1-2-2 is follows.

```
LL S1
S1 S2
LL S1
S1 S2
LL S1
S2 S3
S1 S2
LL S1
S2 S3
S1 S2
begin loop
LL S1
S3 LL
S2 S3
S1 S2
```

Figure 16 shows the flow chart for Algorithm prespecified sequence (PS). Example 5 in Appendix F presents a main portion of the Java codes for a three-stage cluster tool model using the prespecified sequence of wafer handler moves. After creating all moves in the filling-up phase, method get_fixed_sequence_for_3S_tool() repeats the moves of the cycle until all wafers are completed.

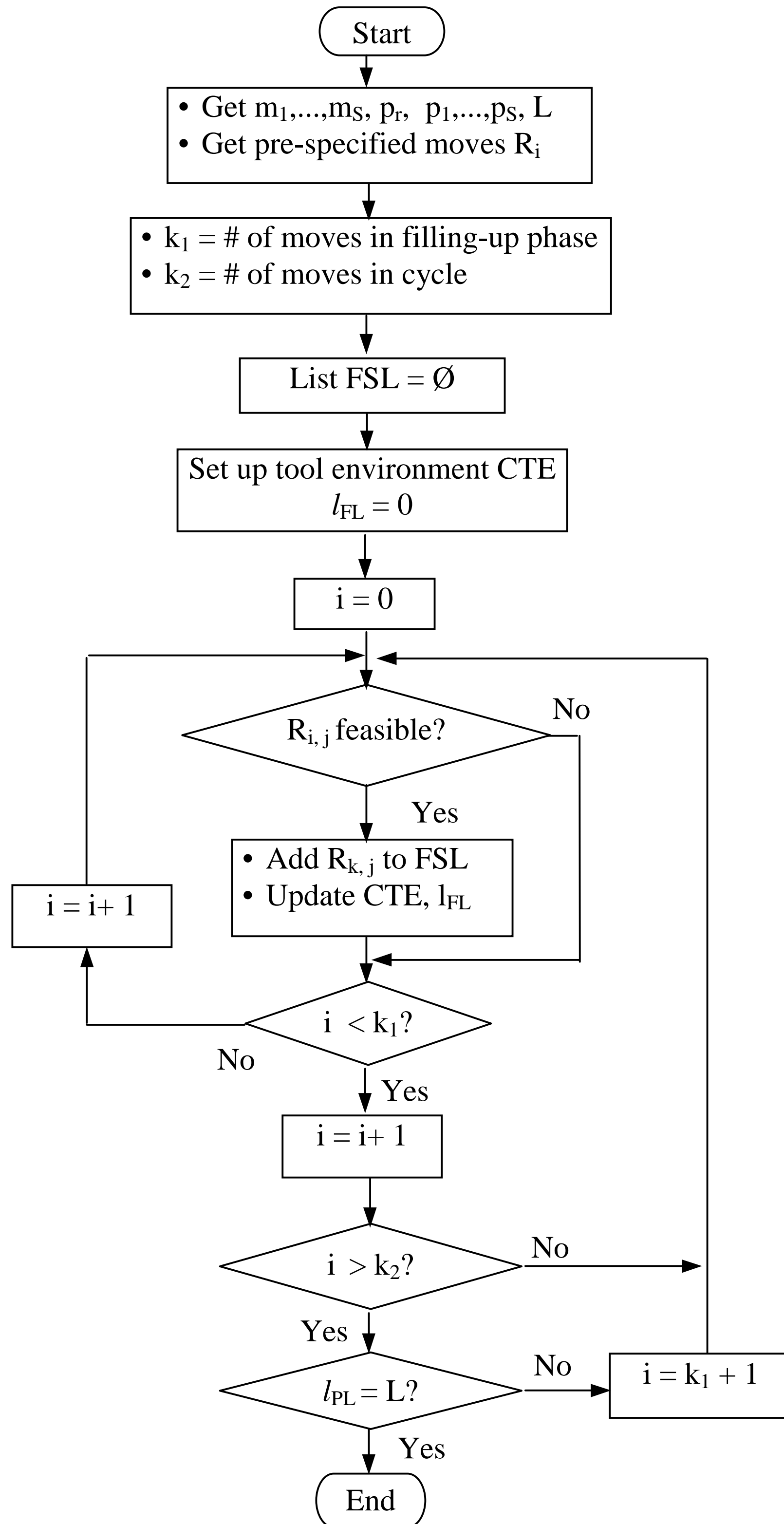


Figure 16. Flow chart for program PS.

6.3 The graphic user interface for the two- and three-stage cluster tool models

After building the simulation models using the prescribed sequence, the dispatching rules, the BB and TBB algorithms, we created the GUI (graphic user interface) for the two- and three-stage cluster tool models. Both GUIs are written using Delphi and compiled into executables. They will work as stand-alone decision-support tools on any personal computer (running Microsoft Windows). The system requirement to execute these GUIs includes the Delphi 4 package and Java Runtime (version 1.2.2). Figure 13 presents the interaction between GUI and the cluster tool models for either two- or three-stage cluster tool. Figures 14 and 15 show the GUI for the two-stage and three-stage cluster tools. The information passed between the GUI and the models are in the format of Strings.

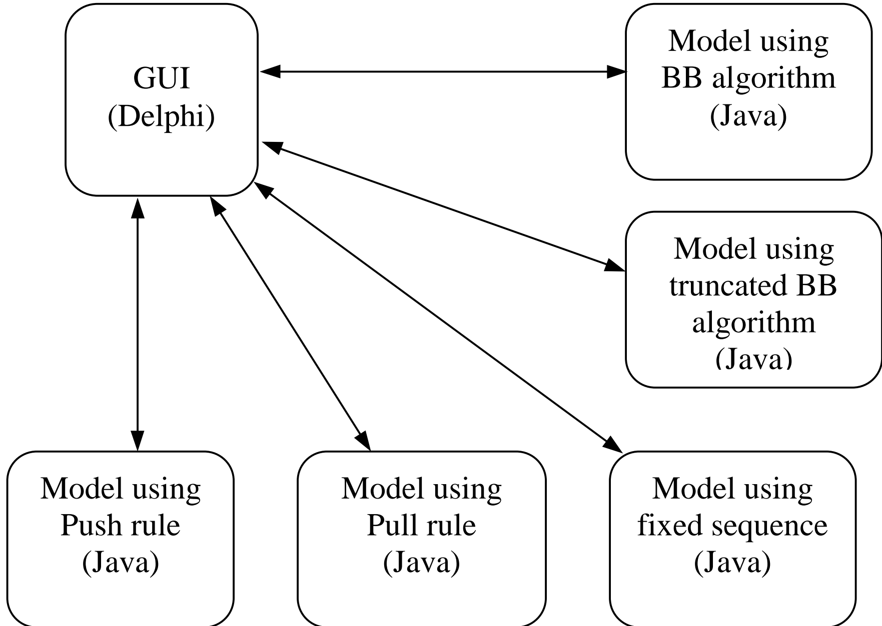


Figure 17. Interaction of GUI and cluster tool models.

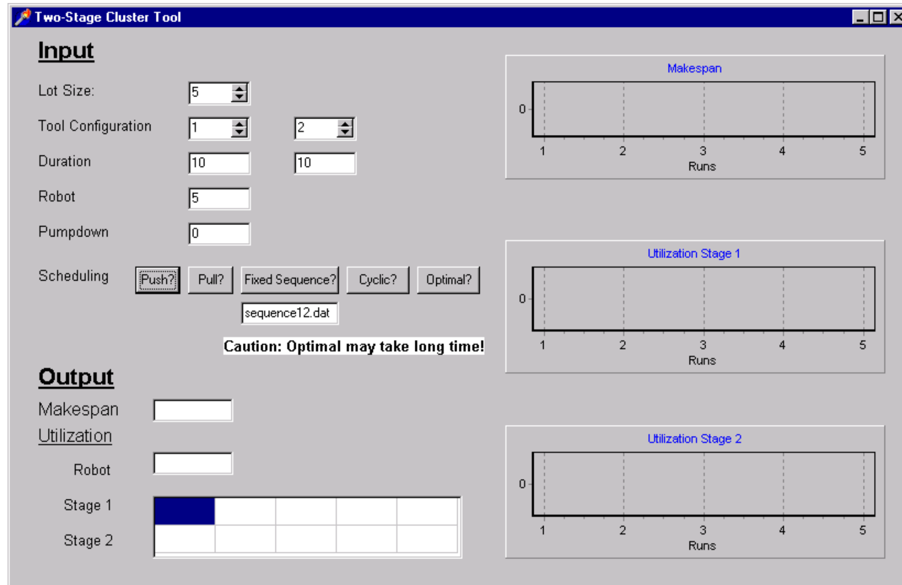


Figure 18. GUI for two-stage cluster tools.

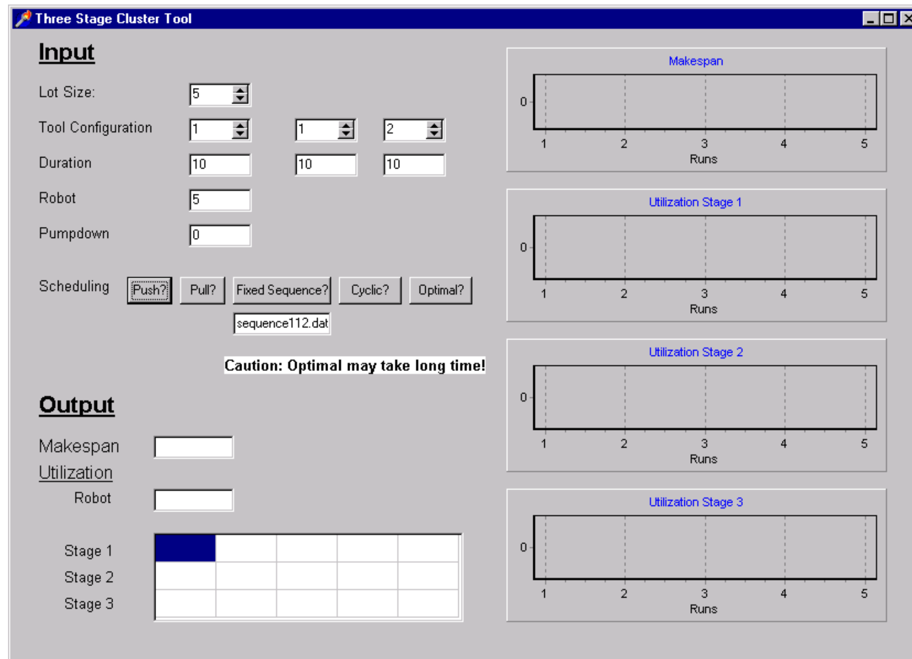


Figure 19. GUI for three-stage cluster tools.

From GUI to cluster tool models, the Strings include (2S + 3) numbers (L, m_1,..., m_S, p_r, p_1,..., p_S, OH If the fixed sequence model is used, the file name of the pre-specified sequence must also be appended to the String. For instance, if the user clicks the "Cyclic?" button in the default window of the two-stage cluster tool GUI, a String "20 1 2 5 10 30 400" will pass to the cluster tool model that uses Algorithm TBB. This model will generate a cyclic sequence and then determine the makespan of a CT1-2, processing 20 wafers per lot, with $p_r = 5$, $p_1 = 10$, $p_2 = 30$, and OH = 400.

After generating the list of all activities, the cluster tool model determines the lot makespan and utilization of the wafer handler and every chamber in the tool. The selected cluster tool model will return to the GUI a String of ($m_1 + \ldots + m_S + 2$) numbers separated by white spaces. These numbers are the lot makespan, utilization of the wafer handler, utilization of every chamber in the first to the last stage, which are then appropriately displayed by the GUI. The GUI also presents the histogram of the runs. For example, Figure 16 displays the GUI for two-stage cluster tool after three runs using push, pull, and cyclic scheduling algorithms. Similarly, Figure 17 shows the GUI for three-stage cluster tool after four runs using different scheduling algorithms: push, pull, fixed-sequence, and cyclic. In each run, the user uses the same lot size, tool configuration, move time, in-chamber processing times, and overhead time.

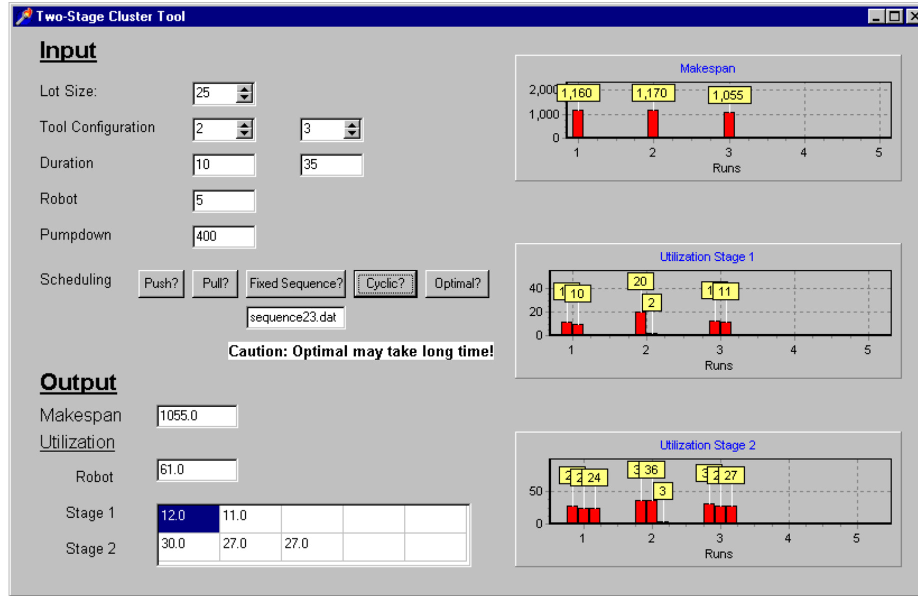


Figure 20. GUI for two-stage cluster tools after 3 runs.

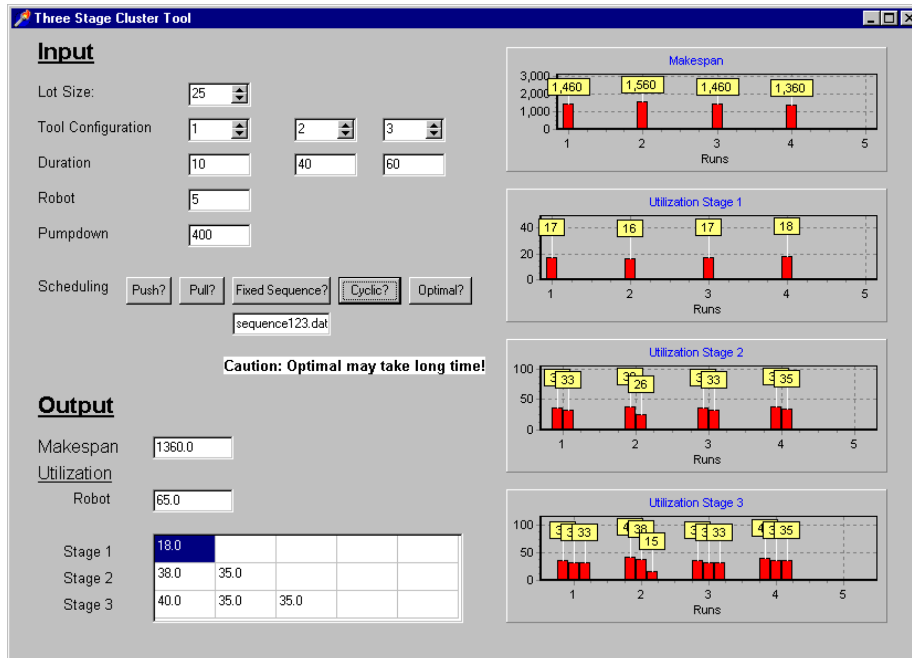


Figure 21. GUI for three-stage cluster tools after 4 runs.

6.4. Integration into subfactory model

The cluster tool models has also been integrated successfully into a subfactory model. This subfactory model integrated operational and process models in a single platform, which allows us to investigate co-optimization of scheduling and process parameters.

In the subfactory models, the cluster tool models take the same input (a String) as that of the stand-alone GUI presented in the previous section. However, the lot makespan is the only required output from the cluster tools models. From GUI of the subfactory model, the user input process parameters (temperature, pressure, and other relevant parameters), lot size, wafer handler move time, overhead time, configuration for each tool, and the preferred scheduling algorithm.

Through a process model, the raw process time is determined from process parameters. The raw process time as well as lot size, wafer handler move time, overhead time, and tool configuration is then fed to a preferred tool simulator to calculates the lot makespan. This lot makespan is then the input to a factory operations simulator. The user is then presented with the factory performance measures, such as work-in-process inventory (WIP) and cycle time, for the process case chosen. Communication between these different models is accomplished by an administrator, which also provides the user interface. Figure 22 summarizes the information flow in the integrated subfactory model.

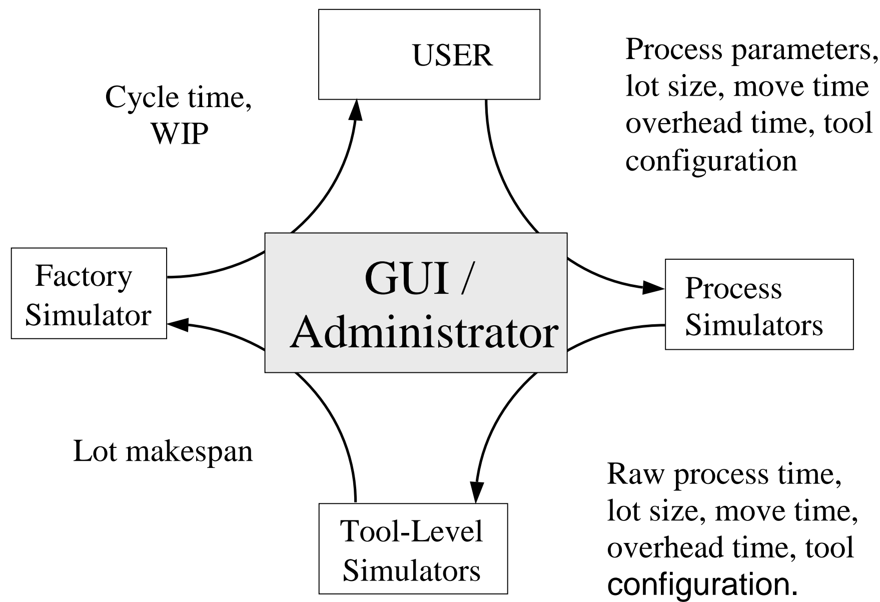


Figure 22. Information flow in the integrated subfactory model.

6.5 Summary

This chapter presented the cluster tool models using Algorithm BB, Algorithm TBB, the push and pull dispatching rules, and the prespecifed sequence of wafer handler moves. The methodology to construct Java codes for these models are also introduced. Examples of the codes are shown in Appendix F.

After building the simulation models, methodology to construct GUIs for the two- and three-stage cluster tool models are presented. Both GUIs are written using Delphi and compiled into executables that worked as stand-alone decision-support tools on any personal computer (running Microsoft Windows), which allow the user to access to all cluster tool models of the same type (two- or three-stage). From GUIs, one can

compute the lot makespan and chamber utilization for any tool configuration, move time, in-chamber processing times, overhead time, and lot size.

The integration of cluster tool models into a subfactory model allows one to investigate co-optimization of scheduling and process parameters. Typically this means revisiting scheduling algorithms and estimating changes in cycle time when process changes are demanded. In some cases this could also mean identifying situations where significant operational improvements could result from changing process parameters without sacrificing quality.

CHAPTER 7

RESULTS AND DISCUSSION

In this chapter, we compare the performance of the optimization models (the BB and TBB algorithms) on two main performance criteria (the lot makespan and CPU time,) to that of the simulation models using the push and pull dispatching rules.

7.1 Performance of the search using Algorithm BB

We tested Algorithm BB on 72 problem sets of two- and three-stage tool configurations. Each problem set included ten randomly generated instances. See Tables 13 and 14 for the parameters used to generate the instances. The problem sets were chosen to determine how the tool configuration and processing time affect algorithm performance. The problem sets cover three cases: short move time and long processing times, approximately equal move and processing times, and a long move time and short processing times. For each instance, we used the branch-and-bound algorithm (Algorithm BB), the push dispatching rule (Algorithm P), and the pull dispatching rule (Algorithm P) to find solutions. On all instances, Algorithm BB halted if it reached 100,000 nodes and reported the best solution found.

Table 15 and Table16 report the performance of Algorithm BB. Table 15 and 16 present, for each problem set, the average CPU time that the branch-and-bound search required, the number of instances that Algorithm BB can completely solved within

100,000 nodes. The fourth, fifth, and sixth columns show the average lot makespan achieved using the branch-and-bound algorithm, the push dispatching rule, and the pull dispatching rule respectively. The last two columns of these tables show the average percent improvement from the push and pull sequences to the best sequence found.

Table 13. Two-stage problem sets.

Problem Sets	Tool Configuration	Associated Lot Size L	Range for move time p_r (s)	Range for processing times p_1, p_2 (s)
1, 2, and 3	CT1-1	5, 10, and 15	[1, 10]	[20,40]
4, 5, and 6	CT1-2	5, 10, and 15	[1, 10]	[20,40]
7, 8, and 9	CT2-1	5, 10, and 15	[1, 10]	[20,40]
10, 11, and 12	CT2-2	5, 10, and 15	[1, 10]	[20,40]
13, 14, and 15	CT1-1	5, 10, and 15	[10, 20]	[10, 20]
16, 17, and 18	CT1-2	5, 10, and 15	[10, 20]	[10, 20]
19, 20, and 21	CT2-1	5, 10, and 15	[10, 20]	[10, 20]
22, 23, and 24	CT2-2	5, 10, and 15	[10, 20]	[10, 20]
25, 26, and 27	CT1-1	5, 10, and 15	[20, 40]	[1, 10]
28, 29, and 30	CT1-2	5, 10, and 15	[20, 40]	[1, 10]
31, 32, and 33	CT2-1	5, 10, and 15	[20, 40]	[1, 10]
34, 35, and 36	CT2-2	5, 10, and 15	[20, 40]	[1, 10]

Table 14. Three-stage problem sets.

Problem Sets	Tool Configuration	Lot Size L	Range for move time p_r (s)	Range for processing times p_1, p_2, p_3 (s)
37, 38, and 39	CT1-1-1	5, 10, and 15	[1, 10]	[20,40]
40, 41, and 42	CT1-2-2	5, 10, and 15	[1, 10]	[20,40]
43, 44, and 45	CT2-2-1	5, 10, and 15	[1, 10]	[20,40]
46, 47, and 48	CT2-2-2	5, 10, and 15	[1, 10]	[20,40]
49, 50, and 51	CT1-1-1	5, 10, and 15	[10, 20]	[10, 20]
52, 53, and 54	CT1-2-2	5, 10, and 15	[10, 20]	[10, 20]
55, 56, and 57	CT2-2-1	5, 10, and 15	[10, 20]	[10, 20]
58, 59, and 60	CT2-2-2	5, 10, and 15	[10, 20]	[10, 20]
61, 62, and 63	CT1-1-1	5, 10, and 15	[20, 40]	[1, 10]
64, 65, and 66	CT1-2-2	5, 10, and 15	[20, 40]	[1, 10]
67, 68, and 69	CT2-2-1	5, 10, and 15	[20, 40]	[1, 10]
70, 71, and 72	CT2-2-2	5, 10, and 15	[20, 40]	[1, 10]

Table 15. Computing time and percentage improvement of Algorithm BB over the push and pull sequences for some two-stage cluster tool configurations (OH = 0).

Tool configuration	L	CPU time min:sec	# instances solved	Average makespan (sec)			% improvement	
				BB	push	pull	push	pull
Case 1: short moves, long processing times.								
CT1-1	5	0:01	10/10	303.1	303.1	303.1	0	0
	10	0:02	10/10	585.6	585.6	585.6	0	0
	15	0:28	10/10	868.1	868.1	868.1	0	0
CT1-2	5	0:01	10/10	280.9	294.1	294.1	4.18	4.18
	10	0:46	7/10	536.2	564.6	564.6	4.65	4.65
	15	1:13	0/10	789.3	835.1	835.1	5.08	5.08
CT2-1	5	0:01	10/10	298.3	306.1	306.1	2.29	2.29
	10	0:57	7/10	575.9	589.6	589.6	2.12	2.12
	15	1:32	0/10	859.4	873.1	873.1	1.43	1.43
CT2-2	5	0:01	10/10	216	221.7	231.7	2.13	6.43
	10	0:51	4/10	378.2	401.5	406.9	4.79	6.21
	15	1:31	0/10	559.7	588.7	609.4	4.32	8.02
Case 2: move time and processing times are approximately equal.								
CT1-1	5	0:01	10/10	372.7	420.3	420.3	10.33	10.33
	10	0:02	10/10	745.2	852.3	852.3	11.41	11.41
	15	1:13	0/10	1117.7	1284.3	1284.3	11.76	11.76
CT1-2	5	0:01	10/10	401.3	468	468	13.64	13.64
	10	1:16	0/10	811	951	951	14.19	14.19
	15	1:36	0/10	1294	1434	1434	9.41	9.41
CT2-1	5	0:01	10/10	383	451.2	451.2	14.71	14.71
	10	1:16	0/10	775.6	919.2	919.2	15.31	15.31
	15	1:27	0/10	1134.4	1387.2	1387.2	18.93	18.93
CT2-2	5	0:03	10/10	382	446.6	419.8	13.84	8.52
	10	1:20	0/10	818.9	908.6	847	9.73	3.17
	15	1:39	0/10	1263.8	1370.6	1263.8	7.72	0.00
Case 3: long move time, short processing times.								
CT1-1	5	0:01	10/10	510.5	812.5	812.5	37.15	37.15
	10	0:02	10/10	1021	1700.5	1700.5	39.94	39.94
	15	0:22	10/10	1531.5	2588.5	2588.5	40.81	40.81
CT1-2	5	0:01	10/10	478.5	767.7	767.7	37.59	37.59
	10	0:44	10/10	957	1607.7	1607.7	40.38	40.38
	15	1:34	0/10	1803.7	2447.7	2447.7	26.23	26.23
CT2-1	5	0:01	10/10	547	897.7	897.7	38.87	38.87
	10	0:55	9/10	1098.9	1884.7	1884.7	41.34	41.34
	15	1:33	0/10	2083.6	2871.7	2871.7	27.29	27.29
CT2-2	5	0:02	10/10	471.5	794.6	697.1	40.13	31.87
	10	1:15	0/10	1210	1616.6	1507	24.71	19.23
	15	1:35	0/10	2032	2438.6	2204.1	16.38	7.52

Table 16. Computing times and percentage improvement of Algorithm BB over the push and pull sequences for some three-stage cluster tool configurations (OH = 0).

Tool configuration	L	CPU time min:sec	# instances solved	Average makespan (sec)			% improvement	
				BB	push	pull	push	pull
	Case 1: short moves, long processing times.							
CT1-1-1	5	0:01	10/10	369	369	371.1	0	0.44
	10	1:04	5/10	677.5	677.5	683.1	0	0.62
	15	1:44	0/10	986	986	995.1	0	0.68
CT1-2-2	5	0:17	10/10	334.9	349.9	350.8	3.97	4.32
	10	1:34	0/10	613.5	638.4	641.1	3.64	4.24
	15	2:00	0/10	889.4	926.9	930.5	3.81	4.38
CT2-2-1	5	0:34	7/10	349.2	359.6	356	2.58	1.85
	10	1:38	0/10	638.8	644.6	641.6	0.78	0.51
	15	2:05	0/10	923.8	929.6	926.6	0.54	0.35
CT2-2-2	5	0:49	6/10	290.1	296.1	318.1	1.73	8.52
	10	1:42	0/10	549.3	563.9	578.6	2.68	4.67
	15	2:10	0/10	817.3	831.9	839.5	1.83	2.72
	Case 2: move time and processing times are approximately equal.							
CT1-1-1	5	0:03	10/10	521.7	591.8	591.8	11.10	11.21
	10	1:37	0/10	1083	1195.8	1195.8	9.07	9.07
	15	2:10	0/10	1687	1799.8	1799.8	6.03	6.06
CT1-2-2	5	1:13	0/10	515.6	573.7	577.5	9.63	10.52
	10	1:43	0/10	1107.9	1161.7	1164.5	4.42	4.74
	15	2:15	0/10	1695.9	1749.7	1753.5	2.93	3.25
CT2-2-1	5	1:14	0/10	537.3	600.6	600.6	10.27	10.21
	10	1:50	0/10	1158	1216.6	1216	4.71	4.64
	15	2:19	0/10	1774	1832.6	1832.6	3.12	3.11
CT2-2-2	5	1:12	0/10	534.4	577.2	549.9	7.11	2.80
	10	1:40	0/10	1027.5	1169.2	1110	14.56	10.00
	15	1:57	0/10	1535.2	1630.3	1535.2	5.18	0.00
	Case 3: long move, short processing times.							
CT1-1-1	5	0:01	10/10	670	1095.4	1071.6	38.63	37.29
	10	1:28	0/10	1585.6	2267.4	2267.4	29.84	29.84
	15	1:54	0/10	2757.6	3439.4	3415.6	19.67	19.12
CT1-2-2	5	0:35	10/10	694.5	1152.4	1105.3	39.28	36.75
	10	1:40	0/10	1984.7	2360.4	2337.1	15.70	14.86
	15	2:06	0/10	3192.7	3568.4	3521.3	10.38	9.21
CT2-2-1	5	0:50	10/10	650	1047.2	1069.6	37.53	38.81
	10	1:40	0/10	1836.3	2167.2	2167.4	15.02	15.04
	15	2:09	0/10	2961.7	3287.2	3309.6	9.77	10.37
CT2-2-2	5	1:09	0/10	780	1092	968.2	28.19	19.08
	10	1:44	0/10	1946.2	2212	2100	11.85	7.15
	15	2:11	0/10	3049.8	3332	3066.4	8.41	0.51

To test the performance of Algorithm BB with additional search time and with more sophisticated lower bounds, we used problem sets 52, 53, and 54: CT1-2-2 and the move and processing times are approximately equal. Table 17 compares the search performances for these problems in three cases: A) the search halted at 100,000 nodes, B) the search halted at 1,000,000 nodes, and C) the search halted at 100,000 nodes and used better lower bounds.

From Tables 15, 16, and 17 we can draw some insights as follows.

- Except for CT1-1 and CT1-1-1 in Case 1, the search was able to find better sequences. This improvement was greater when the processing times are approximately the same as or smaller than the move times. This is true even though the optimal sequence may not satisfy the permutation constraint when the processing times are small.
- The computing effort increases as the lot size, the number of chambers in each stage, and the number of stages increase.
- Conducting longer searches or using more sophisticated lower bounds did not improve the search performance significantly.

Table 17. Computing time and lot makespan of a CT1-2-2 when increasing number of nodes and applying better lower bounds.

L	pr	p1	p2	p3	if # nodes = 10^5			if # nodes = 10^6			if LBs and # nodes = 10^5		
					CPU time min:sec	solved?	MS (sec)	CPU time min:sec	solved?	MS (sec)	CPU time min:sec	solved?	MS (sec)
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)
5					1:18	*	497	1:41		480	1:11		480
10	15	11	14	11	1:42	*	1105	17:00	*	1086	3:11	*	1101
15					2:11	*	1705	22:00	*	1686	4:26	*	1701
5					1:12	*	396	1:43		396	1:25		396
10	11	11	17	19	1:43	*	836	17:11	*	836	2:50	*	836
15					2:20	*	1276	22:02	*	1276	3:57	*	1276
5					1:10	*	638	1:47		615	1:38		615
10	19	13	17	17	1:43	*	1406	16:48	*	1389	3:14	*	1404
15					2:22	*	2166	21:44	*	2149	4:23	*	2164
5					1:18	*	364	1:44		364	1:37		364
10	10	12	13	18	1:42	*	764	16:49	*	754	2:54	*	764
15					2:20	*	1164	21:58	*	1154	4:28	*	1164
5					1:16	*	635	1:43		605	1:20		605
10	19	16	11	18	1:45	*	1404	16:54	*	1384	2:56	*	1401
15					2:22	*	2164	21:55	*	2144	4:35	*	2161
5					1:13	*	442	1:45		442	1:18		442
10	12	20	14	19	1:45	*	928	16:50	*	916	3:08	*	928
15					2:22	*	1408	21:47	*	1396	4:31	*	1408
5					1:11	*	690	1:57		670	1:51	*	690
10	20	15	20	19	1:48	*	1496	17:00	*	1489	3:13	*	1495
15					2:10	*	2296	21:46	*	2289	4:00	*	2295
5					1:11	*	435	1:51		435	1:50		435
10	12	14	13	20	1:40	*	916	17:11	*	916	3:11	*	916
15					2:10	*	1396	22:22	*	1396	4:01	*	1396
5					1:11	*	580	1:58		580	1:47	*	580
10	16	20	18	16	1:43	*	1224	16:59	*	1224	3:13	*	1224
15					2:10	*	1864	21:45	*	1864	4:01	*	1864
5					1:10	*	479	1:45		479	1:29		479
10	13	19	20	18	1:42	*	1000	16:59	*	987	3:13	*	1000
15					2:09	*	1520	21:44	*	1507	4:00	*	1520

Note: * means the problem is not completely solved within the number of nodes specified.

7.2 Performance of the search using Algorithm TBB

We tested Algorithm TBB on the same 72 problem sets of two- and three-stage tool configurations as in the previous section. Each problem set included ten randomly generated instances. See Tables 13 and 14 for the parameters used to generate the instances. TBB halted if it reached 50,000 nodes.

Table 18 and Table 19 report the performance of Algorithm BB. In Tables 18 and 19, for each problem set, the third and fourth columns present the average CPU times that Algorithm BB and Algorithm TBB required. The fifth and sixth columns are the number of instances that Algorithm BB completely solved and Algorithm TBB completely solved. The seventh and eighth columns show the average lot makespan achieved using Algorithms BB and TBB respectively. The last four columns of these tables show the average percent improvement from the push and pull sequences to the best sequence found. Note that the results from Table 15 and 16 about the performance of Algorithm BB are repeated here to enable comparison between the Algorithm BB and Algorithm TBB.

Table 18. Computing times and percentage improvement of Algorithm TBB over the push and pull sequences for some two-stage cluster tool configurations (OH = 0).

Tool configuration	L	CPU time min:sec		# instances solved		Average makespan (sec)		% improvement over push		% improvement over pull	
		BB	TBB	BB	TBB	BB	TBB	BB	TBB	BB	TBB
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
	Case 1: short move time, long processing times.										
CT1-1	5	0:01	0:01	10/10	10/10	303.1	303.1	0	0	0	0
	10	0:02	0:01	10/10	10/10	585.6	585.6	0	0	0	0
	15	0:28	0:01	10/10	10/10	868.1	868.1	0	0	0	0
CT1-2	5	0:01	0:01	10/10	10/10	280.9	285.3	4.49	2.99	4.49	2.99
	10	0:46	0:01	7/10	10/10	536.2	546.6	5.03	3.19	5.03	3.19
	15	1:23	0:01	0/10	10/10	789.3	807.1	5.48	3.35	5.48	3.35
CT2-1	5	0:01	0:01	10/10	10/10	298.3	302.4	2.55	1.21	2.55	1.21
	10	0:57	0:01	7/10	10/10	575.9	585.7	2.32	0.66	2.32	0.66
	15	1:32	0:01	0/10	10/10	859.4	869.2	1.57	0.45	1.57	0.45
CT2-2	5	0:01	0:01	10/10	10/10	216	216.3	2.57	2.44	6.78	6.65
	10	0:51	0:01	4/10	10/10	378.2	377.1	5.80	6.08	7.05	7.32
	15	1:31	0:01	0/10	10/10	559.7	543.3	4.93	7.71	8.16	10.85
	Case 2: move time and processing times are approximately equal.										
CT1-1	5	0:01	0:01	10/10	10/10	372.7	372.7	11.33	11.33	11.33	11.33
	10	0:02	0:01	10/10	10/10	745.2	745.2	12.57	12.57	12.57	12.57
	15	1:13	0:01	0/10	10/10	1117.7	1131.7	12.97	11.88	12.97	11.88
CT1-2	5	0:01	0:01	10/10	10/10	401.3	401.3	14.25	14.25	14.25	14.25
	10	1:16	0:01	0/10	10/10	811	798.3	14.72	16.06	14.72	16.06
	15	1:36	0:01	0/10	10/10	1294	1194.8	9.76	16.68	9.76	16.68

Table 18 continued.

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
CT2-1	5	0:01	0:01	10/10	10/10	383	383	15.12	15.12	15.12	15.12
	10	1:16	0:01	0/10	10/10	775.6	761.2	15.62	17.19	15.62	17.19
	15	1:27	0:01	0/10	10/10	1134.4	1138.2	18.22	17.95	18.22	17.95
CT2-2	5	0:03	0:02	10/10	10/10	382	382	14.46	14.46	9.00	9.00
	10	1:20	0:02	0/10	10/10	818.9	753.7	9.87	17.05	3.32	11.02
	15	1:39	0:03	0/10	10/10	1263.8	1126.6	7.79	17.80	0.00	10.86
	Case 3: long move time, short processing times.										
CT1-1	5	0:01	0:01	10/10	10/10	510.5	510.5	37.17	37.17	37.17	37.17
	10	0:02	0:01	10/10	10/10	1021	1021	39.96	39.96	39.96	39.96
	15	0:22	0:01	10/10	10/10	1531.5	1531.5	40.83	40.83	40.83	40.83
CT1-2	5	0:01	0:01	10/10	10/10	478.5	478.5	37.67	37.67	37.67	37.67
	10	0:44	0:01	10/10	10/10	957	957	40.47	40.47	40.47	40.47
	15	1:34	0:01	0/10	10/10	1803.7	1386	26.31	43.38	26.31	43.38
CT2-1	5	0:01	0:01	10/10	10/10	547	547	39.07	39.07	39.07	39.07
	10	0:55	0:01	9/10	10/10	1098.9	1094	41.69	41.95	41.69	41.95
	15	1:33	0:01	0/10	10/10	2083.6	1641	27.44	42.86	27.44	42.86
CT2-2	5	0:02	0:02	10/10	10/10	471.5	471.5	40.66	40.66	32.36	32.36
	10	1:15	0:03	0/10	10/10	1210	943	25.15	41.67	19.71	37.43
	15	1:35	0:04	0/10	10/10	2032	1414.5	16.67	42.00	7.81	35.82

Table 19. Computing times and percentage improvement of the Algorithm TBB over the push and pull sequences for some three-stage cluster tool configurations (OH = 0).

Tool configuration	L	CPU time min:sec		# instances solved		Average makespan (sec)		% improvement over push		% improvement over pull	
		BB	TBB	BB	TBB	BB	TBB	BB	TBB	BB	TBB
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
	Case 1: short move time, long processing times.										
CT1-1-1	5	0:01	0:01	10/10	10/10	369	369	0	0	0.57	0.57
	10	1:04	0:01	5/10	10/10	677.5	677.5	0	0	0.82	0.82
	15	1:44	0:01	0/10	10/10	986	986	0	0	0.91	0.91
CT1-2-2	5	0:17	0:02	10/10	10/10	334.9	337.8	4.29	3.46	4.53	3.71
	10	1:34	0:03	0/10	10/10	613.5	616.8	3.90	3.38	4.31	3.79
	15	2:00	0:04	0/10	10/10	889.4	895.4	4.05	3.40	4.42	3.77
CT2-2-1	5	0:34	0:03	7/10	10/10	349.2	350.3	2.89	2.59	1.91	1.60
	10	1:38	0:05	0/10	10/10	638.8	634.1	0.90	1.63	0.44	1.17
	15	2:05	0:07	0/10	10/10	923.8	918.1	0.62	1.24	0.30	0.92
CT2-2-2	5	0:49	0:46	6/10	5/10	290.1	294	2.03	0.71	8.80	7.58
	10	1:42	1:27	0/10	0/10	549.6	524.9	2.54	6.92	5.01	9.28
	15	2:10	1:44	0/10	0/10	817.3	760.3	1.76	8.61	2.64	9.43
	Case 2: move time and processing times are approximately equal.										
CT1-1-1	5	0:03	0:01	10/10	10/10	521.7	521.7	11.85	11.85	11.85	11.85
	10	1:37	0:01	0/10	10/10	1083	1032.2	9.43	13.68	9.43	13.68
	15	2:10	0:01	0/10	10/10	1687	1541.7	6.27	14.34	6.27	14.34
CT1-2-2	5	1:13	0:04	0/10	10/10	515.6	509.2	10.13	11.24	10.72	11.83
	10	1:43	0:06	0/10	10/10	1107.9	1005.2	4.63	13.47	4.86	13.68
	15	2:15	0:08	0/10	10/10	1695.9	1501.2	3.07	14.20	3.28	14.39

Table 19 continued

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
CT2-2-1	5	1:14	0:06	0/10	10/10	539.3	525.2	10.21	12.55	10.21	12.55
	10	1:50	0:09	0/10	10/10	1158	1044.5	4.82	14.15	4.77	14.10
	15	2:19	0:11	0/10	10/10	1774	1446.5	3.20	21.07	3.20	21.07
CT2-2-2	5	1:12	1:17	0/10	0/10	534.4	537.9	7.42	6.81	2.82	2.18
	10	1:51	1:25	0/10	0/10	1110	1077.8	5.06	7.82	0.00	2.90
	15	2:10	1:42	0/10	0/10	1659.9	1595.8	5.75	9.39	0.00	3.86
Case 3: long move time, short processing times											
CT1-1-1	5	0:01	0:01	10/10	10/10	670	670	38.84	38.84	37.48	37.48
	10	1:28	0:01	0/10	10/10	1585.6	1340	30.07	40.90	30.07	40.90
	15	1:54	0:01	0/10	10/10	2757.6	2010	19.82	41.56	19.26	41.15
CT1-2-2	5	0:35	0:04	10/10	10/10	694.5	694.5	39.73	39.73	37.17	37.17
	10	1:40	0:06	0/10	10/10	1984.7	1389	15.92	41.15	15.08	40.57
	15	2:06	0:08	0/10	10/10	3192.7	2083.5	10.53	41.61	9.33	40.83
CT2-2-1	5	0:50	0:06	10/10	10/10	650	650	37.93	37.93	39.23	39.23
	10	1:40	0:10	0/10	10/10	1836.3	1300	15.27	40.01	15.28	40.02
	15	2:09	0:12	0/10	10/10	2961.7	1950	9.90	40.68	10.51	41.08
CT2-2-2	5	1:09	1:15	0/10	0/10	780	883.3	28.57	19.11	19.44	8.77
	10	1:44	1:25	0/10	0/10	1946.2	1903.2	12.02	13.96	7.32	9.37
	15	2:11	1:42	0/10	0/10	3049.8	2819.3	8.47	15.39	0.54	8.06

From Table 18 and 19, we can draw some insights as follows.

- Except for CT1-1 and CT1-1-1 when move time is short comparing to processing times (Case 1), the searches were able to find better sequences. This improvement was greater when the processing times are approximately the same as or smaller than the move times.
- If the number of searching nodes is less than 100,000, Algorithm TBB can find better solutions than Algorithm BB when the lot size is a large number in most instances. In other words, the computing effort is significantly reduced and becomes less sensitive with respect to the lot size L, when Algorithm TBB is used. However, Algorithm BB is able to find better results than the Algorithm TBB in Case 1.

7. 3 Summary

This chapter compares the performance of Algorithm BB and Algorithm TBB on two criteria, lot makespan and CPU time, to that of the simulation models using the push and pull dispatching rules. The results suggest that the push and pull dispatching rules only perform well on a small set of problems (1, 2, 3, 37, 38 and 39). That is, the push and pull dispatching rules should be used when the move time is smaller than in-chamber process times and each stage has one chamber. Algorithm BB performs well on two-stage problems when lot size is smaller than 10. Otherwise, Algorithm TBB is the best candidate even though it will not guarantee the optimality of the solutions, because it requires much less computational effort.

CHAPTER 8

SUMMARY AND CONCLUSIONS

This research aims to optimize the total lot processing time (makespan) in cluster tool scheduling for semiconductor manufacturing. Previous research focuses on finding an optimal sequence of wafer handler moves in a cluster tool that has one process chamber in each stage. None of the previous work addresses the problem of finding an optimal sequence of wafer handler moves to improve performance of cluster tools whose number of chambers in each stage can be any integer. Lot makespan is used as the performance measure for the analysis of the cluster tools. The inverse of makespan is an upper bound on the tool throughput; hence, makespan is an important performance measure for the tool in the analysis of the entire system of tools

In this research, the wafer handler sequencing problem are formulated in standard scheduling form. All constraints that a feasible sequence of wafer handler moves must follow are identified. Two cluster tool models have been developed in the research: one model, implementing the forward branch-and-bound algorithm, can find the optimal solution and the other, implementing the truncated branch-and-bound algorithm, can find the best λ-unit cyclic solution. The models attempt to find the optimal or near optimal sequence of wafer handler moves for a given lot size, wafer handler move time, and chamber processing times. This improves cluster tool performance by reducing the total time needed to process the lot. This can reduce cycle time, reduce tool utilization,

and increase tool capacity. However, the research considers only the single-load-lock and single-wafer-handler tools due to the complexity of the problems.

Three dominance criteria are used to reduce the solution space of the problem and improve the search performance. Although Algorithm BB requires additional computational effort, our results show that the tool performance can improve significantly when the wafer handling moves follow the optimal sequence instead of a simple push (or pull) dispatching rule. This is especially true when the move time and processing times are approximately equal and when the move time is longer than processing times. Like other branch-and-bound algorithms, the computational effort is sensitive to the problem size. Increasing the number of chambers in a stage, the number of stages in the tool, or the lot size increases the search space. Because programming the wafer handler can be done off-line, the extra computational effort should be acceptable.

The truncated branch-and-bound algorithm significantly eases computational effort that a complete branch-and-bound algorithm faces, although it does not guarantee the optimality of the solution. Moreover, our results show that the tool performance can still improve significantly when the wafer handling moves follow the cyclic sequence instead of a simple push (or pull) dispatching rule. When the lot size is a large number, Algorithm TBB can find better solutions than Algorithm BB in most instances. The computing effort is significantly improved and becomes less sensitive with respect to the lot size L, if using the Algorithm TBB instead of Algorithm BB.

These models can serve as stand-alone decision support tools that will help the managers select the right tool configuration for given ranges of processing time. They can also be integrated into an integrated simulation model of a semiconductor wafer fab. The integrated simulation will allow engineers to determine how factory performance (such as cycle time) depends upon process parameters and tool configurations.

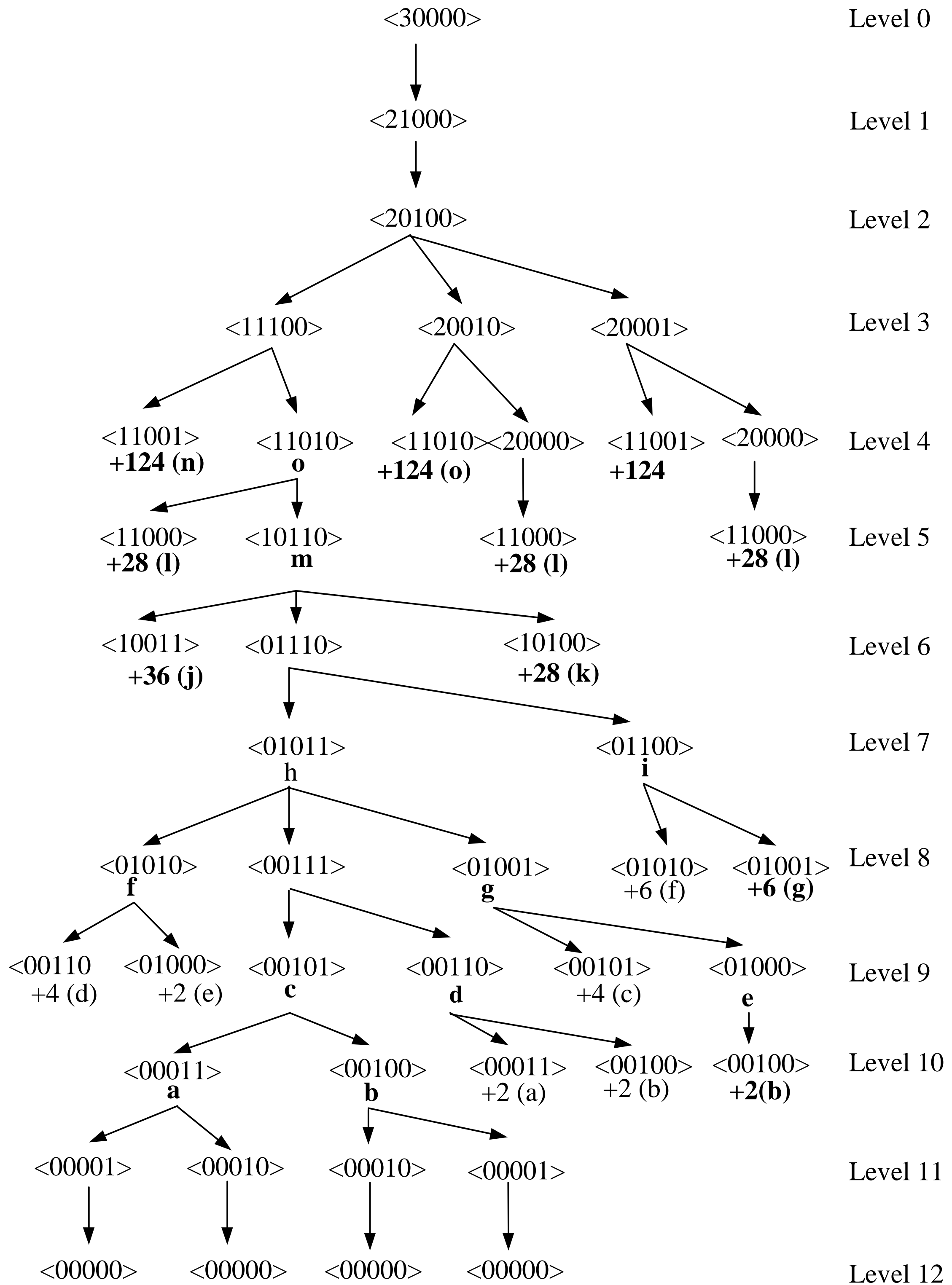
Future work should consider scheduling anticipatory moves, which position the wafer handler at the next chamber before the chamber finishes the wafer. Such anticipatory moves will further improve the cluster tool performance. Other research direction would be a study on the influence of overhead (pumpdown/vent and so on) associated with stop and start of a process in the wafer handler move time. This immediately leads to variability in handler move times.

APPENDIX A

In this Appendix, we will determine the number of feasible sequences of wafer handler moves for a 1-1-2 cluster tool with and without using the first and second dominance criteria.

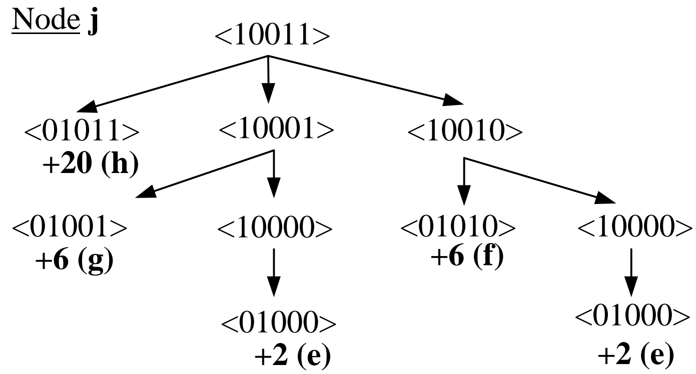
To keep track of the moves easily, define the state variable as <ABCDE>. Where A is the number of unprocessed wafers in LL. B and C are numbers of wafer in the first and second stage respectively. D and E are numbers of wafer in the first and second chamber of the last stage. B, C, C, and D are binary variables with 0 indicated that the chamber is empty and 1 indicated that a wafer occupies the chamber. Thus, the state variable at level zero is <30000> and at the last level is <00000>.

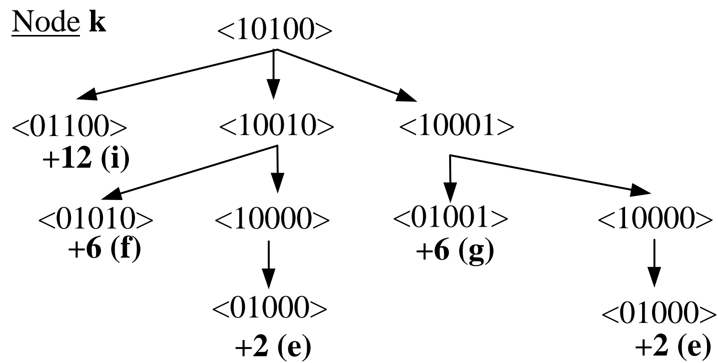
Actually, we do not need to explore all branches. If state variable of a node in a searching branch is the same as one in the previous branches, we can determine the number of branches from the node as the previous cases. For example, at level 10, when node <00011> is repeated to the same state variable as node a, the search is terminated and number of possible sequences is two. If the first and second dominance criteria are not applied, we can find 552 feasible sequences using the graphs in Figure A-1. If applying the first and second dominance criteria, there are 69 feasible sequences (see Figure A-2).

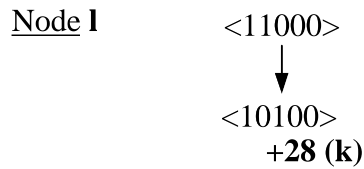


THUS, THERE ARE 552 FEASIBLE

Figure A-1. Complete outtree graph of a CT1-1-2 (L = 3) when the first and second dominance criteria are not active.







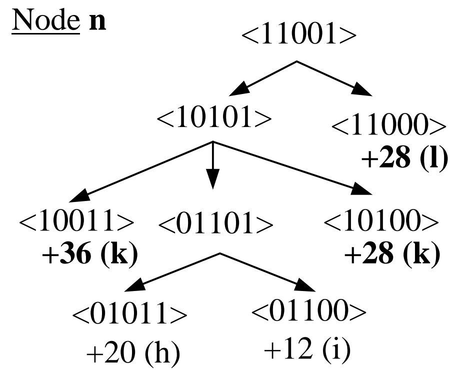
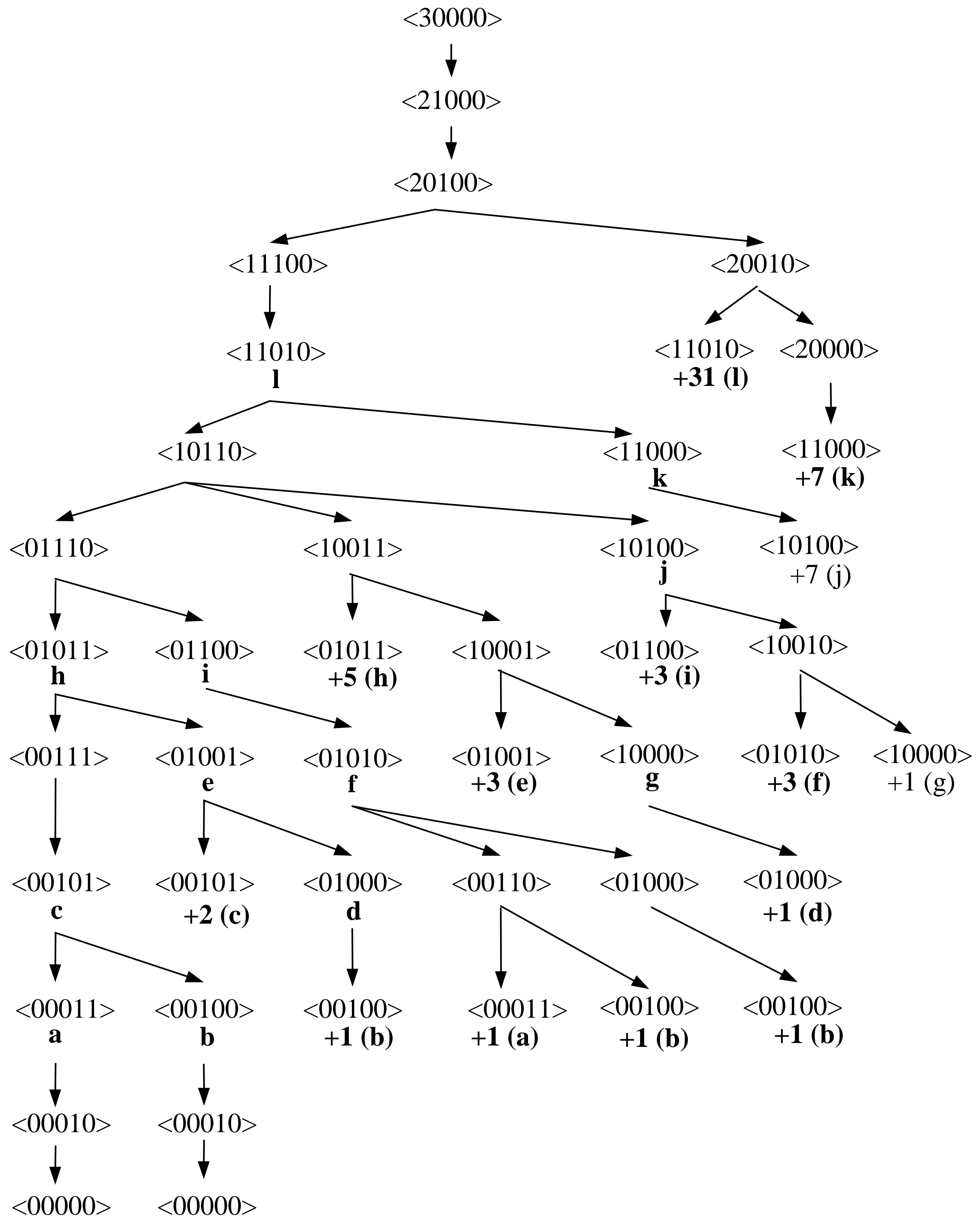


Figure A-1 (cont'd).



Thus, there are 69 feasible sequences.

Figure A-2. Complete outtree graph of a CT1-1-2 (L = 3) when the first and second dominance criteria are active.

APPENDIX B

In this appendix, we formulate the formulas of cycle length and lot makespan for a CT1-1 that follows 1-unit cycle. We will assume that initially the tool is empty and the wafer handler is at load lock.

1) Cycle 1: $R_{2,j-1} - R_{0,j} - R_{1,j} - R_{2,j}$.

Thus, $R_{i,j}$ takes p_r (i = 0, 1, 2). Figure B-1 presents a Gantt chart of a complete σ1-sequence when L = 3.

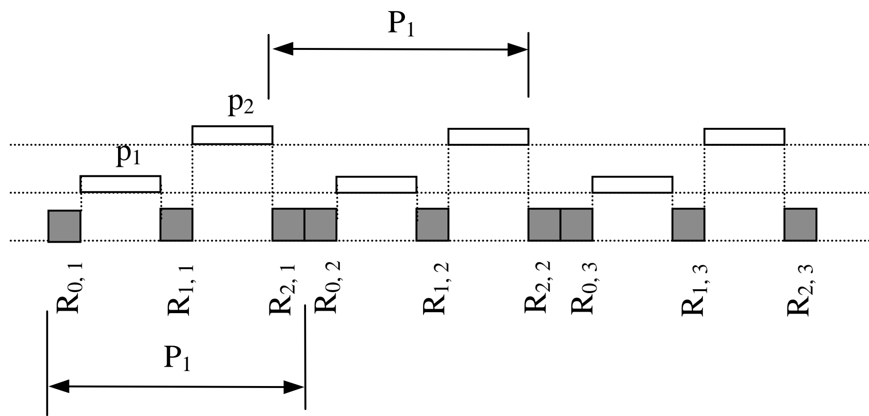


Figure B-1. Gantt chart of a complete σ1-sequence (L = 3).

For convenience, consider the jth cyclic cycle $\sigma 1 = R_{0,j} - R_{1,j} - R_{2,j}$. The complete sequence consists of L occurrences of σ1. Thus, the cycle length $P_1 = 3p_r + p_1 + p_2$, and the makespan $MS_1 = LP_1$.

2) Cycle 2: $R_{2, j-1} - R_{1, j} - R_{0,j+1} - R_{2,j}$. In the cyclic phase, all $R_{i,j}$ are $2p_r$. Figure B-2 presents a Gantt chart of a complete σ2-sequence when L = 4.

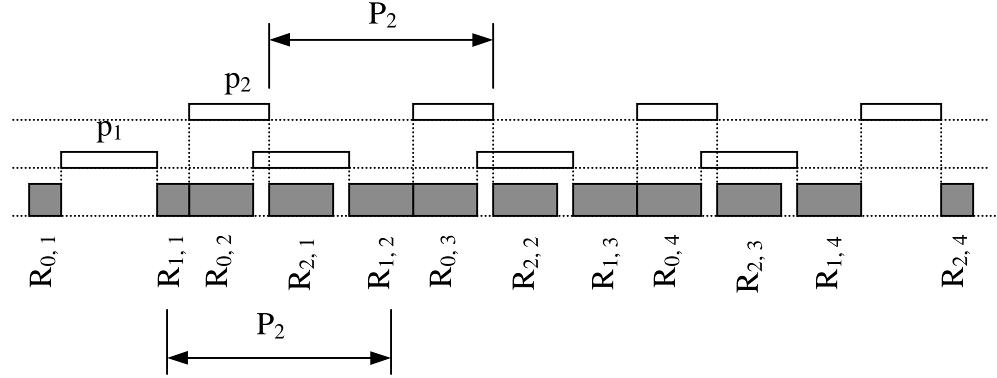


Figure B-2. Gantt chart of a complete σ2-sequence (L = 4).

However, for convenience, we will consider the jth cyclic cycle $\sigma 2 = R_{0, j+1} - R_{2, j} - R_{1, j+1}$, which repeats (L-1) times. Let time t = 0 at the beginning of the sequence. The filling-up phase consists of $R_{0, 1}$ and $R_{1, 1}$. The first cycle begins at $t_1 = 2p_r + p_1$. Let y be the idle time between $R_{0, j+1}$ and $R_{2, j}$, then $y = \max\{p_2, 2p_r\} - 2p_r$. Let x be the idle time between $R_{2, j}$ and $R_{1, j+1}$, then $x = \max\{p_1 - 2p_r - y, 0\} = \max\{p_1, 2p_r + y\} - 2p_r - y$. Thus, the cycle length $P_2 = 6p_r + x + y = 4p_r + \max\{p_1, 2p_r + y\} = 4p_r + \max\{2p_r, p_1, p_2\}$. The completion phase starts at $t_2 = t_1 + (L - 1)P_2 = 2p_r + p_1 + (L-1)P_2$. Note that $R_{2, L}$ takes p_r. Thus, the makespan $MS_2 = 2p_r + p_1 + (L-1)P_2 + p_2 + p_r = 3p_r + p_1 + p_2 + (L-1)P_2$.

APPENDIX C

In this appendix, we derive the formulas for cycle length and lot makespan for a CT1-1-1 that follows 1-unit cycle σx (x = 1, 3, 4, and 5). We will assume that initially the tool is empty and the wafer handler is at load lock.

1) Cycle 1: $R_{3,j-1} – R_{0,j} – R_{1,j} – R_{2,j} – R_{3,j}$. The filling-up phase is $R_{0,1} – R_{1,1} – R_{2,1}$. Thus, $R_{i,j}$ takes p_r. Figure C-1 presents a Gantt chart of a complete σ1-sequence when L = 3.

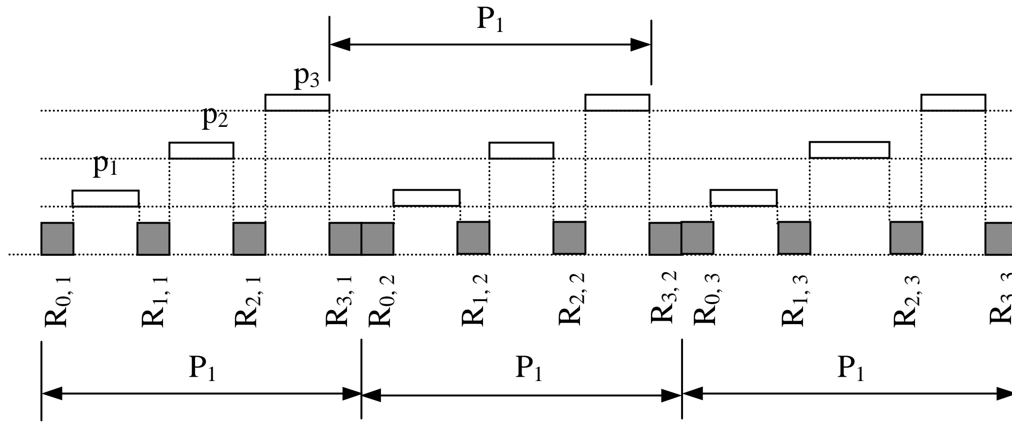


Figure C-1. Gantt chart for a complete σ1-sequence (L =3).

For convenience, consider the cycle $\sigma 1 = R_{0,j} – R_{1,j} – R_{2,j} – R_{3,j}$. The complete sequence consists of L occurrences of σ1. Thus, the cycle length $P_1 = 4p_r + p_1 + p_2 + p_3$, and the makespan $MS_1 = LP_1$.

2) Cycle 3: $R_{3,j-1}– R_{2,j} – R_{0,j+1} – R_{1,j+1} – R_{3,j}$.

In the cyclic phase, $R_{1,j}$ takes p_r and others take $2p_r$. There are two feasible filling-up phases: f1 and f2.

a) f1: $R_{0,1} – R_{1,1} – R_{2,1} – R_{0,2} – R_{1,2}$. Figure C-2a presents a Gantt chart of a complete σ3-sequence with the filling-up f1 when L = 4.

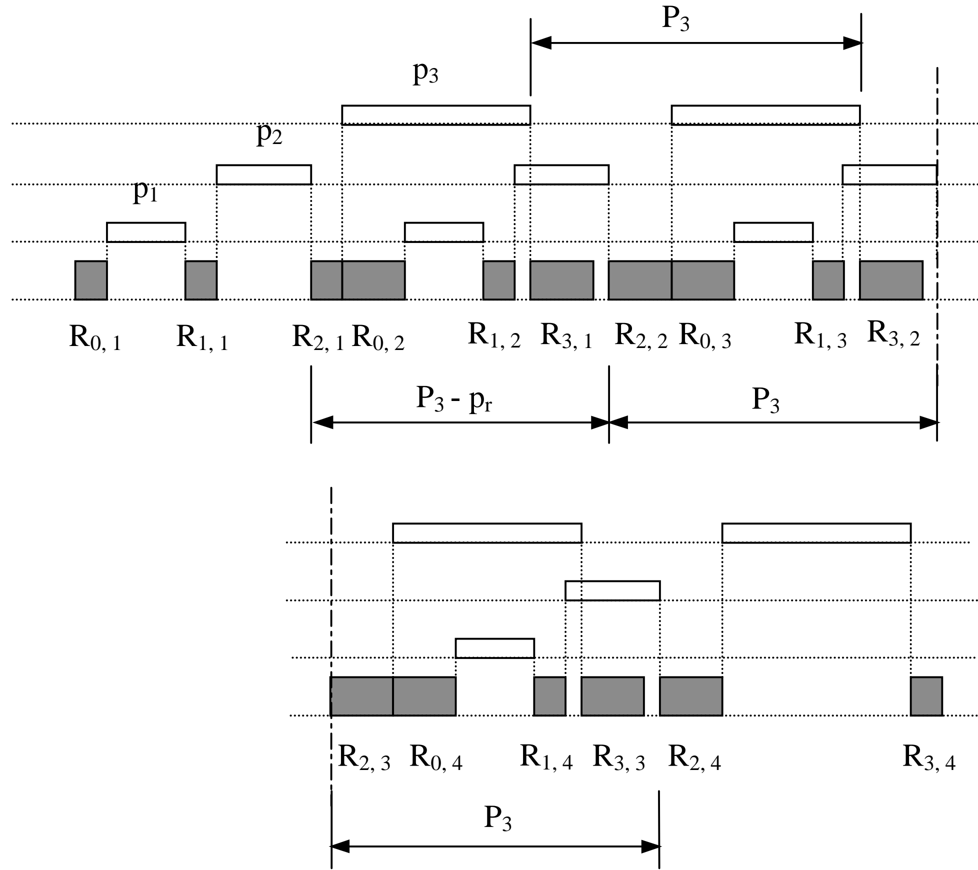


Figure C-2a. Gantt chart of a complete σ3-sequence with filling-up f1 (L = 4).

For convenience, we can consider the jth cyclic cycle $σ3 = R_{2,j}– R_{0,j+1} – R_{1,j+1} – R_{3,j}$, which repeats (L-2) times. Note that the last four moves of the filling-up phase forms a transition cycle whose length equals $P_3 – p_r$. Thus, the first cycle starts at t_1 =

$2p_r + p_1 + p_2 + P_3 - p_r = p_r + p_2 + p_1 + P_3$. We now compute the cycle length. Let t be the starting time of $R_{2,j}$ (j = 2,…, L-1), then

Move	start	complete
$R_{2,j}$	t	$t+2p_r$
$R_{0,j+1}$	$t+2p_r$	$t+4p_r$
$R_{1,j+1}$	$t+4p_r+p_1$	$t+5p_r+p_1$
$R_{3,j}$	$\max\{ t+5p_r+p_1, t+2p_r+p_3\}$	$t+4p_r+\max\{ 3p_r+p_1, p_3\}$
$R_{2,j+1}$	$\max\{ t+4p_r+\max\{ 3p_r+p_1, p_3\},$ $t+5p_r+p_1+p_2\}$ $= t+4p_r +\max\{3p_r+p_1, p_3, p_r+p_1+p_2\}$	$t+6p_r +\max\{ 3p_r+p_1, p_3, p_r+p_1+p_2\}$

Thus, cycle length $P_3 = 4p_r +\max\{3p_r+p_1, p_3, p_r+p_1+p_2\}$. The completion phase starts at time $t_2 = t_1 + (L-2)P_3 = p_r + p_1 + p_2 + (L-1)P_3$. Two moves in the completion phase are $R_{2,L}$ and $R_{3,L}$ (note that $R_{3,L}$ takes p_r). Thus, makespan $MS_{31} = [p_r+p_1+p_2+(L-1)P_3] + [2p_r+p_3+p_r] = 4p_r+p_1+p_2+p_3+(L-1)P_3$.

b) f2: $R_{0,1} - R_{1,1} - R_{0,2} - R_{2,1} - R_{1,2}$. Figure C-2b presents Gantt chart of a complete σ3-sequence with the filling-up f2 when L = 4.

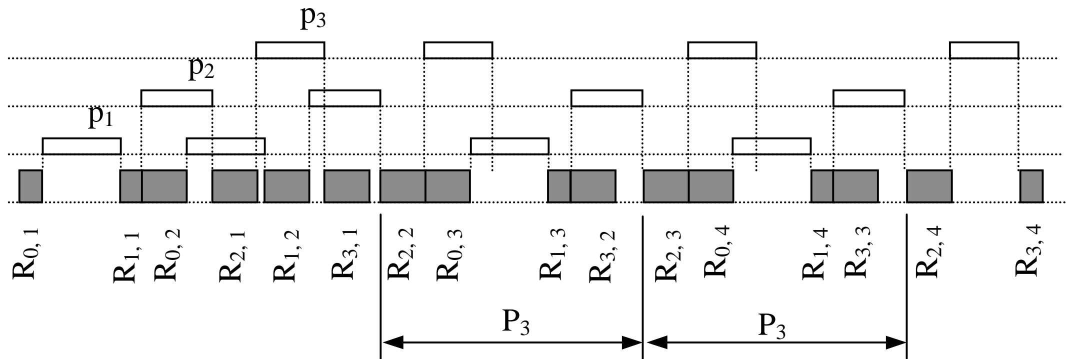


Figure C-2b. Gantt chart of a complete σ3-sequence with filling-up f2 (L = 4).

For convenience, we can consider the jth cyclic cycle $\sigma 3 = R_{2,j} - R_{0,j+1} - R_{1,j+1} - R_{3,j}$, which repeats (L-2) times. We need to determine the starting time t_1 of the first cyclic cycle. Below are the starting and completion times of the moves in the filling-up phase.

Move	Start	Complete
$R_{0,1}$	0	p_r
$R_{1,1}$	p_r+p_1	$2p_r+p_1$
$R_{0,2}$	$2p_r+p_1$	$4p_r+p_1$
$R_{2,1}$	$2p_r+p_1+\max\{p_2, 2p_r\}$	$4p_r+p_1+\max\{p_2, 2p_r\}$
$R_{1,2}$	$\max\{4p_r+2p_1, 4p_r+p_1+\max(p_2, 2p_r)\}$ $= 4p_r+p_1+\max\{p_1, p_2, 2p_r\}$	$6p_r+p_1+\max\{p_1, p_2, 2p_r\}$
$R_{3,1}$	$\max\{4p_r+p_1+p_3+\max(p_2, 2p_r),$ $6p_r+p_1+\max(p_1, p_2, 2p_r)\}$ $= 4p_r+p_1+\max\{p_3+\max(p_2, 2p_r),$ $2p_r+\max(p_1, p_2, 2p_r)\}$	$6p_r+p_1+\max\{p_3+\max(p_2, 2p_r), 2p_r$ $+\max(p_1, p_2, 2p_r)\}$

Hence, $t_1 = \max\{6p_r + p_1 + p_2 + \max(p_1, p_2, 2p_r), 6p_r + p_1 + \max[p_3 + \max(p_2, 2p_r), 2p_r + \max(p_1, p_2, 2p_r)]\}$

$$= 6p_r + p_1 + \max\{p_2, 2p_r\} + \max\{p_1, p_2, p_3, 2p_r\}$$

Note that the steady-state and completion phases are the same as those in Section 3a above. Hence, the cycle length $P_3 = 4p_r + \max\{3p_r + p_1, p_3, p_r + p_1 + p_2\}$. The completion phase starts at time $t_2 = t_1 + (L-2)P_3 = 6p_r + p_1 + \max\{p_2, 2p_r\} + \max\{p_1, p_2, p_3, 2p_r\} + (L-2)P_3$. Two moves in the completion phase are $R_{2,L}$ and $R_{3,L}$ (note that $R_{3,L}$ takes p_r). Thus, makespan

$$MS_{32} = [6p_r + p_1 + \max\{p_2, 2p_r\} + \max\{p_1, p_2, p_3, 2p_r\} + (L-2)P_3] + [2p_r + p_3 + p_r]$$

$$= 9p_r + p_1 + p_3 + \max\{p_2, 2p_r\} + \max\{p_1, p_2, p_3, 2p_r\} + (L-2)P_3.$$

3) Cycle 4: $R_{3,j-1} - R_{1,j} - R_{2,j} - R_{0,j+1} - R_{3,j}$.

In the cyclic cycle, $R_{2,j}$ takes p_r and others take $2p_r$. There are two feasible filling-up phases: f1 and f2.

a) f1: $R_{0,1} - R_{1,1} - R_{2,1} - R_{0,2}$. Figure C-3a presents a Gantt chart of a complete σ4-sequence with the filling-up f1 when L = 4.

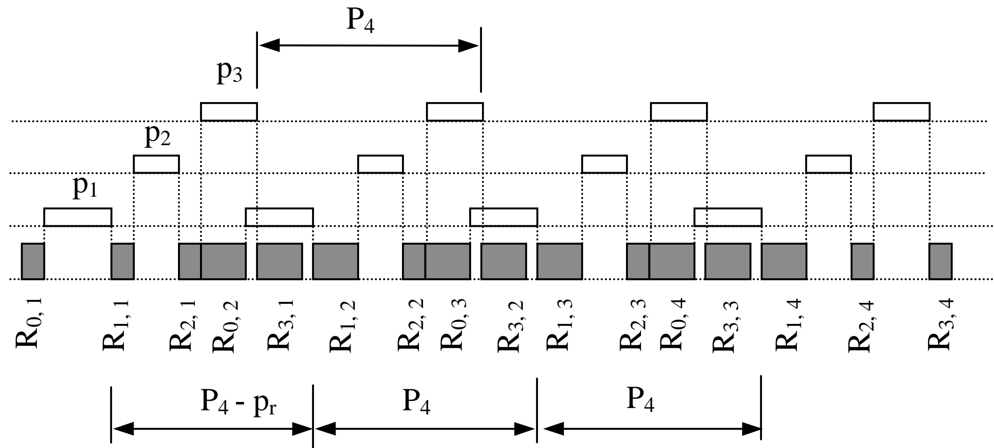


Figure C-3a. Gantt chart of a complete σ4-sequence with filling-up f1 (L = 4).

For convenience, we consider the jth cyclic cycle $\sigma 4 = R_{1,j} - R_{2,j} - R_{0,j+1} - R_{3,j}$, which repeats (L-2) times. Note that the last four moves of the filling-up phase forms a transition cycle whose length equals $P_4 - p_r$. Thus, the first cycle starts at $t_1 = p_r + p_1 + P_4 - p_r = p_1 + P_4$. We now compute the cycle length. Let t be the starting time of $R_{1,j}$ (j = 2,…, L-1), then

Move	start	complete
$R_{1,j}$	t	$t+2p_r$
$R_{2,j}$	$t+2p_r+p_2$	$t+3p_r+p_2$
$R_{0,j+1}$	$t+3p_r+p_2$	$t+5p_r+p_2$
$R_{3,j}$	$t+3p_r+p_2+\max\{2p_r, p_3\}$	$t+5p_r+p_2+\max\{2p_r, p_3\}$
$R_{1,j+1}$	$\max\{t+5p_r+p_2+\max\{2p_r, p_3\},$ $t+5p_r+p_2+p_1\}$ $= t+5p_r+p_2+\max\{2p_r, p_3, p_1\}$	$t+7p_r+p_2+\max\{2p_r, p_3, p_1\}$

Thus, cycle length $P_4 = 5p_r + p_2 + \max\{2p_r, p_3, p_1\}$. The completion phase starts at time $t_2 = t_1 + (L - 2)P_4 = p_1 + (L - 1)P_4$. The completion phase includes $R_{1,L}$, $R_{2,L}$, and $R_{3,L}$. Note that both $R_{2,L}$ and $R_{3,L}$ take p_r. Hence, makespan $MS_{41} = [p_1 + (L - 1)P_4] + [2p_r + p_2 + p_r + p_3 + p_r] = 4p_r + p_1 + p_2 + p_3 + (L - 1)P_4$.

b) f2: $R_{0,1} – R_{1,1} – R_{0,2} – R_{2,1}$. Figure C-3b presents a Gantt chart of a complete σ4-sequence with the filling-up f2 when L = 4.

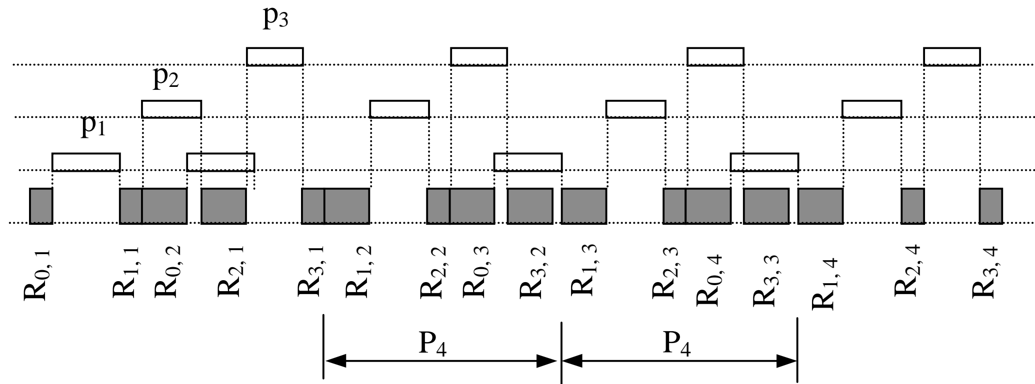


Figure C-3b. Gantt chart of a complete σ4-sequence with filling-up f2 (L = 4).

For convenience, we consider the jth cyclic cycle as $\sigma 4 = R_{1,j} - R_{2,j} - R_{0,j+1} - R_{3,j}$, which repeats (L-2) times. We need to determine the starting time t_1 of the first cyclic cycle. Below are the starting and completion times of the moves in the filling-up phase.

Move	Start	Complete
$R_{0,1}$	0	p_r
$R_{1,1}$	p_r+p_1	$2p_r+p_1$
$R_{0,2}$	$2p_r+p_1$	$4p_r+p_1$
$R_{2,1}$	$2p_r+p_1+\max\{p_2, 2p_r\}$	$4p_r+p_1+\max\{p_2, 2p_r\}$
$R_{3,1}$	$4p_r+p_1+p_3+\max\{p_2, 2p_r\}$	$5p_r+p_1+p_3+\max\{p_2, 2p_r\}$

Hence $t_1 = \max\{4p_r + 2p_1, 5p_r + p_1 + p_3 + \max(p_2, 2p_r)\} = 4p_r + p_1 + \max\{p_1, p_r+p_3+\max(p_2, 2p_r)\}$. Note that the steady-state and completion phases are the same as those in Section 4a above. Hence, the cycle length $P_4 = 5p_r + p_2 + \max\{2p_r, p_3, p_1\}$. The completion phase starts at time $t_2 = t_1 + (L - 2)P_4 = 4p_r + p_1 + \max\{p_1, p_r+p_3 + \max(p_2, 2p_r)\}+ (L - 2)P_4$. The completion phase includes $R_{1,L}$, $R_{2,L}$, and $R_{3,L}$. Note that both $R_{2,L}$ and $R_{3,L}$ take p_r. Hence, makespan $MS_{42} = [4p_r + p_1 + \max\{p_1, p_r+p_3 + \max(p_2, 2p_r)\}+ (L - 2)P_4] + [2p_r + p_2 + p_r + p_3 + p_r] = 8p_r + p_1 + p_2 + p_3 + \max\{p_1, p_r+p_3 + \max(p_2, 2p_r)\}+ (L - 2)P_4$.

4) Cycle 5: $R_{3,j-1}$ - $R_{1,j} - R_{0,j+1} - R_{2,j} - R_{3,j}$.

In the cyclic cycle, $R_{3,j}$ takes p_r and others take $2p_r$. There are two feasible filling-up phases: f1 and f2.

a) f1: $R_{0,1} - R_{1,1} - R_{0,2} - R_{2,1}$. Figure C-4a presents a Gantt chart of a complete σ5-sequence with the filling-up f1when L= 4.

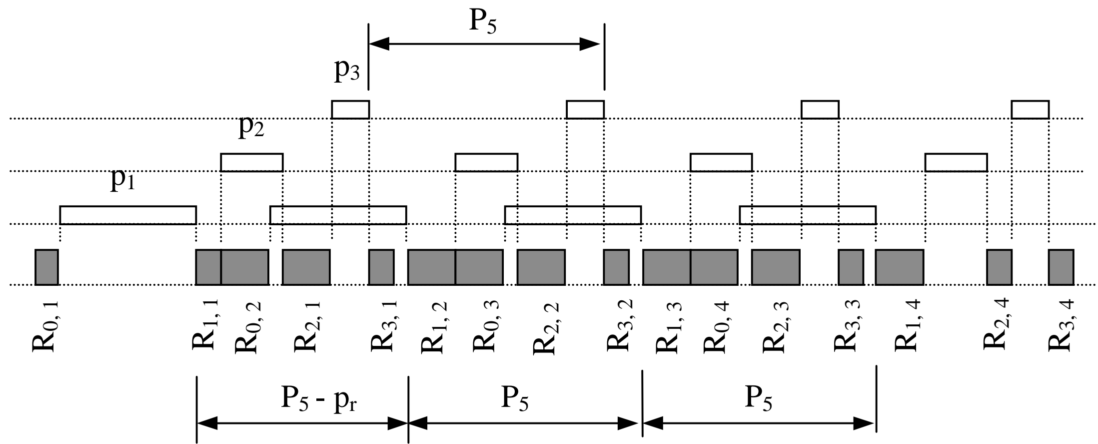


Figure C-4a. Gantt chart of a complete σ5-sequence with filling-up f1 (L = 4).

For convenience, we consider the jth cyclic cycle $\sigma 5 = R_{1,j} - R_{0,j+1} - R_{2,j} - R_{3,j}$, which repeats (L-2) times. Note that the last four moves of the filling-up phase forms a transition cycle whose length equals $P_5 - p_r$. Thus, the first cycle starts at $t_1 = p_r + p_1 + P_5 - p_r = p_1 + P_5$. We now compute the cycle length. Let t be the starting time of $R_{1,j}$ (j = 2,…, L-1). Then,

Move	start	complete
$R_{1,j}$	t	$t+2p_r$
$R_{0,j+1}$	$t+2p_r$	$t+4p_r$
$R_{2,j}$	$t+2p_r+\max\{p_2, 2p_r\}$	$t+4p_r+\max\{p_2, 2p_r\}$
$R_{3,j}$	$t+4p_r+\max\{p_2, 2p_r\}+p_3$	$t+5p_r+\max\{p_2, 2p_r\}+p_3$
$R_{1,j+1}$	$\max\{ t+5p_r+\max\{p_2, 2p_r\}+p_3, t+4p_r +p_1\}$ $= t+4p_r+\max\{p_r+p_2+p_3, 3p_r+p_3, p_1\}$	$t+6p_r+\max\{p_r+p_2+p_3, 3p_r+p_3, p_1\}$

Thus, cycle length $P_5 = 4p_r + \max\{p_r + p_2 + p_3, 3p_r + p_3, p_1\}$. The completion phase starts at time $t_2 = t_1 + (L - 2)P_5 = p_1 + (L - 1)P_5$. The completion phase includes $R_{1,L}$, $R_{2,L}$, and $R_{3,L}$. Note that both $R_{2,L}$ and $R_{3,L}$ take p_r. Hence, makespan $MS_{51} = [p_1 + (L - 1)P_5] + [2p_r + p_2 + p_r + p_3 + p_r] = 4p_r + p_1 + p_2 + p_3 + (L - 1)P_5$.

b) f2: $R_{0,1} – R_{1,1} – R_{2,1} – R_{0,2}$. Figure C-4b presents a Gantt chart of a complete σ5-sequence with the filling-up f2 when L= 4.

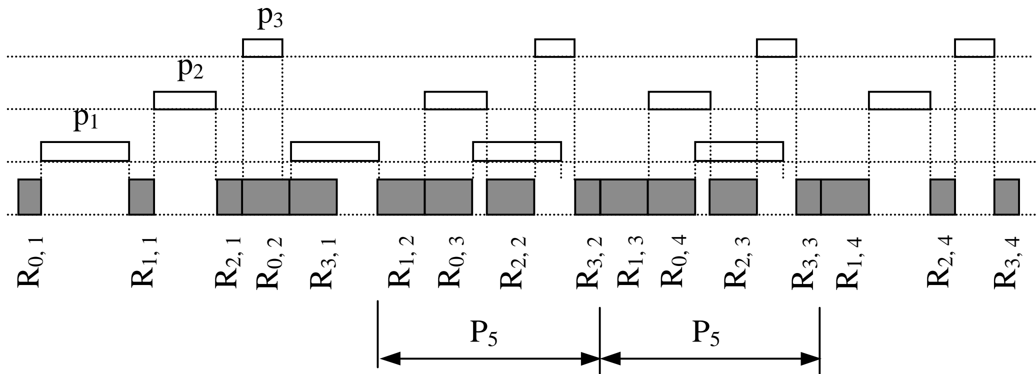


Figure C-4b. Gantt chart of a complete σ5-sequence with filling-up f2 (L = 4).

For convenience, consider the jth cyclic cycle as $\sigma 5 = R_{1, j} - R_{0, j+1} - R_{2, j} - R_{3, j}$, which repeats (L-2) times. We need to determine the starting time t_1 of the first cyclic cycle. Below are the starting and completion times of the moves in the filling-up phase.

Move	Start	Complete
$R_{0, 1}$	0	p_r
$R_{1, 1}$	p_r+p_1	$2p_r+p_1$
$R_{2, 1}$	$2p_r+p_1+p_2$	$3p_r+p_1+p_2$
$R_{0, 2}$	$3p_r+p_1+p_2$	$5p_r+p_1+p_2$
$R_{3, 1}$	$\max\{3p_r+p_1+p_2+p_3, 5p_r+p_1+p_2\}$	$5p_r+p_1+p_2+ \max\{p_3, 2p_r\}$

Hence, $t_1 = \max\{5p_r + 2p_1 + p_2, 5p_r + p_1 + p_2 + \max(p_3, 2p_r)\} = 5p_r + p_1 + p_2 + \max\{p_1, p_3, 2p_r\}$. Note that the steady-state and completion phases are the same as those in Section 5a above. Hence, the cycle length $P_5 = 4p_r + \max\{p_r + p_2 + p_3, 3p_r + p_3, p_1\}$. The completion phase starts at time $t_2 = t_1 + (L - 2)P_5 = 5p_r + p_1 + p_2 + \max\{p_1, p_3, 2p_r\}+ (L - 2)P_5$. The completion phase includes $R_{1,L}$, $R_{2,L}$, and $R_{3,L}$. Note that both $R_{2,L}$ and $R_{3,L}$ take p_r. Hence, makespan $MS_{52} = [5p_r + p_1 + p_2 + \max\{p_1, p_3, 2p_r\}+ (L - 2)P_5] + [2p_r + p_2 + p_r + p_3 + p_r] = 9p_r + p_1 + 2p_2 + p_3 + \max\{p_1, p_3, 2p_r\}+ (L - 2)P_5$.

APPENDIX D

In this appendix, we derive the formulas for cycle length and lot makespan for a CT1-1-1 that follows 1-unit cycle $\sigma 2 = R_{3,j-1} - R_{0,j+1} - R_{2,j} - R_{1,j+1} - R_{3,j}$. Assume that initially the tool is empty and the wafer handler is at load lock. In the cyclic phase, $R_{0,j}$ takes p_r and all other moves require $2p_r$. There are two feasible filling-up phases: f1 and f2.

a) f1: $R_{0,1} - R_{1,1} - R_{0,2} - R_{2,1} - R_{1,2}$. Figure D-1 presents a Gantt chart of a complete $\sigma 2$-sequence with the filling-up phase f1 when L = 5.

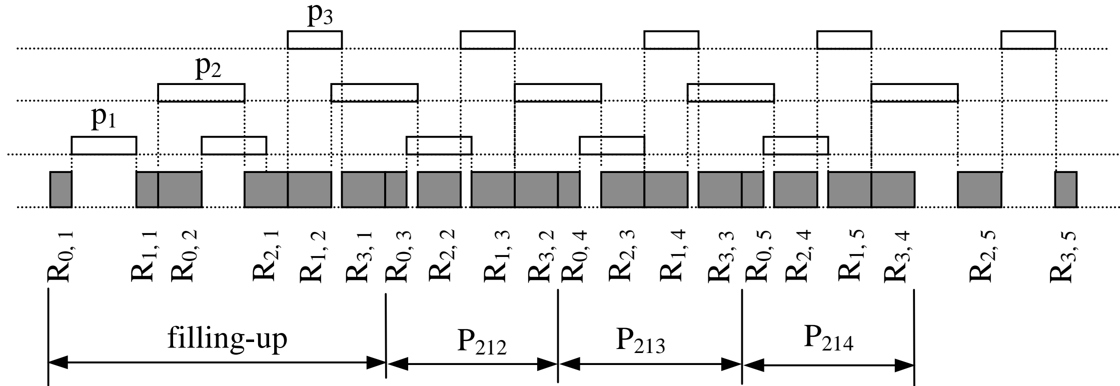


Figure D-1. Gantt chart of a complete $\sigma 2$-sequence with the filling-up phase f1 (L = 5).

For convenience, consider the jth cycle $\sigma 2 = R_{0,j+1} - R_{2,j} - R_{1,j+1} - R_{3,j}$. After the filling-up phase, the tool undergoes (L-2) steady state cycles. The completion phase consists of $R_{2,L}$ and $R_{3,L}$. Note that $R_{3,L}$ takes p_r. Let a_j be the idle time between $R_{2,j}$

and $R_{1, j+1}$. Let b_j be the idle time between $R_{0, j+1}$ and $R_{2, j}$. Let c_j be the idle time between $R_{1, j+1}$ and $R_{3, j}$. Then,

- $j = 1$ (filling-up phase)

  $b_1 = \max(p_2 - 2p_r, 0) = \max(p_2, 2p_r) - 2p_r$

  $a_1 = \max(p_1 - 2p_r - b_1, 0) = \max\{p_1 - \max(p_2, 2p_r), 0\}$

  $c_1 = \max(p_3 - 2p_r - a_1, 0)$

- $1 < j \leq (L - 1)$

  $b_j = \max(p_2 - 3p_r - c_{j-1}, 0)$

  $a_j = \max(p_1 - 2p_r - b_j, 0)$

  $c_j = \max(p_3 - 2p_r - a_j, 0)$

Thus, the length of the jth cycle is $P_{21j} = 7p_r + a_j + b_j + c_j$ ($j = 2,\ldots,L-1$). The average cycle time is

$$P_{21} = \frac{1}{L-2}\sum_{j=2}^{L-1} P_{21j}.$$

The first cycle start at $t_1 = 10p_r + p_1 + a_1 + b_1 + c_1$. The last move of the steady-state phase $R_{3, L-1}$ completes at $t_2 = t_1 + (L - 2)P_{21} = 10p_r + p_1 + a_1 + b_1 + c_1 + (L-2) P_{21}$. The first move of the completion phase can start at time $t_2 + \max(p_2 - 2p_r - c_{L-1}, 0)$, hence, the makespan is

$$\begin{aligned} MS_{21} &= 10p_r + p_1 + a_1 + b_1 + c_1 + (L-2) P_{21} + \max(p_2 - 2p_r - c_{L-1}, 0) + 3p_r + p_3 \\ &= 13p_r + p_1 + p_3 + a_1 + b_1 + c_1 + (L-2) P_{21} + \max(p_2 - 2p_r - c_{L-1}, 0) \end{aligned}$$

b) f2: $R_{0,1} – R_{1,1} – R_{2,1} – R_{0,2} – R_{1,2}$. Figure D-2 presents a Gantt chart of a complete σ2-sequence with the filling-up phase f2 when L = 5.

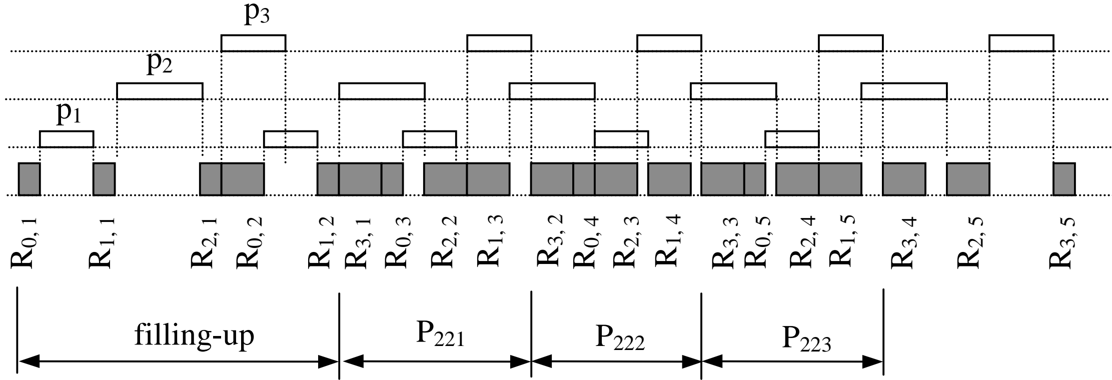


Figure D-2. Gantt chart of a complete σ2-sequence with the filling-up phase f2 (L = 5).

Consider the jth cyclic cycle $\sigma 2 = R_{3,j} – R_{0,j+2} – R_{2,j+1} – R_{1,j+2}$. The first cycle starts at time $t_1 = 3p_r + p_1 + p_2 + \max\{p_3, 3p_r + p_1\}$. Let d_j be the idle time between $R_{2,j+1}$ and $R_{1,j+2}$. Let e_j be the idle time between $R_{0,j+2}$ and $R_{2,j+1}$. Let g_j be the idle time between $R_{1,j+2}$ and $R_{3,j+1}$. Then,

- $j = 1$

  $e_1 = \max\{p_2 – 3p_r – \max(p_3 – p_1 – 3p_r, 0), 0\} = \max\{p_1 + p_2 – \max(p_3, p_1 + 3p_r), 0\}$

  $d_1 = \max\{p_1 – 2p_r – e_1, 0\}$

  $g_1 = \max\{p_3 – 2p_r – d_1, 0\}$

- $1 < j \le (L-2)$

  $e_j = \max\{p_2 – 3p_r – g_{j-1}, 0\}$

  $d_j = \max\{p_1 – 2p_r – e_j, 0\}$

  $g_j = \max\{p_3 – 2p_r – d_j, 0\}$

Thus, the length of the jth cycle is $P_{22j} = 7p_r + d_j + e_j + g_j$ ($j = 1,\ldots, L-2$). The average cycle time is

$$P_{22} = \frac{1}{L-2}\sum_{j=1}^{L-2} P_{22j}.$$

The completion phase starts at $t_2 = t_1 + (L-2)P_{22} = 3p_r + p_1 + p_2 + \max\{p_3, 3p_r + p_1\} + (L-2)P_{22}$. We can compute the starting times of the moves in completion phase as follows.

Move	Starting time	Completion time
$R_{3, L-1}$	t_2	$t_2 + 2p_r$
$R_{2, L}$	$t_2 + 2p_r + \max\{p_2 - 2p_r - g_{L-2}, 0\}$	$t_2 + 4p_r + \max\{p_2 - 2p_r - g_{L-2}, 0\}$
$R_{3, L}$	$t_2+4p_r+p_3+\max\{p_2 - 2p_r - g_{L-2}, 0\}$	$t_2+5p_r+ p_3 +\max\{p_2 - 2p_r - g_{L-2}, 0\}$

Hence, the makespan is

$$\begin{aligned} MS_{22} &= 3p_r+p_1+p_2+\max\{p_3, 3p_r+p_1\} + (L-2)P_{22} + 5p_r + p_3 +\max\{p_2 - 2p_r - g_{L-2}, 0\} \\ &= 8p_r + p_1 + p_2 + p_3 + \max\{p_3, 3p_r + p_1\} +\max\{p_2 - 2p_r - g_{L-2}, 0\} + (L-2)P_{22}. \end{aligned}$$

APPENDIX E

In this appendix, we derive the formulas for cycle length and lot makespan for a CT1-1-1 that follows 1-unit cycle $\sigma 6 = R_{3,j-1} - R_{2,j} - R_{1,j+1} - R_{0,j+2} - R_{3,j}$. We will assume that initially the tool is empty and the wafer handler is at load lock. In the cyclic phase, all moves take $2p_r$. There are two feasible filling-up phases: f1 and f2.

a) f1: $R_{0,1} - R_{1,1} - R_{0,2} - R_{2,1} - R_{1,2} - R_{0,3}$. Figure E-1 presents a Gantt chart of a complete σ6-sequence with the filling-up f1 when L = 5.

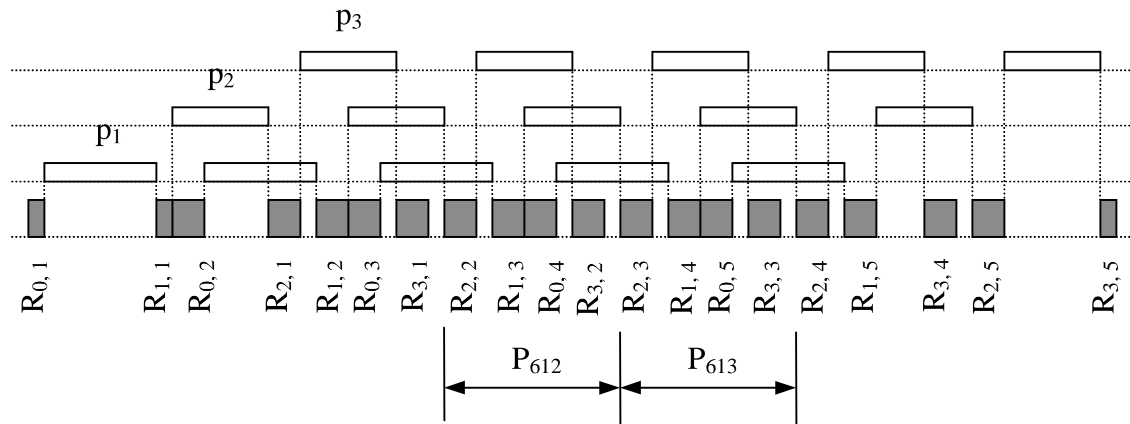


Figure E-1. Gantt chart of a complete σ6-sequence with filling-up f1 (L = 5).

For convenience, we consider the jth cyclic cycle as $\sigma 6 = R_{2,j} - R_{1,j+1} - R_{0,j+2} - R_{3,j}$, which repeats (L-3) times. Below are the starting and completion times of the moves in the filling-up phase.

Move	start	complete
$R_{0,1}$	0	p_r
$R_{1,1}$	p_r+p_1	$2p_r+p_1$
$R_{0,2}$	$2p_r+p_1$	$4p_r+p_1$
$R_{2,1}$	$2p_r+p_1+\max\{2p_r, p_2\}$	$4p_r+p_1+\max\{2p_r, p_2\}$
$R_{1,2}$	$4p_r+p_1+\max\{2p_r, p_1, p_2\}$	$6p_r+p_1+\max\{2p_r, p_1, p_2\}$
$R_{0,3}$	$6p_r+p_1+\max\{2p_r, p_1, p_2\}$	$8p_r+p_1+\max\{2p_r, p_1, p_2\}$
$R_{3,1}$	$4p_r+p_1+\max\{4p_r+\max(2p_r, p_1, p_2), \max(2p_r, p_2)+p_3\}$	$6p_r+p_1+\max\{4p_r+\max(2p_r, p_1, p_2), \max(2p_r, p_2)+p_3\}$

Hence, the first cycle starts at $t_1 = \max\{6p_r + p_1 + \max[4p_r + \max(2p_r, p_1, p_2), \max(2p_r, p_2) + p_3], p_2 + 6p_r + p_1 + \max\{2p_r, p_1, p_2\}\} = 6p_r + p_1 + \max(2p_r, p_1, p_2) + \max\{4p_r, p_2, p_3 - \max[0, p_1 - \max(2p_r, p_2)]\}$. Let w_2 be the idle time between $R_{2,2}$ and $R_{1,3}$, then $w_2 = \max\{p_1 + 8p_r + p_1 + \max(2p_r, p_1, p_2) - 8p_r - p_1 - \max(2p_r, p_1, p_2) - \max\{4p_r, p_2, p_3 - \max[0, p_1 - \max(2p_r, p_2)]\}, 0\} = \max\{p_1 - \max\{4p_r, p_2, p_3 - \max[0, p_1 - \max(2p_r, p_2)]\}, 0\}$.

We now compute the cycle length. Let w_j be the idle time between $R_{2,j}$ and $R_{1,j+1}$. Let t be the starting time of $R_{2,j}$ ($j = 2, \ldots, L-3$), then the starting and completion times of the moves in the cyclic cycle are:

Move	start	complete
$R_{2,j}$	t	$t+2p_r$
$R_{1,j+1}$	$t+2p_r+w_j$	$t+4p_r+w_j$
$R_{0,j+2}$	$t+4p_r+w_j$	$t+6p_r+w_j$
$R_{3,j}$	$t+2p_r+\max\{p_3, 4p_r+w_j\}$	$t+4p_r+\max\{p_3, 4p_r+w_j\}$
$R_{2,j+1}$	$t+4p_r+w_j+\max\{p_2, p_3-w_j, 4p_r\}$	$t+6p_r+w_j+\max\{p_2, p_3-w_j, 4p_r\}$

Thus, cycle length of the jth cycle is $P_{61j} = 4p_r + w_j + \max\{p_2, p_3 - w_j, 4p_r\}$, and the idle time between $R_{2,j+1}$ and $R_{1,j+2}$ is $w_{j+1} = \max\{p_1 - \max(p_2, p_3 - w_j, 4p_r), 0\}$. The average cycle time is

$$P_{61} = \frac{1}{L-3}\sum_{j=2}^{L-2} P_{61j}.$$

The completion phase can start at time $t_2 = t_1 + (L-3)P_{61}$. The completion phase consists of $R_{2,L-1}$- $R_{1,L}$ - $R_{3,L-1}$ - $R_{2,L}$ - $R_{3,L}$. Note that $R_{3,L}$ takes p_r. We now compute starting and completion times of the moves in the completion phase as follows.

Move	Start	Complete
$R_{2,L-1}$	t_2	t_2+2p_r
$R_{1,L}$	$t_2+2p_r+ w_{L-2}$	$t_2+4p_r+w_{L-2}$
$R_{3,L-1}$	$t_2+2p_r+\max\{2p_r+ w_{L-2}, p_3\}$	$t_2+4p_r+\max\{2p_r+ w_{L-2}, p_3\}$
$R_{2,L}$	$t_2+4p_r+ w_{L-2} +\max\{2p_r, p_2, p_3-w_{L-2}\}$	$t_2+6p_r+ w_1 +\max\{2p_r, p_2, p_3-w_{L-2}\}$
$R_{3,L}$	$t_2+6p_r+p_3+ w_{L-2}+\max\{2p_r, p_2, p_3 - w_{L-2}\}$	$t_2+7p_r+p_3+w_{L-2}+\max\{2p_r, p_2, p_3- w_{L-2}\}$

Thus, makespan

$$MS_{61} = 6p_r + p_1 + \max(2p_r, p_1, p_2) + \max\{4p_r, p_2, p_3 - \max[0, p_1 - \max(2p_r, p_2)]\}$$
$$+ (L-3)P_{61} + 7p_r + p_3 + w_{L-2} + \max\{2p_r, p_2, p_3 - w_{L-2}\}$$
$$= 13p_r + p_1 + p_3 + w_{L-2} + \max(2p_r, p_1, p_2) + \max\{4p_r, p_2, p_3 - \max[0, p_1 - \max(2p_r, p_2)]\} + \max\{2p_r, p_2, p_3 - w_{L-2}\} + (L-3)P_{61}.$$

b) f2: $R_{0,1} - R_{1,1} - R_{2,1} - R_{0,2} - R_{1,2} - R_{0,3}$. Figure E-2 presents a Gantt chart of a complete σ6-sequence with the filling-up f2 when L = 5.

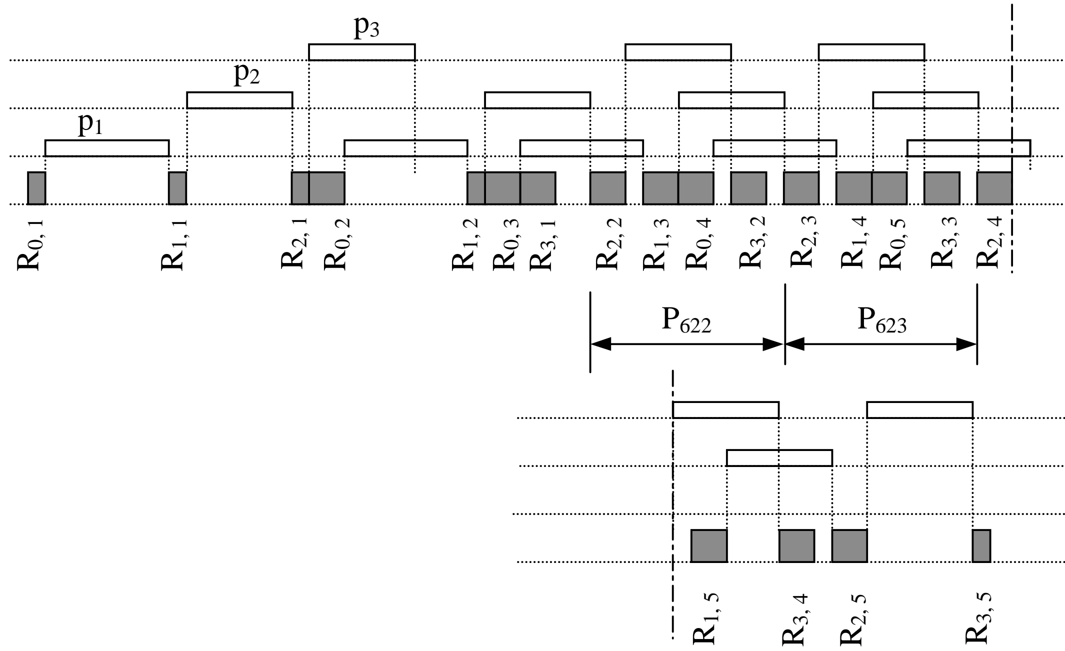


Figure E-2. Gantt chart of a complete σ6-sequence with filling-up f2 (L = 5).

For convenience, we consider the jth cyclic cycle as $\sigma 6 = R_{2,j} - R_{1,j+1} - R_{0,j+2} - R_{3,j}$, which repeats (L-3) times. Below are the starting and completion times of the moves in the filling-up phase.

Move	start	complete
$R_{0,1}$	0	p_r
$R_{1,1}$	p_r+p_1	$2p_r+p_1$
$R_{2,1}$	$2p_r+p_1+p_2$	$3p_r+p_1+p_2$
$R_{0,2}$	$3p_r+p_1+p_2$	$5p_r+p_1+p_2$
$R_{1,2}$	$5p_r+2p_1+p_2$	$6p_r+2p_1+p_2$
$R_{0,3}$	$6p_r+2p_1+p_2$	$8p_r+2p_1+p_2$
$R_{3,1}$	$\max\{3p_r+p_1+p_2+p_3, 8p_r+2p_1+p_2\}$	$5p_r+p_1+p_2+\max\{p_3, 5p_r+p_1\}$

Hence, the first cycle starts at $t_1 = \max\{6p_r + 2p_1 + 2p_2, 5p_r + p_1 + p_2 + \max(p_3, 5p_r + p_1)\}$ $= 5p_r + p_1 + p_2 + \max\{p_r + p_1 + p_2, p_3, 5p_r + p_1\}$. And the idle time of robot between $R_{2,2}$ and $R_{1,3}$ is

$$v_2 = \max\{8p_r + 3p_1 + p_2 - t_1 - 2p_r, 0\}$$

$$= \max\{8p_r + 3p_1 + p_2 - 5p_r - p_1 - p_2 - \max\{p_r + p_1 + p_2, p_3, 5p_r + p_1\} - 2p_r, 0\}$$

$$= \max\{p_r + 2p_1 - \max(p_r + p_1 + p_2, p_3, 5p_r + p_1), 0\}$$

$$= \max\{p_1 - \max[p_2, 4p_r, p_3 - p_1 - p_r], 0\}.$$

The procedure to determine the cycle length is the same as in Section 6a above. In fact, substituting w_j by v_j yields the cycle length $P_{62j} = 4p_r + v_j + \max\{p_2, p_3\text{-}v_j, 4p_r\}$. The average cycle time is

$$P_{62} = \frac{1}{L-3}\sum_{j=2}^{L-2} P_{62j}.$$

The completion phase starts at time $t_2 = t_1 + (L - 3)P_{62}$. Also, the idle time between $R_{2,j+1}$ and $R_{1,j+2}$ in the $(j+1)^{th}$ cycle is $v_{j+1} = \max\{p_1\text{-}\max(p_2, p_3 - v_j, 4p_r), 0\}$. The completion phase consists of $R_{2,L-1}$- $R_{1,L}$ - $R_{3,L-1}$ - $R_{2,L}$ - $R_{3,L}$. Note that $R_{3,L}$ takes p_r. We can compute the starting and completion times of the moves in the completion phase as the same procedure in Section 6a. The lot makespan is

$$MS_{62} = t_2 + 7p_r + p_3 + v_{L-2} + \max\{2p_r, p_2, p_3\text{-} v_{L-2}\}$$

$$= 5p_r + p_1 + p_2 + \max\{p_r + p_1 + p_2, p_3, 5p_r + p_1\} +$$

$$+ (L - 3)P_{62} + 7p_r + p_3 + w_{L-2} + \max\{2p_r, p_2, p_3\text{-} v_{L-2}\}$$

$$= 12p_r + p_1 + p_2 + p_3 + v_{L-2} + \max\{p_r + p_1 + p_2, p_3, 5p_r + p_1\} +$$

$$+ \max\{2p_r, p_2, p_3\text{-} v_{L-2}\} + L - 3)P_{62}.$$

APPENDIX F

F.1 The Basic Objects: Job and Machine

- The Job

```
public class Job {
  private String jobName;
  private int jobID;
  private double jobDueDate;
  private double jobPriority;
  private double jobReleaseDate;
  private double jobProcessTime;
  private Vector jobOperating;

  public Job() { jobOperating = new Vector(); }
  public void setJobName(String name) { jobName = name; }
  public String getJobName() { return jobName; }
  public void setJobID(Integer index){ jobID = index.intValue(); }
  public int  getJobID() { return jobID; }
  public void setJobDueDate(Double duedate) {
   jobDueDate = duedate.doubleValue();
                                             }
  public double  getJobDueDate() { return jobDueDate; }
  public void  setJobPriority(Double priority) {
   jobPriority = priority.doubleValue();
  }
  public double  getJobPriority() { return jobPriority; }
  public void  setJobReleaseDate(Double releaseDate) {
   jobReleaseDate = releaseDate.doubleValue();
  }
  public double  getJobReleaseDate(){ return jobReleaseDate; }
  public void setJobProcessTime(Double processTime) {
  jobProcessTime = processTime.doubleValue();
  }
  public double getJobProcessTime() { return jobProcessTime; }
  public void setJobOperating(Machine aMachine) {

   jobOperating.addElement(aMachine);

 }
```

- The Machine

```
public class Machine implements Cloneable
{
  private int machineID;
  private double machineProcessTime;
  private double startTime;
  private double completionTime;
  private Vector machineTask;

  public Machine() {
    machineTask = new Vector(); }
  public void setMachineID(Integer  id)  {
    machineID = id.intValue(); }
  public int getMachineID ()  {
    return machineID; }
  public void setMachineProcessTime(Double time)  {
    machineProcessTime = time.doubleValue(); }
  public double getMachineProcessTime ()  {
    return machineProcessTime;  }
  public void setStartTime(Double time) {
    startTime = time.doubleValue();  }
  public double getStartTime() {
    return startTime;  }
  public void setCompletionTime(Double time) {
    completionTime = time.doubleValue();  }
  public double getCompletionTime() {
    return completionTime;  }
  public void setMachineTask(Job aJob)  {
    machineTask.addElement(aJob);  }
  public Vector getMachineTask()  {
    return machineTask;  }
}
```

The purpose of the “set” methods is to store information into the Job and Machine, while the purpose of the “get” methods is to extract information from the Job and Machine. The Job also contains a vector of Machines, the jobOperating. And the Machine also contains a vector of Jobs, the machineTask. By this structure, we are able to build a changeably sized array of any dimension.

F.2 Java codes for three-stage cluster tool models.

Example 1. A portion of the Java codes for a three-stage cluster tool model using Algorithm BB.

```
public Vector search(char[] feasible_moves, int num_wafer, int wafer, int c_wafer,
          int act, double robot_durat, double S1_durat, double S2_durat,double
S3_durat,
          Job upper_bound , Job robot_pos, Machine global_time, Vector S1s,
          Vector S2s, Vector S3s, Vector list, Vector best_list )
{

   Schedule2 T2 = new Schedule2();
   for (int i = 0; i < feasible_moves.length; i++) {
   NEXT_I:
          if (feasible_moves[i] = = 'y') {
            // make a temporary list
            Vector t_list = new Vector();
            for (int j = 0 ; j < list.size(); j++)
              t_list.addElement((Job)list.elementAt(j));

           // make temporary tool environment

            Job t_C_wafer = new Job(); t_C_wafer.setJobID(new Integer(c_wafer));
            Job t_Wafer = new Job(); t_Wafer.setJobID(new Integer(wafer));
            Job t_Act = new Job(); t_Act.setJobID(new Integer(act));

            Job t_robot_pos = new Job();
            t_robot_pos.setJobName(robot_pos.getJobName());

            Machine t_global_time = new Machine();
            t_global_time.setStartTime(new Double(global_time.getStartTime()));

            Vector t_S1s = create_new_handler(S1s);
            Vector t_S2s = create_new_handler(S2s);
            Vector t_S3s = create_new_handler(S3s);

            make_the_move(i, num_wafer, t_Wafer, t_C_wafer, t_Act, robot_durat,
                         S1_durat, S2_durat, S3_durat,t_robot_pos,
                         t_global_time, t_S1s, t_S2s, t_S3s, t_list);
             // return values
            int t_wafer = t_Wafer.getJobID();
            int t_c_wafer = t_C_wafer.getJobID();
            int t_act = t_Act.getJobID();
```

Example 1 continued.

```
            //discard the trial sequence if its partial makespan is greater than
upper_bound

            if(T2.getMakespan(t_list) >= upper_bound.getJobDueDate())
              break NEXT_I;
            if (t_c_wafer = = num_wafer) { // last wafer finished
              double pMS = T2.getMakespan(t_list);
              // update upper_bound
              upper_bound.setJobDueDate(new Double(pMS));
              // update best_list
              best_list.removeAllElements();
              for (int u = 0; u < t_list.size(); u++)
                best_list.addElement( (Job)t_list.elementAt(u));
              break NEXT_I;
            }
            else { // not complete, search again
              char[] feasible_moves_b =
                get_feasible_moves(num_wafer-t_wafer+1, t_S1s, t_S2s, t_S3s,
                                   t_robot_pos, t_global_time, robot_durat,
                                   S1_durat, S2_durat, S3_durat);

              t_list =
                    search(feasible_moves_b, num_wafer, t_wafer,
                            t_c_wafer, t_act,  robot_durat, S1_durat, S2_durat,
                            S3_durat, upper_bound ,  t_robot_pos, t_global_time,
                            t_S1s, t_S2s, t_S3s, t_list,  best_list);
            }
          }
      }
      return best_list;
  }
}
```

Example 2. A main portion of the Java code for a three-stage cluster tool model using Algorithm TBB.

```
public Vector smart_search(int k,int K,Job limiter,char[] feasible_moves,
                                int num_wafer, int wafer, int c_wafer, int act,
                                double robot_durat, double S1_durat, double S2_durat,
                                double S3_durat, Job upper_bound ,Job robot_pos,
                                Machine global_time, Vector S1s, Vector S2s,
                                Vector S3s, Vector list, Vector best_list )
 {
  Schedule2 T2 = new Schedule2();
  Schedule7 T7 = new Schedule7();
  Schedule9a T9a = new Schedule9a();
  for (int i = 0; i < feasible_moves.length; i++) {
   NEXT_I:
      if (feasible_moves[i] == 'y') {
        Vector t_list = new Vector();
       for (int j = 0 ; j < list.size(); j++)

      // make a temporary list
        {  t_list.addElement((Job)list.elementAt(j));   }
      // make temporary tool environment

       Job t_C_wafer = new Job(); t_C_wafer.setJobID(new Integer(c_wafer));
       Job t_Wafer = new Job(); t_Wafer.setJobID(new Integer(wafer));
       Job t_Act = new Job(); t_Act.setJobID(new Integer(act));
       Job t_robot_pos = new Job();
       t_robot_pos.setJobName(robot_pos.getJobName());

       Machine t_global_time = new Machine();
       t_global_time.setStartTime(new Double(global_time.getStartTime()));

       Vector t_S1s = T9a.create_new_handler(S1s);
       Vector t_S2s = T9a.create_new_handler(S2s);
       Vector t_S3s = T9a.create_new_handler(S3s);

       T9a.make_the_move(i, num_wafer, t_Wafer, t_C_wafer, t_Act,
                    robot_durat,   S1_durat, S2_durat, S3_durat,t_robot_pos,
                    t_global_time, t_S1s, t_S2s, t_S3s, t_list);

       // return values
       int t_wafer = t_Wafer.getJobID();
       int t_c_wafer = t_C_wafer.getJobID();
       int t_act = t_Act.getJobID();
```

Example 2 continued

```
// discard the trial sequence if its partial makespan is greater than or equal to
//upper_bound.

            double pMS = T2.getMakespan(t_list);
            if( pMS >= upper_bound.getJobDueDate())
              break NEXT_I;

            if (t_c_wafer == num_wafer) {  // sequence reaches its end
            // update upper_bound
            upper_bound.setJobDueDate(new Double(pMS));
              // update best_list
              best_list.removeAllElements();
              for (int u = 0; u < t_list.size(); u++)
                best_list.addElement( (Job)t_list.elementAt(u));
              break NEXT_I;
            }
            else { // sequence not reach its end.
            if(t_c_wafer > k){
                Vector moves = get_departures_of_moves(t_list);
                int[] pattern =
                     is_pattern_found(S1s.size(), S2s.size(), S3s.size(),moves) ;
                if ( pattern[0] = = 1 ) {
                     //pattern found, stop here, get the pattern, run the program to
                     // assign fixed sequence to tool.
                     Vector pattern_list =
                             get_fixed_sequence(pattern[1], pattern[2], moves);
                     Vector aList =
                            T7.get_fixed_sequence_for_3S_tool(pattern_list, num_wafer,
                                                              S1s.size(), S2s.size(), S3s.size(),
                                                              robot_durat, S1_durat, S2_durat,
                                                              S3_durat);
                     pMS = T2.getMakespan(aList);
                     if (pMS  < upper_bound.getJobDueDate() ) {
                      // update upper_bound
                         upper_bound.setJobDueDate(new Double(pMS));
                         // update best list
                         best_list.removeAllElements();
                         for (int p = 0; p < aList.size(); p++)
                           best_list.addElement( (Job)aList.elementAt(p) );
                     }
                     break NEXT_I;
                }
```

Example 2 continued

```
          else {// sequence not long enough, search again
              char[] feasible_moves_b =
                    T9a.get_feasible_moves(num_wafer-t_wafer+1,t_S1s,t_S2s,t_S3s,
                                             t_robot_pos,t_global_time,robot_durat,
                                             S1_durat,S2_durat,S3_durat);

               t_list =
                    smart_search(k, feasible_moves_b, num_wafer, t_wafer, t_c_wafer,
                                   t_act, robot_durat, S1_durat, S2_durat, S3_durat,
                                   upper_bound , t_robot_pos, t_global_time, t_S1s,
                                   t_S2s, t_S3s, t_list, best_list);
                 }
            }
          }
     }
  return best_list;
}
```

Example 3. A main portion of the Java code for a three-stage cluster tool model using the push dispatching rule.

```
public Vector get_sequence_by_push_rule_for_3S_tool(int num_wafer,
                                    int num_S1, int num_S2, int num_S3,
                                    double robot_durat, double S1_durat,
                                    double S2_durat, double S3_durat)
  {
// set up tool configuration
    Machine global_time = new Machine();
    global_time.setStartTime(new Double(0));
    Job robot_pos = new Job();
    robot_pos.setJobName("LL");
    Vector S1s = set_up_new_chambers(num_S1);
    Vector S2s = set_up_new_chambers(num_S2);
    Vector S3s = set_up_new_chambers(num_S3);
    Vector list = new Vector();
    int c_wafer = 0; // number of completed wafer
    int act = 1, wafer = 1;  // wafer is the number of wafer started
    while(c_wafer < num_wafer)  {
      char S1s_status;
      if (wafer <= num_wafer)
           S1s_status = is_a_chamber_free_or_all_busy(global_time, S1s);
      else { S1s_status = 'b';}
      char S2s_status = is_a_chamber_free_or_all_busy(global_time, S2s);
      char S3s_status = is_a_chamber_free_or_all_busy(global_time, S3s);
      if(S1s_status == 'f' && wafer <= num_wafer)
           { //  one free cham. in S1 and loadlock still has wafer(s)
             // priority 1
             load_S1(wafer, act, robot_durat, S1_durat, robot_pos,
                        global_time, S1s, list);
             wafer++; act = act + 2;

           }

           else if(is_there_a_done_chamber(global_time, S1s) == 'y' &&
                S2s_status == 'f')
           { // one done cham. in S1 others busy and one free cham. in S2
              // --> must load wafer from S1 to S2, move name is R2
              // priority 2
```

Example 3 continued.

```
        load_stage("S1","R2","S2",act, robot_durat, S2_durat, robot_pos,
                    global_time, S1s, S2s, list);
        act = act + 2;
       }

    else if(is_there_a_done_chamber(global_time, S2s) == 'y' &&
          S3s_status == 'f')
       { // one done cham. in S2 others busy and one free cham. in S3
        // --> must load wafer from S2 to S3, move name is R3
        // priority 3
        load_stage("S2","R3","S3",act, robot_durat, S3_durat, robot_pos,
                    global_time, S2s, S3s, list);
        act = act + 2;
       }

    else if(is_there_a_done_chamber(global_time, S3s) == 'y')
       { //S3 is the last stage --> must unload this chamber from S3, move R4
        // priority 4
        unload_stage("S3", "R0", act, robot_durat, robot_pos,
                     global_time, S3s, list);
        act++; c_wafer++;
       }

    else if(S2s_status == 'b' && S3s_status == 'f')
       { // S2 busy and one S3 free other S3 are busy
        // priority 5
        if(push_cham_from_2_stages(S2s,S3s)== '2') {
          // one cham in S3s done first
          unload_stage("S3", "R0", act, robot_durat, robot_pos,
                       global_time, S3s, list);
          act++; c_wafer++;
         }
        else {// one S2 cham. done first (there is a free S3)-->load S3
          load_stage("S2","R3","S3",act, robot_durat, S3_durat, robot_pos,
                       global_time, S2s, S3s, list);
          act = act + 2;
         }
       }

      else if(S1s_status == 'b' && S2s_status == 'f' && S3s_status == 'f')
       { // all S1 busy, one S2 free other S2 busy, S3 free
        // priority 6
```

Example 3 continued.

```
        char which = push_cham_from_3_stages(S1s,S2s,S3s);
         if(which == '1') {
           // a cham in S1s done first and always has a free S2s' cham
           // --> load from S1s to S2s
           load_stage("S1","R2","S2",act, robot_durat, S2_durat, robot_pos,
                    global_time, S1s, S2s, list);
           act = act + 2;
         }
         else if(which == '2') {
           // a cham in S2s done first and there is always a free cham in S3s
           // --> load from S2s to S3s
           load_stage("S2","R3","S3",act, robot_durat, S3_durat, robot_pos,
                    global_time, S2s, S3s, list);
            act = act + 2;
         }
         else { // a cham in S3s done first
           unload_stage("S3", "R0", act, robot_durat, robot_pos,
                        global_time, S3s, list);
           act++; c_wafer++;
         }
        }
    else if(S1s_status == 'b' && S2s_status == 'f' && S3s_status == 'b')
        { // all S1 busy, one S2 free other S2 busy, all S3 busy
         // priority 7
         if(push_cham_from_2_stages(S1s,S3s)== '1') {
           // one cham in S1s done first --> load from S1s to S2s
           load_stage("S1","R2","S2",act, robot_durat, S2_durat, robot_pos,
                    global_time, S1s, S2s, list);
           act = act + 2;
         }
         else {   // one S3 cham. done first --> unload S3s
           unload_stage("S3", "R0", act, robot_durat, robot_pos,
                        global_time, S3s, list);
           act++; c_wafer++;
         }
        }
    else { // otherwise case
        unload_stage("S3", "R0", act, robot_durat, robot_pos,
                  global_time, S3s, list);
        act++; c_wafer++;
          }
      }
  return list;
}
```

Example 4. A main portion of the Java code for a three-stage cluster tool model using the pull dispatching rule.

```
public Vector get_sequence_by_pull_rule_for_3S_tool(int num_wafer, int num_S1,
int num_S2,
                                int num_S3, double robot_durat, double S1_durat,
                                double S2_durat, double S3_durat)
 {
  // set up tool configuration
  Machine global_time = new Machine();
  global_time.setStartTime(new Double(0));
  Job robot_pos = new Job();
  robot_pos.setJobName("LL");
  Vector S1s = set_up_new_chambers(num_S1);
  Vector S2s = set_up_new_chambers(num_S2);
  Vector S3s = set_up_new_chambers(num_S3);
  Vector list = new Vector();
  int c_wafer = 0; // number of completed wafer
  int act = 1, wafer = 1;  // wafer is the number of wafer started

  while(c_wafer < num_wafer)  {
   char S1s_status;
   if (wafer <= num_wafer)
       S1s_status = is_a_chamber_free_or_all_busy(global_time, S1s);
   else { S1s_status = 'b';}
   char S2s_status = is_a_chamber_free_or_all_busy(global_time, S2s);
   char S3s_status = is_a_chamber_free_or_all_busy(global_time, S3s);

   if(is_there_a_done_chamber(global_time, S3s) == 'y')
       { //S3 is the last stage --> must unload this chamber from S3, move R4
        // priority 1
        unload_stage("S3", "R0", act, robot_durat, robot_pos,
                  global_time, S3s, list);
        act++; c_wafer++;
       }
   else if(is_there_a_done_chamber(global_time, S2s) == 'y' &&
          S3s_status == 'f')
       {// one done cham. in S2 others busy and one free cham. in S3
        // --> must load wafer from S2 to S3, move name is R3
        // priority 2
        load_stage("S2","R3","S3",act, robot_durat, S3_durat, robot_pos,
                 global_time, S2s, S3s, list);
        act = act + 2;
       }
```

Example 4 continued.

```
   else if(is_there_a_done_chamber(global_time, S1s) == 'y' &&
            S2s_status == 'f')
        {// one done cham. in S1 others busy and one free cham. in S2
         // --> must load wafer from S1 to S2, move name is R2
         // priority 3
         load_stage("S1","R2","S2",act, robot_durat, S2_durat, robot_pos,
                      global_time, S1s, S2s, list);
         act = act + 2;
        }

    else if(S1s_status == 'f' && wafer <= num_wafer) {
        //  one free cham. in S1 and loadlock still has wafer(s)
        // priority 4
        load_S1(wafer, act, robot_durat, S1_durat, robot_pos,
                  global_time, S1s, list);
        wafer++; act = act + 2;
         }
    else if(S2s_status == 'b' && S3s_status == 'f')
        {// S2 busy and one S3 free other S3 are busy
         // priority 5
         if(pull_cham_from_2_stages(S2s,S3s)== '2') {
           // '2' means the second stage between S2s and S3s
           // one cham in S3s done first
         unload_stage("S3", "R0", act, robot_durat, robot_pos,
                         global_time, S3s, list);
           act++; c_wafer++;
        }
         else {// one S2 cham. done first (there is a free S3)-->load S3
           load_stage("S2","R3","S3",act, robot_durat, S3_durat, robot_pos,
                        global_time, S2s, S3s, list);
           act = act + 2;
        }

        }
     else if(S1s_status == 'b' && S2s_status == 'f' && S3s_status == 'f')
        {// all S1 busy, one S2 free other S2 busy, S3 free or busy
         // priority 6

         char which = pull_cham_from_3_stages(S1s,S2s,S3s);
```

Example 4 continued.

```
  if (which == '3') {
        // a cham in S3s done first  --> pull
        unload_stage("S3", "R0", act, robot_durat, robot_pos,
                     global_time, S3s, list);
        act++; c_wafer++;
      }
       else if(which == '2') {
        // a cham in S2s done first and always have a free S3s' cham
        // --> load from S2s to S3s
        load_stage("S2","R3","S3",act, robot_durat, S3_durat, robot_pos,
                    global_time, S2s, S3s, list);
        act = act + 2;
      }
       else {
        // a cham in S1s done first and there is always a free cham in S2s
        // --> load from S1s to S2s
        load_stage("S1","R2","S2",act, robot_durat, S2_durat, robot_pos,
                   global_time, S1s, S2s, list);
        act = act + 2;
       }
   }

   else if (S1s_status == 'b' && S2s_status == 'f' && S3s_status == 'b'){
      // all S1 busy, one S2 free other S2 busy, all S3 busy
      // priority 7
      if(pull_cham_from_2_stages(S1s,S3s)== '2') {
       // '2' means the second stage between S1s and S3s
       // one cham in S3s done first
       unload_stage("S3", "R0", act, robot_durat, robot_pos,
                    global_time, S3s, list);
       act++; c_wafer++;
      }
      else {// one S1 cham. done first (there is a free S2)-->load S2
       load_stage("S1","R2","S2",act, robot_durat, S2_durat, robot_pos,
                  global_time, S1s, S2s, list);
       act = act + 2;
      }
   }
  return list;
 }
```

Example 5. A main portion of the Java code for a three-stage cluster tool model using the prespecified sequence of wafer handler moves.

```
public Vector get_fixed_sequence_for_3S_tool(Vector event_list, int num_wafer,
                                             int num_S1, int num_S2, int num_S3,
                                             double robot_durat, double S1_durat,
                                             double S2_durat, double S3_durat)
  {
   // set up tool configuration
   Schedule6 tool6 = new Schedule6();
   Machine global_time = new Machine();
   global_time.setStartTime(new Double(0));
   Job robot_pos = new Job();
   robot_pos.setJobName("LL");
   Vector S1s = tool6.set_up_new_chambers(num_S1);
   Vector S2s = tool6.set_up_new_chambers(num_S2);
   Vector S3s = tool6.set_up_new_chambers(num_S3);
   int c_wafer = 0; // number of completed wafer
   int act = 1, wafer = 1; // wafer is the number of wafer started
   Vector list = new Vector();
   int k = count_lines(event_list, 0);
   Job event;
   // fill up the tool
   for(int i = 0; i < k; i++)
     {    event = (Job)event_list.elementAt(i);
          String condition =  event.getStarting().substring(0,2);
          if ( condition.equals("LL") && wafer <= num_wafer)  {
             tool6.load_S1(wafer, act, robot_durat, S1_durat, robot_pos,
                           global_time, S1s, list);
             wafer++; act = act + 2;
            }
          else if(condition.equals("S1") && feasible_move(S1s) == 'y')  {
             tool6.load_stage("S1","R2","S2",act, robot_durat, S2_durat, robot_pos,
                              global_time, S1s, S2s, list);
             act = act + 2;
            }

          else if(condition.equals("S2") && feasible_move(S2s) == 'y')  {
             tool6.load_stage("S2","R3","S3",act, robot_durat, S3_durat, robot_pos,
                              global_time, S2s, S3s, list);
             act = act + 2;
            }
```

Example 5 continued.

```
        else if (condition.equals("S3") && feasible_move(S3s) == 'y')
          { // unloading S3
            tool6.unload_stage("S3", "R0", act, robot_durat, robot_pos,
                              global_time, S3s, list);
            act++; c_wafer++; if(c_wafer == num_wafer) {break;}
          }
    }
   // end filling up

   while ( c_wafer < num_wafer) {
     for(int i = k; i < event_list.size(); i++)   {
        event = (Job)event_list.elementAt(i);
        String condition =  event.getStarting().substring(0,2);
        if (condition.equals("LL") && wafer <= num_wafer)  {
            tool6.load_S1(wafer, act, robot_durat, S1_durat, robot_pos,
                          global_time, S1s, list);
            wafer++; act = act + 2;
          }
        else if(condition.equals("S1") && feasible_move( S1s)=='y')  {
            tool6.load_stage("S1","R2","S2",act, robot_durat, S2_durat, robot_pos,
                            global_time, S1s, S2s, list);
            act = act + 2;
          }
        else if(condition.equals("S2") && feasible_move( S2s)=='y') {
            tool6.load_stage("S2","R3","S3",act, robot_durat, S3_durat, robot_pos,
                            global_time, S2s, S3s, list);
            act = act + 2;
          }
        else if (condition.equals("S3") &&  feasible_move(S3s)=='y')
          { // unloading S3
            tool6.unload_stage("S3", "R0", act, robot_durat, robot_pos,
                              global_time, S3s, list);
            act++; c_wafer++;if(c_wafer == num_wafer) { break;}
          }
        if (i == event_list.size() && c_wafer < num_wafer) { i = k; }
     }
   }
  return list;
}
```

REFERENCES

1. Chandrasekaran, Niranjan, “Operational models for evaluating the impact of process changes on cluster tool performance.” MS thesis. Mechanical Engineering Dept., University of Maryland College Park, 1999.

2. Crama, Yves and Joris van de Klundert, “Cyclic scheduling of identical parts in a robotic cell”, *Operations Research*, vol. 45, n. 6, Dec. 1997.

3. Dance, Daren L., Devid W. Jimenez, and Alan L. Levine, “Understanding equipment cost-of-ownership,” *Semiconductor International*, pp. 117-122, July 1998.

4. 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.

5. Herrmann, Jeffrey W., Niranjan Chandrasekaran, Brian F. Conaghan, Manh-Quan Nguyen, Gary W. Rubloff, and Rock Z. Shi, “Evaluating the impact of process changes on cluster tool performance”, *IEEE Transactions on Semiconductor Manufacturing*, vol.13, n. 2, May 2000.

6. International Sematech, “Semiconductor manufacturing process.” Sematech. 29 March, 2000. <http://www.sematech.org/public/news/mfgproc/mfgproc.htm>

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. Kamoun, Hichem, Nicholas G. Hall, and Chelliah Sriskandarajah, “Scheduling in Robotic Cells: Heuristics and Cell Design.” *Operations Research*, vol. 47, n. 6, pp. 821-835, 1999.

9. Kise, Hiroshi, Tadayoshi Shioyama, and Toshihide Ibaraki, “Automated two-machine flowshop scheduling: a solvable case.” *IIE Transactions*, vol. 23, n. 1, pp. 10-16, March 1991.

10. Murphy, Robert, Puneet Saxena, William Levinson, “Use OEE; don’t let OEE use you,” *Semiconductor International*, pp. 125-132, Sep. 1996.

11. Perkinson, Terry L., Peter K. McLary, Ronald S. Gyucsik, and Ralph K. Cavin. 1994. Single-wafer cluster tool performance: an analysis of throughput. *IEEE Transactions on Semiconductor Manufacturing*, vol. 7, n. 3, pp. 369-373, August 1996.

12. Schruben, L. W., “Deadlock detection and avoidance in cluster tools.” *International Conference on Semiconductor Manufacturing Operational Modeling and Simulation*, 1999.

13. 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>

14. Sethi, S. P., C. Sriskandarajah, G. Sorger, J. Blazewicz, and W. Kubiak, “Sequencing of parts and robot moves in a robotic cell,” *Int. J. Production Res.*, vol. 4, pp. 331-358, 1992.

15. 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.

16. 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.

17. Wood, Samuel C, “Simple performance models for integrated processing tools.” *IEEE Transactions on Semiconductor Manufacturing*, vol. 9, n. 3, pp. 320-328, 1996.