INTELLIGENT DECISION SUPPORT AND AGENT-BASED TECHNIQUES APPLIED TO WOOD MANUFACTURING

by

Eman Elghoneimy
B.A.Sc., Cairo University, 1995
M.A.Sc., University of Waterloo, 2000

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

In the
School of Engineering Science

SIMON FRASER UNIVERSITY

Summer 2010

APPROVAL

Name: **Eman Elghoneimy**

Degree: **Doctor of Philosophy**

Title of Thesis: **Intelligent decision support and agent-based techniques applied to wood manufacturing.**

Examining Committee:

Chair: **John Jones**
Associate Professor of Engineering

William A. Gruver
Senior Supervisor
Professor Emeritus of Engineering Science

Shahram Payendah
Supervisor
Professor of Engineering Science

Carlo Menon
Internal Examiner
Assistant Professor of Engineering Science

Robert Brennan
External Examiner
Associate Professor of Mechanical and Manufacturing Engineering
University of Calgary

Date Defended/Approved: June 2, 2010

Declaration of
Partial Copyright Licence

ABSTRACT

A rough mill is a manufacturing facility where loads of lumber of approximate dimensions are cut into components of specific sizes, priorities and qualities. These components are used in making furniture, doors and windows.

Lumber is a very valuable natural resource and is a significant expense to the rough mill. By improving the processes in the rough mill, the cost is reduced and the waste of natural material is decreased.

In this research, the operations in a Canadian rough mill are described, and the decisions that operators take are identified. The rough mill scheduling of components on machines is a challenging problem that cannot be solved by traditional methods because the defects in the wood are not known in advance. In addition, the wood sizes are approximate and often inaccurate.

Scheduling algorithms are implemented using constraint satisfaction and heuristic methods. An agent-based system for decision support and simulation is designed and implemented.

Keywords: intelligent systems, decision support, multi-agent systems, scheduling algorithms, operations research, wood manufacturing, rough mill operations, cutlist scheduling, distributed intelligent systems

To my family

QUOTATION

". . . and say, 'My Lord, increase me in knowledge.'" (Qur'an, 20:114)

ACKNOWLEDGEMENTS

It is a pleasure to thank those who made this thesis possible. I would like to thank Dr. Bill Gruver for his supervision and mentorship. I am grateful for Dr. Ozge Uncu for his support. I would like to thank Dr. Robert Brennan for his review of the thesis. I am indebted to many of my advisors and colleagues at the National Research Council and at the Intelligent Distributed Enterprise Automation Lab: Mr. Dilip Kotak, Dr. Yunli Wang, Martin Fleetwood, Hiroshi Tamoto and Nestor Siu.

I would like to show my gratitude to my husband Dr. Mohamed Allam who motivated me in many ways. My children Lobna Allam and Hassan Allam managed to keep me inspired and delighted. I am grateful for their teachers who filled in where I missed.

I am deeply grateful for my parents: Dr. Mohy Elghoneimy and Mrs. Sawsan Mowafi for their encouragement and prayers. My sister Dr. Hanan Elghoneimy and my brother Dr. Ayman Elghoneimy supported me throughout the ups and downs of this process.

I am thankful for many advisors, colleagues, family members and friends who encouraged me in their own ways.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1: INTRODUCTION

1.1 Problem description

Naturally-defective materials such as wood have defects for which location, size and type are not known in advance. Moreover, the parameters of the materials such as length, width and grade are approximate. A schedule for cutting the material into defect-free fixed-size components cannot be done in advance since the processing time of each task is unknown. Therefore, scheduling cannot be done using traditional methods. Moreover, the schedule has to be dynamically updated; once one task is done, the next is scheduled.

In addition to the above challenges, most factories have problems such as adding or canceling orders, and *jammed* machines (*breakdowns*). This requires flexibility in material selection and order scheduling processes. As factories grow and upgrade its machines and tools, new solutions are required to be flexible, adaptable and expandable.

1.2 Current solution in the rough mill

Current solutions depend on a human operator to create a schedule and dynamically schedule a component on a machine once the previous component is done. Such critical decisions are taken on the shop floor while the operator is busy doing other tasks such as sorting cut components or supervising other

workers. Such decisions are rarely optimal, which leads to wasted time and materials, and therefore cost.

Material selection is also based on human operators. Fork-lift drivers select material based on general recommendation from their supervisor.

1.3 Objectives of the rough mill production system

The objectives of a rough mill production system can be outlined as follows:

- Decision support system, to give recommendations to operators for scheduling and material selection.
- Scheduling jobs of unknown processing times, and dynamic replacement scheduling.
- Ability to tolerate changed orders and jammed machines.
- Distributed system with ability to operate in different locations.
- Expandable to new machines or lines of production.
- Ability to upgrade to enterprise integration, monitoring over the Internet or supply-chain management.

1.4 Existing agent and scheduling solutions for manufacturing

Agent-based systems are hailed as the next technology for manufacturing systems due to their flexibility, fault-tolerance and expandability. Agents have been used for manufacturing applications ranging from control, scheduling and

planning to enterprise operations and supply-chain management. A survey of agent-based manufacturing systems is presented in Chapter 4.

1.4.1 Why existing solutions are not fit for naturally-defective material

There is no agent-based solution that is fit for naturally-defective materials. While some agent solutions feature dynamic scheduling, this actually is in the context of fault-tolerance. If there is some interruption to the schedule, then the schedule is modified dynamically. There is no current solution which does scheduling where processing times are unknown in advance. Formally, most manufacturing applications address the job shop scheduling problem, while the problem at hand is parallel machine scheduling with unknown processing times. In addition, decision support for material selection is not addressed in existing agent-based manufacturing systems.

1.5 Overview of thesis

The following chapter presents background on rough mill operations in a Canadian windows and doors manufacturing plant. Then, I present the history and the joint work done on this rough mill project and define the scope of the rest of the thesis and the two challenges that I am addressing. Chapter 4 provides a survey of multi-agent systems in manufacturing. Chapter 5 addresses the first challenge: designing and implementing an agent-based system for decision support and simulation. Chapter 6 presents solutions to the rough mill scheduling problem. Finally, a summary and conclusions are presented.

2: WOOD MANUFACTURING: ROUGH MILLS

2.1 Introduction

In a rough mill, *jags* (bundles of lumber) are stored in a warehouse, and then transferred to a *ripsaw* which cuts the boards into *strips*. Strips are cut by a *chopsaw* into *components* of specific lengths. Components are then processed to produce manufactured products such as furniture, windows and doors. One of the most valuable resources in a rough mill is lumber. A small increase in the overall yield results in large cost savings and better use of natural resources.

Lumber is a natural material which has a variety of defects that are not known in advance. While lumber is usually stored as jags having boards with an average length, width, and grade, these values are subjective and can vary from one mill to another. Therefore, cutting lumber into components is a complicated process that involves dealing with unknown or imprecise dimensions and random defects.

2.2 Information gathering

I was introduced to the operations of the rough mill during an extended visit to C.P. Loewen Enterprises Ltd., a Canadian windows and doors manufacturing company in Steinbach, Manitoba. The purpose of this trip was to gather information on the rough mill production system, with focus on the cutlist scheduling process, the work done by operators, and the workflow between

different operations [21]. I stayed there for two weeks to interview operators and managers, document operations and work on the line to learn about the process by practice [37]. This was a part of an NSERC project with the objective of improving the operations in the rough mill.

2.3 Rough mill operations

As required, a jag is brought from the warehouse by a forklift to a lumber feeder that presents one board at a time to the ripsaw which cuts the board lengthwise into strips. Strips are inspected and sorted by an operator and are passed through a conveyor belt one at a time to the chopsaw. The chopsaw cuts the strips into components of different lengths, based on the nature of the defects and the components being sorted. These components are routed to the appropriate sorting bin called *kicker*. Finally, the components are inspected by operators and are stacked into loads of components of equal dimensions. Figure 2.1 shows the operations in the rough mill with two lines of production.

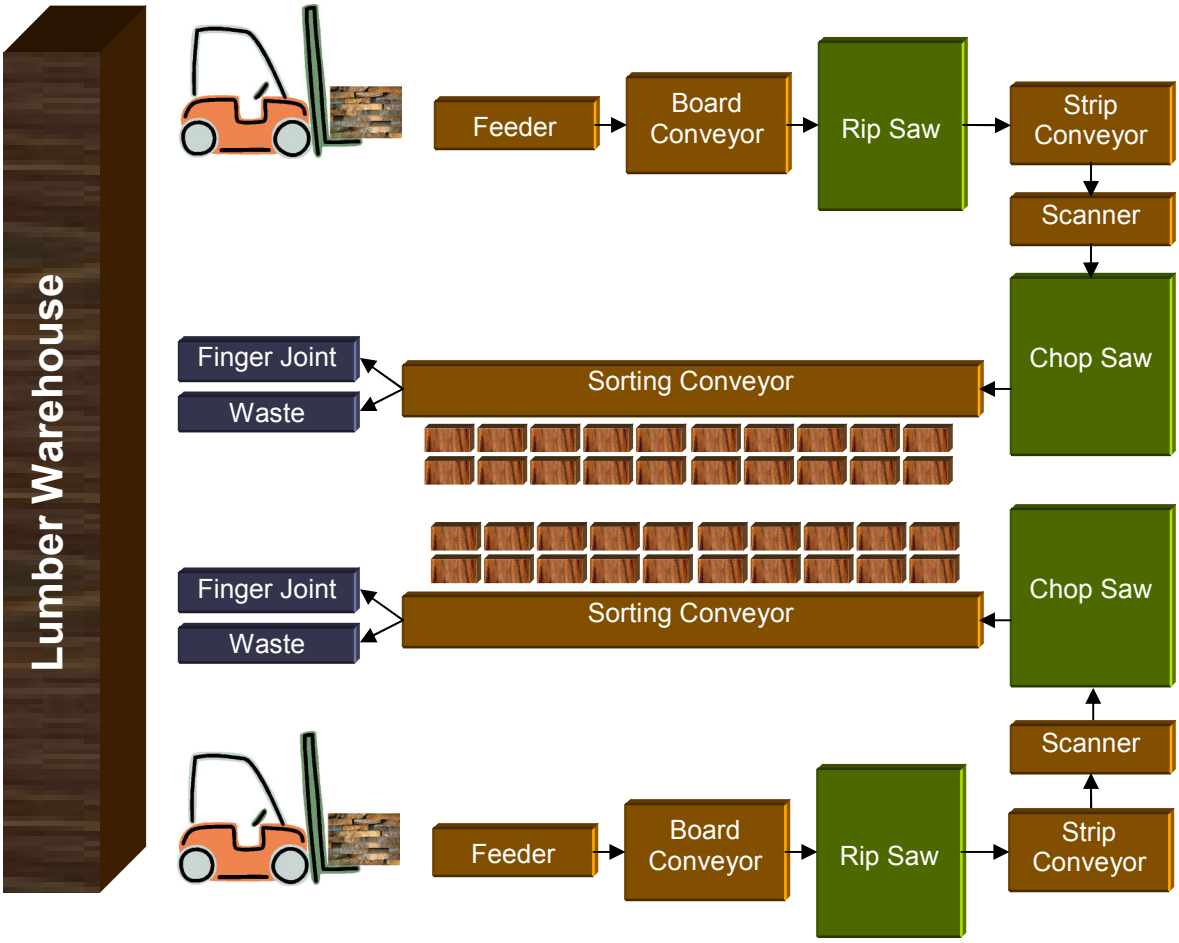

Figure 2.1 Rough mill production lines: Forklifts transfer jags from the warehouse to both ripsaws. Chopsaws cut wood into components of specific sizes.

2.3.1 Lumber warehouse

Jags are stored in the warehouse. The jags are characterized by mill, species, grade, width, and length. Forklift operators select jags from the warehouse to present to the ripsaw feeder based on guidelines from their supervisor.

2.3.2 Ripsaw

The ripsaw processes the boards of the jag to produce strips which are conveyed to the chopsaw as shown in Figure 2.2. The ripsaw cuts the boards of lumber into strips of different widths. The commonly used widths are 57, 67 and 22 mm. The 22 mm strips (called *rippings*) are sent for manual chopping and thus are not routed to the chopsaw. The cutting pattern of the board is based on the pattern of the blades of the ripsaw. A *fence* automatically locates the board across the saws to produce maximum board yield. The ripsaw scanner cameras take a top view of each board to determine its exact dimensions.

An operator inspects and sorts the strips as they come out of the ripsaw. 57 and 67 mm are conveyed to the chopsaw and the 22 mm strips are separated. Strips less than 22 mm wide are considered waste and are conveyed to a wood grinder. The operator also inspects all the pieces and some of the defective 57 and 67 strips are carried back to the ripsaw to form 22 mm strips, other defective strips are thrown away as waste.

The operator enters the jag information to the software and attaches the jag information to a marker board. This board is passed on to the chopsaw indicating the end of the jag.

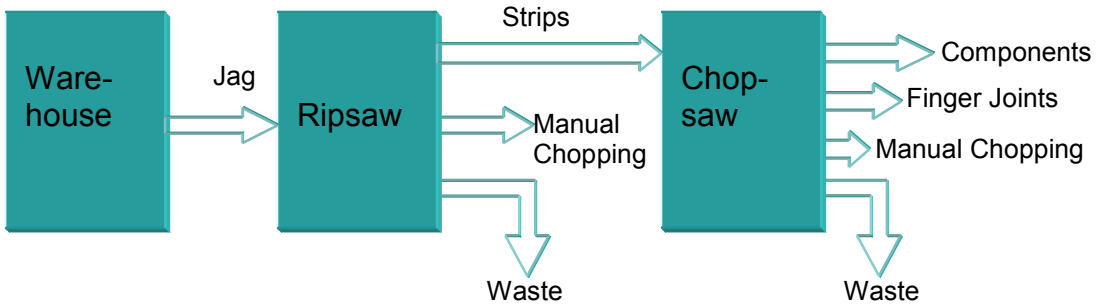

Figure 2.2 Flow of lumber in the rough mill

2.3.3 Conveyor

There are four conveyors in series, each with different length and speed. Conveyor one carries ripped strips parallel to the conveyor's moving direction, with several strips ripped from the same board placed side by side, and with one foot space between the strips ripped from different boards. At conveyor two, the orientation changes ninety degrees as strips are perpendicular to the moving direction of the conveyor. Only one strip is moving at a time with a one foot space between each strip. Conveyor three is similar to conveyor two except that it has a different length and speed. Conveyor four is similar to conveyor one, except that only a single strip is transferred at a time.

There is a photocell sensor in the buffer between the second conveyor and third conveyor. If there is more than one strip in that buffer, conveyor three and conveyor four keep moving while conveyor two stops. Conveyor one continues to move until it is full, at which time the conveyor and ripsaw are stopped.

2.3.4 Chopsaw

The chopsaw receives 57 and 67 mm wide strips from the ripsaw. The chopsaw *scanner* views the four sides of the strip and detects the different types of defects in the strip. According to the components scheduled on the line (*cutlist* or *cutting bill)*, the chopsaw cuts the strips into the desired lengths of components, and chops off the defects into waste bins as shown in Figure 2.2. Smaller components are cut as *finger joints*[1], used for manual chopping or collected as waste. Several operators inspect and sort the components into loads. Defective components could be fed back to the chopsaw, sent for manual chopping or thrown away as waste.

When the chopsaw operator receives the board marking the end of the jag, he enters the jag number into the chopsaw optimization software to keep track of the data per jag. This does not affect the process of scheduling components on the cutlist.

The supervisor receives work orders with breakout due date within six days. She assigns work orders - with the same thickness - that are due within two days, to the chopsaw operator. These work orders form the *order list*. Usually work orders are received at the chopsaw at the beginning of the shift. Rush orders may be received during the shift.

The operator creates the cutlist based on information in the work orders, according to certain decision criteria. When one or more of the components are

[1] Finger joints (FJ) are low priority products. There are quality numbers associated with FJ, each with assigned thickness, width and grade. There is no specific length for finger joints, but a range of 180 to 600 mm.

done, a new component is scheduled based on the order list and the current cutlist. The operator enters the components on the cutlist into the chopsaw optimization software when they are scheduled. The new cutlist influences the process of jag selection as well as the setup of the ripsaw.

2.3.5 Kickers

A kicker is a pneumatic arm that pushes the components from the conveyor to sorting stations where they are inspected and sorted by the operators.

Each kicker has its own properties that restrict the length of the component assigned to it. This is due to the physical layout of the sorting stations. For example, kickers close to each other with no barrier between them are not assigned components with similar sizes so that they do not get mixed together during sorting. Another example is that large components can pile up against short ones and fall down, which creates a safety hazard. For each production line, there are ten kickers for assignment to 20 - 25 components at the same time (depending on quantities required, due dates and floor space).

2.4 Decisions in the rough mill

By observing the operations in the rough mill and talking to operators and domain experts, I identified the exchange of information that takes place between different rough mill operations as shown in Figure 2.3. It can be summarized as follows: the order list is the list of all the components to be cut in the next few days (work orders). The chopsaw operator selects a subset of these components

(cutlist) and schedules them on the kickers of the chopsaw. Another operator selects jags from the warehouse according to the cutlist. The ripsaw operator sets the arbor and priority of the ripsaw based on the cutlist and the selected jag.

While the lumber processing starts with a jag at a ripsaw and ends with cut components in the sorting stations, the decisions in the rough mill start at the cutlist on the chopsaw, then the jag selection and the ripsaw settings.

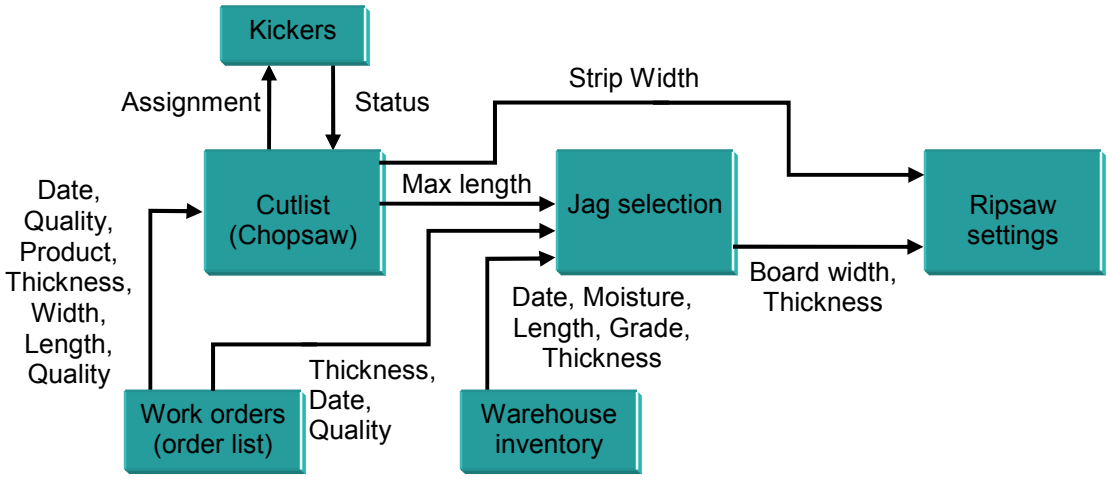

Figure 2.3 Flow of information in the rough mill

2.4.1 Jag selection problem

Jags are received from different mills and stored in a warehouse. The supervisor performs the jag selection process, and asks the forklift drivers to bring jags from the inventory with the selected properties, and drive them to the ripsaw as required. Each jag is identified by parameters such as width, length, grade, mill, thickness, moisture level, and date received. One jag at a time is selected from the warehouse and transferred by a forklift to the ripsaw feeder. A

solution of the jag selection problem requires choosing from the warehouse the best jag that can be used to cut components scheduled on the chop line forming the cutlist.

The jag selection is based on the following criteria:

1. The thickness of the jag must match the components in the work orders being processed.
2. The length of the jag must be larger than the longest component in the cutlist. For example, if the longest component was 10 ft long, the jag length would be 12-16 ft depending on the grade.
3. The moisture level of the jag must be between 8 and 12.
4. The date of receiving the jag: it is preferred to use older jags first.
5. Grade selection:
   5.1. The *quality*[2] of components in the work orders sometimes affects the grade of lumber selected. For example, a good grade of lumber is required when the cutlist includes large quantities of long components that allow no defects.
   5.2. The due date of components in the work order. If items on the cutlist are due soon, a good grade of wood is used to finish the components quickly. If the components are not due soon, it is ok to use lower grade even if it takes longer.

[2] Quality is a code of the allowed type and location of the defects in cut components.

5.3. For 6/4-thickness lumber: higher grades (clear, door and moulding) should only compose 11% of the jags used, and 89% for shop grades.

Based on the above criteria, the supervisor would ask the forklift drivers to bring jags with certain properties, for example, thickness 6/4, high grade, 14 or 16 ft long, with preference to older jags.

In case of changing jag width, the new width should be coordinated with the ripsaw to change ripsaw parameters (arbor).

2.4.2 Rough mill cutlist scheduling problem

Components from several work orders are selected and grouped together to form the order list. The chopsaw operator examines all the work orders and adds up the quantities for components that have the same thickness, quality, width and length.

All components in the cutlist must have the same thickness. This produces component lists with same thickness but different quality, length, width and quantity. Each kicker is assigned one or more of these component lists.

For example, line 2 is processing thickness 6/4 and kicker 3 is assigned two components: The first has quality Q51, width 57, length 1180 and quantity 200. The second component on kicker 3 has quality Q21, width 67, length 1510 and quantity 100. Kicker 12 is assigned one component that has quality Q111, width 67, length 480 and quantity 180. And so on with the rest of the kickers.

The operator keeps track of components belonging to different orders, because they need to be handled separately by the forklift drivers. For example,

if there are 50 components required for one order and 30 for another order, the operator write it down as (50+30) not 80. Components with large quantities (more than 400) are split into smaller groups to be easily handled by the forklift drivers. For example, if the quantity required for a certain component is 1000, it is scheduled on the chopsaw as 1000, but will be loaded as (250+250+250+250).

The cutlist is created when switching to a new width or thickness. Otherwise, it is only updated when a kicker becomes available (when the component assigned to it is done).

Components are chosen to be scheduled on the kickers based on the decision criteria detailed in Chapter 6.

Therefore, the rough mill cutlist scheduling is the problem of selecting a list of components to be scheduled on the chopsaw (cutlist) from the list of several components to be cut in the next few days (order list). The cutlist should present a good mix of components of different lengths to the chopsaw. Components must be done by their due dates, while giving priority to certain types of components. I present several solutions to this problem in Chapter 6.

2.4.3 Ripsaw settings

The ripsaw has control of two main parameters of operation: priority and arbor selection.

2.4.3.1 Priority

The priority selection is based entirely on the desired strip width. The priority can be changed during the operation of the ripsaw. There are three priorities for the ripsaw

1. P57: more 57 mm strips are desired
2. P67: more 67 mm strips are desired
3. YLD: both 57 and 67, maximum yield of the board is desired.

2.4.3.2 Arbor selection

Selecting one of several arbors sets the distance between the blades of the ripsaw. The change of arbor requires extensive machine adjustment; therefore it is preferred not to change it often. In practice, the priority and the arbor are usually changed at the same time.

The operator selects the arbor based on the following jag parameters:

1. The board width code,
2. The strip width (priority), and
3. Thickness

The operator chooses the arbor based on a table. This table states the arbor number recommended for each combination of the above parameters, in order to provide maximum yield. For example, arbor 252 is recommended for board width code of 20, thickness 6/4 and priority YLD. If only 57 or only 67 are desired, the arbor must be changed (codes 263 and 267 respectively).

Ripsaw settings are straight-forward and require only a look-up table, therefore this decision in the rough mill is not investigated any further.

2.4.4 Thickness switch-over process

The switch-over process is carried out when a line of production is switching from one thickness to another. The supervisor decides to switch to a new thickness according to the work orders and their due dates. Both the ripsaw and the chopsaw have to be adjusted to process the new thickness. Jags with the new thickness are selected and presented to the ripsaw.

The switch-over process takes 20-30 minutes, therefore it is preferred not to be done so often. Usually a line of production runs the same thickness for one to three days.

If the quantity of components of a certain thickness is very small, e.g. 5/4 and 8/4, they are not processed until about one day before their deadline. This is to process all of the components of that thickness together. If the quantity is less than 800 components, the work orders are sent to manual chopping (rework) instead of the chopsaw.

The switch-over process results in new order lists, which are considered as the input to the scheduling. Therefore, this decision is not addressed in this work.

2.5 Summary

In this chapter, the operations that take place in a rough mill were described. Loads of lumber (jags) are stored in a warehouse. As required, one

jag at a time is brought to a lumber feeder, which feeds one board at a time to a ripsaw. The ripsaw cuts the boards into strips lengthwise. Strips are inspected and sorted by an operator, then fed one at a time to the chopsaw. The chopsaw cuts the strips into components of fixed lengths. The components are inspected by operators and are sorted into loads.

Two major decision problems that are addressed by human operators are defined: jag selection and cutlist scheduling. Ripsaw settings and thickness switch-over are other decisions in the rough mill, but they are straight-forward processes and therefore do not require further investigation.

3: HISTORY AND JOINT WORK

3.1 Introduction

In this chapter expected benefits of developing a decision support and simulation system are described. Next, I describe the previous work related to the rough mill that has been done before I joined this project. I also describe the joint work done with other researchers. The rest of the thesis contains my individual contribution.

3.2 Expected benefits

The following are the expected benefits of developing a decision support and simulation system for the rough mill:

- Cost savings through reducing wasted lumber;
- Cost savings by better utilization of grade;
- Time savings by automating (or providing support) for component scheduling and jag selection decisions;
- Consistent and standardized decisions among different operators;
- It allows the system to be operated by inexperienced operators when experienced operators are not available or busy;
- It can be used as a training tool for new operators;

- Operators have the choice to change the recommendations of the system, providing flexibility;
- The system can be used with a simulator, allowing the user to test the outcome of suggested decisions before running it in production;
- Operations personnel can use it to examine their existing business rules and discover new ways of operation;
- The system helps researchers understand operations in the rough mill; and
- Researchers can use the simulation as a tool for evaluating different methods to improve performance.

3.3 History

In the following subsections, I outline the work that was done by others on this project before I started my research, namely, simulation and case-based jag selection.

3.3.1 Simulation

The rough mill operation was analyzed using Quest software, which is based on discrete- event simulation with 3D visualization [13]. An application (Editor) developed in Delphi is used for user interface to the simulator. The rough mill operator can plan operation in the rough mill using this package. By selecting different jags and different component lists for running the simulation, the operator can predict the outcome of the mill and run several what-if scenarios.

The Quest simulation model was run at the National Research Council in Vancouver, BC, and was used remotely by the rough mill operators. Later in this project, the simulator and editor programs were re-engineered to Java to be able to run locally in the rough mill, without the need for remote connection or special licensed software. 3D simulation was not used, and run time of the simulation was decreased from hours to a few seconds.

3.3.1.1 Ripsaw

The board generation is done by using two different random generations. The width is derived from a specific width distribution that is linearly interpolated, while the length of the board is done by using a triangular distribution that has a base of one inch and center at the average length of the jag.

Once the board is created, it is ripped by the ripsaw according to the arbor configuration. The arbor configuration includes two criteria: the arbor set and the priority as described in section 2.4.3.

Once the arbor configuration is set, the boards are ripped by using a look-up table. For each configuration set, there is a table that is ordered by the width with number of resultant strips, and also includes associate value for each width category. The look-up table technique involves finding the appropriate length groups that allow the ripping to be possible, and then find the group with the highest value to determine the number of strips to be produced.

The simulation rip time is five seconds per board, regardless of the number of strips produced per board.

3.3.1.2 Conveyor

The conveyor transfer strips from the ripsaw to the chopsaw. When the conveyor is full, the ripsaw is blocked.

3.3.1.3 Chopsaw

The chopsaw cuts the strips crosswise into components with lengths and values assigned to the kickers. Mathematically, this is known as the knapsack problem. Due to defects in the wood and the desire for a fast algorithm for simulation, a heuristic algorithm is used to simulate the chopsaw, which can be summarized as follows:

For every strip, generate clear pieces randomly from a cumulative density function which is based on historical data. If the clear piece is shorter than any component in the cultist, it is used for finger joint or waste. For every clear piece of lumber, find the component in the current cultist that yields the maximum value that can be cut from the clear piece. Then, subtract the length of the component from length of the clear piece and decrement the required quantity of the component. Repeat for the remaining length of the leftover piece to cut more components. If no component can be cut and there is leftover from the clear piece, then the leftover is considered as finger joint or waste. More clear pieces are generated in the same fashion until the sum of clear piece lengths exceeds the strip length.

Simulation chop time is set to 0.5 seconds per cut.

3.3.1.4 Other simulation systems

In this section, I mention other simulation systems that are available for rough mills.

ROMI-RIP and ROMI-CROSS [25] are simulators for rip-first and cross-first rough mills respectively, which were developed by the USDA Forest Service. Rip-first processing involves gang ripping the board into strips. Next, these strips are crosscut to primary part (component) lengths, either specified or random. Chop-first processing cuts the boards to primary part lengths and removes any wide defective areas. Next, the board segments are straight-line ripped to the required widths, specified or random. Part (component) scheduling is done by ranking of parts by the user. Another example of rough mill simulator is a simulation system that was developed for crosscut-first (chop-first) roughmills [38].

Since the simulation described earlier was developed for the particular rough mill of interest (Loewen), I decided to use it in my research rather than ROMI-RIP.

3.3.2 Jag selection: case-based reasoning

In this approach, historical data is used to select jag types that produce similar length distributions as the cutlist. Jag types are defined by mill, grade, production line, length code, and thickness. An ideal jag for a given cutlist allocates production components equally over the entire production. Hence, a similar proportion of produced components to the requirement of the cutlist

corresponds to a suitable jag type. The length distribution is used to describe the differences of production capabilities of jag types.

A distribution of the percentage of cut quantities in each predetermined sort length bin is calculated for each jag type based on the average of historical cut quantities achieved by using jags that belong to the corresponding jag type.

The case-based reasoning (CBR) algorithm can be summarized as follows [99]. For a given cutlist, the distribution of the percentage of required quantities for the components in the cutlist is created. The difference between the distribution of the cutlist and the historical distribution of each jag type is calculated. This difference, which can be considered as a penalty measure, is used to sort the jag types. Starting from the most suitable jag type, the jags available in the inventory (current data) are selected for each jag type until a maximum allowed number of jags has been reached. The list of jags for each jag type is sorted by using criteria such as age (it is preferred to use older jags to increase the turnover in the inventory), proximity of the board footages (which is a measure of volume used in lumber industry) of the jag and the cutlist, and the suitability of the width of the selected jag for the given ripsaw arbor configuration and priority. The sorted list of jags is presented to the user to select the jag.

3.4 Joint work

The following sections outline the work done in collaboration with other researchers [21].

3.4.1 Scheduling: Genetic algorithms

The scheduling problem can be viewed as the problem of searching for one cutlist solution from a large solution space of possible cutlists. Genetic Algorithms (GA) are a suitable approach for searching large solution spaces. Therefore, GAs are used as a solution to the cutlist scheduling problem [58].

The objective function is chosen to represent the heuristic rules and the constraints of the problem which is presented in detail in Chapter 6. The objective function is composed of two functions: the first is the *due date* objective that favours components that are due first, the second is the *length distribution* objective that favours cutlists with good length distributions.

As a problem representation for the Simple Genetic Algorithm (SGA), a variable x_i was assigned to each entry in the order list, and it was set to 1 if the item was chosen to be in the cutlist, and 0 if not. The x_i variables were concatenated together to form a binary solution string. To handle the kicker constraints, a heuristic algorithm is used to pick the kicker assignments to satisfy physical constraints. If the algorithm was unable to assign all the chosen items to a kicker, the solution cutlist is deemed infeasible and discarded. Two way tournament selection and single point crossover operators are used. To improve the algorithm's performance, the order list items are pre-sorted by length.

The Ordering messy Genetic Algorithm (OmeGA), specialized for permutation problems, searches through different orderings of a set of numbers to obtain the best ordering. To use OmeGA, a method to decode the number orderings into a cutlist with their kicker assignment was developed. Each number

in the ordering represented an item in the cutlist. Beginning with the first item specified by the first number in the ordering, it is assigned to an available kicker. The next item in the ordering is then assigned to the best fitting kicker remaining. This process continues until all available kickers are assigned. The solution decoding process ensures that every sequence represents a valid cutlist, so all solutions generated are feasible.

For replacement scheduling, the objective function used by the GA was used. A single unscheduled component is added to the currently scheduled components to form a temporary cutlist, and the objective function value is calculated. The process is repeated and the unscheduled component that is part of temporary cutlist with the highest objective function value is then chosen as the replacement component.

3.4.2 Jag Selection: MCDM and FMCDM

Jag selection is a multi-criteria decision making process. There are four criteria for selecting jags [98]

1. Yield: Yield is the volume of wood in the final product as a percentage of volume of wood in a jag. A higher yield of a jag indicates higher volumetric utilization of the wood. Average values of chopsaw yield of jag types in the historical file are utilized to measure the yield.
2. Material Cost: Material cost of a jag is the standard cost (average unit cost of that particular grade) times its board feet volume.

3. Percentage of orders satisfied: Usually a cutlist is completed by using several jags. Using one jag, only part of a cutlist could be produced. Matching the produced cutlist, the cumulative length distribution of components produced from each jag type, with the required cutlist can be used to measure the percentage of orders satisfied for the jag.
4. Processing time: Production cost can be reduced by finishing orders with processing time as short as possible. The average processing time of a jag type indicates its potential capabilities in processing time.

The TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) multiple criteria optimization method is used to rank alternative candidates. The chosen alternative has the shortest distance from the ideal solution and the farthest distance from the negative-ideal solution. TOPSIS is a fast method compared to methods such as dynamic programming.

Depending on the different situations, the importance of the above four criteria may be different. If the cutlist includes many components that are urgently required, the percentage of orders satisfied is more significant than other criteria. If there is a longer lead time, then the yield and cost become more important. To achieve overall optimization, an appropriate weight must be assigned to each criterion.

The user is expected to input the weights for different criteria for each cutlist. Then, the TOPSIS method is used to rank alternative jag types, thus recommending jags that are most suitable to produce the input cutlist.

In the above method, it might be difficult for the operator to set individual weight values for each cutlist. In an alternative approach to solve the jag selection problem, a fuzzy system is used to set the weights of the multiple criteria using data from the cutlist and linguistic fuzzy rules. The fuzzy system is fast and runs in real-time.

A Fuzzy Inference System (FIS) is used to set the pair-wise comparison weights. The inputs of the fuzzy system are statistics from the cutlist. Since it is difficult to make rules linking cutlist information directly to the weights, pair-wise comparison values were used as an output of the FIS. Next, the AHP (Analytic Hierarchy Process) method is used to convert these pair-wise comparison values into individual weights to be used for the multiple criteria method described above. The TOPSIS method mentioned above is extended to Fuzzy TOPSIS with weights from the output of the AHP process.

3.4.3 Jag sequencing methods

Jag sequencing is the process of selecting a list of jags to be presented in sequence on the ripsaw. Four alternative methods for jag sequencing are discussed in this section [60]. The first method is ZI (Zero Intelligence), which repeatedly selects jags based on the current cutlist. The second method uses beam search based on a fitness function. Fuzzy rulebase based AHP is the third method presented. The last jag sequencing method is weight determination using fuzzy rulebase tuned by genetic algorithms.

In the *ZI method*, the jags are selected repeatedly by using the CBR jag selection approach until the order is fulfilled.

Beam search method: Current conditions are defined by the cutlist, order, kicker assignments, ripsaw and chopsaw statistics. A new list of suitable jags is created for each position in the sequence for which a jag is selected by using the CBR jag selection algorithm. Beam search limits the window of neighborhood creation to creating the ones for only *q* most suitable candidates in the current state. The fitness function that is used to evaluate the candidates is based on the statistics collected through the simulator [61].

Fuzzy rulebase based AHP (Simplified FMCDM) method utilizes conventional AHP with fuzzy pair-wise comparisons. The weights are determined by the Row Means of Normalized Columns method. The jag types with scores calculated by using these weights are ranked by using the conventional TOPSIS method as explained in a previous section.

AHP method is used to find the most suitable weights associated with the decision criteria used to rank the jag types. The pair-wise comparisons of importance of decision criteria under different cutlist characteristics, made by domain experts, were used to determine these weights. The uncertainty in these comparisons for different cutlist conditions is captured by using fuzzy rulebases.

Weight determination using fuzzy rulebase tuned by GA: GA metaheuristic search mechanism is used to identify the most suitable weights associated with the decision criteria without pair-wise comparisons acquired from experts. A

fuzzy rulebase model is proposed to imply the crisp weights under different cutlist conditions [59].

3.5 Goal of this research

Since the jag selection decision problem was addressed in detail, it will not be further investigated in this research. Similarly, the simulation algorithms are already developed and are not addressed. The rest of the thesis focuses on two challenges: solving the scheduling problem as well as providing an agent-based design and implementation to allow for flexibility and future enhancements.

3.5.1 Agent-based system architecture

Design overall system architecture using the multi-agent paradigm, and implement it using existing simulation algorithms and new or previously developed decision support algorithms.

3.5.2 Scheduling

The scheduling problem is a significant and complicated decision in the rough mill. My second challenge is to develop scheduling algorithms that address the unique features of the rough mill scheduling problem.

3.6 Summary

In this chapter, the history of this research was presented. The simulation was first developed, and then some algorithms for decision support were developed for the rough mill. The rest of this thesis focuses on two goals: developing agent-based system architecture, and solving the scheduling

problem. The next chapter provides a survey of agent-based manufacturing systems.

4: AGENT-BASED MANUFACTURING SYSTEMS

4.1 Introduction

Manufacturing is a highly dynamic process that requires real-time, flexible, reactive decisions. This chapter presents a survey of agent technology and multi-agent systems, and their suitability for manufacturing applications [20]. An overview of agent technologies that have been developed for scheduling, planning, and control of manufacturing systems is presented. Hierarchical, heterarchical, and hybrid architectures for manufacturing systems, learning methods, and features of selected technologies and applications are discussed. Finally, Internet-enabled manufacturing systems and Holonic Manufacturing Systems are discussed.

4.2 Agents and multi-agent systems

Agent-based computing is a multi-disciplinary field rooted in Distributed Artificial Intelligence and distributed objects technologies [53].

4.2.1 Artificial intelligence

Artificial intelligence (AI) is a computing field that goes back to the 1950s. Its goal is to understand and build intelligent entities [75]. AI currently encompasses a large variety of sub-fields, ranging from learning and perception, to game theory, proving mathematical theorems and diagnosing diseases. Current research areas in AI include natural languages, machine learning,

automated reasoning, computer vision, knowledge representation, robotics, search methods, cognitive sciences and agents.

4.2.2 Agent definition

There are several definitions of an agent. A discussion of agent definitions is presented by Franklin and Graesser [74]: “an autonomous agent is a system situated and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to affect what it senses in the future.”

For the purpose of this work, I follow the approach of Jennings and Woolridge [55] and Wooldridge [54], and define an agent as a software component or entity that is capable of autonomous action, has partial control on its environment, and can decide for itself what it needs to do in order to satisfy its design objectives. An agent-based system is one in which the key abstraction is that of an agent. Multi-agent systems (MAS) contain multiple agents interacting together to pursue goals beyond their individual capabilities.

4.2.3 Agent characteristics

Agents are usually autonomous, intelligent, flexible, reactive, proactive and social. An autonomous agent should be able to act without direct intervention of humans or other agents and should have control over its own actions and internal state. An intelligent agent is a system that is capable of flexible autonomous action in order to meet its design objective. Flexibility of an agent implies that an agent is responsive to the environment, user, or other agents. A

proactive agent exhibits goal-directed behavior and takes initiatives. Social agents interact together with other agents and humans to achieve their goals and help others with their activities. Additional aspects of agents are mobility and adaptability. Agents are also persistent and continuously perceive their environment [75].

Lockemann [77] identify four properties of a software agent: it resides in an environment and interacts continuously with it; it offers a useful service while its internal processes remain encapsulated; it is capable of autonomous action in order to meet its design objectives (provide its service); and the autonomy of the software agent is guided by its own goals. Furthermore, intelligent software agents are characterized by being reactive to the environment, and balancing their goal and reactive behaviour. Intelligent software agents may also take initiatives in pursuing their goals, and may interact with other agents to provide their services.

Agents can be distinguished from objects in that they are autonomous entities capable of exercising choice over their actions and interactions. However, they may be constructed using object technology. Agents do not invoke methods (actions) on agents, rather request actions to be performed. The decision to act upon the request or not lies within the recipient. Moreover, agents typically run in their own thread of control, as opposed to standard object systems which have a single thread [48]. There are several agent-oriented methodologies that can be used to develop multi-agent systems. These methodologies are listed and compared in reference [6].

4.2.4 MAS applications

Multi-agent systems are used in a wide range of applications requiring flexibility and adaptability to a rapidly changing environment due to their distributed nature, modularity, and ease of implementation. Examples of such applications are [77] [1] process control, manufacturing, information management, text and web searching, wireless sensor networks, nano-technology, distributed energy management, e-commerce, health care including patient management, intelligent personal assistant software, computer games, air traffic control, satellite imaging data, network security and multi-robot teams.

In a manufacturing plant, new tasks are added during operation, machines may fail unexpectedly, and order deadlines can change. In addition, several machines could perform the same type of tasks and the number of tasks is usually higher than the capacity of the machines. All the above factors make manufacturing a highly dynamic process, which requires real-time, flexible, reactive decisions. To meet these challenges in manufacturing, multi-agent systems have been applied. An early use of agent-based systems in manufacturing was introduced by Shaw [47] where distributed planning and negotiation techniques were used for flexible scheduling and control.

Jennings and Bussmann [57] suggest that MAS have three key aspects that make them suitable for control and software engineering in general. These aspects are decomposition of large problems into smaller problems, abstraction by defining a simplified model of the system, and organization by defining and managing the relationship among different components.

Agent technology is widely recognized as a promising paradigm for the next generation of manufacturing systems [90]. Agent technology satisfies the fundamental requirements of modern manufacturing, including enterprise integration, distributed organization, agility, scalability, fault-tolerance and integration of humans with software and hardware [49].

Holonic Manufacturing Systems (HMS) are based on highly decentralized manufacturing control systems built from autonomous, cooperative intelligent systems [72] [87] [35], and these can be viewed as a specialized type of multi-agent system.

4.3 Background of Agent-Based Manufacturing Systems

Agent-based manufacturing systems can be classified according to several criteria. In the following section, I present examples of agent-based manufacturing systems and discuss them according to these criteria.

4.3.1 Scope of the manufacturing systems

Some manufacturing systems are concerned with supply-chain management and integrating shop floor operations such as scheduling, order management, material handling, monitoring, and simulation. I refer to these as “integrated systems.”

Other systems are concerned only with scheduling. These systems might also include planning and control. Shen provides an in-depth look at manufacturing scheduling [89] and process planning and scheduling [93].

In the following sections, I present examples of integrated systems, and scheduling, planning and control systems. I focus on recent developments. Shen and Norrie [88] describe earlier MAS projects in intelligent manufacturing.

4.3.2 Physical agents versus logical agents

In manufacturing systems, agents represent physical machinery, e.g., machine controllers, conveyors and automated guided vehicles. Agents can directly connect to programmable logic controllers through a software wrapper. Alternatively, agents represent operators or business processes. For example, a scheduling agent can perform the task of an operator who is responsible for scheduling. Several tasks are carried out logically at the same time.

4.3.3 Hierarchical, heterarchical, and hybrid architectures

MAS architectures can be classified in three categories. The first category is a hierarchical architecture in which agents are arranged in a hierarchy with a master-slave relationship. The second category is a heterarchical architecture in which agents have a peer-to-peer relationship. This architecture is distributed, with decentralized communication to avoid bottlenecks that occur in centralized systems. It is frequently used for scheduling and control applications.

The third category is the hybrid architecture which has federating agents that help with the communication of agents, and maintain a global view of the system. Hybrid architectures also include modular systems such as integrated systems.

4.3.4 Learning ability

Learning is “the acquisition of new knowledge and motor and cognitive skills, and the incorporation of the acquired knowledge and skills in future system activities, providing that this acquisition and incorporation is conducted by the system itself and leads to an improvement in its performance” [30]. Panait [43] presents a survey of the applications of machine learning to such problems in the MAS area.

Soft computing techniques such as fuzzy logic, neural networks, and genetic algorithms are widely used in manufacturing applications of control and decision support.

4.4 Agent-based systems in manufacturing

4.4.1 Integrated systems

The following describes examples of MAS used in integrating operations and processes in manufacturing applications.

4.4.1.1 Super-agent architecture

Early systems involve a super-agent that supervises subagents. An example of this approach includes subsystem agents for receiving, shipping, and maintenance, and a super-agent that contains knowledge of the master production interaction [68].

Marik et al. [84] present an integrated system at the control level. A meta-agent organizes work and performs auto-configuration.

4.4.1.2 Hybrid architectures

Since super-agents formed a bottleneck in the system, subsequent systems moved towards a modular design or hybrid architecture. The following are examples of such systems.

An MAS for printed circuit board manufacturing is presented by Sikora and Shaw [70]. Agents represent physical entities as well as logical decision modules. Agents exist on the system, process, and decision levels. Coordination mechanisms for each level are detailed. A constantly updated knowledge base is used for learning. Performance tests indicate that the MAS is superior to a traditional system.

Shen et al. [91] presents a collaborative agent approach for an integrated production planning system. Agents are categorized as interface, collaboration, knowledge management, template mediator, dynamic mediator and product model database.

Metamorph I is a planning and scheduling system with a hybrid architecture. A mediator agent learns from history and propagates behaviors to the future. This system was modified to produce Metamorph II that integrates design, planning, simulation and execution for supplier, customer, and partners. Shen et al. [92] includes an in-depth look at hybrid architectures.

An MAS for manufacturing integration is presented by Deen and Fletcher [78]. Agents occur on three levels: coordination agents, skill agents, and class minder agents. Rescheduling negotiation is done using a temperature model.

ManAge is an agent based architecture for flexible manufacturing control [82]. This framework was tested in an electronics manufacturing demonstration test system. The agent architecture is based on the Planning, Execution, Monitoring (PEM) concept. Agent models are implemented in Java using Visual Café rapid application tools.

An MAS for order fulfillment in a virtual foundry fab is introduced by Yu and Huang [12]. Agents represent subprocesses in the system. A direct messaging protocol called GMPP (Generic Message Passing Platform) is used for application-to-application communication. The system also includes a learning agent that uses a distributed neural network model for on-line learning.

The PABADIS model is a distributed Manufacturing Execution System (MES) with focus on production management functions: resource allocation, scheduling, and dispatching [14]. Residential agents provide information about resources to other agents. Product agents negotiate scheduling and resource allocation of individual work-pieces using a simplified Contract Net Protocol. Plant manager agents organize the process by performing quality management, reporting, and monitoring. The system was developed using JINI, which facilitates a “plug-and-participate” service.

IntaPS is an multi-agent system for process planning and scheduling that features order, resource, and service (interface) agents [5]. It is implemented using a FIPA-OS tool.

Paolucci and Sacile [49] present PS-Bikes as a case study of a multi-agent control system for manufacturing of custom bikes using Java Agent DEvelopment Framework (JADE).

4.4.1.3 Supply-chain and Internet monitoring

Recent systems involve supply-chain management, virtual enterprise and web-based monitoring and control.

A multi-agent framework for production planning, simulation, and supply-chain management is introduced by Pechoucek et al. [50].

Agentsteel [76] is an agent-based online supply chain system for planning and observation of steel production using an InteRRaP generic agent architecture.

Wise-shopfloor is a real-time web-based monitoring and control for the shop floor using Java 3D models which transmits only sensor data and control commands and deploys control logic [45].

iShopfloor is an Internet-enabled agent-based intelligent shopfloor [94]. A distributed control approach is applied from the virtual enterprise or supply chain to the shop floor and also to each machine.

Shen et al. presents a service-oriented integration framework based on software agents and web services to establish a dynamic collaborative environment for inter-enterprise collaboration [95].

A survey of online simulation-based manufacturing systems (including virtual manufacturing) is presented by Yoon and Shen [32] where simulation is

based on historical data. Current data is used for simulation, and new decisions according to simulation are directly transmitted to execution systems in the shop floor.

4.4.2 Scheduling, Planning and Control

The following are examples of MAS that are used in scheduling, planning, and control.

4.4.2.1 Control systems

A shop control system presented by Park et al. [33] includes one planning agent, three manufacturing system agents, and one control agent.

A robotic manufacturing control system features three types of agents: manufacturing agents, part agents, and interface agents [16]. The agents were implemented using the JACK intelligent agent language in which agents have a Believe-Desire-Intention (BDI) architecture.

A survey of intelligent MAS in control applications, involving fuzzy logic and genetic algorithms is presented by Naso and Maione [17]. A simulation of a benchmark system shows the superiority of the agent-based systems over conventional systems.

Tichý et al. present a control and planning system based on a three-layer modified hierarchal structure with directory facilitator agents [64].

Marik et al. [86] describes an industrial control system with high-level control agents implemented in JADE and low-level control agents in a programmable logic controller.

Brennan [71] provides a survey of recent real-time control systems, and evaluates them against several primary needs for next-generation systems.

Dynamically Integrated Manufacturing Systems (DIMS) [19] manage optimal fulfillment of customer orders while simultaneously considering alternative system structures to suit changing conditions by integrating manufacturing planning and control decisions with systems reconfiguration and restructuring.

A mobile agent-based framework supports dynamic deployment of control algorithms and tasks in automation systems [80]. The framework is based on a mobile agent system called Mobile-C and uses Ch, an embeddable interpretive C/C++ environment for mobile agent execution.

4.4.2.2 Hybrid scheduling systems

The following systems include federating agents for scheduling and/or control.

Ramos [10] describes a system for dynamic scheduling with resource agents and task agents. A resource manager agent negotiates a schedule with the resource agents. The negotiation process includes a re-negotiation phase when exceptions occur. A task manager is responsible for generating task

agents. This system was later modified to exclude the resource manager agent, thus achieving a fully distributed architecture [11].

A market-based scheduling control system features product and resource agents that negotiate a schedule [36]. A supervisor agent handles blocking and delay situations.

Another system for scheduling, planning, and control has a hybrid architecture with system and cell agents in addition to parts, machines, and material handling agents [79]. In this system a modified contract net protocol is used for negotiation. This paper also offers an overview of various MAS architectures.

In Araujo et al. [42] a control scheduling system with hybrid architecture features mediator agents that resolve conflicts among local agents. Scheduling agents coordinate with process activities and resources agents to achieve global process control. A human agent interfaces with the scheduling agent. Agents are modeled using high-level Petri-nets.

4.4.2.3 Fully distributed scheduling systems

In a real-time heterarchical scheduling system [83], producer agents use a game theory philosophy called “coopitition” in which agents simultaneously compete and cooperate with other producer agents. Agents use a memory-based reasoning technique for learning. The user is represented through a “human agent.” The system was developed using the Java-based Voyager Platform.

A fully distributed job shop scheduling system with auction-based negotiation using Lagrangian relaxation is proposed by Dewan and Joshi [62]. Results show that the proposed system outperforms distributed dispatching heuristics.

Bussmann and Schild [73] proposes a prototype for a self-organizing Flexible Manufacturing System where work-pieces, machines, and switches agents negotiate a schedule using auction-based protocols, where switches move pallets across conveyors in order to bypass machines. To avoid deadlocks resulting from capacity bottlenecks, a mechanism is introduced whereby machines do not bid for new work-pieces when the machine's output is blocked. Tests show that the performance of the MAS is nearly optimal.

Another system for job shop scheduling (shop floor task allocation) is presented by Glanzer et al. [40], a heterarchical system using Contract Net Protocol. The system was implemented using the Zeus toolkit.

Csaji et al. [4] presents a machine learning functionality with reinforcement learning neural networks to improve the scheduling in the PROSA holonic architecture [34].

Another dynamic scheduling MAS with fully-distributed (heterarchical) architecture from Liu et al. [56] features a hybrid auction/Lagrangian relaxation approach with a rolling time horizon procedure for formulating and solving the scheduling problem.

The bottleneck station scheduling problem is addressed using Ant Colony Optimization (ACO) to solve it metaheuristically [96]. The system was implemented in an Intel chipset.

4.4.3 Holonic Manufacturing Systems

The concept of holons was first introduced by Arthur Koestler [3]. Holons are collaborative self-configuring agents that are capable of communicating with other holons to achieve overall system objectives [18]. In holonic systems, the holons are organized in a hierarchical structure called a holarchy.

One of the most comprehensive treatments of holonic systems for industrial applications was provided by the Holonic Manufacturing Systems (HMS) Consortium, an international industrially driven project addressing research, standards, pre-competitive development, deployment, and support of architectures and technologies for open, distributed, intelligent, autonomous and cooperative systems. During its ten-year program, more than 40 companies, R&D laboratories, and universities developed specifications of holonic architectures, a computer-aided environment for the encapsulation, reuse and integration of holonic systems technologies, and libraries of demonstrated, reusable technologies and tools for the construction of holonic manufacturing systems. Details of the organization and the accomplishments of the HMS Consortium are provided by Gruver et al. [87] and the Intelligent Manufacturing Systems (IMS) web site [35].

The extension of the original holonic visions, aimed at real-time low-level control, towards production planning and scheduling, and supply chain management issues covers the same area as current MAS research [85]. For this reason, the technology and results achieved in MAS can be applied to holonic system applications.

An example of a holonic manufacturing system is a planning, scheduling and control system that uses fuzzy logic to model multi-agent systems with holonic self-organizing systems [52]. The system has hybrid architecture with mediator agents.

In another holonic system, resource and component agents integrated in a holarchy perform scheduling using a distributed algorithm based on the theory of constraints [41]. The system was implemented using the JADE platform.

Recently, there is a direction to combine service-oriented architecture with MAS. In one application [39], a service-oriented agent acts as a mediator of a virtual unidirectional conveyor to transport pallets from one port to another. The agent communicates via service-orientation and controls the physical device. In order for the agent to provide its service (transfer of pallets), it requests service from a neighbouring device.

4.5 Summary

Manufacturing operation, scheduling, planning, and control are complex issues, due to the highly dynamic nature of manufacturing. Because of their

distributed characteristics, flexibility, abstraction, and ease of implementation, MAS have been widely applied to manufacturing.

I presented examples of agent-based manufacturing systems developed for the integration of several operations in manufacturing. Earlier systems involved a super-agent that supervises sub-agents in a hierarchical architecture. Subsequently, modular or hybrid architecture was used in integrated systems. More recent systems involve virtual enterprise, supply-chain and Internet monitoring and control.

I also provided examples of agent-based manufacturing systems that deal with the scheduling, planning, and control. Those systems were classified into control, hybrid architecture and heterarchical architecture. An heterarchical architecture (fully-distributed approach) is typically used to solve the job shop scheduling problem, especially in Flexible Manufacturing Systems (FMS). Several MAS in manufacturing incorporate intelligence algorithms include neural networks, optimization, and knowledge-base techniques.

5: AGENT-BASED DECISION SUPPORT AND SIMULAION SYSTEM FOR ROUGH MILLS

5.1 Introduction

In this chapter, I introduce agent-based solutions for the rough mill. Agents represent the physical machines in the simulation such as the ripsaw, as well as the business processes involved in the decision making such as scheduling.

The outline of this chapter is as follows: First, I introduce the agent-based platform middleware that is used in developing the solutions. Then, a prototype system is implemented to demonstrate the architecture and inter-agent communication. This prototype is then extended to two lines of production and the negotiation protocol used is presented. The possibility of extending to multiple lines of production is discussed. The prototype is used to implement the system with real data and full functionality and the challenges of implementing the system are discussed. Evaluation of each of the three systems is presented as verification and validation testing. Finally, a discussion of the agent-based approach is presented as well as an evaluation of using the agent paradigm.

5.2 JADE and FIPA standards

Java Agent DEvelopment Framework (JADE) [26] [27] is a Java-based open-source middleware for agent development and execution. It is compliant with standards of the Foundation for Intelligent Physical Agents (FIPA) [69] [77]. FIPA is an IEEE Computer Society standards organization that promotes agent-

based technology and the interoperability of its standards with other technologies [29].

FIPA specifications define the reference model of an agent platform and a set of services that should be provided to realize truly interoperable MAS. The FIPA performatives and other features of FIPA standardization can be applied to control problems, production planning and supply-chain management [85].

Using the JADE platform ensures following the FIPA standards, while providing other features such as agent communications and a graphical user interface (GUI) for monitoring agents. JADE provides the support to run agents on several computers and mobile platforms over a network or the Internet. Java is platform-independent which makes it a flexible and portable choice.

JADE has been used for several agent applications, such as providing e-services to mobile users [63], integrating agents into distributed virtual environments [44], reducing traffic in peer-to-peer systems [97], manufacturing simulation for material handling [66] and mobile ad hoc information and services [2].

5.3 RoughMill ontology

An application-specific ontology is developed for defining the concepts and agent actions. An ontology provides a formal explicit description and a common vocabulary of concepts and their properties and relations. An ontology together with a set of individual instances of classes constitutes a knowledge base [67].

The ontology is generated using Protégé semantic editor [67] with a Bean Generator plug-in reference [7]. This graphical tool is an open source ontology editor and knowledge acquisition system that can be easily used to add concepts, agent actions, and predicates. The ontology Java files are automatically generated, facilitating future modification of the ontology. Figure 5.1 shows the jag concept in the Protégé GUI.

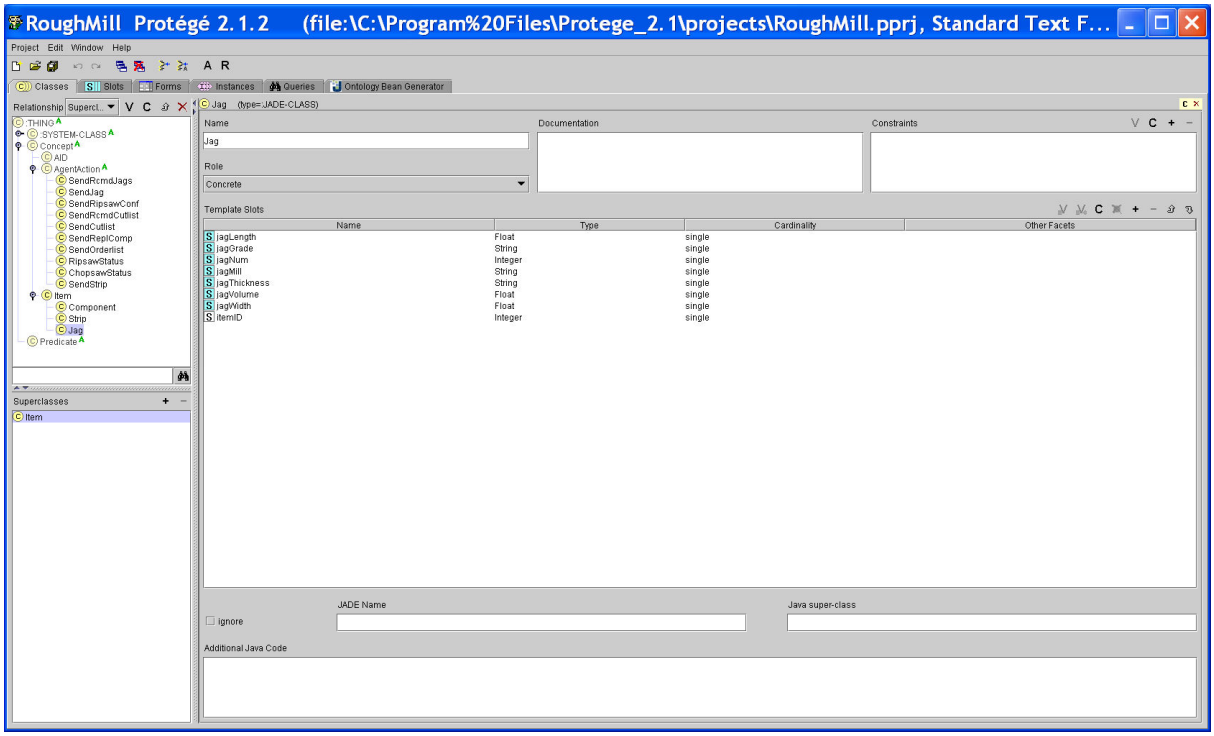

Figure 5.1 Protégé GUI showing the jag concept

Concepts used in the RoughMill ontology are item, strip, Jag and component. Examples of agent actions are SendJag, SendStrip and SendRcmdCutList. Figure 5.2 shows the SendJag action as the content of an ACL (Agent Communication Language) message sent from the ripsaw agent.

```
((action
(agent-identifier
:name
RSA@eman:1099/JADE)
(SendJag
:ripJag
(Jag
:itemID 5
:jagLength 14.0
:jagVolume 2845.0
:jagGrade M
:jagNum 123423
:jagWidth 2.0))))
```

Figure 5.2 SendJag message

5.4 Prototype multi-agent system for one line of production

5.4.1 System architecture

Figure 5.3 shows the system architecture for one line of production. There are three categories of agents used in the proposed system: user interface, decision support and simulation agents. Figure 5.4 shows the inter-agent communications.

This design is a hybrid model, i.e., neither a master-slave super-agent architecture, nor a fully-distributed peer to peer system. It has a modular architecture where agents represent different physical and logical entities in the

rough mill such as chopsaw, ripsaw, and jag selection. The system was designed to mimic the functionality of the real-life system.

The agents use JADE messaging system to send and receive messages. All agents (except the start agent) check continuously on received messages. Each agent has several templates of expected messages. When the agent receives one of those messages, it takes action according to its internal logic. If the message is unknown, the agent records the unknown message to the output debug log. The block() function is used to keep the agent idle between receiving messages.

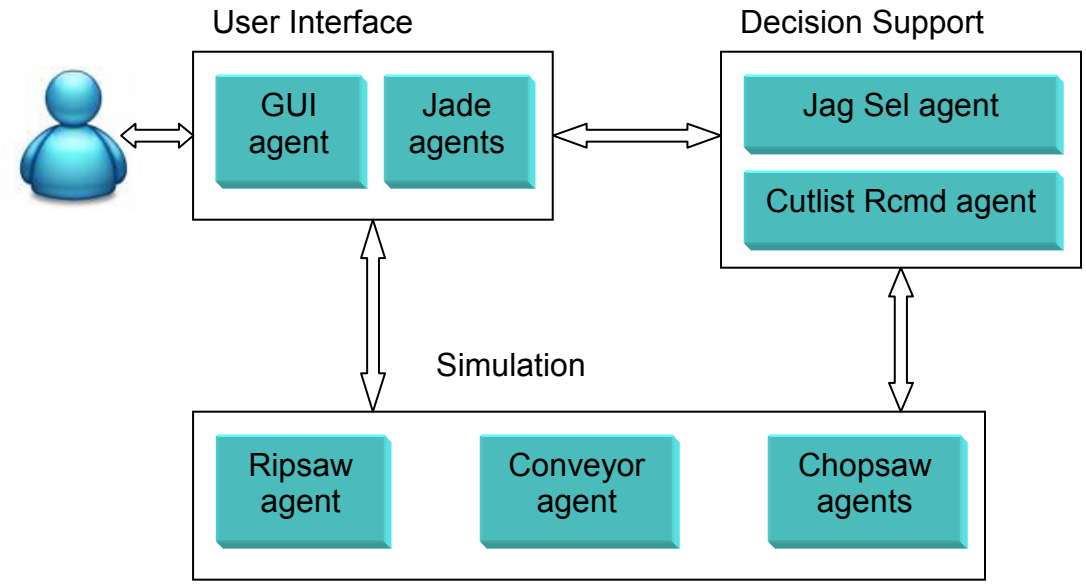

Figure 5.3 System architecture showing different types of agents

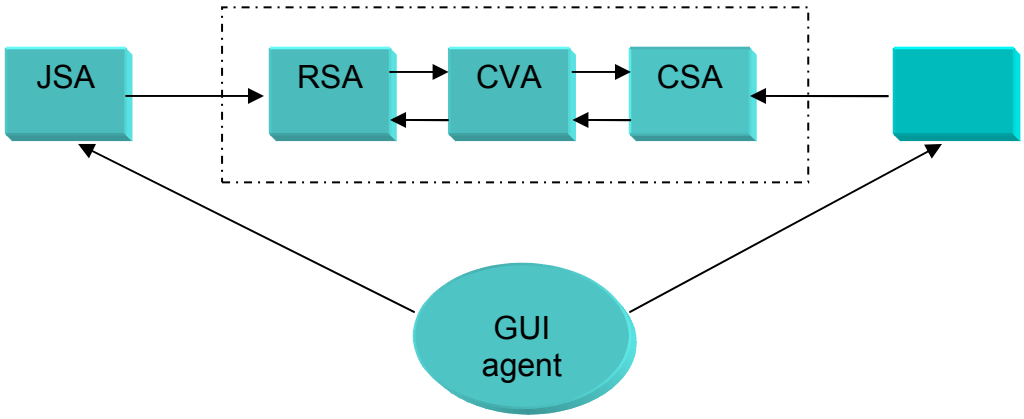

Figure 5.4 System architecture with agent communications

Every JADE agent is composed of a single execution thread. In order to be able to execute several tasks, the agent must be able to carry out several concurrent tasks, each in its own behaviour. Cyclic behaviour, one shot behaviour and simple behaviour are used to implement the agents. JADE framework also provides other behaviours such as FSM behaviour and Parallel behaviour.

5.4.2 User interface agents

5.4.2.1 Start/GUI agent (SA)

The GUI of the prototype is a simple GUI with start and end simulation buttons. When the user presses the respective buttons on the GUI, this agent sends start simulation and end simulation messages to the decision support agents.

5.4.2.2 JADE agents

JADE provides agents for debugging and platform management, e.g., remote management agent, dummy agent and sniffer agent.

5.4.3 Decision support agents

Decision support agents provide recommendation for selecting jags and cutlists.

5.4.3.1 Jag selection agent (JSA)

This agent selects a jag and sends it to the ripsaw agent to start the simulation.

5.4.3.2 Cutlist recommend agent (CRA)

The agent recommends a cutlist and sends it to the chopsaw for simulation.

5.4.4 Simulation agents

These agents simulate the operations of the machines in the rough mill. Agents representing the ripsaw, conveyor and chopsaw process selected jags into individual components according to the required cutlist.

5.4.4.1 Ripsaw agent (RSA)

The ripsaw agent receives a message including the suggested jag. It simulates cutting of the strips from boards of the jag. Generated strips are saved in a linked list data structure in the agent knowledge base. Another behaviour sends strips one by one to the conveyor agent as individual messages. The

ripsaw agent stops/resumes processing a jag upon receiving a message from the conveyor agent indicating jamming in the line (as discussed below).

5.4.4.2 Conveyor agent (CVA)

The conveyor agent acts as a buffer between the ripsaw agent and the chopsaw agent, and represents the physical conveyor that exists in the rough mill. When the conveyor agent detects jamming, it sends a message to the ripsaw agent to stop. It sends another message to the ripsaw to resume when the jamming has ended.

The conveyor agent receives strip messages from the ripsaw agents and saves them to a strips buffer with a linked list data structure. It also receives confirmation messages from the chopsaw for strips received. The conveyor agent keeps track of the number of strips waiting on the conveyor as follows:

number of strips waiting on the conveyor = number of strips received from ripsaw - number of strips confirmed by chopsaw

Another cyclic behaviour sends the strips to the chopsaw agent if there is no jamming, or when it receives a strip acknowledgement message from the chopsaw.

Jamming is detected if the number of strips waiting is greater than the max number of strips waiting. This number is set as an arbitrary constant number in the prototype. Using real data implementation (to be discussed later in this chapter), this number is calculated based on the dimensions of the strips on the conveyor.

5.4.4.3 Chopsaw agent (CSA)

The chopsaw agent simulates chopping of strips into components. It receives strip messages from the conveyor agent, sends confirmation message back to the conveyor, and simulates chopping of strips into components of the cutlist.

5.4.5 Evaluation

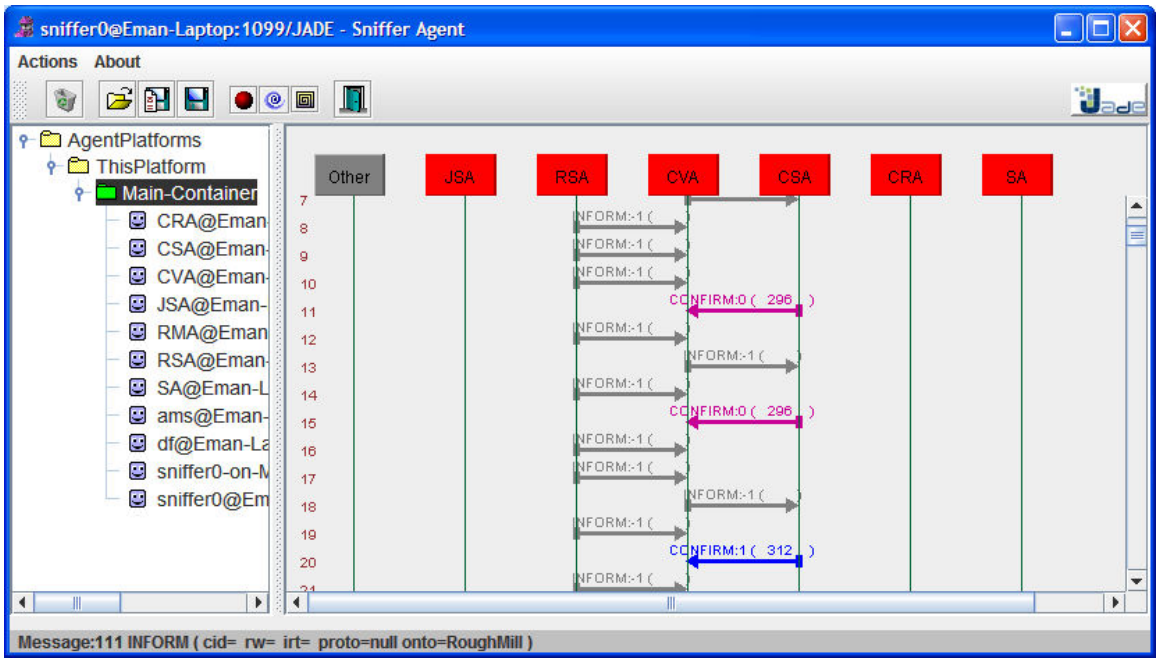

Figure 5.5 Jade sniffer agent shows message exchange between agents

In Figure 5.5, the JADE sniffer agent shows the messages exchanged between agents during the simulation. Strip messages are generated by the ripsaw agent (RSA), and sent to the conveyor agent (CVA), which in turn sends the strip messages to the chopsaw agent (CSA). CSA then sends a confirm

message back to the conveyor to keep track of the number of strips on the conveyor.

The content of a message is displayed by double-clicking on the arrow representing the message. Figure 5.6 shows the content of an ACL message exchanged between CVA and CSA. The sniffer agent can be used to save a log of all exchanged messages to a text file, as well as a snapshot of the session.

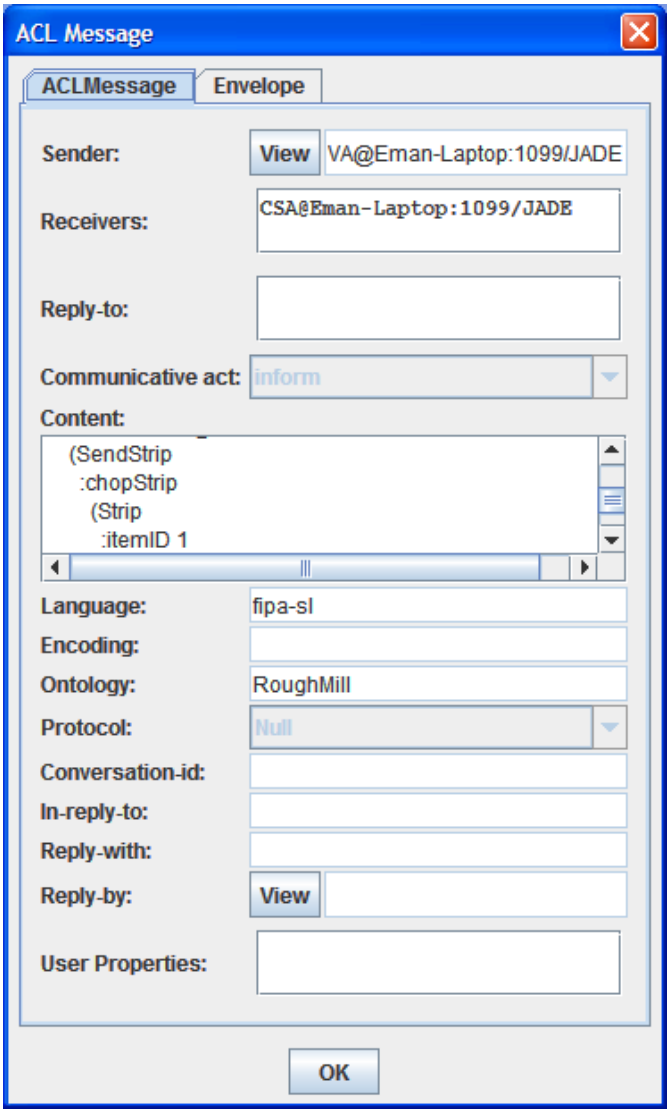

Figure 5.6 ACL message containing a strip sent from the conveyor agent to the chopsaw agent

The system is verified and validated using the sniffer agent, as well as a debug output, as shown in Figure 5.7.

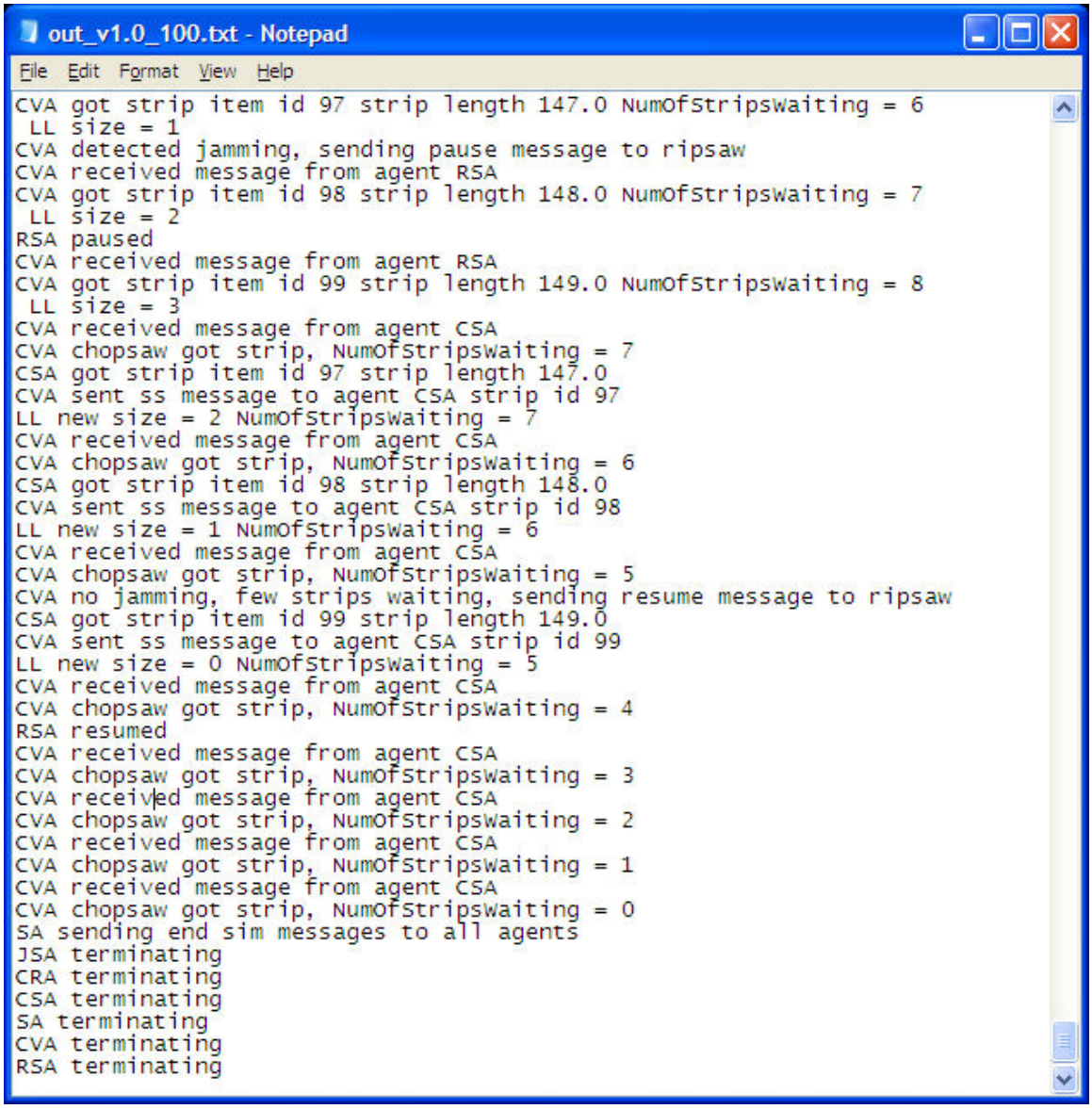

Figure 5.7 A section of the output file of simulation with 100 strips

5.5 Agent-based prototype for two lines of production

In the rough mill, there are two lines of production. One warehouse stores all the jags that are used for jag selection by both lines, and components can be scheduled on either lines. Therefore, decision support agents (namely jag

selection and cutlist recommend agents) of one line need to negotiate with decision support agents from the other line.

However, the machines on the production line work in sequence, i.e., the conveyor agent of line1 (CVA1) transfers strips from RSA1 to CSA1. Similarly, for line2, CVA2 transfers strips from RSA2 to CSA2. Therefore, agents representing machines of one line (namely ripsaw, chopsaw, and conveyor agents) do not interact with agents from the other line.

5.5.1 System architecture

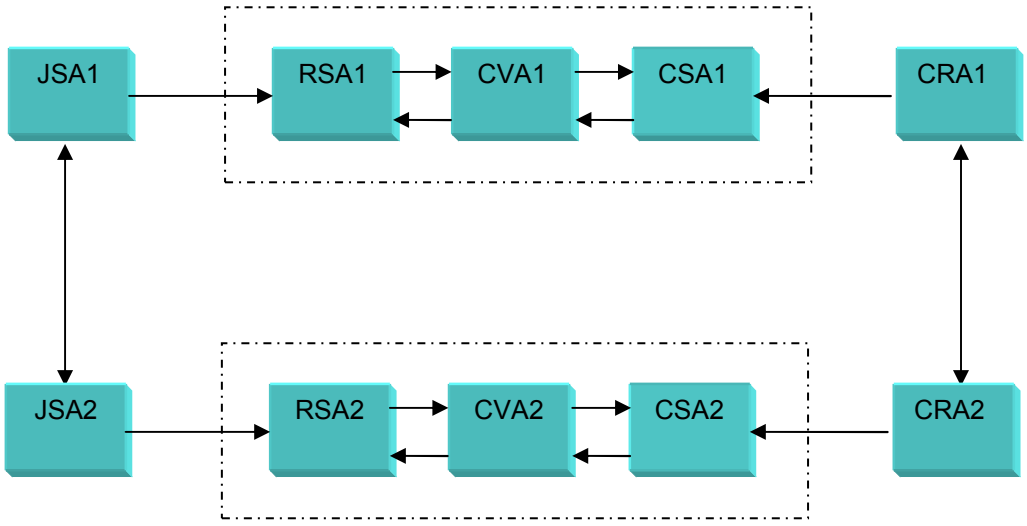

Figure 5.8 System architecture of two lines of production

Figure 5.8 represents the two-line prototype. Jag selection agents negotiate together to avoid conflict over selected jags. Similarly, cutlist recommend agents negotiate over components of the cutlist. Simulation agents

do not interact with agents from the other line. The start agent (not shown in the figure) sends start and end simulation messages to the four decision-support agents.

5.5.2 Negotiation protocol

FIPA-Propose negotiation protocol is used. The initiator sends a propose message indicating that it will propose some action. The participant (responder) responds by accept or reject message. The initiator performs the action and returns a status response message. This protocol is a direct negotiation protocol with no need for moderators. Other FIPA interaction protocols supported by JADE are contract-net, achieve rational effect, and subscribe.

5.5.3 Evaluation

Figure 5.9 shows the log of messages exchanged between the jag selection agents of the two lines. JSA2 initiates of the protocol, JSA responds with an accept message and JSA2 receives it and ends the protocol.

JSA Responder received the following message: (PROPOSE

:sender (agent-identifier :name JSA2@Eman-Laptop:1099/JADE :addresses (sequence http://192.168.199.25:7778/acc))

:receiver (set (agent-identifier :name JSA@Eman-Laptop:1099/JADE))

:reply-with R8549963_0 :protocol fipa-propose

:conversation-id C8549963_1272134595062)

JSA2 Protocol finished. Received the following accept message: (ACCEPT-PROPOSAL

:sender (agent-identifier :name JSA@Eman-Laptop:1099/JADE :addresses (sequence http://192.168.199.25:7778/acc))

:receiver (set (agent-identifier :name JSA2@Eman-Laptop:1099/JADE :addresses (sequence http://192.168.199.25:7778/acc)))

:content "accept_proposal"

:reply-with JSA2@Eman-Laptop:1099/JADE1272134595093 :in-reply-to R8549963_0 :protocol fipa-propose

:conversation-id C8549963_1272134595062)

Figure 5.9 Negotiation protocol log

5.6 Multiple lines of production

While the rough mill of interest has only two lines of production, it is interesting to examine whether the system can be extended to more than two lines of production. As seen in the previous section, the simulation agents of each line of production are independent of agents of other lines. Only the decision support agents (JSA and CRA) interact with agents from other lines. The FIPA-Propose negotiation protocol can also be used since it is 1:N (the initiator can handle several responders at the same time) [28]. Figure 5.10 is the suggested design for three lines.

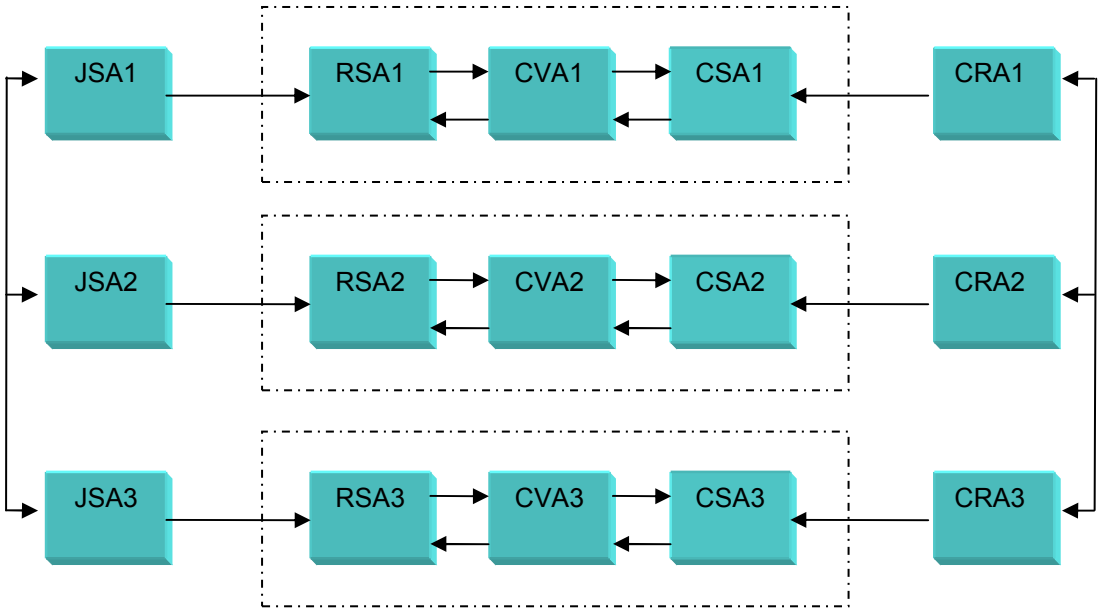

Figure 5.10 Three lines of production

5.7 Rough Mill Decision Support and Simulation System (RMDSSS)

5.7.1 System architecture

The one-line prototype is extended by implementing it with real data and algorithms. Figure 5.11 shows the RMDSSS system architecture.

The simulation described in Section 3.3.1 is used to develop the simulation agents. The agents read historical data to generate boards, strips and clear pieces as described earlier.

The case-based reasoning jag selection algorithm mentioned earlier in Section 3.3.2 is used for jag selection. This algorithm reads historical data files used for case-based reasoning. It also reads current data to get information about jags in the warehouse.

For cutlist recommend, the heuristic algorithm proposed later in this thesis (Sections 6.6.1 and 6.6.2) is used

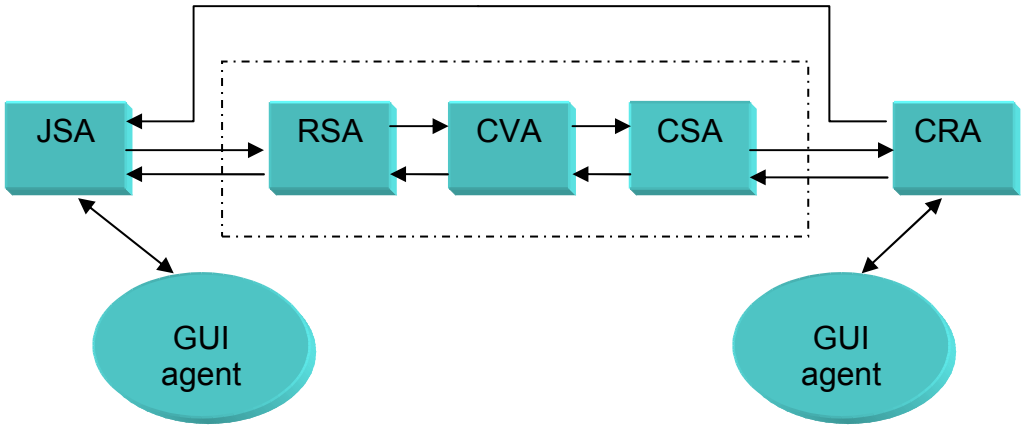

Figure 5.11 System architecture using real data

Components can be added by the GUI to change the order list. The production line configuration can also be changed through a configuration file to reflect jammed machines (kickers).

Like most object-oriented programs, the code of the simulation and decision support was divided into modules each with a specific functionality, such as GUI, algorithms, parsers, etc. While it is possible to do an agent-wrapper for each of these modules, this is not straightforward because most modules use methods from other modules. It becomes necessary to re-design the system into a more logical design that represents the physical system and the decisions in the rough mill.

Implementing the full system with real data uncovered some information that is missing from the prototype that is then added in RMDSSS, namely:

1. The cutlist is required for JSA,
2. Exchange of cutlist between CSA and CRA for component replacement,
3. RSA sends status update to JSA for GUI update, and
4. Some fields are missing from the ontology.

There are two GUIs for this system, which are slightly modified from the original simulation Delphi simulator interface GUIs. The first one is for jag selection and simulation (Figure 5.12), and the other GUI is for cutlist recommend and simulation (Figure 5.13).

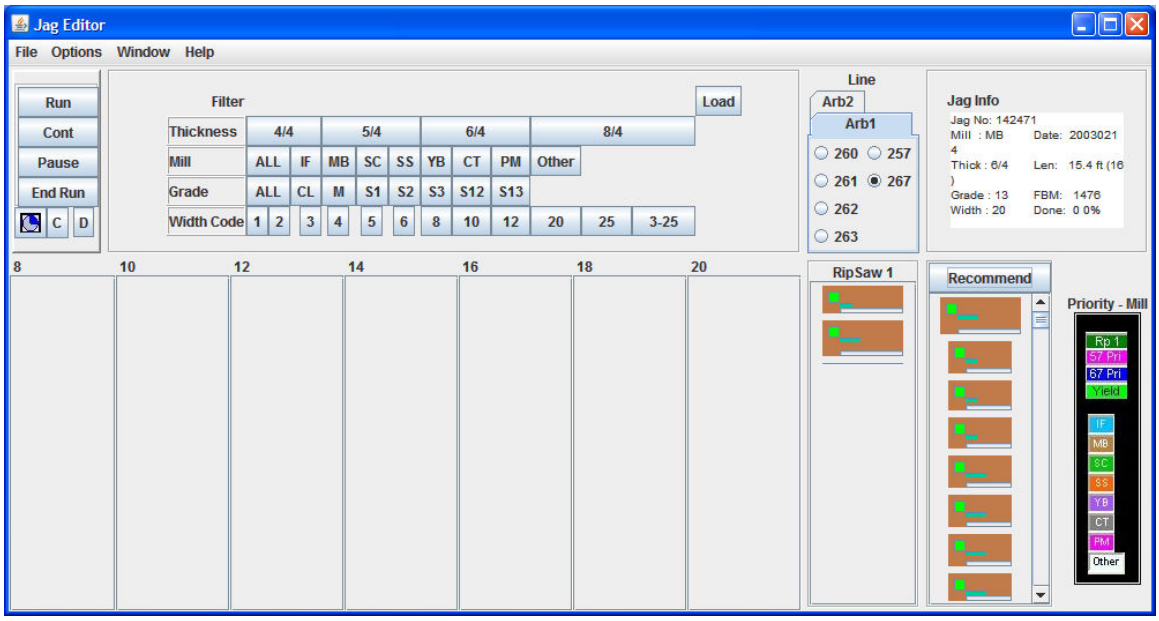

Figure 5.12 GUI for Jag Selection Agent

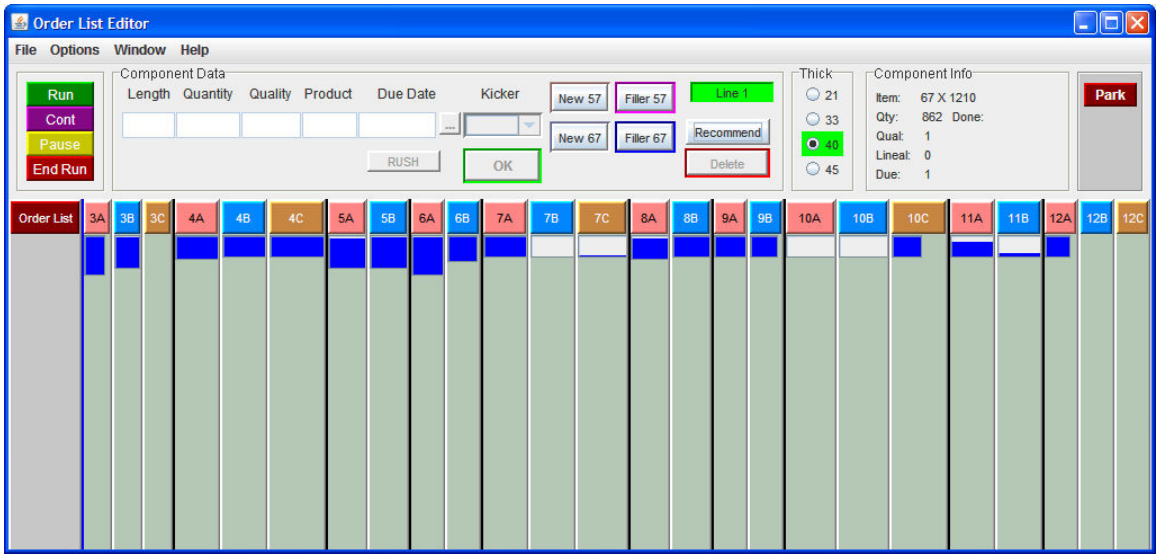

Figure 5.13 GUI for Cutlist Recommend Agent

Figure 5.14 shows the package layout. The three main packages are DecisionSupport, Simulation and config. This hierarchy is used to represent the logical function of the classes implemented. For example CutlistRcmd is a DecisionSupport package, that does not import any files from the JagSelection package as well as the Simulation packages. Similarly, the chopsaw package does not import any files from the conveyor or the ripsaw, as well as DecisionSupport. The config package contains other files such as the ontology classes, configuration and data files and the simulation report classes.

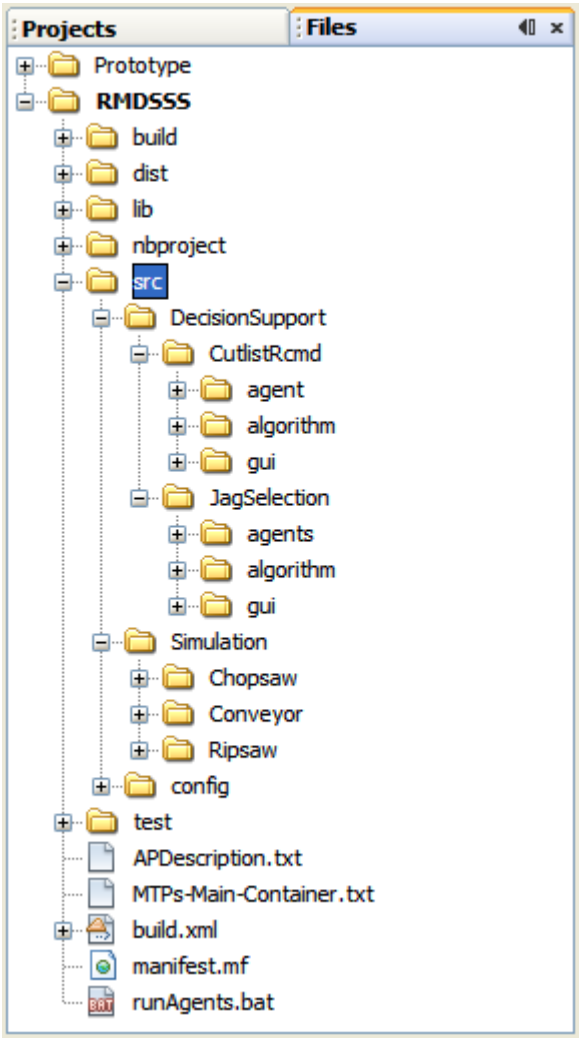

Figure 5.14 Package layout

5.7.2 Challenges of using real data

Using real data and simulation algorithms rather than prototypes results in the following challenges:

1. Following the agent paradigm on existing simulation code. In the existing code, as with most object-oriented programs, one module

directly uses the methods of another module. In agent-based systems, one agent cannot directly manipulate another agent's logic as this contradicts the autonomy characteristic of the agent. This requires reengineering the code to make sure the design agrees with the agent paradigm. However, using the JADE middleware made developing agents easier.

2. Discovering missing information that is not considered in the prototype as described above.
3. GUI functions and display of progress to the user is more complicated than the simple GUI of the prototype, which contains only two buttons to start and end the simulation. In addition, there are two GUI agents instead of one.
4. It is hard to trace the large number of messages exchanged between agents, especially given the distributed characteristics of the agent system. This challenge is the motivation for building the prototype and verifying its function before implementing the system with full functionality and real data.

5.7.3 Evaluation

The system is verified using the JADE GUI (for agent communication) and debug status statements as in prototype evaluation. In addition, the simulation report and the information displayed on the two GUIs (as shown in Figure 5.12 and Figure 5.13) and are both used for verification. The simulation is validated

against the Quest model [22], which was used and tested for several years by researchers and rough mill staff.

5.8 Discussion of agent implementation

In this section, some of the issues that resulted from the use of agent technology are discussed.

5.8.1 Jamming detection

In the rough mill, conveyors transfer strips from the ripsaw to the chopsaw. If the chopsaw is running slower than the ripsaw, jamming or congestion will occur on the conveyor. A photocell is used to detect jamming. The ripsaw and conveyor are stopped until the jamming ends.

The conveyor agent detects jamming by keeping track of the number of messages (strips) it received from the ripsaw agent, and the number of confirmation messages it received from the chopsaw agent. The difference represents the number of strips waiting to be processed by the chopsaw. Given the speed of the conveyors, and the dimensions of the strips sent to the chopsaw, the maximum number of strips waiting to be processed (conveyor capacity) can be calculated.

This process mimics the conveyor functionality in the rough mill. It also provides approximate synchronization in the operation of the ripsaw and chopsaw.

5.8.2 Synchronizing the simulation

In order to achieve synchronized simulation, we can mimic discrete event simulation by using an event-logger (supervisor or super-agent) to keep track of the simulation events, and trigger each agent to perform its action in the correct timing and the right sequence of events. However, such centralized approach can cause bottleneck problems since one supervisor agent controls the rest of the agents.

In addition, such time accuracy is not needed since the purpose of this system is decision support and simulation rather than control. An approximate synchronization is sufficient to provide consistent information about the ripsaw and chopsaw progress, and display the progress on the GUI. Conveyor agent jamming detection is used to approximately synchronize the simulation as described above.

5.8.3 Simulation time calculation using bottleneck detection

Simulation time is the time it will take to run the operations in real-life. In the previous discrete-event simulation system, simulation time is calculated according to a model. In the agent-based system, ripsaw simulation time is calculated based on the number of boards processed by the ripsaw, conveyor simulation time is based on the velocity of the conveyor, and chopsaw simulation time is based on the number of chops performed by the chopsaw.

Since all machines run simultaneously, simulation time is not simply adding these values together. Discrete event simulation provides accurate simulation time because a list of events can be tracked accurately and used to

calculate the time. However, if we want to avoid a centralized supervisor agent that runs the whole system, alternative approximations are necessary.

The proposed approach is to find out the bottleneck of operations. This means that we decide which machine is slower, the chopsaw or the ripsaw. By calculating the simulation time for the ripsaw and the simulation time for the chopsaw and picking the higher number, we find out the approximate simulation time. A problem arises is that since the system is not accurately synchronized (as described above), we cannot use equal time intervals (such as the ticker behaviour provided by JADE) to find out the bottleneck. Therefore, one jag is used as the interval where the bottleneck is seeked. The ripsaw and the chopsaw each send their simulation time per jag, the GUI agent pick the maximum of the two numbers to calculate the simulation time displayed in the simulation report.

5.8.4 Computational time

Computational time is the time it takes to run the simulation on the computer since agents run locally on one computer. It can be easily calculated from the start until the end of the simulation. Several runs using RMDSSS with real data show that this time is acceptable with a few seconds per jag.

5.9 Evaluation of using the multi-agent paradigm

The advantages of using agent-based design are as follows:

1. **Modelling**: In the rough mill, there is no single centralized entity that controls the machines or the decisions in the rough mill. The

agent design reflects the actual rough mill system as it exists in real life, which provides a good model of how the rough mill operates.

2. **Communication**: Once the rough mill ontology of the agents is defined and implemented, agents communicate easily with messages that contain ontology elements. There is no discrepancy even if multiple programmers are involved.
3. **Integration**: Similar to communication between agents in the system, it is possible to add new agents and connect them to existing agents.
4. **Expandability**: Agent design allows for expanding the current system. For example moving from one line to two lines is facilitated by using a standard negotiation protocol between the old and new agents. Similarly, adding more lines or other rough mill operations such as creating order lists from work orders can be integrated to the system.
5. **Mobility**: JADE framework supports agent mobility, as agents can exist on several JADE containers (different computers or PDAs). For example, jag selection GUI can run on one container and cutlist recommend GUI on another. It is also possible to allow for remote or web monitoring.

On the other hand, the disadvantages of using agent-based systems are:

1. **Synchronization**: It takes careful design and experimentation to get the expected functionality of the system. In a centralized system, one program controls the flow of the rest of the modules, thus achieving the desired functionality. With a distributed decentralized system, the control is shared among all running agents, each running in its own thread. The desired functionality is achieved by implementing several behaviours in each agent, for example one behaviour to receive messages, and another to send messages or do other tasks.
2. **Paradigm-specific issues:** The design and implementation of the agent-based system required a learning process in several areas such as AI, agent-based systems, distributed systems, ontologies, FIPA standards and JADE framework.

5.10 Summary

In this chapter, three rough mill agent-based systems were presented and evaluated using rough mill data. The first system is a prototype for one line of production. The second one is an extension of the prototype for two lines of production or more. The third system RMDSSS (Rough Mill Decision Support and Simulation System) uses the prototype to implement a complete system with real data, simulation algorithms and two GUIs. The purpose of these systems is to present recommendation for the operators to select material and schedule orders and to view the simulation. JADE framework is used for implementation, as it is compliant with FIPA IEEE standards.

6: THE ROUGH MILL SCHEDULING PROBLEM (RMSP)

6.1 Introduction

In this chapter, the second part of this work which is the scheduling problem is investigated. In a previous chapter, the operations in the rough mill are described, including scheduling. In the following sections the Rough Mill Scheduling Problem (RMSP) is described in detail, followed by a problem formulation of the RMSP as a search problem and Constraint Satisfaction Problem (CSP). In the rest of the chapter I propose several solutions to the problem including full search, domain search, backtrack, heuristic and randomized heuristic methods. The implementation and evaluation of these methods is presented.

6.2 Problem description

Figure 6.1 shows the inputs and outputs of the scheduling process. The inputs are the components from the work orders (order list) and the current status on the kickers. The outputs are new assignments to the kickers, maximum length on the cutlist to the jag selection process and strip width to the ripsaw for arbor setting.

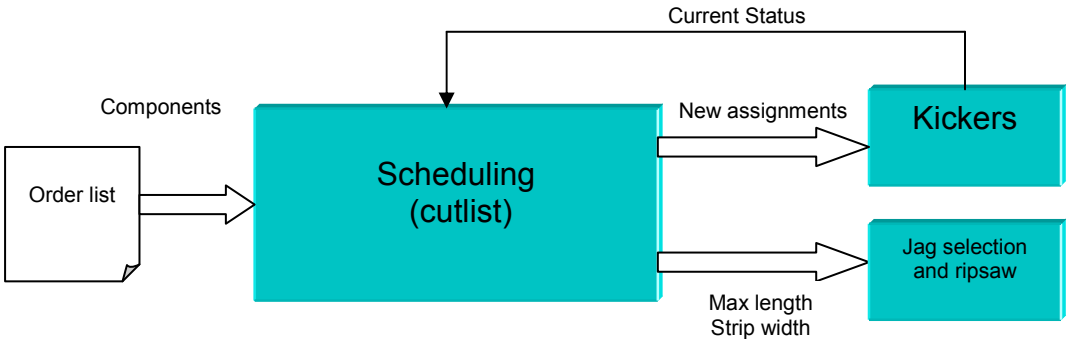

Figure 6.1 Rough mill cutlist scheduling

6.2.1 Physical constraints

Table 6.1 and Table 6.2 list the properties of *kickers (sorting bins)* of production lines one and two respectively. Each kicker can be assigned more than one component. The components are manually sorted and stacked into loads. However, due to limited ground floor space and limited number of kickers, usually 20 to 25 components are scheduled at one time. Since kickers can be assigned more than one component, 'logical' kickers or *sort numbers (kicker numbers)* are used to assign one component per logical kicker.

Table 6.1 Kicker (Sorting bin) configuration for production line number one

	Sort #	Length range (mm)	Nbrhd
Kicker #3	1,2,3	300 – 800	1
Kicker #4	4,5,6	1815 – 3700	2
Kicker #5	7,8	1300 – 1810	2
Kicker #6	9,10	800 – 1299	2
Kicker #7	11,12,13	1815 – 3700	3
Kicker #8	14,15	1300 – 1810	3
Kicker #9	16,17	800 – 1299	3
Kicker #10	18,19,20	1815 – 3700	4
Kicker #11	21,22	1300 – 1810	4
Kicker #12	23,24,25	300 – 900	5

Table 6.2 Kicker (sorting bin) configuration for production line number two

	Sort #	Length range (mm)	Nbrhd
Kicker #2	1,2	1810 – 2599	1
Kicker #3	3,4	1300 – 1810	1
Kicker #4	5,6,7	300 – 1299	1
Kicker #5	8,9	1810 – 3610	2
Kicker #7	10,11	1300 – 1810	2
Kicker #8	12,13	1810 – 3610	3
Kicker #9	14,15	1300 – 1810	3
Kicker #10	16,17	900 – 1299	3

Kicker #11	18,19	300 – 1200	4
Kicker #12	20	300 – 700	4

To formulate the physical constraints of close-by kickers, the concept of *neighbourhood* is introduced. Kickers physically close-by are said to be in proximity to each other and are assigned the same neighbourhood value (*Nbrhd*) N_j as shown in Table 6.1. Components assigned to one kicker affect close-by kickers in two ways. The difference in lengths between components assigned to kickers in the same neighbourhood should not be too small (less than 100 mm) because components can get mixed up during the manual sorting and loading process. If the length difference between components on nearby kickers is too big (greater than 2000 mm), it can form a hazard by stacking up and falling on workers.

The neighbourhood conflict constraint for the two components C_{i1} and C_{i2} can be expressed as follows:

$$n_{\min} < |L_{i1} - L_{i2}| < n_{\max} \quad \forall\, i \tag{6.1}$$

where $n_{\min} = 100mm$ and $n_{\max} = 2000mm$

For each kicker K_j, the minimum and maximum component length allowed are L_{jmin} and L_{jmax} respectively, as shown in Table 6.1. The *range* of kicker K_j is defined as follows:

$$R_j = L_{j\max} - L_{j\min} \quad \forall j \tag{6.2}$$

A valid kicker K_j for a component C_i is one for which the length of the component L_i falls within K_j's range of valid lengths. The length constraint of kicker K_j and component C_i can be expressed as follows:

$$L_{j\min} < L_i < L_{j\max} \quad \forall i, j \tag{6.3}$$

The maximum length represent the physical limit of the kicker, however, the minimum length can be ignored to achieve a cutlist with more components.

$$L_{j\min} = 0 \quad \forall j \tag{6.4}$$

In order to present a variety of component lengths to the chopsaw, it is preferable to have a uniform length distribution of components. The Length distribution constraint for components C_{i1} and C_{i2} can be expressed as follows:

$$|L_{i1} - L_{i2}| < Gap_{\min} \quad \forall i \tag{6.5}$$

where $Gap_{\min} = 100 or 50 or 5 mm$

The value of Gap_{min} is not set to zero to prevent assigning same-length components on two different kickers.

6.2.2 Initial and replacement scheduling

The scheduling problem can be divided into two sub-problems. The first is *initial scheduling* in which the chop line is empty, so we attempt to schedule as many components as possible on the line at once. The second sub-problem is *replacement scheduling* where the required quantity of one of the components scheduled on the line is completed, and we need to replace it with another component from the order list.

6.3 Theoretical background

6.3.1 Scheduling problems

Scheduling problems can be classified into the following major types: single machine, parallel machine, flow shop, job shop and open shop [51].

In the *single machine problem*, the jobs are processed on one machine only. In the *parallel machine problem*, there are *m* identical machines in parallel. Job *j* requires a single operation and may be processed on any one of the *m* machines or on any one that belongs to a given subset M_j.

In the *flow shop problem*, there are *m* machines in series, and each job *j* has to be processed on each one of the *m* machines following the same route. In the *job shop problem,* each job has its own pre-determined route to follow. Finally, in the *open shop problem,* there are no restrictions on the routing of each job through the machines and some machines may not be used.

6.3.2 Constraint Satisfaction Problems (CSP)

CSP is defined by a set of *variables V*, a set of *domain values D* and a set of *constraints C* (relations between these variables). A solution to the CSP gives an assignment to variables of values that satisfy all the constraints [75].

If we add to the CSP an objective function *f* that can be used to evaluate the solutions, we get the *Constraint Optimization* Problem (COP), which attempts to find solutions to the CSP with optimum value of *f*.

6.3.3 Search methods

Search methods can be classified into two main categories: constructive search and local search [31].

In *constructive search*, variables are assigned one by one until all the variables are assigned. If we search through all the possible values of the variables, the search is *complete*. A complete search discovers a solution if it exists. In the case of optimization, a complete search finds the optimum value of an optimization function.

On the other hand, *local search* methods start with an initial assignment of all variables. The initial assignment can be an invalid solution, or a non-optimum value in case of optimization search. Iteratively, the variable assignments are changed according to the search algorithm, until a certain stopping criterion is reached such as runtime limit, valid solution, or no change in the objective function. Local search methods are not complete; a solution that exists can be missed, or for optimization problems, the global optima can be missed.

6.4 Problem formulation: RMSP

The RMSP is a parallel machine scheduling problem as several machines (kickers) are available for processing the given jobs (components); each job consists of a single operation that is performed by one machine. Furthermore, the processing time of a job is independent of the machine on which it is processed.

However, in the RMSP, due to the natural defects in the wood and inaccurate dimensions of the wood, the processing time is unknown at the time of scheduling. Therefore, the RMSP cannot be solved using traditional scheduling methods that find the schedule by optimizing measures like the makespan or the completion time of the jobs [31] [51].

6.4.1 Configuration

We can view the RMSP as a CSP with the following specifications:

- *Variables* are the kickers (sorting bins)
- *Domain values* are the components that can be assigned to the kickers
- *Constraints* are the feasibility conditions namely, length of the component in the range of the kicker, neighbourhood constraints, and size distribution gap constraints according to Eq. (6.3), Eq. (6.1) and Eq. (6.5) respectively.

6.4.2 Exhaustive search

To find out all the possible cultists, we can conduct an exhaustive search. The domain values of each kicker are all the components on the order list.

Let *c* be the number of components in the order list and *k* the number kickers on the production line. Since unscheduled kickers can be a part of the cultists, the effective number of components is *(c+k)*. The total number of cultists for *c* components and *k* kickers can be calculated as follows:

$$^{c+k}P_k = \frac{(c+k)!}{c!} \qquad \textbf{(6.6)}$$

These are all the possible lists, but not all of them satisfy the constraints. By conducting *consistency checks* on these lists, we can find out the feasible cultists. This can be done by exploring a search tree as shown in Figure 6.2 of three variables and three domain values.

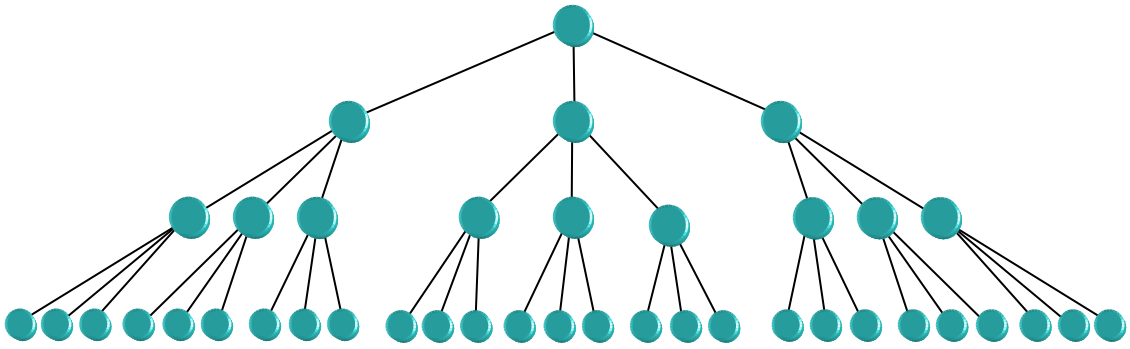

Figure 6.2 Search tree of three variables, with three domain values each

The time complexity [46] is calculated by specifying an upper bound of the growth rate in terms of the order of the algorithm. The order of the search algorithm for *n* variables and *d* domain values is:

$$T(n,d) \text{ is } O(d^n) \qquad \textbf{(6.7)}$$

6.4.3 Partial search

Given the physical constraints of the problem, we can find out all the valid component of each kicker according to Eq. (6.3). In this case, the domain value of each kicker is reduced to the number of valid components for that kicker, which results in a smaller search tree as shown in Figure 6.3.

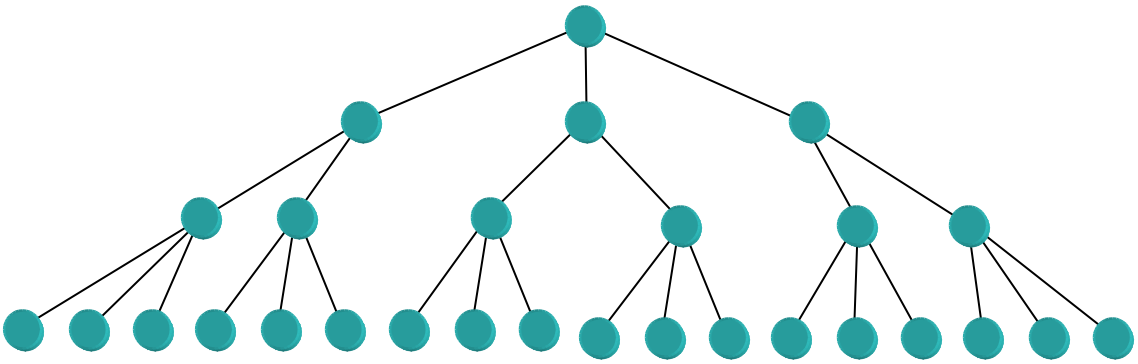

Figure 6.3 Search tree for three variables: variable1 has three domain values, variable2 has two domain values and variable3 has three domain values

The order of the search for *n* variables, each with d_v valid domain values, can be calculated as follows:

$O(d_v^n)$ where $d_v \leq d$ **(6.8)**

6.5 Backtrack scheduling algorithm

Backtrack algorithm is a complete search method; it produces all the feasible solutions to the problem. One by one, the variables are assigned domain values, which form a consistent solution. In RMSP, if one kicker assignment results in an infeasible solution, further assignments to other kickers will produce infeasible solutions as well as shown in Figure 6.4. In the backtrack algorithm, if one solution is not feasible, we do not continue to search its sub-tree since all solutions in the sub-tree are infeasible as well, as shown in Figure 6.5.

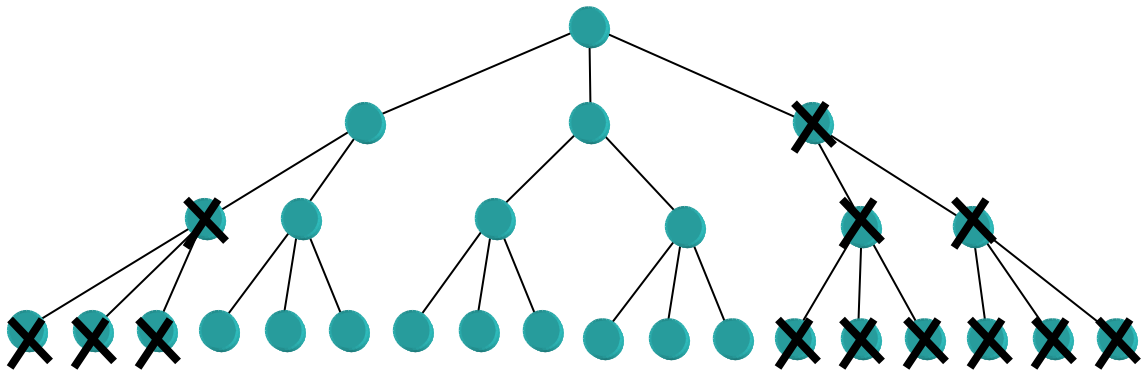

Figure 6.4 If one node is infeasible solution, the rest of the sub-tree nodes are infeasible solutions too

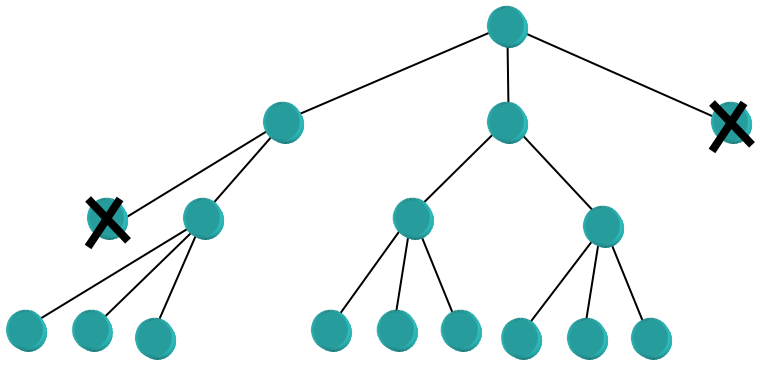

Figure 6.5 Backtrack search tree

6.5.1 Backtrack initial scheduling algorithm for RMSP

The notion of *live domain* is introduced. Each variable is assigned a live domain of all the possible domain values still to be explored. Once a domain value is checked, it is removed from the live domain of the variable.

For the backtrack algorithm, in order to assign one variable in the tree, the variable before it should be assigned. However, some variables (kickers) may not have any valid components to assign, i.e., left unscheduled. An unscheduled kicker can be a part of a feasible cutlist. In order to implement this, the notion of a *dummy component* is introduced. A dummy component has the component number '*-1*' and is added to the end of each variable live domain as shown in Figure 6.6.

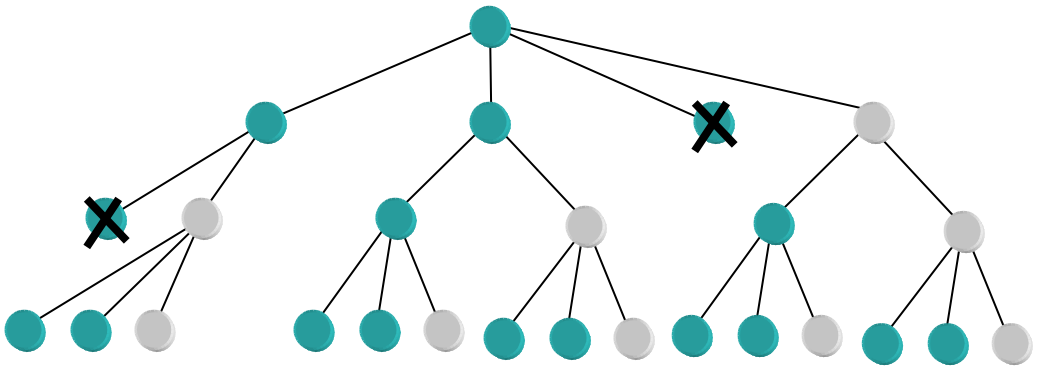

Figure 6.6 RMSP backtrack algorithm. Light grey nodes represent dummy components or unscheduled kickers

The backtrack algorithm can be outlined as follows:

Step 1. For all the variables (kickers), find all the domain values d = valid comps for that kicker plus a dummy component to represent unscheduled kickers

Step 2. Use depth first search to explore the tree:

a. If the solution is feasible: save or count the number of feasible cutlists, explore the next node.

b. If the solution is infeasible: backtrack – do not explore the rest of the subtree.

The order of the algorithm for k kickers, each with c_f feasible components, and c_v valid components is as follows:

$O(c_f^k)$ where $c_f \leq c_v \leq c$ **(6.9)**

6.5.2 Complete-search replacement scheduling algorithm for RMSP

Replacement scheduling is triggered when one component on the line is done. Since we only need to replace one component, replacement scheduling can be done by complete search. The algorithm is as follows:

For each unscheduled kicker, test each valid component, if the cutlist is feasible, add to a list of feasible replacement cultists.

6.5.3 Evaluation

To evaluate the backtrack algorithm, four order lists are used. The test is repeated for 5,10,15,20 and 25 kickers. Gap_{min} is set to 100mm (Eq. (6.5)) and L_{jmin} is not set to zero (Eq. (6.3)).

Table 6.3 shows the results of backtrack scheduling, where *k* is the number of kickers, *c* is the number of components, *TotalNum* is the calculated total number of lists for a complete search (according to Eq. (6.6)), *Feasblnum* is the number of feasible lists, *cc* is the number of consistency checks, *maxK* is the maximum number of scheduled kickers.

As the number of components and the number of kickers increase, the number of feasible cultists and the runtime increase exponentially. This makes this method not useful for large numbers, since it is impossible go through all the nodes, as well as the impractical runtime. However, the production line contains 25 kickers, and the order lists used in testing are typical ones.

The results show the complexity of the RMSP, as it is not a trivial problem that can be solved by complete search methods. Therefore, it is not possible to find all feasible cultists (or search for optimal cultists) through a complete search. It is essential to develop other methods that find a reasonable number of cultists in acceptable time.

Table 6.3 Backtrack results

k		*c*	*TotalNum*	*FeasblNum*	*cc*	*maxK*	*runtime*
k = 5	Orderlist1	19	$5.1*10^{6}$	241	576	4	0.0 sec
	Orderlist2	51	$4.6*10^{8}$	24,199	99,740	5	0.1 sec
	Orderlist4	104	$1.4*10^{10}$	$2.9*10^{5}$	$1.5*10^{6}$	5	2.5 sec
	Orderlist5	207	$4.1*10^{11}$	$2.8*10^{6}$	$5.4*10^{6}$	4	42.2 sec
k = 10	Orderlist1	19	$7.2*10^{13}$	$2.6*10^{5}$	$5.0*10^{5}$	8	0.8 sec
	Orderlist2	51	$3.3*10^{17}$	$6.8*10^{8}$	$1.2*10^{9}$	9	54.0 min
k = 15	Orderlist1	19	$2.4*10^{21}$	$2.5*10^{7}$	$1.1*10^{8}$	10	81.8 sec
	Orderlist2	51	$3.5*10^{26}$	$> 4.8*10^{10}$	-	-	> 4 days
k = 20	Orderlist1	19	$1.6*10^{29}$	$4.4*10^{8}$	$>2.1*10^{9}$	11	41.0 min
k = 25	Orderlist1	19	$2.1*10^{37}$	$4.4*10^{8}$	$>2.1*10^{9}$	11	17.0 hrs

The complete-search rescheduling algorithm is tested using one random test cutlist (cutlist number 121), unscheduling the first scheduled kicker. Gap_{min} is set to 5mm (Eq. (6.5)). Table 6.4 shows the results; where *InitialNumComp* is the number of components on the test cutlist before rescheduling and

FeasblNumLists is the number of feasible replacement cultists, *cc* is the number of consistency checks. *runtime* is 0.0 seconds for all tests.

Table 6.4 Rescheduling test

K		*InitialNumComp*	*FeasblNumLists*	*cc*
K = 5	Orderlist1	1	8	13
	Orderlist2	3	12	23
	Orderlist4	4	9	26
	Orderlist5	3	22	66
K = 10	Orderlist1	6	4	11
	Orderlist2	8	5	15
	Orderlist4	8	9	26
	Orderlist5	7	51	113
K = 15	Orderlist1	8	9	23
	Orderlist2	10	14	39
	Orderlist4	10	40	95
	Orderlist5	12	52	135
K = 20	Orderlist1	9	14	37
	Orderlist2	10	28	77
	Orderlist4	12	51	114
	Orderlist5	14	106	261
K = 25	Orderlist1	10	18	55
	Orderlist2	11	34	143

	Orderlist4	13	82	244
	Orderlist5	14	263	513

6.6 Heuristic scheduling

In this approach, scheduling is done using heuristic rules derived from observing operations in the rough mill. These rules are implemented to meet several heuristics that are specific to the RMSP such as physical constraints of the sorting area, priority and due date of components, and maintaining a uniform size distribution to provide the chopsaw with a variety of lengths.

This method is the one used by the cutlist recommend agent (CRA) in the RMDSSS described in Chapter 5.

6.6.1 Heuristic initial scheduling algorithm

Step 1. Components in the order list are pre-sorted according to the following criteria:

a. Due Date to reflect urgency

b. Priority (according to product type)

c. Quantity to allow higher quantities longer time for cutting

Step 2. For each component, all the valid kickers are determined (based on each kicker's length restrictions Eq. (6.3)). Valid kickers are then sorted by range R_j (Eq. (6.2)).

Step 3. The longest and shortest components on the order list are assigned. The longest component is given priority because it takes longer

time to produce, while the shortest component helps in reducing waste. If the second-shortest component has a closer due date than the shortest component, and the difference between their lengths is less than 100 mm, the second-shortest component is scheduled instead of the shortest one.

Step 4. The rest of components are scheduled to the first valid kicker (the one with the smallest range). If that kicker is already scheduled for another component, the next kicker is checked, and so on. Other conditions for scheduling are:

a. Conflict in length with components scheduled to neighbouring kickers according to Eq. (6.1), and

b. Conflict in length with other scheduled components (for a uniform length distribution) according to Eq. (6.5). The minimum gap value Gap_{min} is initially set to *100 mm*.

Step 5. If the scheduling rate (number of assigned components) is too low (less than 18 components[3]), the scheduling process is repeated relaxing the length distribution restriction by setting Gap_{min} to *50 mm*, then *5 mm*.

Step 6. If the scheduling rate is still too low, the scheduling process is repeated ignoring the minimum length requirement of the kickers, keeping only the maximum length requirement (Eq. (6.4)) in order to allow

[3] Using the ROMI-RIP simulator, significant yield increases were observed as a result of increasing sorting capacities (number of components in the cutlist). However, a plateau is reached around 18 to 20 components, where additional sorting capacity increases result in negligible yield gains [81].

for more sorting locations to be used. Gap_{min} is set to 100 mm, 50 mm and then 5 mm to relax the length distribution condition.

The order of the heuristic search algorithm for c components and k_v valid kickers is as follows:

$$O(c^2+ck_v^2) \quad \textbf{(6.10)}$$

And since mostly the number of components is greater than the number of kickers and valid kickers ($k_v < c$), the order of the algorithm can be approximated to:

$$O(c^3) \quad \textbf{(6.11)}$$

6.6.2 Heuristic replacement scheduling algorithm

Step 1. Order list components are pre-sorted according to the same criteria used in step 1 of the initial scheduling algorithm.

Step 2. Search the sorted order list for a component similar to the finished component (having the same length, width and quality requirements). This is done in order to maintain the previous length distribution and to ease the manual stacking process of components. Terminate if a component is found, otherwise go to the next step.

Step 3. If the longest item is not currently on the cutlist, attempt to schedule it, with minimal length distribution conditions (Gap_{min}=*5mm*). Terminate if a component is found, otherwise go to the next step.

Step 4. The shortest (or second-shortest as described above) component is examined for scheduling with minimal length distribution conditions. Terminate if a component is found, otherwise go to the next step.

Step 5. Components are examined in the pre-sorted order, and are assigned to available kickers using the same constraints of step 4 of the initial scheduling algorithm (Eq. (6.1) and Eq. (6.5)).

Step 6. If there is no replacement component found, the above step is repeated with less restrictions, similar to those of steps 5 and 6 of the initial scheduling method.

6.6.3 Heuristic scheduling as a search method

Pre-sorted component represent the variables of the search tree. The domain values are the valid kickers. Sorting the domain values represent a *best-first* strategy for searching[4]. Heuristics focus on choosing the next neighbour in the search to move to [65]. The first feasible domain value is selected and no further domain values are explored. Figure 6.7shows an example of three pre-sorted components, domain values represent sorted valid kickers, and light grey nodes represent unassigned components.

[4] Assigning a component to the kicker with the smallest range is a *best-fit* strategy.

Repeating the search is equivalent to relaxing the consistency check conditions. In Figure 6.8, when the conditions are relaxed, more components are assigned compared to Figure 6.7.

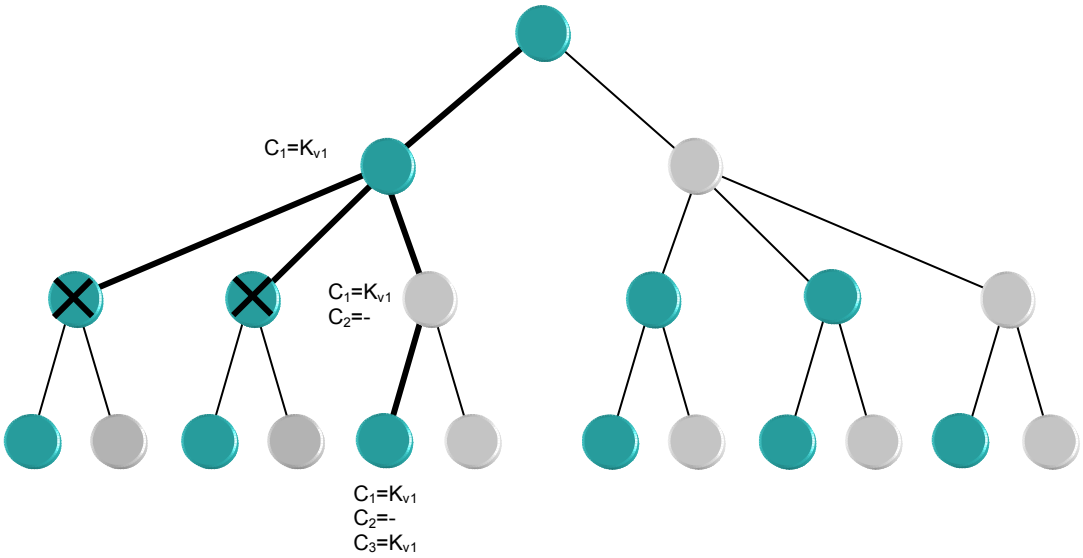

Figure 6.7 Heuristic scheduling as a search problem

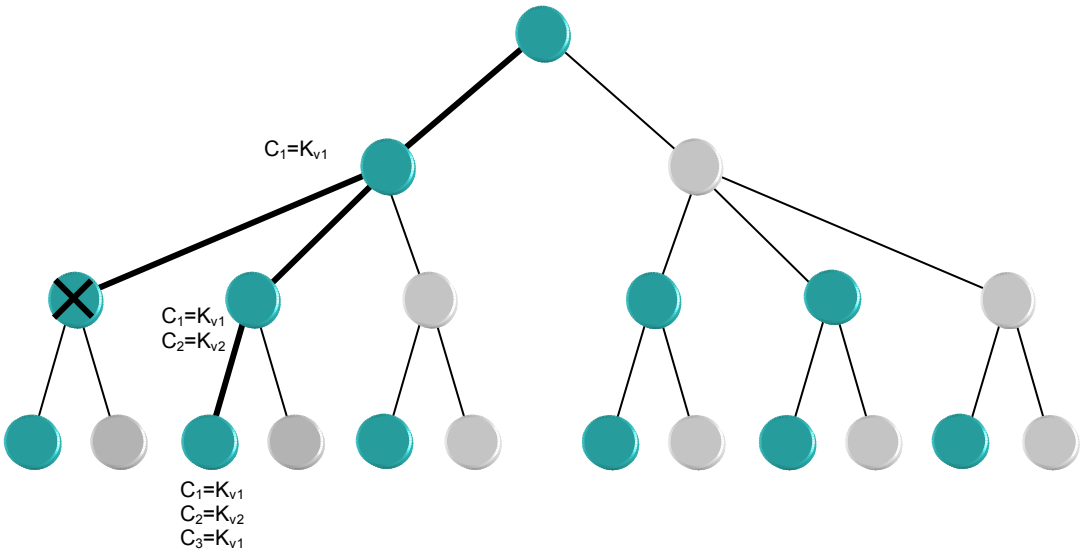

Figure 6.8 Repeating heuristic search with relaxed conditions results in assigning more components

6.7 Randomized heuristic scheduling

In the above sections, it was shown that the backtrack algorithm finds all feasible solutions, which can be a large number and take a long time to run. In addition, a heuristic algorithm based on best-first search that provides one solution was presented.

It is interesting to explore providing the user with alternative recommended cutlists to choose from, all of which are feasible solutions. The randomized heuristic algorithm is presented, which adds a random factor to explore different areas of the search space.

6.7.1 Randomized heuristic *n*-initial scheduling algorithm

Randomized heuristic scheduling algorithm is similar to the heuristic algorithm, but instead of sorting the valid kickers of components by range, their order is randomized (shuffled) to get a different solution every time. This is repeated for *n* times and the schedules are presented to the user. Suggested value of *n* is *5* to *20*.

Therefore, the randomized heuristic algorithm is the same as the heuristic algorithm listed in section 6.6.1, except for step 2:

> Step 2. For each component, all the valid kickers are determined (based on each kicker's length restrictions). Randomize the order of the valid kickers.

6.7.2 Heuristic *n*-replacement scheduling algorithm

The algorithm provides the first *n* replacement components according to the heuristic replacement scheduling algorithm. This gives the user more options to choose from.

6.8 Evaluation of heuristic algorithms

Both heuristic initial algorithm and randomized heuristic *n*-initial algorithms were tested on dataset *orderlist5* with number of components *c = 207,* number of kickers *k = 25*, and number of lists *n = 10*. All the heuristic initial and replacement algorithms provide instant results.

Output is shown in Table 6.5. Each cutlist is displayed in one column as a list of components in the format of component number (unique number for each

component) and component length *(compNo/compLength).* The heuristic initial cutlist is displayed first (*HL*), followed by the 10 randomized heuristic initial cultists (*L1-L10*). The results are validated by tracing debug statements, and analyzing the order list and kicker information.

The randomized heuristic algorithm provided more choices of cultists, some with more components than the heuristic method.

Table 6.5 Output of heuristic and randomized heuristic *n*-initial scheduling (compNo/compLength)

	HL	*L1*	*L2*	*L3*	*L4*	*L5*	*L6*	*L7*	*L8*	*L9*	*L10*
K1	0 / 580	83 / 780	105 / 730				0 / 580	105 / 730	105 / 730		
K2	83 / 780	0 / 580		105 / 730	105 / 730	105 / 730			0 / 580	170 / 673	83 / 780
K3			0 / 580	0 / 580	0 / 580	0 / 580	105 / 730	0 / 580		83 / 780	0 / 580
K4	13 / 2810		75 / 2110	75 / 2110	79 / 3010	54 / 2376		79 / 3010			54 / 2376
K5	54 / 2376	79 / 3010	13 / 2810	79 / 3010	54 / 2376		141 / 2010	141 / 2010	141 / 2010	13 / 2810	79 / 3010
K6	125 / 3610	125 / 3610	124 / 3551	125 / 3610	124 / 3551	125 / 3610	125 / 3610	124 / 3551	124 / 3551	125 / 3610	
K7	6 / 1510	6 / 1510	72 / 1810	179 / 1410	115 / 1670		72 / 1810	6 / 1510	115 / 1670		28 / 1610
K8	179 / 1410	115 / 1670	179 / 1410	72 / 1810	72 / 1810	28 / 1610	179 / 1410		72 / 1810	6 / 1510	6 / 1510
K9	194 / 876	154 / 946	196 / 1086	175 / 1028	175 / 1028	66 / 1210	66 / 1210	175 / 1028	194 / 876	66 / 1210	196 / 1086
K10	175 / 1028	196 / 1086	194 / 876	66 / 1210	66 / 1210	175 / 1028	175 / 1028	194 / 876	196 / 1086	175 / 1028	66 / 1210
K11	79 / 3010	141 / 2010	79 / 3010	13 / 2810		141 / 2010		75 / 2110	13 / 2810	54 / 2376	13 / 2810
K12	75 / 2110	54 / 2376	141 / 2010	54 / 2376	13 / 2810	75 / 2110	54 / 2376	13 / 2810	79 / 3010	75 / 2110	141 / 2010
K13		124 / 3551	125 / 3610	141 / 2010	75 / 2110	13 / 2810	79 / 3010	125 / 3610	125 / 3610	141 / 2010	125 / 3610

K14	72 / 1810		6 / 1510	28 / 1610	28 / 1610	179 / 1410	28 / 1610	179 / 1410	28 / 1610	115 / 1670	
K15	28 / 1610	72 / 1810	28 / 1610	6 / 1510	179 / 1410	72 / 1810	6 / 1510	28 / 1610		179 / 1410	115 / 1670
K16	196 / 1086	175 / 1028	175 / 1028	196 / 1086	194 / 876	194 / 876	196 / 1086	66 / 1210	175 / 1028	194 / 876	194 / 876
K17	66 / 1210	66 / 1210	66 / 1210	194 / 876	196 / 1086	196 / 1086	194 / 876	196 / 1086	66 / 1210	196 / 1086	175 / 1028
K18	141 / 2010	75 / 2110	54 / 2376		141 / 2010		75 / 2110		54 / 2376		
K19		13 / 2810				79 / 3010	13 / 2810	54 / 2376	75 / 2110	79 / 3010	75 / 2110
K20	124 / 3551			124 / 3551	125 / 3610	124 / 3551	124 / 3551			124 / 3551	124 / 3551
K21	115 / 1670	179 / 1410	115 / 1670		6 / 1510	115 / 1670	115 / 1670	72 / 1810	179 / 1410	28 / 1610	179 / 1410
K22		28 / 1610		115 / 1670		6 / 1510		115 / 1670	6 / 1510	72 / 1810	72 / 1810
K23	170 / 673	170 / 673	83 / 780	170 / 673	83 / 780	170 / 673	83 / 780	170 / 673	83 / 780	0 / 580	170 / 673
K24			170 / 673				170 / 673	83 / 780	170 / 673	105 / 730	
K25		194 / 876		83 / 780	170 / 673	83 / 780					
num	**19**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**19**

L10 in Table 6.5 is used as a test cutlist for replacement algorithms with component on kicker number eight (K8) marked as done.

Table 6.6 displays the results of the heuristic replacement and *n*-replacement algorithms in the same format as above. The heuristic replacement results are displayed first (*HL*), followed by the ten replacement cultists (*L1-L10*). The cultists are validated by examining the suggested replacement component against the available unscheduled components in the order list.

The heuristic *n*-replacement provided more choices for replacement components, starting with components of the same length as the component that was done.

Table 6.6 Output of heuristic replacement and heuristic *n*-replacement scheduling (compNo/compLength), done Kicker is K8, done component number is 6

	HL	***L1***	***L2***	***L3***	***L4***	***L5***	***L6***	***L7***	***L8***	***L9***	***L10***
K1											
K2	83 / 780	83 / 780	83 / 780	83 / 780	83 / 780	83 / 780	83 / 780	83 / 780	83 / 780	83 / 780	83 / 780
K3	0 / 580	0 / 580	0 / 580	0 / 580	0 / 580	0 / 580	0 / 580	0 / 580	0 / 580	0 / 580	0 / 580
K4	54 / 2376	54 / 2376	54 / 2376	54 / 2376	54 / 2376	54 / 2376	54 / 2376	54 / 2376	54 / 2376	54 / 2376	54 / 2376
K5	79 / 3010	79 / 3010	79 / 3010	79 / 3010	79 / 3010	79 / 3010	79 / 3010	79 / 3010	79 / 3010	79 / 3010	79 / 3010
K6											
K7	28 / 1610	28 / 1610	28 / 1610	28 / 1610	28 / 1610	28 / 1610	28 / 1610	28 / 1610	28 / 1610	28 / 1610	28 / 1610
K8	**138 / 1510**	**138 / 1510**	**181 / 1510**	**92 / 1510**	**48 / 1510**	**70 / 1510**	**160 / 1510**	**114 / 1510**	**201 / 1510**	**27 / 1510**	**26 / 1480**
K9	196 / 1086	196 / 1086	196 / 1086	196 / 1086	196 / 1086	196 / 1086	196 / 1086	196 / 1086	196 / 1086	196 / 1086	196 / 1086
K10	66 / 1210	66 / 1210	66 / 1210	66 / 1210	66 / 1210	66 / 1210	66 / 1210	66 / 1210	66 / 1210	66 / 1210	66 / 1210
K11	13 / 2810	13 / 2810	13 / 2810	13 / 2810	13 / 2810	13 / 2810	13 / 2810	13 / 2810	13 / 2810	13 / 2810	13 / 2810
K12	141 / 2010	141 / 2010	141 / 2010	141 / 2010	141 / 2010	141 / 2010	141 / 2010	141 / 2010	141 / 2010	141 / 2010	141 / 2010
K13	125 / 3610	125 / 3610	125 / 3610	125 / 3610	125 / 3610	125 / 3610	125 / 3610	125 / 3610	125 / 3610	125 / 3610	125 / 3610
K14											
K15	115 / 1670	115 / 1670	115 / 1670	115 / 1670	115 / 1670	115 / 1670	115 / 1670	115 / 1670	115 / 1670	115 / 1670	115 / 1670
K16	194 / 876	194 / 876	194 / 876	194 / 876	194 / 876	194 / 876	194 / 876	194 / 876	194 / 876	194 / 876	194 / 876

K17	175 / 1028	175 / 1028	175 / 1028	175 / 1028	175 / 1028	175 / 1028	175 / 1028	175 / 1028	175 / 1028	175 / 1028	175 / 1028
K18											
K19	75 / 2110	75 / 2110	75 / 2110	75 / 2110	75 / 2110	75 / 2110	75 / 2110	75 / 2110	75 / 2110	75 / 2110	75 / 2110
K20	124 / 3551	124 / 3551	124 / 3551	124 / 3551	124 / 3551	124 / 3551	124 / 3551	124 / 3551	124 / 3551	124 / 3551	124 / 3551
K21	179 / 1410	179 / 1410	179 / 1410	179 / 1410	179 / 1410	179 / 1410	179 / 1410	179 / 1410	179 / 1410	179 / 1410	179 / 1410
K22	72 / 1810	72 / 1810	72 / 1810	72 / 1810	72 / 1810	72 / 1810	72 / 1810	72 / 1810	72 / 1810	72 / 1810	72 / 1810
K23	170 / 673	170 / 673	170 / 673	170 / 673	170 / 673	170 / 673	170 / 673	170 / 673	170 / 673	170 / 673	170 / 673
K24											
K25											

6.9 Summary

In this chapter, the RMSP (Rough Mill Scheduling Problem) was presented. The physical constraints and the unique characteristics of this problem were described. Traditional methods cannot be applied to the problem because of unknown processing times.

Backtrack, heuristic and randomized heuristic solutions were presented. The backtrack solution provides all feasible cultists. However, the number of lists is very large, and it takes a long processing time. The heuristic solution provides one solution that can be used directly in the simulation (as in Chapter 5). The randomized heuristic solution provides *n* solutions for decision support.

For component replacement, I presented a heuristic method that suggested one replacement component, a full-search method that presented all

possible replacement components and heuristic *n*-replacement method that provides *n* replacement cutlists.

7: CONCLUSION

In this section, a summary of the content of the thesis is presented and a list of contributions is outlined. Finally, future research directions are suggested.

7.1 Summary

In a rough mill, boards of lumber of approximate sizes are cut to produce components of fixed sizes and qualities. Improving the processes in the rough mill provides cost-saving of expensive lumber and reduces the waste of natural resources.

The operations in a Canadian rough mill are investigated and documented. The decisions taken by operators are analyzed as well during a trip to the rough mill. The two main decisions in the rough mill are the jag selection problem (material selection) and the scheduling problem (component or part scheduling).

The history of the project is outlined which starts with a 3D simulation of the rough mill. Collaborative work is done in the areas of jag selection, jag sequencing and scheduling. The goal of the rest of this research is defined with two challenges: developing an agent-based decision support and simulation system and developing scheduling algorithms to solve the initial and replacement rough mill scheduling problem.

Before tackling the first challenge of designing the agent-based system, the concept of an agent is defined and a survey of agent-based manufacturing systems is presented. Agents are used for several manufacturing applications ranging from control, scheduling and planning to enterprise operations and supply-chain management.

The agent solution is first implemented as a prototype or a framework for one line of production. This prototype is used to develop and validate the behaviours of the agents and the communications between them. The system is composed of three simulation agents (ripsaw, conveyor and chopsaw) and two decision support agents (jag selection and cutlist recommend). This prototype is extended to two lines using a negotiation protocol.

The prototype is implemented into the Rough Mill Decision Support and Simulation System (RMDSSS). Simulation agents use historical data to simulate rough mill operations. Jag selection agent uses historical and current data to select jags from the warehouse inventory. Cutlist Recommend Agent uses heuristic scheduling to recommend cultists from a given order list.

Synchronization of the simulation is done using the conveyor, which mimics the real conveyor behaviour that detects jamming and stops the ripsaw. Simulation time is calculated using bottleneck detection. The pros and cons of using the agent paradigm are discussed.

The second challenge is the Rough Mill Scheduling Problem (RMSP). The problem is detailed and formulated. Natural defects in the wood and inaccurate dimensions result in unknown processing times, therefore traditional scheduling

methods cannot be used to solve this problem. The problem is formulated as a constraint satisfaction and search problem. Backtrack search is implemented and evaluated. The number of feasible solutions and the runtime grow exponentially when increasing the number of components and kickers.

A heuristic method for scheduling is developed. It can be viewed as a best-first search method. Pre-sorted components are assigned to the first valid kicker with the smallest range. If the number of components on the list is small, the search is repeated with relaxed parameters. A randomized heuristic method is also implemented that assigns pre-sorted components to the first valid random kicker, and *n*-solutions are presented to the user.

Three replacement scheduling algorithms are also developed to dynamically replace components done on the line with new ones. The first method presents all the feasible replacement components, the second method selects one component, and the third method presents *n*-components.

7.2 Contributions

1. Identifying the operations and decisions in a Canadian rough mill.
2. Designing and developing a rough mill ontology.
3. Designing and implementing a decision support and simulation system for rough mills.
4. Applying the multi-agent paradigm to rough mills.
5. Adding a second line of production that shares the same material resources and orders.

6. Formulating the problem of scheduling naturally-defective material.
7. Developing backtrack, heuristic and randomized heuristic scheduling methods and evaluating them with rough mill data.
8. Jointly developing decision support algorithms.

7.3 Future research

7.3.1 Agent-based decision support and simulation

1. Agent-based implementation makes it easy to extend the system to other operations in the rough mill such as generating the order list from the work orders. Moreover, it is possible to run the agents on several containers in a distributed fashion and provide remote or web-based monitoring [45] [95].
2. Learning algorithms can be used to improve the simulation agents [75].

7.3.2 Scheduling

1. Several local search methods with various objective functions can be implemented and compared using the simulation. An example of the objective function is the one used in [58] as described in Section 3.4.1. Examples of local search methods that can be used are taboo search [24], simulated annealing and constraint optimization methods [31] [65]. It is also possible to apply swarm intelligence as a local optimization method [8] [9], such as ant

colony optimization, particle swarm optimization and intelligent water drops.

2. Scheduling can gather data from the simulation and improve on existing heuristics using learning methods such as the one presented by Park et al. [15].

REFERENCE LIST

[1] A. Abraham, K. Franke and M. Köppen, “Intelligent Systems Design and Applications,” *Third International Conference on Intelligent Systems Design and Applications, Berlin, New York, Springer,* 2003.

[2] A. Genco, S. Sorce, G. Reina and G. Santoro, “An agent-based service network for personal mobile devices,” *IEEE Pervasive Computing,* vol. 5, pp. 54-61, 2006.

[3] A. Koestler, *The Ghost in the Machine.* New York: Macmillan, 1968.

[4] B. C. Csaji, B. Kadar and L. Monostori, “Improving multi-agent based scheduling by neurodynamic programming,” *Holonic and Multi-Agent Systems for Manufacturing, Lecture Notes in Artificial Intelligence,* vol. 2744, Springer-Verlag, 2004, pp. 110-123.

[5] B. Denkena, M. Zwick and P. Woelk, “Multiagent-based process planning and scheduling in context of supply chains,” *Holonic and Multi-Agent Systems for Manufacturing, Lecture Notes in Artificial Intelligence,* vol. 2744, Springer-Verlag, 2004, pp. 100-109.

[6] B. Henderson-Sellers and P. Giorgini, *Agent-Oriented Methodologies.* Hershey, PA,Idea Group Pub., 2005.

[7] Beangenerator plugin, “Acklin BV – Beangenerator,” http://acklin.nl/beangenerator, accessed July 7, 2005.

[8] C. Blum and D. Merkle, *Swarm Intelligence: Introduction and Applications: Swarm Intelligence in Optimization.* Natural Computing Series, Springer, 2008.

[9] C. Lim, L. Jain and S. Dehuri, *Innovations in Swarm Intelligence: A Review of Particle Swarm Optimization Methods Used for Multimodal Optimization*, Springer-Verlag, Berlin, Heidelberg, 2009.

[10] C. Ramos, “An architecture and a negotiation protocol for the dynamic scheduling of manufacturing systems,” *Proc. of the IEEE International Conference on Robotics and Automation*, 1994, pp. 3161-3166.

[11] C. Ramos, “Task negotiation for distributed manufacturing systems,” *Proc. of the IEEE International Symposium on Assembly and Task Planning,* 1995, pp. 259-264.

[12] C. Yu and H. Huang, “Development of virtual foundry fab based on distributed multi-agents,” *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics, 2001*, pp. 1030-1035.

[13] D. B. Kotak, M. Fleetwood, H. Tamoto and W. A. Gruver, “Operational scheduling for rough mills using a virtual manufacturing environment,” in *Proc. of IEEE International Conference on Systems, Man & Cybernetics,* 2001, pp. 140-145.

[14] D. Diep, P. Massotte and A. Meimouni, “A distributed manufacturing execution system implemented with agents: The PABADIS model,” *Proc. of the IEEE International Conference on Industrial Informatics,* 2003, pp. 301-306.

[15] D. E. Brown and W. T. Scherer, *Intelligent Scheduling Systems.* Boston, Kluwer Academic Publishers, 1995.

[16] D. Jarvis, J. Jarvis, D. McFarlane, A. Lucas and R. Ronnquist, “Implementing a multi-agent systems approach to collaborative autonomous manufacturing operations,” *Proc. of the IEEE Aerospace Conference, 2001,* pp. 2803-2811.

[17] D. Naso and G. Maione, “Recent developments in the application of computational intelligence to multi-agent manufacturing control,” Proc. of the *IEEE International Conference on Fuzzy Systems*, 2001, pp. 990-994.

[18] D. Sabaz, W. A. Gruver, and M. H. Smith, “Distributed systems with agents and holons,” *Proc. of the 2004 IEEE International Conference on Systems, Man, and Cybernetics*, October 2004, pp. 1958 - 1963 vol.2.

[19] D. Zhang, A. Anosike and M.K. Lim, “Dynamically Integrated Manufacturing Systems (DIMS) - A multiagent approach,” *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 37, no. 5, 2007, pp. 824-850.

[20] E. Elghoneimy and W. A. Gruver, “Agent-based manufacturing systems: A survey,” *Proc. of the IEEE SMC International Conference on Distributed Human-Machine Systems*, 2008, pp. 127-132.

[21] E. Elghoneimy, “Loewen Windows rough mill production system: Information gathering report,” *Technical Report #204, Intelligent Robotics and Manufacturing Systems Laboratory, Simon Fraser University,* October 2004.

[22] E. Elghoneimy, O. Uncu, W. A. Gruver and D. B. Kotak, “Simulation and decision support models for rough mills: A multi-agent perspective,” *Proc. of the IEEE International Conference on System, Man and Cybernetics*, 2005, pp. 3723-3728.

[23] E. Elghoneimy, O. Uncu, W. A. Gruver, D. B. Kotak and M. Fleetwood, “An intelligent decision-support system for rough mills,” *International Journal of Manufacturing Technology and Management*, A. Ramirez-Serrano; R. W. Brennan, (eds.), vol. 8, issue 1/2/3, Inderscience, 2006, pp. 203-225.

[24] E. Nowicki and C. Smutnicki, “A fast taboo search algorithm for the job shop problem,” *Management Science Journal,* vol 42, issue 6, June 1996, pp. 797-813.

[25] E. Thomas and J. Weiss, “Rough mill simulator version 3.0: An analysis tool for refining rough mill operations,” *Forest Products Journal*, 2006, pp. 53-58.

[26] F. Bellifemine, A. Poggi, G. Rimassa, “Developing multi-agent systems with a FIPA-compliant agent framework,” *Software - Practice & Experience*, John Wiley & Sons, Ltd. vol 3, no. 3, 2001, pp. 103-128,.

[27] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, “JADE, a white paper,” *EXP, In Search of Innovation*, vol. 3, no. 3, Telecom Italia Lab, Turin, Italy, September 2003, http://jade.tilab.com/, accessed February, 2010.

[28] F. Bellifemine, G. Caire, T. Trucco, and G. Rimassa, “JADE programmer’s guide,” http://jade.tilab.com/doc/programmersguide.pdf, accessed February, 2010.

[29] FIPA, “The Foundation of Intelligent Physical Agents,” http://www.fipa.org/, accessed February, 2010.

[30] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* Cambridge, MA: MIT Press, 2000.

[31] H. H. Hoos, *Stochastic Local Search: Foundations and Applications.* Morgan Kaufmann / Elsevier, 2004.

[32] H. J. Yoon and W. Shen, “Simulation-based real-time decision making for manufacturing automation systems: A review,” *International Journal of Manufacturing Technology and Management,* vol. 8, 2006, pp. 188-202.

[33] H. Park and W. Lee, “Agent-based shop control system under holonic manufacturing concept,” *Proc. of the 4th Korea-Russia International Symposium on Science and Technology,* 2000, pp. 116-121.

[34] H. Van Brussel, J. Wyns, P. Valckenaers, L. Bongaerts and P. Peeters. “Reference architecture for holonic manufacturing systems: PROSA,” *Computers in Industry,* vol. 37, 1998, pp. 255-74.

[35] Holonic Manufacturing Systems Consortium. Available: http://www.ims.org/sites/default/files/2.4.15.1%20Project%20Summary%20-%20%20HMS%20PHASE%20I%20and%20II.pdf, accessed February 2010.

[36] I. Dumitrache, S. I. Caramihai and A. M. Stanescu, “Intelligent agent-based control systems in manufacturing,” *Proc. of the 2000 IEEE International Symposium on Intelligent Control,* 2000, pp. 369-374.

[37] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland and Tom Carey, *Human Computer Interaction.* Addison Wesley, 1994.

[38] J. Wiedenbeck, “Simulation for rough mill options,” *Wood & Wood Products Journal,* November 1992.

[39] J.M. Mendes, P. Leitão, F. Restivo and A.W. Colombo, “Service-oriented agents for collaborative industrial automation and production system,” *Proc. of the 4th International Conference on Industrial Applications of Holonic and Multi-Agent Systems, Linz, Austria,* 2009, pp. 13-24.

[40] K. Glanzer, A. Hammerle and R. Geurts, “The application of ZEUS agents in manufacturing environments,” *Proc. of the 2001 Conference on Database and Expert Systems Applications, 2001*, pp. 628-632.

[41] L. B. Sheremetov, J. Martinez and J. Guerra, “Agent architecture for dynamic job routing in holonic environment based on the theory of constraints,” *Holonic and Multi-Agent Systems for Manufacturing, Lecture Notes in Artificial Intelligence,* vol. 2744, Springer-Verlag, 2004, pp. 124-133.

[42] L. O. Araujo Jr., N. Maruyama, P. E. Miyagi, L. A. Moscato and D. J. Santos F, “A control architecture for distributed production systems using a virtual cellular manufacturing and agents society based approach,” *Proc. of the IEEE International Symposium on Industrial Electronics,* 2003, pp. 862-867.

[43] L. Panait and S. Luke, “Cooperative multi-agent learning: The state of the art,” *Autonomous Agents and Multi-Agent Systems.,* vol. 11, 2005, pp. 387-434.

[44] L. Wang, S. J. Turner and F. Wang, “Interest management in agent-based distributed simulations,” *Proc. of the IEEE International Symposium on Distributed Simulation and Real-Time Applications*, October 2003, pp. 20–27.

[45] L. Wang, W. Shen and S. Lang, “Wise-ShopFloor: A web-based and sensor-driven shop floor environment,” *Proc. of 7th International Conference on Computer Supported Cooperative Work in Design,,*2002, pp. 413-418.

[46] M. A. Weiss, *Data Structures and Algorithm Analysis*, Redwood City, Calif.: Benjamin/Cummings Pub. Co., 1995.

[47] M. J. P. Shaw, “Distributed planning in cellular flexible manufacturing systems,” *INFOR Canadian Journal of Operational Research and Information Processing,* vol. 25, 1987, pp. 13-25.

[48] M. Luck, R. Ashri and M. D'Inverno, *Agent-Based Software Development.* Boston: Artech House, 2004, pp. 208.

[49] M. Paolucci and R. Sacile, *Agent-Based Manufacturing and Control Systems: New Agile Manufacturing Solutions for Achieving Peak Performance.* Boca Raton, FL: CRC Press, 2005, pp. 269.

[50] M. Pechoucek, J. Vokrinek and P. Becvar, "ExPlanTech: Multiagent support for manufacturing decision making," *IEEE Intelligent Systems,* vol. 20, 2005, pp. 67-74.

[51] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems.* Upper Saddle, N.J., Prentice Hall, 2002.

[52] M. Ulieru, D. Stefanoiu and D. Norrie, "Holonic self-organization of multi-agent systems by fuzzy modeling with application to intelligent manufacturing," *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics,* 2000, pp. 1661-1666.

[53] M. Wooldridge and N. R. Jennings, "Intelligent agents: theory and practice," *Knowledge Engineering Review,* vol. 10, 1995, pp. 115-52.

[54] M. Wooldridge, "Intelligent agents: The key concepts," *Multi-Agent-Systems and Applications II: 9th ECCAI-ACAI/EASSS*, 2002, pp. 151-190.

[55] N. Jennings and M. J. Wooldridge, *Agent Technology: Foundations, Applications, and Markets.* Berlin; New York: Springer-Verlag, 1998, pp. 325.

[56] N. Liu, M. A. Abdelrahman and S. Ramaswamy, "A multi-agent model for reactive job shop scheduling," *Proc. of the Thirty-Sixth Southeastern Symposium on System Theory,* 2004, pp. 241-245.

[57] N. R. Jennings and S. Bussmann, "Agent-based control systems: Why are they suited to engineering complex systems?" *IEEE Control Systems Magazine,* vol. 23, 2003, pp. 61-73.

[58] N. Siu, E. Elghoneimy, Y. Wang, W. A. Gruver, M. Fleetwood and D. B. Kotak, "Rough mill component scheduling: Heuristic search versus genetic algorithms," *Proc. of the IEEE International Conference on Systems, Man and Cybernetics,* 2004, pp. 4226-4231.

[59] O. Uncu and E. Elghoneimy, W. A. Gruver, D. B. Kotak and M. Fleetwood, "Identifying aggregation weights of decision criteria: Application of fuzzy systems to wood product manufacturing," *in Forging New Frontiers: Fuzzy Pioneers I: Studies in Fuzziness and Soft Computing, M. Nikravesh.; J. Kacprzyk; L. A. Zadeh, (eds.),* vol. 217, 2007, pp. 415-435.

[60] O. Uncu, E. Elghoneimy, W. A. Gruver, D. B. Kotak and M. Fleetwood, "A model for identifying aggregation operator weights using a fuzzy rulebase and

genetic algorithms: Application to the wood products manufacturing industry," *Proc. of the Berkeley Initiative in Soft Computing Symposium,* 2005.

[61] O. Uncu, E. Elghoneimy, W. A. Gruver, D. Kotak and M. Fleetwood, "Jag sequencing in rough mill operations," *Proc. of the IEEE International Conference on System, Man and Cybernetics,*2005, pp. 300-305.

[62] P. Dewan and S. Joshi, "Distributed scheduling of job shop using auctions," *Proc. of the Second International Workshop on Intelligent Manufacturing Systems,* 1999, pp. 33-40.

[63] P. Moraitis and N. Spanoudakis, "Combining Gaia and JADE for multi-agent systems development," 4th International Symposium 'From Agent Theory to Agent Implementation' *Proc. of the 17th European Meeting on Cybernetics and Systems Research*, Vienna, Austria, April 2004.

[64] P. Tichý, P. Slechta, F. Maturana and S. Balasubramanian, "Industrial MAS for planning and control," *Multi-Agent Systems and Applications II, Lecture Notes in Artificial Intelligence,* vol. 2322, Springer-Verlag, 2002, pp. 280-295.

[65] P. V. Hentenryck and L. Michel, *Constraint-Based Local Search.* Cambridge, MA, MIT Press, 2005.

[66] P. Vrba, "MAST: Manufacturing agent simulation tool," *Proc. of the IEEE Conference on Emerging Technologies and Factory Automation*, vol.1, September 2003, pp. 282 - 287.

[67] Protégé ontology editor, "The Protégé ontology dditor and knowledge acquisition system," Stanford Medical Informatics, Stanford University School of Medicine, http://protege.stanford.edu/, accessed February, 2010.

[68] R. D. Lathon, A. F. Claassen, D. M. Rochowiak and L. E. Interrante, "Negotiation among scheduling agents to achieve global production goals," *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics,* 1994, pp. 1541-1546.

[69] R. H. Bordini, M. Dastani and J. Dix, A. E. Seghrouchni (editors), *Multi-Agent Programming : Languages, Platforms and Applications*. New York, Springer, 2005.

[70] R. Sikora and M. J. Shaw, "Coordination mechanisms for multi-agent manufacturing systems: Applications to integrated manufacturing scheduling," *IEEE Transactions on Engineering Management,* vol. 44, 1997, pp. 175-187.

[71] R. W. Brennan, "Toward real-time distributed intelligent control: A survey of research themes and applications," *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews, vol. 37, 2007,* pp. 744-765.

[72] R. W. Brennan, J. H. Christensen, W. A. Gruver, D. B. Kotak, D. H. Norrie and E. van Leeuwen, “Holonic manufacturing systems – A technical overview,” *The Industrial Information Technology Handbook, ed. Richard Zurawski,* CRC Press, 2005.

[73] S. Bussmann and K. Schild, “Self-organizing manufacturing control: An industrial application of agent technology,” *Proc. of the Fourth International Conference on Multi Agent Systems,* 2000, pp. 87-94.

[74] S. Franklin and A. Graesser, “Is it an agent, or just a program? A taxonomy for autonomous agents,” *Intelligent Agents III Agent Theories, Architectures, and Languages,* 1997, pp. 21-35.

[75] S. J. Russell and P. Norvig, *Artificial Intelligence : A Modern Approach.* Upper Saddle River, N.J., Prentice Hall, 2010.

[76] S. Jacobi, C. Madrigal-Mora, E. Leon-Soto and K. Fischer, “AgentSteel: An agent-based online system for the planning and observation of steel production,” *Proc. of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems,* 2005, pp. 114-119.

[77] S. Kirn, O. Herzog, P. Lockemann and O. Spaniol (editors), *Multiagent Engineering : Theory And Applications In Enterprises.* Berlin, New York, Springer, 2006.

[78] S. M. Deen and M. Fletcher, “Temperature equilibrium in multi-agent manufacturing systems,” *Proc. of the 11th International Workshop on Database and Expert Systems Applications,* 2000, pp. 259-264.

[79] S. S. Heragu, R. J. Graves, Byung-In Kim and A. St Onge, “Intelligent agent based framework for manufacturing systems control,” *IEEE Transactions on Systems, Man and Cybernetics, Part A,* vol. 32, 2002, pp. 560-573.

[80] S.S. Nestinger, B. Chen and H.H. Cheng, “A mobile agent-based framework for flexible automation systems,” *IEEE/ASME Transactions on Mechatronics, to be published.*

[81] T. Edward and B. John, “Determining the impact of sorting capacity on rip-first rough mill yield,” *Forest Products Society*, vol. 53, no7-8, 2003, pp. 54-60.

[82] T. Heikkila, “An agent architecture for manufacturing control: ManAge,” *Computers in Industry,* vol. 46, 2001, pp. 315-31.

[83] T. Teredesai and V. C. Ramesh, “A multi-agent mixed initiative system for real-time scheduling,” *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics,* 1998, pp. 439-444.

[84] V. Marik, K. Kraus, O. Flek and J. Bezdicek, “Multi-agent decision making architecture and distributed control,” *Proc. of the 2nd IEEE/ECLA/IFIP International Conference on Architectures and Design Methods for Balanced Automation Systems,* 1996, pp. 315-328.

[85] V. Marik, M. Pechoucek, P. Vrba and V. Hrdonka, “FIPA standards and holonic manufacturing,” *Agent Based Manufacturing: Advances in Holonic Approach,* Edited by S. M. Deen, Springer-Verlag, 2003, pp. 89-120.

[86] V. Marik, P. Vrba, K. Hall and F. Maturana, “Rockwell automation agents for manufacturing,” *Proc. of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems,* 2005, pp. 107-113.

[87] W. A. Gruver, D. B. Kotak, E. H. van Leeuwen and D. H. Norrie, “Holonic manufacturing systems: Phase II,” *Holonic and Multi-Agent Systems for Manufacturing, Lecture Notes in Artificial Intelligence,* vol. 2744, Springer-Verlag, 2004, pp. 1-14.

[88] W. Shen and D. H. Norrie, “Agent-based systems for intelligent manufacturing: A state-of-the-art survey,” *International Journal of Knowledge and Information Systems,* vol. 1, 1999, pp. 129-156.

[89] W. Shen, “Distributed manufacturing scheduling using intelligent agents,” *IEEE Intelligent Systems,* vol. 17, 2002, pp. 88-94.

[90] W. Shen, D. H. Norrie and J. Barthès, *Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing.* London; New York: Taylor & Francis, 2000, pp. 386.

[91] W. Shen, D. H. Norrie and R. Kremer, “Developing intelligent manufacturing systems using collaborative agents,” *Proc. of the Second International Workshop on Intelligent Manufacturing Systems,* 1999, pp.157-166.

[92] W. Shen, F. Maturana and D. H. Norrie, “MetaMorph II: An agent-based architecture for distributed intelligent design and manufacturing,” J*ournal of Intelligent Manufacturing,* vol. 11, 2000, pp. 237-251.

[93] W. Shen, L. Wang and Q. Hao, “Agent-based distributed manufacturing process planning and scheduling: A state-of-the-art survey,” *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews,* vol. 36, 2006, pp. 563-77.

[94] W. Shen, S. Y. T. Lang and L. Wang, “iShopFloor: An Internet-enabled agent-based intelligent shop floor,” *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews,* vol. 35, 2005, pp. 371-81.

[95] W. Shen, Y. Li, Q. Hao, S. Wang and H. Ghenniwa, “Implementing collaborative manufacturing with intelligent web services,” *Proc. of the Fifth International Conference on Computer and Information Technology CIT,* 2005, pp. 1063-9.

[96] Y. Song, M.T. Zhang, J. Yi, L. Zhang, and L. Zheng, “Bottleneck station scheduling in semiconductor assembly and test manufacturing using Ant Colony Optimization,” *IEEE Transactions on Automation Science and Engineering,* vol. 4, no. 4, 2007, pp. 569-578.

[97] Y. Upadrashta, J. Vassileva and W. Grassmann, “Social networks in peer-to-peer systems” *Proc. of the Annual Hawaii International Conference on System Sciences* (HICSS '05), January 2005, pp 200c - 200c.

[98] Y. Wang, E. Elghoneimy, W. A. Gruver, M. Fleetwood and D. B. Kotak, “A fuzzy multiple decision support for jag selection,” *Proc. of the Annual Meeting of the North American Fuzzy Information Processing Society*, 2004, pp. 717-722.

[99] Y. Wang, W. A. Gruver, D. B. Kotak, and M. Fleetwood, “A distributed decision support system for lumber jag selection in a rough mill,” *Proc. of the IEEE International Conference on Systems, Man and Cybernetics,* Vol. 1, October 2003, pp. 616-621.