DYNAMIC JOB SHOP SCHEDULING USING ANT COLONY OPTIMIZATION ALGORITHM BASED ON A MULTI-AGENT SYSTEM

ZHOU RONG

(B.Eng., South China University of Technology, P.R. China)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2007

Acknowledgement

The thesis would have been impossible without the invaluable guidance and constant support of my supervisors: Professor Andrew Nee Yeh Ching and Associate Professor Lee Heow Pueh.

I would like to thank Professor Nee for guiding me throughout the entire course with his insights into the field and his unfailing help over the years on a wide range of problems. I am extremely fortunate to be his student.

I cannot thank Professor Lee enough for helping me to clarify my thoughts through many meetings and for providing advanced experimental facilities. His interests in various fields have enabled me to identify a number of directions for future research.

I am also grateful to late Dr. Cheok Beng Teck for leading me into the field of agent technology and Dr. Bud Fox for providing valuable opinions and helping me to improve my writing skills.

I also thank the most important people in my life: my husband, Zou Chunzhong, for his patience and encouragement throughout my Ph.D study, my lovely daughter, Zou Yi Catherine, for lighting up my life like sunshine, and my mom, He Yuru, for her endless love, tolerance, and support.

Lastly, I wish I could share the moment with my father, Zhou Wenzhong. His love and expectations are always the source of courage for me to overcome difficulties and to pursue high goals.

Table of Contents

Summary

A job shop manufacturing system is specifically designed to simultaneously produce different types of products in a shop floor. Job shop scheduling problems (JSSPs) have been studied extensively and most instances of JSSP are NP-hard, which implies that there is no polynomial time algorithm to solve them. As a result, many approximation methods have been explored to find near-optimal solutions within reasonable computational efforts. Furthermore, in a real world, JSSP is generally dynamic with continuous incoming jobs and providing schedules dynamically within constrained computational times in order to optimize the system performance becomes a great challenge.

The developments in both areas of multi-agent systems (MAS) and the behaviour of foraging ants have inspired the current studies to build a scheduling system that can provide quality schedules for a dynamic shop floor. A group of foraging ants is a natural MAS with an internal mechanism to dynamically optimize the routes between their nest and a food source. This optimization mechanism is realized through simple interaction rules among ants and modeled as an algorithm titled Ant Colony Optimization (ACO), which is promising in solving dynamic JSSPs.

In this thesis, a common test bed simulating a generic job shop is firstly built to facilitate a systematic study of the performance of the proposed dispatching rules and algorithms in a dynamic job shop; this is first simulated as a discrete event system (DES) to provide long-term performance evaluations; thereafter it is implemented as an MAS so that data collecting and analysis can be naturally distributed to the most related entities and events can be executed simultaneously at different locations.

Secondly, the test bed further includes a scheduler agent employing ACO to dynamically generate the schedules. The effectiveness of ACO is demonstrated in two dynamic JSSPs with the same mean total workload but different dynamic frequencies and disturbance severity. The effects of its adaptation mechanism are next studied. Furthermore, two important parameters in the ACO algorithm, namely the minimal number of iterations and the size of searching ants per iteration, which control the computational time and the quality of the intermediate solutions, are also examined. The results show that ACO performs effectively in both cases; the adaptation mechanism can significantly improve the performance of ACO; increasing the numbers of iterations and ants per iteration do not necessarily improve the overall performance of ACO.

Finally, experiments were carried out to identify the appropriate application domains defined by machine utilizations, ranges of processing times, and performance measures. The steady-state performances of ACO are compared with those from dispatching rules including first-in-first-out, shortest processing time, and minimum slack time. The experimental results show that ACO can outperform other approaches when the machine utilization or the variation of processing times is not high, otherwise, the dispatching rules will have a better performance.

Nomenclature

A	the machine environment in the *n/m/A/B* classification scheme
ACO	ant colony optimization
ACS	ant colony system
AC^2	ant colony control
A_i	accessible operation list
ANTS	approximate non-deterministic tree search
AS	ant system
AS_{rank}	the ***rank-based*** AS
B	the field of performance measures in the *n/m/A/B* classification scheme
BMS	biological manufacturing system
c	the tightness index for setting the due date of jobs
C_i	the completion time of job J_i
C_{max}	the makespan of job J_i, $C_{max} = \max[C_i]$, where $i = 1, \ldots, m$.
DES	discrete event system
d_i	the due date of job J_i
d_{ij}	the heuristic distance between nodes i and j
e	the base of the natural logarithm (e = 2.71828…)
ev	the event of a new arrival job
EDD	the earliest due date dispatching rule
EAS	the elitist strategy for AS
F_i	the flowtime of job J_i, $F_i = r_i - C_i$
FIFO	first-in-first-out dispatching rule
FMS	flexible manufacturing system

FrMS	fractal manufacturing system
FSP	flow shop problem
$\overline{F}$	the mean flowtime of all the jobs in a schedule, $\overline{F} = \frac{1}{n}\sum_{i=1}^{n} F_i$
G	a job shop
GSSP	group shop scheduling problem
h	the index of iteration number in the ACO scheduling procedure
HMS	holonic manufacturing system
JADE	Java Agent Development Framework
J_i	the i^{th} job arrived at the shop floor
JSSP	job shop scheduling problem
k	the number of occurrences of an event
l	the starting point of the steady state
m	the total number of machines or workcenters
M	machine
MAS	multi-agent system
MHS	material handling system
M_i	the i^{th} machine
M_{ij}	the available times of all machines in workcenter j maintained by ant i
MST	minimum-slacktime dispatching rule
n	the total number of jobs
NA_i	non-accessible operation list
O_{ij}	the j^{th} elementary task of job i to be performed on a machine
P-ACO	population-based ACO
p_{ij}	the processing time of O_{ij}

$p_{ij}(h)$	the probability for an ant to travel from node *i* to node *j* at h^{th} iteration
$\overline{P}$	the mean processing time
PC_i	the total processing times of all the operations of job J_i
P-O-P-M	position-operation-pheromone-matrix
Q	the constant representing the total quality of pheromone on a route;
r_i	the release/arrival time of job J_i
s	the size of iterations
s_{max}	the maximal sets of ants that can be initiated
s_{min}	the minimal sets of ants that can be initiated
S_i	scheduled operation list
SPT	shortest-processing time dispatching rule
t	time
T_i	the tardiness of job J_i, $T_i = \max[0,(C_i - d_i)]$
TSP	traveling salesman problem
$\overline{T}$	the mean tardiness of all jobs in a schedule, $\overline{T} = \frac{1}{n}\sum_{i=1}^{n} T_i$
TC_i	the technical order of job J_i
TWK_i	the total work content of job J_i
u	the number of ants per iteration
U	the utilization rate of a resource
UML	unified modeling language
α	the importance index of pheromone
β	the importance index of distance heuristic
λ	a positive real number in a poisson distribution
D	the mean inter-arrival time

ρ the evaporation coefficient, which can be a real number between 0 and 1.0.

τ_{ij} the quantity of pheromone on the edge connecting node *i* and node *j*

$\tau_{ij}(h)$ is the quantity of pheromone on the edge connecting nodes *i* and *j* at h^{th} iteration

$\Delta\tau_{ij}(h)$ the quantity of increased pheromone on the edge connecting nodes *i* and *j* at h^{th} iteration;

ϑ the rate parameter in the exponential distribution, $\vartheta > 0$

List of Figures

List of Tables

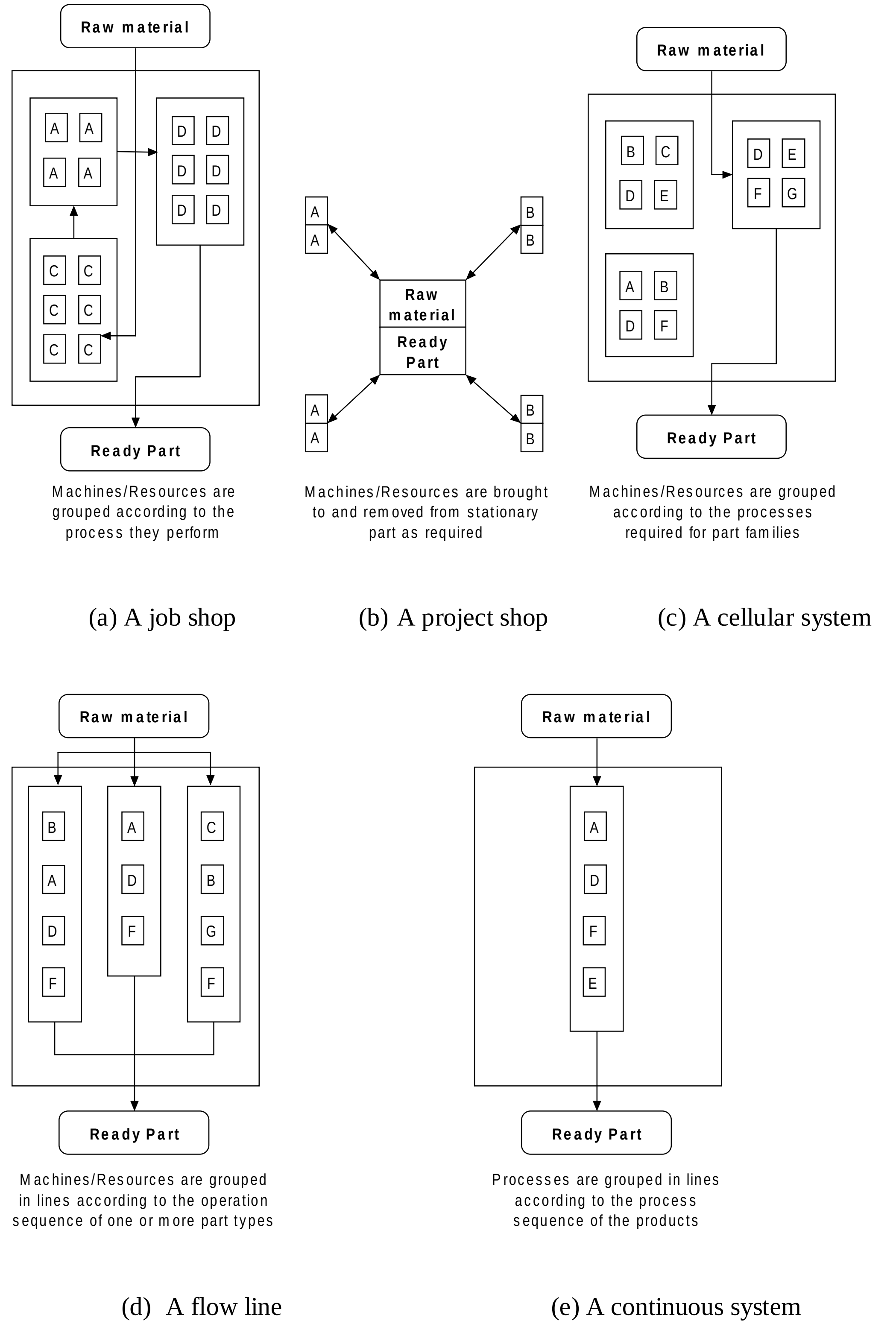
1 Introduction

A background of the research in dynamic job shop scheduling is presented in this chapter. Section 1.1 classifies manufacturing environments and gives the roles of scheduling in manufacturing production management. Section 1.2 presents the notions, definition, representation, roles, and the classification of classic scheduling problems. The classification of schedules and the complexity of classical job shop scheduling problems are also described. Section 1.3 introduces dynamic scheduling problems and discusses the main approaches to solve them in the fields of industry and academic research. Section 1.4 gives the motivations for this research and section 1.5 identifies the research goals and the methodologies. Finally, section 1.6 elaborates the outline for the remaining parts of the thesis.

1.1 Manufacturing environments

1.1.1 Classification

Manufacturing environments can be classified into five types: job shop, project shop, cellular system, flow line and continuous systems (Chryssolouris, 2006) (Fig. 1.1). In a ***job shop*** (Fig. 1.1, (a)), machines with the same or similar material processing capabilities are grouped together in workcenters. A part moves through the system by visiting the different workcenters according to the part's process plan. In a ***project shop*** (Fig. 1.1, (b)), a product's position remains fixed during manufacturing because of its size and/or weight and materials are brought to the product as needed.



Machines/Resources are grouped according to the process they perform

Machines/Resources are brought to and removed from stationary part as required

Machines/Resources are grouped according to the processes required for part families

(a) A job shop

(b) A project shop

(c) A cellular system

Machines/Resources are grouped in lines according to the operation sequence of one or more part types

Processes are grouped in lines according to the process sequence of the products

(d) A flow line

(e) A continuous system

Fig. 1.1 Schematics of five types of manufacturing systems (Chryssolouris, 2006)

In a ***cellular system*** (Fig. 1.1, (c)), the equipment or machinery is grouped according to the process combinations that occur in families of parts. Each cell contains machines that can produce a certain family of parts. In a ***flow line*** (Fig. 1.1, (d)), the machines are ordered according to the process sequences of the parts to be manufactured. Each line is typically dedicated to one type of parts. Finally, a ***continuous system*** (Fig. 1.1, (e)) produces liquids, gases, or powders in a continuous production mode.

One lot of jobs refers to a batch of jobs which are simultaneously released to a manufacturing shop floor and the lot size directly affects inventory and scheduling. Generally, the lot sizes that can be processed by a discrete manufacturing system, which works on discrete pieces of products like metal parts, are related to the types of manufacturing systems. Normally, job shops and project shops are most suitable for small lot size production, flow lines are most suitable for large lot size production, and cellular systems are most suitable for production of lots of intermediate size. It can be seen from Fig. 1.2 that lot sizes in job shops range from 1 to 100 jobs.

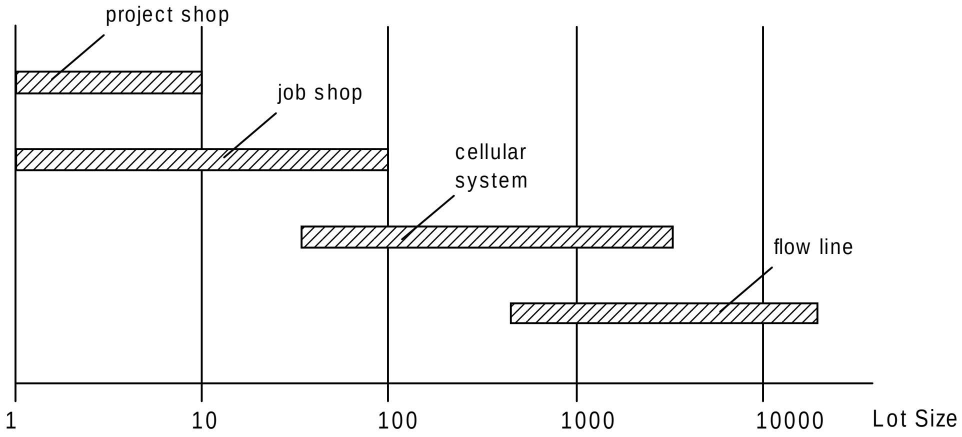


Fig. 1.2 Suitable manufacturing system types as a function of lot sizes (Chryssolouris, 2006)

1.1.2 Manufacturing production management

The production management and control activities in a manufacturing system can be classified as strategic, tactical and operational activities, depending on the long, medium or short term nature of their tasks (Hopp and Spearman, 2000; Chryssolouris, 2006).

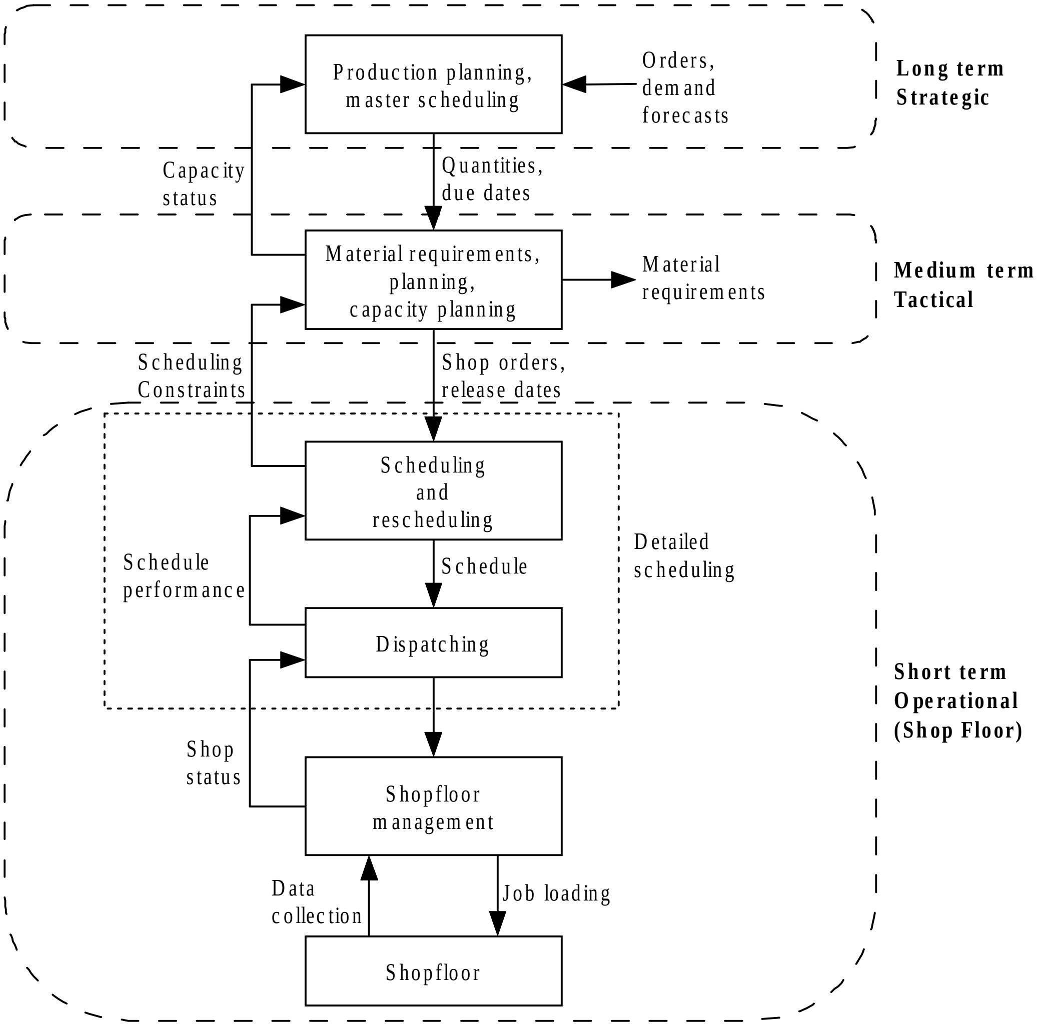


Fig. 1.3 The information flow diagram in a manufacturing system (Pinedo, 2002)

The information flow diagram in a manufacturing system modified from Pinedo (2002) is given in Fig.1.3 to illustrate the relationship of those activities at different levels. The ***strategic*** production management decides issues related to the determination of products according to the market demands or forecasts, the design of the manufacturing systems to produce those products, the generation of master schedule to meet the capacity requirement, etc. The ***tactical*** production management decides issues relating to the generation of detailed plans according to the master schedule. The results of this stage, such as shop orders with release and due dates are passed to the lower control level, *i.e.*, the ***operational*** production management, which decides the processing of those orders on the shop floor in order to fulfill the order requirements, and at the same time, optimizes the performance of the manufacturing system. It needs proper scheduling strategies to meet those requirements. After scheduling, the schedule is transferred to the shop floor and the implementation of a schedule is often referred to as ***dispatching*** (Vollmann *et al*, 1992).

1.2 Classical scheduling problems

1.2.1 Notions

Important notions adopted in the current thesis are defined as follows.

An ***operation*** ($\boldsymbol{O_{ij}}$) refers to the j^{th} elementary task of job i to be performed on a machine.

A ***job*** ($\boldsymbol{J_i}$) refers to the i^{th} job which has a set of operations that are interrelated by precedence constraints derived from technological restrictions.

The ***processing time*** ($\boldsymbol{p_{ij}}$) of an operation is the amount of time required to process operation O_{ij}.

The ***setup time*** refers to the time required by a machine to shift from the current status to the next one in order to process the next operation. In the current studies, setup times are independent of operation sequence and are included in the processing time.

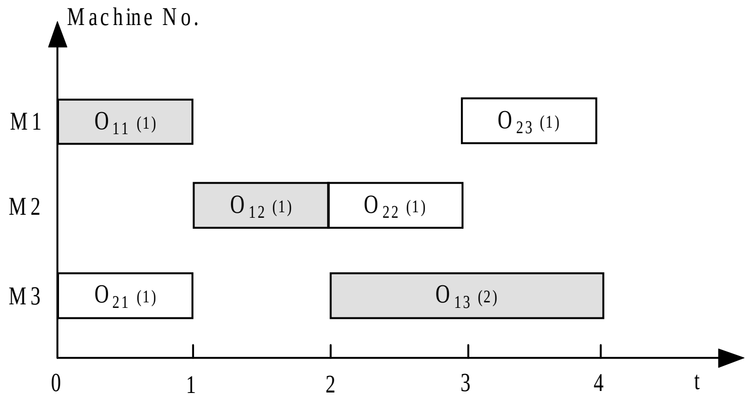
A ***machine*** (***M***) is a piece of equipment, a device, or a facility capable of performing an operation.

The ***due date*** ($\boldsymbol{d_i}$) of job *i* is the time by which the last operation of the job should be completed.

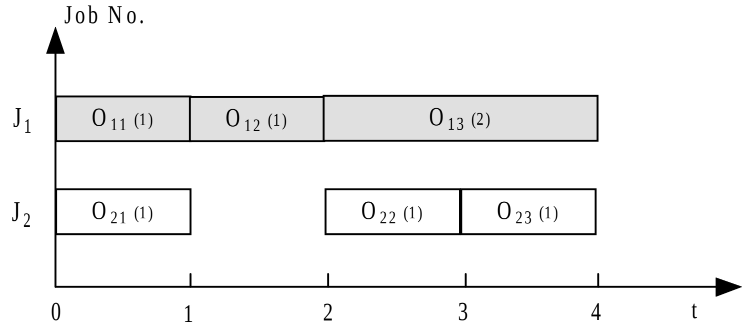
The ***completion time*** ($\boldsymbol{C_i}$) of job *i* is the time at which processing of the last operation of the job is completed.

1.2.2 Definition, representation, and roles

Scheduling deals with the allocation of scarce resources to tasks over time. It is a decision-making process with the goal of optimizing one or more objectives (Pinedo, 2002). The result of a scheduling procedure generates one or several ***schedules***, which are defined as plans with reference to the sequence of and time allocated for each item or operation necessary to complete the item (Vollmann *et al*, 1992). A schedule can be represented as a ***Gantt Chart***, which is a two-dimensional chart showing time along the horizontal axis and the resources along the vertical axis. Each rectangle on the chart represents an operation of a job, which is allocated to certain time slots on that resource. A Gantt Chart can be machine-oriented or job-oriented and examples for both types are presented in Fig. 1.4, where jobs J_1 and J_2 are scheduled. O_{11}, O_{12}, and O_{13} are three operations of J_1 and O_{21}, O_{22}, and O_{23} are operations of J_2. The processing time of each operation is included in parentheses.



(a) Machine-oriented Gantt Chart



(b) Job-oriented Gantt Chart

Fig. 1.4 Examples of machine- and job-oriented Gantt Chart

The main goal of manufacturing production management is to meet demands in a timely and cost-effective manner. In most manufacturing environments, especially in those with a wide variety of products, processes, and production levels, the construction of advance schedules is recognized as central to achieving this goal. Scheduling in manufacturing systems is very important for its roles in maximizing throughput and resource utilization, meeting due dates of orders, reducing inventory levels and cycle time, etc. Even small improvements in those measures can lead to considerable profit and thus increase the competitiveness of a factory.

Furthermore, a production schedule can enable the anticipation of potential performance obstacles and provide opportunities to minimize their harmful effects on the overall system behavior; it can enable better coordination to increase productivity and minimize operating costs; it can identify resource conflicts, control the release of jobs to the shop floor, and ensure that the required raw materials are ordered in time. A production schedule can also determine whether delivery promises can be met and identify time periods available for preventive maintenance; it gives shop floor personnel an explicit statement of what should be done so that supervisors and managers can measure their performance (Vieira *et al*, 2003). All these contribute to decreasing the cost of production and increasing profits for a factory.

1.2.3 Classification of scheduling problems

A scheduling problem can be described based on the *n/m/A/B* classification scheme of Graham *et al* (1979). ***n*** is the number of jobs; ***m*** is the number of machines; the ***A*** field describes the machine environment. The ***B*** field describes the objective to be optimized and usually contains a single entry.

1.2.3.1 Machine environments

The possible machine environments are single machine, flow shop, job shop, etc. The current studies focus mainly on job shop with the definition as follows.

In a ***job shop*** (***G***), there are *m* machines , M_1, … M_m, which are different from each other, and a set of *n* jobs J_1, … J_n, which are to be processed on those machines subject to the sequence constraints of their operations. Job J_i ($1 \le i \le n$) consists of m_i operations O_{i1}, … O_{im_i} ($0 \le m_i \le m$) and their respective number of machines can be given in a vector v_i, where $v_i(k)$ ($0 \le k \le m_i$, $1 \le v_i(k) \le m$) is the number of the machine that processes operation O_{ik}. The processing times of those operations are $p_{i1}, \ldots\ p_{im_i}$. A schedule has to be found so that all jobs are routed in the shop floor in a manner that the performance measures of the system can be optimized. The schedule decides the starting time t_{ik} for each operation O_{ik} of job J_i and the following formula holds:

$$t_{ik} = \max\left(t_{i,k-1} + p_{i,k-1}, t_{hl} + p_{hl}\right).$$

(1.1)

t_{hl} is the starting time of job J_h, which is the job processed on the same machine immediately before job J_i. t_{ik} is decided by either the completion time of its direct preceding operation or the earliest available time of its machine.

1.2.3.2 Objectives

The objectives to be optimized are always a function of the completion times of the jobs. The objective criteria considered in this study include makespan, mean flowtime, and mean tardiness, which are most commonly used in the literature of job shop scheduling. Performance measures related to inventory status like throughput, work-in-process and the size of jobs in a queue are also considered.

Makespan (C_{max}) is the “length” of the schedule, or an interval between the time at which the schedule begins and the time at which the schedule ends. Thus, the makespan of a schedule equals to $\max[C_i]$, where i = 1, …, m.

Flowtime (F_i) (also called cycle time) is the amount of time job J_i spends in the shop floor. It corresponds to the time interval between the release time r_i and the completion time C_i of job J_i: $F_i = C_i - r_i$.

Mean Flowtime **($\overline{F}$)** is the average flowtime of the schedule. $\overline{F} = \frac{1}{n}\sum_{i=1}^{n} F_i$, where *n* is the number of jobs.

Tardiness *(T_i)* of a job J_i is the non-negative amount of time by which the completion time exceeds the due date d_i: $T_i = \max[0,(C_i - d_i)]$.

Mean Tardiness **($\overline{T}$)** is the average tardiness of all jobs in the schedule: $\overline{T} = \frac{1}{n}\sum_{i=1}^{n} T_i$, where *n* is the number of jobs.

Throughput (TP) is the average output of a production process (machine, workcenter, plant) per unit time (e.g., parts per hour).

Work-In-Process **(*WIP*)** includes all unfinished parts or products that have been released to a production line; it represents the inventory in the shop floor and is preferred to be low so that less possibility of congestion in the shop floor is expected and less extra capital is expensed in inventory. However, the production rate cannot be guaranteed if WIP is too low according to ***Little's Law***, which is described as follows: at every WIP level, WIP is equal to the product of throughput and cycle time (Hopp and Spearman, 2000).

Size of jobs in a queue refers to the number of jobs waiting in the queue of a resource (machine) or a workcenter.

The above performance measures can be put into four categories: utilization-based objectives, flow-based objectives, due-date-based objectives, and inventory-based objectives. Makespan corresponds to the ***utilization-based objective***, which is related to the resource utilization. A schedule with a shorter makespan implies higher resource utilization. mean flowtime, throughput, and WIP are ***flow-based objectives***, which measure the turnaround times of the jobs in the shop floor; ***tardiness related objectives*** measure the ability to meet due dates; finally, the size of jobs in a queue and *WIP* are ***inventory-based objectives*** which measure the inventory status of the shop floor.

Given a measure of performance Z, which is defined as a function of the set of job completion times, and $Z = f\left(C_1, C_2, ...C_n\right)$, Z is ***regular*** if: 1) the scheduling objective is to minimize Z, and 2) Z can increase only if at least one of the completion times in the schedule increases (Baker, 1974). Makespan is a regular performance measure while mean tardiness-related objectives are ***non-regular***.

Thus, a scheduling problem given as $n/m/G/\overline{T}$ refers to a job shop scheduling problem (JSSP) with *n* jobs, *m* machines; and the objective is to minimize the mean tardiness. $n/m/G/\overline{F}$ refers to a JSSP with *m* workcenters and the objective is to minimize the mean flowtime.

1.2.4 Classes of schedules

In scheduling theory, schedules from optimizing regular measures of performance can be categorized into three types, semi-active, active and non-delay. A feasible schedule is called ***semi-active*** if no operation can be completed earlier without changing the order of processing on any one of the machines; it implies that there is no unnecessary idle time inserted before the starting time of a job. A semi-active schedule is called ***active*** if there is at least one operation which can be started earlier without delaying any other operation. It is sufficient to consider only active schedules in order to find an optimum. An active schedule is called a ***non-delay*** schedule if no machine is kept idle at the time when it can begin processing some operations.

The set of non-delay schedules is the subset of the set of active schedules for the same scheduling problem but the optimal schedule could be found in either sets. Fig. 1.5 shows a Venn diagram of the relationships among the three classes of schedules (Pinedo, 2002). Generally, the best non-delay schedule can usually be expected to provide a very good solution, if not an optimum (Baker, 1974).

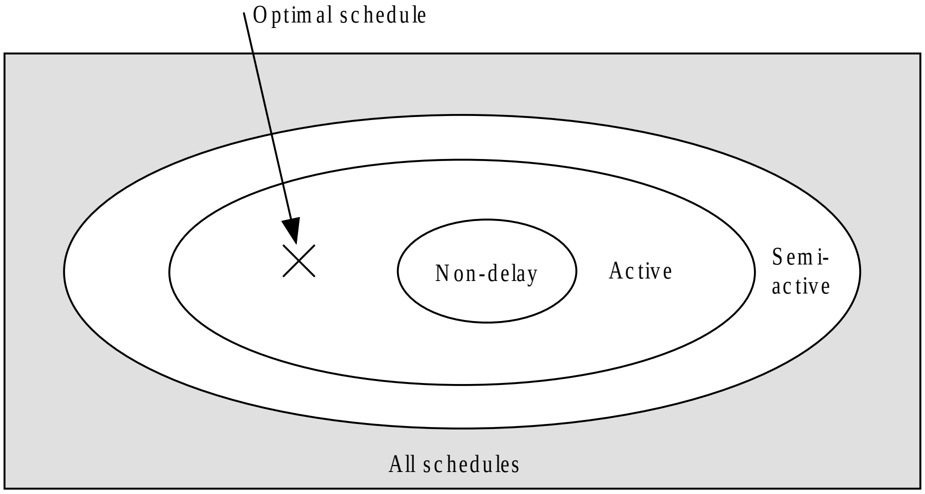


Fig. 1.5 Venn diagram of classes of schedules

1.2.5 Complexity of classical job shop scheduling problems

The inherent complexity of a classical JSSP arises mainly from the large size of its possible solutions as well as its objective functions. Both of them are decided by the medium- to long-term strategies of a manufacturing management system (Fig. 1.3).

The solution space including the optimum or a near-optimum solution is directly decided by the number of machines *m* and jobs *n* in the problem. It could be comprised of $(n!)^m$ schedules assuming that each job has one operation on each type of machine. Research has been focused on finding efficient algorithms for optimal solutions in a computational time that grows polynomially as the size of jobs increases. However, there are no such algorithms for most scheduling problems and these scheduling problems are thus called NP-hard problems (Garey and Johnson, 1979; Blazewicz *et al*., 1996). This fact also implies that it is impossible to find optimal solutions for most realistically sized scheduling problems in reasonable times. Hopp and Spearman (2000, pp.493-497) illustrated the complexity of a scheduling problem caused by the size of possible solutions and also concluded that there was little help by improving the speed of the computer. Thus the "optimal solution" mentioned in this thesis would mean a reasonably good solution unless it is otherwise indicated.

Given the same scheduling problems, the time complexities to optimize different performance measures may be different. For example, optimal solutions can be found in a polynomial time of $O(n \log n)$ with Johnson's algorithm (Johnson, 1954) to minimize the makespan of a two-machine flow shop problem while the time complexities to optimize other objectives for the same problem are considered NP-hard.

1.3 Dynamic scheduling problems

Scheduling in the real world is dynamic and stochastic in nature. A scheduling problem is ***dynamic*** if there are continuous arrivals of new jobs and ***stochastic*** if uncertain events like machine breakdowns or variant processing times are considered.

Those events are introduced into the system due to two factors. Quantities may either have inherent variability or they cannot be measured exactly (Ovacik and Uzsoy, 1994, 1997). The main consequence of those uncertainties for a scheduling system is that a predetermined schedule can become obsolete immediately.

In dynamic/stochastic manufacturing environments, managers, production planners, and supervisors must not only generate high-quality schedules but also react promptly to unexpected events in order to revise schedules in a cost-effective manner. In an attempt to construct an effective reactive scheduling system, various approaches have been proposed and they can be categorized as industrial and academic studies.

1.3.1 Main approaches in industry

Industry often uses simple but robust tools to guide production, like interactive schedulers, human involvement and self-developed software, often in combination with a Material Requirements Planning (MRP) system, which is one of the earliest applications of computers for medium- to long-term material and resource capacity planning for the entire production cycle.

However, the simplistic model of MRP undermines its effectiveness because: 1) it assumes infinite capacity; 2) it uses one lead time for offsetting, which results in earlier release, larger queues, and hence longer cycle times; and 3) the small change in its master production schedule may result in a large change in planned order releases, which is called ***system nervousness*** (Hopp and Spearman, 2000).

The problems in MRP prompted some scheduling researchers and practitioners to turn to enhancements in the form of Manufacturing Resource Planning (MRP II) and more recently, Enterprise Resource Planning (ERP). However, the fundamental problems of assuming infinite capacity and fixed lead times are still with the basic models underlying those improved systems. Some just rejected MRP altogether in favor of Just-In-Time (JIT).

JIT, which originated in mid-1950s, is a method to avoid scheduling by changing the production environment where the production is driven by the need of downstream workstations. This type of production system is also called the *pull system*. JIT demonstrates very good performance in automobile industries in Japan by removing idle intermediate WIP jobs. However, this approach assumes steady demand and is most suitable for a flow shop pull system. It may not equally benefit dynamic job shops where demands are variable.

Finally, dispatching rules are widely adopted in practice and they are also well studied in literature. Their detailed description will be given in Chapter 2.

1.3.2 Main approaches reported in open literature

In open literature, there are basically two approaches to accommodate those dynamic events: proactive and reactive scheduling. In ***proactive scheduling***, the events are considered predictable and some slacks are reserved in the original schedule so that disturbances can be absorbed without re-scheduling. In **r*eactive scheduling***, actions have to be taken to revise or repair a complete schedule that has been “overtaken” by events on the shop floor (Zweben *et al*, 1994). The latter approach is the main focus of this study.

Three main ideas underlie the enormous number of approaches under the umbrella of reactive scheduling and they are: queuing theory, predictive-reactive scheduling, and artificial intelligence. Early research has used the queuing theory to explore the collective effect of several types of dynamics on a shop floor using simple rules to decide the orders of jobs. Later, researchers proposed to use schedules generated by more advanced scheduling techniques in order to improve overall production performance. Finally, the development in the field of artificial intelligence, especially multi-agent systems (MAS), has been inspiring its applications in dynamic scheduling.

1.3.2.1 Queuing theory

Queuing theory is inspired by the real-world applications where jobs are assumed to arrive in a random process in some statistical forms; the processing times of operations and dynamic events are random variables with known distributions. Jobs are queued in the buffer of their waited machine until it is free. A job is selected from the buffer to be processed according to some predetermined priority rules or dispatching rules. Jobs are discharged from the system if all of its operations are completed. The randomness in the arriving jobs, processing times, and stochastic events like machine breakdowns together implies the distributions of job flow times and machine busy/idle times. Different dispatching rules may be compared and the best ones can be chosen for production.

The advantage of using the queuing theory is that a system reacts to events and makes allocation decision one at a time only if necessary for keeping execution going based on the current status of the system. This strategy is insensitive to unexpected events and thus yields quite robust behaviour. Furthermore, it is highly effective computationally. However, the performance of factory operations may be sacrificed since there is no attempt for optimization.

1.3.2.2 Predictive-reactive scheduling

In the *predictive-reactive scheduling* approach, a schedule is generated for a set of jobs in order to optimize certain criteria before those jobs are actually executed and the schedule is refined when dynamic events occur. It is a common strategy to reschedule dynamic manufacturing systems (Jain and ElMaraghy, 1997, Mehta and Uzsoy, 1998).

There are two parts for the actions in this approach: namely generating predictive schedules and reacting to disturbances. The generation of predictive schedules may use the methods from the field of classic scheduling and the reaction to disturbances implies decisions about *what*, *when,* and *how* to react (Sabuncuoglu and Bayiz, 2000) in order to optimize system performance in the face of dynamic events (Church and Uzsoy, 1992; Abumaizar, and Svestka, 1997). Different scheduling generation and refining procedures may be explored and compared in order to find the best one for a particular problem.

Generally, the *predictive-reactive scheduling* approach requires more computational efforts to generate optimal or sub-optimal solutions as compared to dispatching rules in the queuing theory. It is also different from queuing in that queuing decides only the order of tasks while scheduling also decides their starting times.

1.3.2.3 Multi-agent systems

Parunak (1997) defined an intelligent ***agent*** as “an active object with initiative” and views it as a software design paradigm, which is the next extended step to object-oriented programming in software evolution. An agent has at least two important capabilities. First, it is capable of autonomous/pro-active action to decide its actions in order to realize its objectives. Second, it is capable of interacting with other agents through exchanging data or through cooperating, coordinating or negotiating with other agents (Wooldridge, 2001).

An ***MAS*** is a loosely-coupled network of agents that work together in a group to solve a common problem (Pendharkar, 1999). As a distributed problem-solving paradigm, an MAS can transform a complex scheduling problem into smaller and manageable sub-problems to be solved by individual agents co-operatively. Like in the queuing theory, no schedules are calculated in advance but the core is to find appropriate protocols and architectures for agents to interact and share information dynamically. The overall performance emerges as the result of the interactions among agents using certain co-operation protocols.

1.4 Motivations

The essential motivation of the current study is to develop a scheduling system that can keep on optimizing the performance of a job shop manufacturing system in real time in the face of dynamic events.

The idea is first inspired by the advancement in the field of MAS. Durfee (1988) and Durfee and Lesser (1989) proposed a ***heterarchical MAS*** where independent agents interact with each other using only local information and a global optimization can emerge from those local interactions. The emphasis of this approach is to find appropriate interaction rules or coordination protocols for agents and model problem components into appropriate agents. However, this approach has the disadvantages of unsatisfactory optimality, unpredictability, and high communication overhead.

In order to improve optimality and predictability as well as to reduce communication overhead, researchers have developed hierarchical MAS and furthermore, hybrid

MAS for dynamic control and scheduling. In a pure ***hierarchical MAS***, agents at higher levels can allocate tasks to their immediate lower level agents, which execute their assigned tasks without any opinion. The system will produce schedules with good global performance since the agent at the higher level can have a wider view of the system. However, this architecture lacks reactivity to dynamic events since events are first forwarded from the lowest level agents to the upper level agents and then the reaction decision is passed down from the upper level agents to the lowest level agents to be executed. This type of MAS may assume the schedules to guide production in a similar manner performed in predictive scheduling. To cope with the disadvantages and combine the advantages of the previous two types of MAS, some hybrid architectures have been proposed. Basically, agents in a ***hybrid MAS*** have the autonomy to promptly react to dynamic changes and simultaneously be guided by those agents with global views.

Recent research on the foraging behaviour of a natural MAS, namely an ant colony, has found that autonomous agents like ants can find the shortest route from their nest to a food source based on the pheromone strength on their ways. Each ant affects the environment by leaving behind itself some amount of pheromone. This type of optimization mechanism is a collective effect of the interactions between the ants and the pheromone environment. Furthermore, it is also found that an alternative shortest path can soon be formed by foraging ants if the current one is not available. Both features are of great research interests in the view of their applications in dynamic/stochastic scheduling environments.

In order to realize this mechanism for the optimization purpose in scheduling problems, two implementations had been proposed. One is the pure MAS approach;

the other is through the ant colony optimization (ACO) algorithm. The former involves not only the indirect correspondence between a modeled agent and a facility in the real world problem, but also a great number of communications among agents. Thus, the ACO approach is adopted in the current study.

Meanwhile, the previous applications of ACO on JSSPs have been mainly focused on static cases and its performance on dynamic JSSPs has not been systematically studied. The current research aims to explore the effectiveness of ACO in dynamic JSSPs, the factors affecting its performance, the effects of the adaptation mechanism, and its application domains based on the research findings in the areas of the queuing theory, ACO algorithms, and MAS. As dynamic JSSPs continues to be a challenge (Smith, 2003; Stoop and Wiers, 1996), the research of exploring an advanced scheduling system is considered valuable.

1.5 Research goals and methodologies

1.5.1 Goals

In summary, the goals of the current study include:

- To analyze a dynamic JSSP, identify the systematic manners of research in this field, and define the domains of the dynamic JSSP.

- To build a generic test bed that can provide problem scenarios for systematically evaluating a proposed scheduling approach.

- To present the effectiveness of ACO in solving dynamic JSSPs, and demonstrate the effectiveness of its adaptation mechanism.

- To improve its performance through adjusting its parameters.

- To find the best application domains of ACO in dynamic JSSPs.

1.5.2 Methodologies

Normally, the evaluation of an approach for a static scheduling problem includes the comparisons in two aspects: schedule quality and computational time. Schedule quality is evaluated in terms of target performance measures like makespan, total/mean flowtime, total/mean tardiness, *etc*. Computational time refers to the time spent by computers to find the schedule and can be measured through the analysis of the ***computational complexity***, which describes how the computational time and memory requirements of the algorithm change as the size of the input to an algorithm increases (Garey and Johnson, 1979). A good scheduling approach performs well in either providing high quality schedules or obtaining acceptable schedules within limited computational times.

However, the evaluation of approaches for dynamic scheduling problems is different. Jobs continue to arrive during the entire period of the evaluation while the proposed dynamic scheduling procedure continues to working simultaneously during the same period. Occasional good schedules do not guarantee a long-term good performance of a proposed approach. Thus, it is important to decide a reasonable test period in order to obtain a fair evaluation of the proposed approach. The approach in the queuing theory is to execute a simulation until the system reaches a steady state and the performance data are recorded from that point. Next the simulation continues for a certain period of time and an average steady-state performance of the approach can be obtained.

A similar approach is adopted in the current study and all experiments were carried out on a simulated test bed as the experiments on real factories are generally expensive and sometimes impossible. First, a discrete job shop manufacturing system is simulated using discrete event simulation (DES) in order to provide adequate scenarios. Several replications of the experiments for the same problem configurations were tested with only variations in the generation of initial random numbers. DES can facilitate the examination of a long-term average performance of the tested approach since the time intervals that do not change the system state are skipped. The experimental results are analyzed or compared statistically.

Furthermore, the DES will be implemented as an MAS based on the following two considerations. On the one hand, the optimization mechanism of foraging ants can be implemented in different types of MAS, which will be described in Chapter 5. On the other hand, the MAS implementation of a job shop has many advantages, which will be mentioned in Chapter 4. Briefly, the test bed not only can properly model a shop floor as a distributed system but also provide a long-term performance evaluation for a proposed approach.

In order to build the above simulated job shop, commercial simulation tools like ARENA have been considered at first. However, the effort of interfacing them with the ACO scheduler would be about the same effort as building a new tool. In particular, there should be communications between shop floor entities like workcenters, jobs and the scheduler in order that it resembles the similar structure and the logic in a real job shop. Thus, a test bed is thereafter built from scratch based on Java Agent DEvelopment Framework (JADE), which is a software framework fully implemented in Java language.

After the test bed is built, the ACO algorithm implemented as an MAS is used to generate schedules for dynamic JSSPs, which are systematically designed to achieve the goals set in section 1.5.1. The predictive-reactive scheduling procedure is used in all experiments.

1.6 Outline of the thesis

Chapter 2 first reviews the approaches for solving static JSSPs in order to pave the way of reviewing the approaches for dynamic JSSPs, which is followed immediately by focusing on predictive-reactive scheduling. Next, MAS approaches and the applications of ACO in the scheduling related fields are also reviewed in detail to give a background of the current research.

Chapter 3 first analyzes the static JSSPs, then the dynamic JSSPs. Finally, the factors affecting the evaluation of a scheduling technique in a predictive-reactive approach are analyzed.

Chapter 4 builds a common test bed to facilitate a systematic examination of the performance of control policies and algorithms in a dynamic job shop environment. The definition of a generic job shop is first given, and a generic job shop is modeled as a DES. A prototype of the job shop is implemented as an MAS. The communication of agents in the MAS is presented and a case study is described.

Chapter 5 extends the test bed to include a scheduler, which uses ACO as an optimizer simulating the scheduling function in a factory. It discusses the additional coordination of the scheduler agent to the main existing agents like job, job shop and workcenters agents and among the behaviours within the scheduler agent itself. The procedure to dynamically update the pheromone matrix of ACO is also discussed. Finally, the implementation of ACO as an MAS is presented.

Chapter 6 tests the performance of ACO on two dynamic JSSPs with the same mean load but different dynamic frequency and severity. The effectiveness of its adaptation mechanism is studied. Furthermore, two important parameters in the ACO algorithm, namely the number of iterations and the size of searching ants per iteration, which control the computing time and the quality of solutions, are also examined.

Chapter 7 first defines the three dimensions describing the ***domain*** of dynamic JSSPs: namely the frequency of the arriving jobs, the variation of the processing times, and performance measures. Two series of experiments are next carried out to find the appropriate application domains of ACO in terms of the ranges of job arriving levels and the variation of the processing times. The performances of the experiments are compared and the proper ranges that ACO outperforms the best dispatching rule are identified. In this manner, the domains that ACO can be effectively applied can be identified.

Chapter 8 concludes the work, highlights the contributions, and identifies a number of future works.

2 Literature Review

Scheduling as a research discipline dated back to early 1900s but serious analysis of scheduling problems did not begin until the advent of computer age in the 1950s and 1960s. Since then, a great amount of theoretical work has been reported. A good historical overview of the different approaches was given by Froeschl (1993) and an early introductory work on scheduling was reported by Baker (1974), French (1982), Buxey (1989), and Sule (1997). Literature reviews on static deterministic scheduling can be found in (Graves, 1981, Jain and Meeran, 1998, 1999, MacCarthy and Liu, 1993, Blazewicz *et al*, 1996, Sellers, 1996, Weirs, 1997, Jones and Rabelo, 1998, and Pinedo, 2002). Nowicki and Smutnicki (1995) provided an excellent review of minimum makespan job shop problems. Suresh and Chaudhari (1993) reviewed the dynamic scheduling literature.

This review starts with the approaches for static JSSPs; then the emphasis is put on the approaches for handling dynamic environments. Furthermore, the applications of ACO in the scheduling related fields are reviewed in detail to give a background of the current research.

2.1 Approaches for the classical job shop scheduling problems

2.1.1 An overview

The main approaches to solve the classical JSSPs include exact mathematical algorithms, dispatching rules, metaheuristics, and artificial intelligence methods. These approaches and some of their examples are listed in Fig. 2.1.

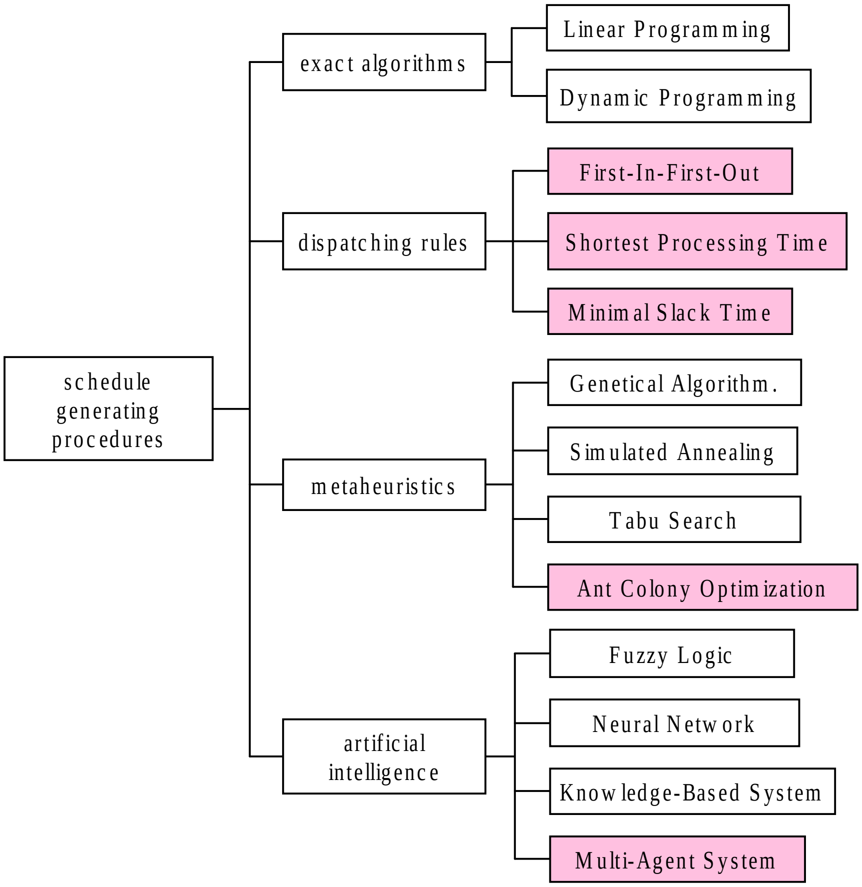


Fig. 2.1 Approaches to solve classic job shop scheduling problems

2.1.2 Exact mathematical algorithms

Balas (1965, 1967) developed modern integer programming, which allows rather realistic JSSPs to be formulated in a manner that would theoretically permit them to be solved exactly. Two popular solution techniques for integer-programming problems are branch-and-bound and Lagrangian relaxation. Branch-and-bound is an enumerative technique, which systematically curtails undesired solutions by dynamically setting lower bounds through modeling the JSSP as a decision tree. Lagrangian relaxation solves integer-programming problems by omitting specific integer-valued constraints and adding the corresponding costs to the objective function.

Another exact mathematical algorithm reported is *dynamic programming*, which enumerates in an intelligent manner all the possible solutions. During the enumeration process, schedules which are not optimal are eliminated.

However, both integer programming and dynamic programming are computationally intensive. Thus large problems remain intractable although very small problems can be solved with optimal solutions. Subsequently, the majority of scheduling problems has to be solved using *heuristics*, which are techniques seeking good solutions instead of the optimal ones at a reasonable computational cost (Voß, 2001). Main heuristic approaches include dispatching rules, metaheuristics, and artificial intelligence.

2.1.3 Dispatching rules

The simplest heuristic to find a solution is using dispatching rules, where a schedule is constructed in one iteration with generally a very light computational effort even for a large problem. A ***dispatching rule*** is used to prioritize jobs waiting for processing at the time that the waited machine/resource becomes available. The job with the highest priority is selected to be processed on the machine. An early survey can be found in Panwalkar and Iskander (1977).

Common dispatching rules employ processing times and due dates as deciding factors in simple rules or their complex combinations. Some dispatching rules are extensions of policies that work well on simple machine scheduling problems, for example, First-In-First-Out (FIFO), Shortest Processing Time (SPT), Minimal Slack Time (MST), and Earliest Due Date (EDD). In FIFO, the first operation coming into a workcenter has the highest priority; in SPT, the operation with the shortest processing time has the highest priority; in MST, the operation with the shortest slack time has the highest priority. The slack time indicates the temporal difference between the due time, the current time and the remaining computation time. In EDD, the operation with the earliest due date has the highest priority.

Other dispatching rules can be found in Panwalkar and Iskander (1977), which provides an extensive list of dispatching rules and their classification includes five categories: simple dispatching rules, combinations of simple rules, weighted priority indices, heuristic scheduling rules, and similar findings by others, such as Blackstone *et al* (1982), Ramasesh (1990), and Morton and Pentico (1993).

Owing to their inexpensive computational effort and robustness, dispatching rules are widely adopted, especially, in dynamic environments (Li *et al*, 1993). However, they do not guarantee the realization of the full potential of a shop floor as they do not aim at optimization. Scheduling systems using algorithms, especially metaheuristic algorithms, have continuously been studied to provide optimized solutions.

2.1.4 Metaheuristics

A ***Metaheuristic*** is a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of problems (Voß *et al*, 1999). It refers to an iterative master process that guides and modifies subordinate heuristics in order to efficiently produce high-quality solutions. There may be a complete (or incomplete) single solution or a collection of solutions per iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method. A ***local search algorithm*** is a metaheuristic iteratively moving from solution to solution in the space of candidate solutions (the search space) until a solution deemed optimal is found or a time bound has elapsed. A ***construction method*** generates a schedule by adding in an operation one at a time until all operations are considered.

Examples of metaheuristics include genetic algorithms (Goldberg, 1989), simulated annealing (Kirkpatrick, *et al*, 1983), Tabu search (Glover, 1989, 1990; Glover and Laguna, 1997), ACO (Dorigo and Di Caro, 1999), and their hybrids. Each has its own perturbation methods, stopping rules, and methods for avoiding local optimum. The use of metaheuristics has significantly increased the ability of finding very high-quality solutions to hard, practically relevant combinatorial optimization problems in a reasonable time (Dorigo and Stützle, 2004).

2.1.5 Artificial intelligence

The approaches to solve scheduling problems in the artificial intelligence field are based on the inspirations from either human society or natural phenomena (Weiss, 1999). Many sophisticated procedures have been proposed including fuzzy logic, neural network, knowledge-based systems and MAS (Kusiak, 2000).

Fuzzy set theory has been used to develop hybrid scheduling approaches. It can model and solve job shop scheduling problems with uncertain processing times, constraints, and set-up times, which are represented by fuzzy numbers. A neural network is trained with historical data and some desired relationships between the inputs and the outputs have been captured. The network can be used to estimate solutions for new inputs.

Knowledge-based scheduling systems employ domain specific problem solving information to derive schedules and this information knowledge is encoded as rules, which are often obtained by eliciting knowledge from experienced schedulers (Randhawa and McDowell, 1990). The work on constraint satisfaction problems is also of direct relevance to scheduling, if the latter is regarded as their incremental construction of a solution that satisfies the constraints in a problem space in which each additional assignment imposes a new set of constraints on the remainder of the solution (Sadeh, 1991). The most well-known systems include ISIS (Fox and Smith, 1984), OPIS (Smith *et al*, 1990), CABINS (Miyashita, 1995), and IOSS (Park *et al*, 1996).

A knowledge-based system does not aim to guarantee optimal solutions; instead, it just provides feasible good solutions (Randhawa and McDowell, 1990). Its performance is not beyond what has been provided by rules in the system. Furthermore, a great amount of domain-dependent heuristics is required and the most difficult operation is to decide which knowledge source has to be activated (Akturk and Gorgulau, 1999). Besides, scheduling decisions can only be evaluated locally.

MAS is a relatively new sub-field of computer science which was started around 1980 and has gained widespread recognition since the mid-1990s. It has been an active research topic in the manufacturing arena (Jennings and Wooldridge, 1998; Jennings *et al*, 1998; Parunak, 1994). Although an MAS can solve static scheduling problems, its more promising applications are in dynamic/stochastic ones. Therefore, its detailed description is specifically presented in section 2.3.

2.2 Approaches for dynamic job shop scheduling problems

Only two of the three approaches for dynamic JSSPs described in section 1.3 are reviewed based on their importance and relevance to the current work. They are predictive-reactive scheduling and MAS approaches. Reviews for the other approaches can be found as follows. The survey on priority-rules in dynamic job shop can be found in Haupt (1989); a detailed discussion of knowledge-based systems related to reactive scheduling can be found in Blazewics *et al* (1994) and Szelke and Kerr (1994). Conway *et al* (1967, Chapter 11) provided an excellent introduction to simulation in the context of the job shop. Parunak (1991) characterized the manufacturing scheduling problems.

2.2.1 Predictive-reactive scheduling

2.2.1.1 An overview

Predictive-reactive scheduling is an approach most commonly used in practice (Vieira *et al*, 2003). Basically, its study in manufacturing systems should consider the following factors: 1) the applied production systems identified by the types of manufacturing systems (flow shop, job shop, etc., or their extensions) and the types of dynamic events (dynamic incoming jobs, machine breakdowns, or processing variations) as well as their respective patterns of occurrences, 2) schedule generation/regeneration methods (algorithms, dispatching rules, or cooperation), 3) control rules (what, when and how to reschedule), 4) performance measuring criteria, 5) the testing period (short or long term), and 6) evaluation methods (comparison or statistical analysis). Those factors are illustrated in Fig. 2.2.

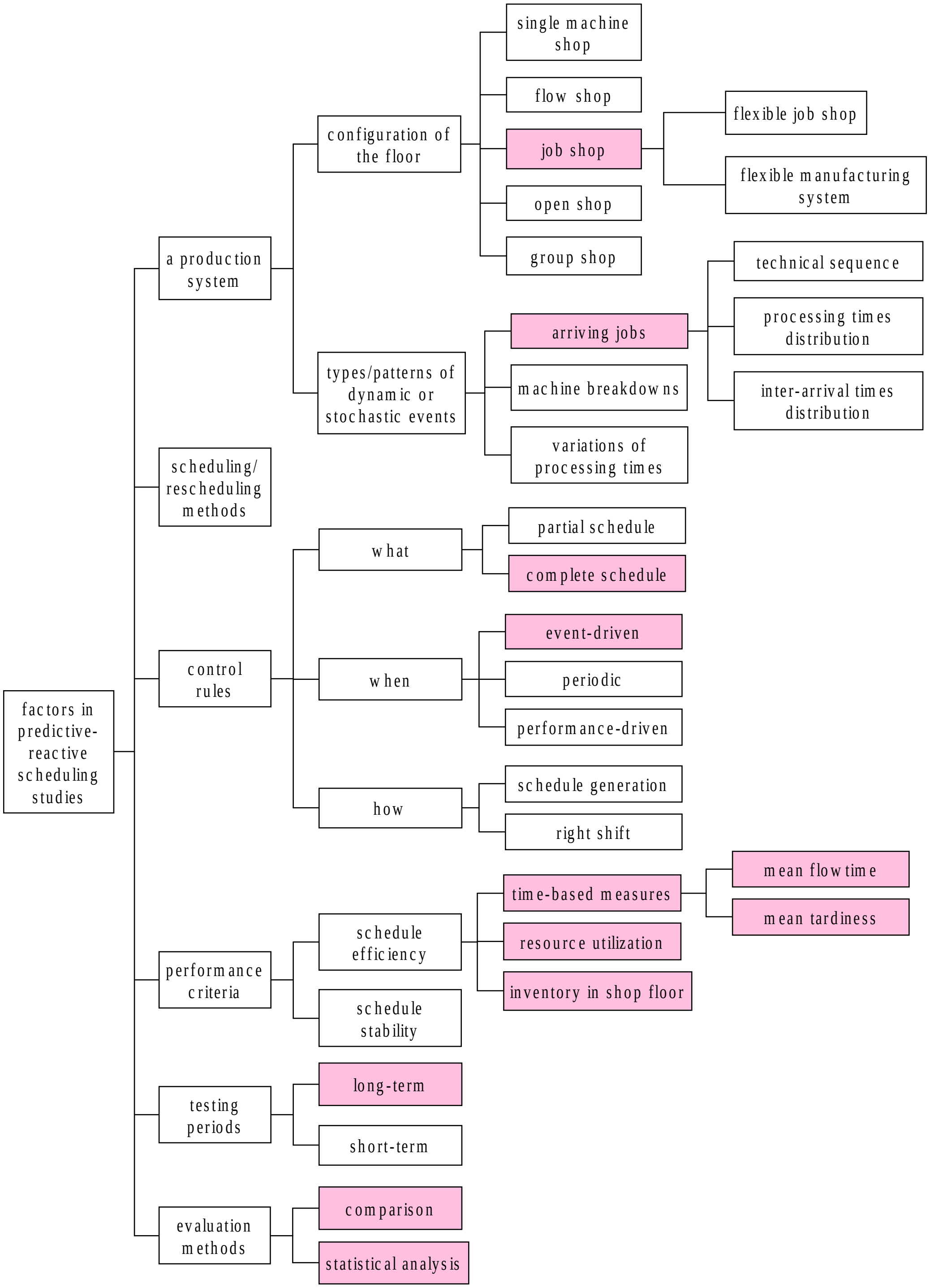


Fig. 2.2 Factors considered in the predictive-reactive scheduling research

A thorough study of a proposed scheduling procedure may include testing it on every different production system, control rules, performance criteria, and testing periods. That implies a huge number of experiments. In fact, researchers have done a lot of work studying dispatching rules in dynamic/stochastic scheduling environments due to their lower computational requirements. For scheduling procedures requiring similar computational efforts as those in predictive-reactive scheduling, it is important to identify the main domains that a proposed approach can perform well. In the following sections, selected works are reviewed focusing on the framework described in Fig. 2.2. Other reviews of dynamic scheduling can also be found in Smith (1995), Raheja and Subramaniam (2002), Vieira *et al* (2003) and Aytug *et al* (2005). A good survey of the simulation models for dynamic scheduling environments is provided by Ramasesh (1990).

2.2.2 Literature review

Holloway and Nelson (1974) proposed a multi-pass heuristic scheduling procedure to generate schedules in a job shop where processing time variations of the operations are considered. This centralized scheduling procedure is later used in the dynamic job shop environments (Nelson *et al*, 1977) to generate schedules periodically. They concluded that a periodic policy (scheduling/rescheduling periodically) is very effective.

Muhleman *et al* (1982) analyzed the periodic scheduling policy in a dynamic and stochastic job shop system and their experiments showed that a more frequent revision can improve scheduling performance. Church and Uzsoy (1992) studied the period and event driven policies in a dynamic one-machine system. They concluded that the performance of periodic scheduling was affected by the length of the rescheduling period while event-driven policy performs well in the given problem.

Bean *et al* (1991) proposed a match-up scheduling procedure to match up with the schedule, which was optimal or near optimal before the disturbance occurred. The match-up procedure ensures that the revised schedule is consistent with the original one after the “match-up point” as soon as possible. The procedure is applied to a set of real problems in the automotive manufacturing industry where a partial schedule is produced to minimize total tardiness at each rescheduling point. The results from the proposed match-up procedure are significantly better than those from pure static and dynamic strategies that are often used in practice. It also performs well when machine utilization is high. Later, Arturk and Gorgulu (1999) used match-up scheduling to react to disturbances. Their methods improve the schedule quality, the stability, and the computational time compared to several match-up alternatives under different experimental settings.

Raman and Talbot (1993) decomposed a dynamic problem into a series of static problems, which were then solved in their own entirety and then implemented on a rolling basis. A heuristic is used to construct the schedule for the entire system at each rescheduling moment. The experiments on dynamic scheduling problems are carried out with balanced and unbalanced machine utilizations. Their results indicate that a significant due date performance improvement over several dispatching rules is obtained.

Bierwirth *et al* (1995) explored the adaptive optimization ability of GA for reactive scheduling in dynamic job shops and their work was continued by Lin *et al* (1997). However, the size of their tested jobs was only 100, which is not enough to give a fair evaluation of the average performance of GA. Later, Bierwirth and Mattfeld (1999) again studied the similar problem by using two versions of improved GA to generate a new schedule every time a new job arrives reusing the previous solution. Furthermore, they tested on 1000 jobs instead of 100 and considered only the steady state performance, which was the performance between the times that jobs 201 and 800 arrived at the system. Both versions of GA outperformed SPT dispatching rule at reasonable computational times for the minimization of the mean flow-time of jobs.

Holthaus and Rajendran (1997) examined the performance of several dispatching rules in a dynamic job shop. They found their proposed dispatching rules efficient in minimizing flowtime and tardiness related criteria. They also described the simulated test bed and experimental designs in detail. These methods have been followed by Bierwirth and Mattfeld (1999). Holthaus (1999) further analyzed dispatching rules in dynamic job shop scheduling considering machine breakdowns. The results revealed that the relative performance of scheduling rules can be affected by changing the levels of the breakdown parameters.

Lawrence and Sewell (1997) compared the static and the dynamic applications of heuristic and optimal solution methods to JSSPs when processing times were uncertain and the performance measure was the makespan. They demonstrated that simple dispatch heuristics provide performance comparable or superior to that of algorithmically more sophisticated scheduling policies.

Sabuncuoglu and Bayiz (2000) proposed a heuristic algorithm basing on a filtered beam search to analyze reactive scheduling problems under different job shop environments considering machine breakdowns. They concluded that: 1) there was not much difference between the optimum methods and heuristics when uncertainty or variability was high, which was a conclusion also made by Lawrence and Sewell (1997), Hopp and Spearman (2000), Sabuncuoglu and Bayiz (2000), and Hall and Posner (2001); 2) the performance of the off-line algorithm was affected more than the on-line method in a stochastic environment; 3) the solution quality improved as the scheduling frequency increased; and 4) the quality of schedule deteriorated as the length of the partial schedule decreased. From these results, one could infer that the effort to reduce the variability and uncertainty in the systems might worth more than the difficulties in using more sophisticated algorithms (Sabuncuoglu and Bayiz, 2000).

Sabuncuoglu and Kizilisik (2003) studied reactive scheduling in a simulated Flexible Manufacturing System (FMS) considering a multi-machine environment and a material handling system (MHS) under variant system configurations, processing time variations, and machine breakdowns. Some of their conclusions were: 1) it would be more beneficial to use the online scheduling systems in dynamic and stochastic environments; and 2) full rescheduling was generally better than partial rescheduling at a cost of higher CPU times.

2.2.3 Main conclusions

In summary, some observations can be drawn from the research of the last thirty years. Firstly, there is not much difference between the optimum methods and heuristics when the uncertainty or variability is high (Lawrence and Sewell, 1997; Sabuncuoglu and Bayiz, 2000). Secondly, the performance of a scheduling procedure is affected by control policies like the frequency of scheduling and the length of the intermediate schedule. Thus the performance of a scheduling method is problem-dependent.

2.3 Multi agent systems

Many MAS scheduling systems have been proposed to generate schedules through the interactions of distributed and independent agents using certain protocols based on appropriate architecture. Manufacturing scheduling systems built as MAS had been surveyed by Shen and Norrie (1999) and Baker (1998). They are further reviewed according to their architecture: heterarchical, hierarchical, hybrid, and nature-inspired MAS.

2.3.1 Heterarchical MAS

A heterarchical MAS was built at a General Motors factory to assign trucks to paint booths using a simple bidding mechanism and each paint booth made decision whether it would take a job through negotiation (Morley and Schelberg, 1993, Morley, 1996). The MAS outperformed the previous centralized scheduling system in terms of throughput and paint costs. Liu (1996) proposed an MAS which sequentially initiated two groups of agents representing resources and jobs for distributed manufacturing scheduling and agents in the same group communicate based on several coordination schemes. The MAS was tested on several deterministic benchmark JSSPs and the results showed that it could provide equivalent or superior performance to centralized scheduling techniques.

Heterarchical MAS can provide a highly distributed structure to the manufacturing system and it is very robust and reactive against disturbances. However, banning all forms of hierarchy, it cannot perform global optimization and the behaviour of a system under heterarchical control can be hardly predicted. Furthermore, many heterarchical algorithms need to be properly fine-tuned, which is a labour intensive work (Bongaerts, 1998). Thus it is believed that in the unstructured environments, heterarchical control without explicit schedulers can be the most suitable approach. In other situations, however, the incorporation of a scheduler in a distributed system will enhance the stability, predictability and performance.

2.3.2 Hierarchical MAS

Parunak (1987) proposed YAMS for real time task allocation and control. A factory is modeled as a hierarchy of work cells and each work cell corresponds to a node in a contract net (Smith, 1980) and negotiates with others nodes vertically and laterally. Zhou *et al* (2004) used a hierarchical MAS to solve a deterministic scheduling problem using heuristic dispatching rules and Contract Net Protocol. Their results show that the MAS can generate good solutions for a given problem as compared to a mathematical approach. Cavalieri *et al* (2000) compared the performances of heterarchical and hierarchical MAS experimentally.

2.3.3 Hybrid MAS

Hybrid MAS includes holonic manufacturing system (HMS) (Bongaerts, 1998; Bongaerts *et al*, 2000; Wyns, 1999), biological manufacturing system (BMS) (Okino, 1993), and fractal manufacturing system (FrMS) (Warnecke, 1993; Ryu and Jung, 2003). Basically, agents in those systems have the autonomy to promptly react to dynamic changes and simultaneously to be guided by the agents with global views. Valckenaers *et al* (1994) compared the above three architectures and found that the hybrid one performed well in a wider range of situations. Wong *et al* (2006a, 2006b) proposed a hybrid MAS for integrating process planning with scheduling/rescheduling in job shops in cases of machine breakdown and new part arrival.

2.3.4 Nature-inspired MAS

Bonabeau *et al* (1999) gave a comprehensive survey of adaptive MASs, which were inspired by natural insect behaviors. Cicirello and Smith (2001) reviewed those MASs focusing on manufacturing applications. Valckenaers et *al* (2001) discussed multi-agent coordination and control using techniques inspired by the behavior of social insects. It presents a system design that enables desirable overall behavior to emerge without exposing the individual agents to the complexity and dynamics of the overall system. Cicirello and Smith (2004) proposed a new coordination rule inspired by the behaviour of a wasp colony for dynamic shop floor routing.

2.4 Ant colony optimization algorithm

2.4.1 ACO overview

ACO is a class of distributed algorithms used for solving NP-hard combinatorial optimization problems. Its introduction can be found in (Dorigo *et al*, 1996, 1999), (Dorigo and Gambardella, 1997a, 1997b), and (Dorigo and Di Caro, 1999).

The first form of ACO, Ant System (AS), was introduced by Dorigo *et al* (1991) and is based on the foraging behaviour observed in a real ant colony. The cooperation of ants and how they efficiently find the shortest routes have been formulated into an algorithm used to solve combinatorial optimization problems.

The first improvement of the initial AS is called the ***elitist strategy*** for AS (EAS) (Dorigo *et al*, 1996), where only the best-so-far solution is used to update the pheromone trails. The idea is to enhance the promising search space. Another improvement is called the ***rank-based*** AS (AS_{rank}), proposed by Bullnheimer *et al* (1999). The amount of pheromone that each ant deposits on the trails decreases according to its rank. Meanwhile, the best-so-far ant still deposits pheromone at each iteration. The results of an experimental evaluation suggest that AS_{rank} performs slightly better than EAS and significantly better than AS.

A ***MAX-MIN Ant System*** is another improvement proposed by Stützle and Hoos (1997, 2000). It limits the possible range of pheromone trail values to an interval [τ_{min}, τ_{max}] in order to avoid stagnation caused by exploring best-so-far solutions; all trails are initiated with the upper pheromone value and the pheromone evaporation rate is small; finally, pheromone trails are reinitiated whenever stagnation is met or a solution has not been improved for a certain number of consecutive iterations.

There are also a few extensions of AS, for example, the Ant Colony System (ACS) by Dorigo and Gambardella (1997a, b), Approximate Non-deterministic Tree Search (ANTS) by Maniezzo (1999) and population-based ACO (P-ACO) by Guntsch and Middendorf (2002a). Some local search methods can also be combined with ACO to improve the solutions.

ACO has been used to solve the traveling salesman problem, the quadratic assignment problem, data network routing problem (Schoonderwoerd *et al*, 1996), and scheduling problem (flow shop or job shop). It has been successful in finding near-optimal solutions comparable to those found using the state-of-the-art approaches in most of those problems except JSSP (Dorigo and Stüzle, 2004, pp.168). The following review presents results obtained from previous work of ACO related to scheduling problems and dynamic problems which may give insights for reactive scheduling in a dynamic job shop.

2.4.2 ACO for static scheduling problems

The AS was first applied to JSSP by Colorni *et al* (1994). It was successful in finding solutions within 10% of the optima for both instances of 10x10 and 10x15 job shop scheduling problems (Dorigo *et al*, 1996). However, despite showing the viability of the approach, the computational results were not competitive with state-of-the-art algorithms for classic JSSPs (Stützle and Dorigo, 1999).

EAS was applied to three benchmark JSSPs in 1999 (Zwaan and Marques, 1999). The results were within 8% and 26% of the best known optima for the 10/10/G/C_{max} Muth-Thompson problem and the 20/10/G/C_{max} Lawrence problem (OR-Library), respectively. The authors considered the results promising since the tests were only partially executed with an iteration number of 2000. The study also presented the importance of parameter settings.

There are also reports of other forms of JSSPs. Blum (2002) applied MMAS to solve the Group Shop Scheduling Problem (GSSP), which is a general Shop Scheduling problem covering JSSP and Open Shop Scheduling (OSSP). Several versions of MMAS were compared and the proposed algorithm could find optima for the tested benchmark JSSP (15x15) and OSSP. Stützle (1998) applied MMAS integrating a local search for a series of benchmark flow shop problems (FSP). The results were compared with several other heuristics and showed that the MMAS gave high quality solutions to FSP in a shorter time, performing better or at least comparable to other state-of-the-art algorithms.

2.4.3 ACO for dynamic problems

The dynamic problems that ACO has been applied include routing problems in communication networks, dynamic traveling salesman problem (TSP), and dynamic JSSP. The applications of ACO in dynamic TSP are reviewed in this study because of its close relevance to dynamic JSSP and the reports of ACO in dynamic scheduling problems are few.

2.4.3.1 ACO for dynamic TSP

The main concern for ACO being applied to a dynamic TSP is about the updating of its problem graph and pheromone matrix, which are the main procedures consuming computational space and time. Thus, the strategies to modify the pheromone matrix become a main topic.

Angus and Hendtlass (2002) applied ACO to dynamic TSP and their motive was based on the following observation: ants did not retreat to their nest and start all over if something blocked their current efficient path; rather, they adapted the path to suit the new constraint. All the pheromone levels at each city were normalized relative to the path segment involving that city with the highest pheromone concentration whenever a city is added in or removed. The result was that the adaptation rate was very high, significantly faster than finding the result by starting all over.

Guntsch and Middendorf (2001) proposed one global and two local strategies to update the pheromone matrix for dynamic TSP considering the compromise between resetting (through equalization) and keeping enough information. The strategies are 1) Restart-Strategy – reinitializes all the pheromone values by the same degree. 2) η–Strategy – uses distances between cities to decide to what degree equalization is done on the pheromone values on all edges incident to each city. 3) τ-Strategy – uses pheromone based information to define another concept of distance between cities. They concluded that the first two strategies performed be the best, closely followed by the τ-Strategy.

Guntsch *et al* (2001) proposed several strategies for ACO to solve a highly dynamic TSP in order to provide a good solution quality averaged over time. Their motive is that the new optimal solutions might be in some sense related to the old ones if changes of the problem instances occur frequently and each change is not too large. The highly dynamic TSP refers to the problem where k cities are exchanged every t iterations between an initial TSP with 200 cities and a spare city pool of 200 cities. EAS was used to update the pheromone matrix. Empirical evaluation showed that the η–Strategy was the best overall strategy.

Guntsch and Middendorf (2002a) proposed P-ACO to keep some recent information for adapting to a new solution in a reasonable time when there was a change in the problem instances. Such recent information was represented by a group of *k* best solutions. A series of TSP benchmarks were tested and the comparison shows that the performance of P-ACO was as least as good as the standard ACO and MAX-MIN ACO for static problems.

The P-ACO was further tested on dynamic TSP by Guntsch and Middendorf (2002b). Their main approach was that a set of solutions was transferred from one iteration to the next rather than transferring pheromone information as in most ACO algorithms. The advantage was that it would usually be faster to modify a few solutions directly than to modify the whole pheromone information of a standard ACO algorithm. Five new population updating strategies were tested on the TSP problem similar to that in (Guntsch *et al*, 2001). The experimental results showed that P-ACO performs superior than the approach that restarted the procedure upon dynamic events.

2.4.3.2 ACO for dynamic job shop scheduling problems

Vogel *et al* (2002) proposed a ***continuously operating Ant Algorithm***, which could easily adapt to sudden changes in the production system. A position-operation-pheromone-matrix (P-O-P-M) and an allocation table were maintained. Pheromone values were reset whenever there was a change. Pheromone updating depends on two factors: temporal buffer and the priorities of jobs (which was reflected in the time of initiating pheromone). The dynamic ACO was tested on a record based on the real-world practice for two months and was compared to manual, priority-rule and GA approaches. The result generated by ACO was only inferior to the GA approach.

2.4.4 ACO as an MAS

There are two approaches to implement the ACO algorithm as MAS. The first approach is to take the advantages of parallel computation of concurrent ant agents, for example, Xiang *et al* (2005). The other is to analogize the co-ordination strategy among foraging ants and their decision-making rules in the field of manufacturing control in order to reach a similar emergent global optimal performance. A good overview of solving difficult real-life problems mimicking natural phenomena can be found in Bonabeau *et al* (1999).

In (Peeters *et al*, 2001), the ant in AS was modeled as an order and resource agent to find solutions while the pheromone environment was modeled according to the layout of a physical flow shop. The test results showed that the proposed approach offered clear benefits in terms of change management. However, the main disadvantages of the pheromone concept were time delays and the need for tuning.

Cicirello and Smith (2001, 2001a) proposed the Ant Colony Control (AC^2) applying the analogy of ACO to the problem of dynamic shop floor routing. The main idea was to assign a new incoming job to an ant, which was responsible for the routing of this job. All communication was carried out indirectly in the form of pheromone that the ants left on the trail between resources. Four experiments were conducted to the AC^2 with different problem configurations. They concluded that the global behaviour emerged was comparable to following the optimal routing strategy on simple problems.

2.4.5 Summary

In summary, the reactive scheduling problem in a dynamic job shop has been studied using dispatching rules, optimum seeking algorithms, and ACO inspired MAS. Dispatching rules are robust in situations where uncertainty or variability is high as compared to optimum seeking approaches. The nature-inspired MAS has only been tested on very simple problem models. For systems where uncertainty or variability is not so high, reactive scheduling using optimum seeking algorithms may provide better solutions with global optimization.

The application of ACO in dynamic TSP inspires the current study of using ACO for dynamic JSSPs although its performance for static JSSPs is not competitive with the other state-of-the-art approaches. Although the ACO algorithm has been tested on the data of a real-world dynamic job shop, a general understanding of its performance in dynamic JSSP is still not clear. In this work, ACO is tested to optimize the throughput and the resource utilization of a simulated dynamic job shop.

3 Analysis of Dynamic Job Shop Scheduling Problems

It can be seen from Chapter 1 that scheduling is a function with a short-term effect in the hierarchy of production management. The decisions from a higher level of management related to production planning internally determine the complexity of a dynamic scheduling problem. However, a proper scheduling system can facilitate the realization of the full potential of a given production system and the general challenge is to explore efficient procedures to find best possible solutions within the time limit demanded by a specific problem.

This chapter first analyzes static JSSPs in section 3.1, then dynamic JSSPs in section 3.2. A simple example in section 3.3 illustrates that an appropriate scheduling approach is decided based on the particular properties of a dynamic JSSP itself. Thereafter the factors affecting the evaluation of a scheduling technique in a predictive-reactive approach are analyzed in section 3.4 and finally, section 3.5 summarizes the chapter.

3.1 Analysis of classical job shop scheduling problem

The factors determining the complexity of a classical JSSP include the sizes of jobs and machines as well as the performance measures, which have been illustrated in section 1.2.5. The factors affecting the solution quality of a classical JSSP are described as follows.

Given a static scheduling problem, the quality of its solution can be determining by: 1) the complexity of the JSSP, 2) the quality of the scheduling procedure, and 3) the available computing time.

The complexity of a problem is determining by the factors mentioned in the above section; the scheduling procedure can be any of those ranging from exact mathematical methods, dispatching rules, meta-heuristics, to artificial intelligence. The available computing time determines how thorough a procedure can be allowed to explore the solution space of the scheduling problem. Some parts of the solution space may never be searched and thus the good schedules in those parts may not be discovered due to the limited computing time. In fact, computing time may hardly be sufficient for finding optimal solutions for most static JSSPs with even moderate sizes due to their NP-hard nature.

The optimality of a schedule should be measured by how near the solution is to the optimal one, if it is known, in terms of solution quality. However, this is generally not measurable since the optima are unknown. Thus either advanced scheduling techniques or extended computing time has to be adopted in order to improve the optimality of a schedule.

3.2 Analysis of the dynamic scheduling problem

The dynamism of a scheduling problem is usually treated following the approach of a rolling time horizon (Raman and Talbot, 1993), *i.e.*, a deterministic scheduling problem consisting of all known jobs is solved at each rescheduling moment. When a new job arrives at time *t*, the part of the solution consisting of operations already started before *t* is fixed and a new problem is constructed, consisting of the backlog to be starting after time *t*, plus all the operations from the newly arrived job. The dynamic problem is thus decomposed into a series of static intermediate scheduling problems over time (Branke, 2002).

Therefore, in a dynamic JSSP, each incoming job changes the current setting of the intermediate scheduling problem and the task of reactive scheduling is to continuously generate schedules for the set of existing unprocessed jobs in a timely manner so that an overall optimality of performance can be reached for the given period of time. A specific intermediate scheduling problem is internally decided not only by the characteristic of the new job but also by the status of the shop floor at the moment that the job arrives.

3.2.1 Factors that characterize an intermediate JSSP

The two factors that characterize an intermediate JSSP are the arrival time of a new job and its characteristics determined by the technical sequence, which refers to the order of workcenters that the job has to be processed, and the processing time distribution over workcenters. Their effects are illustrated in the following sections.

3.2.1.1 The arrival time

Given a set of jobs with *a priori* schedule, the subsequent intermediate JSSP varies as the arrival time of the new job varies. For example, given a $2/3/G/C_{\max}$ JSSP with a technological matrix T and a processing time matrix P:

$$T = \begin{bmatrix} M1 & M2 & M3 \\ M3 & M2 & M1 \end{bmatrix} \qquad P = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.3 & 0.6 & 0.6 \end{bmatrix},$$

an optimal schedule can be given in Fig. 3.1.

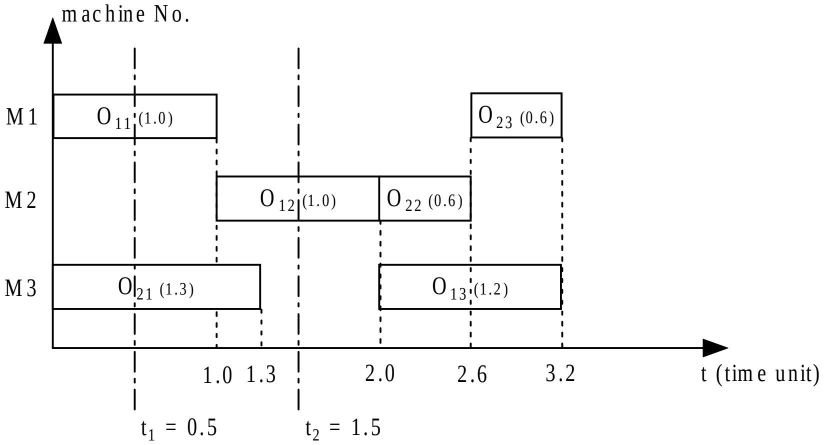


Fig. 3.1 An optimal schedule for the example JSSP

A new job J_3 coming at $t_1 = 0.5$ incurs a new scheduling problem that is different from the one incurred by the same job but coming at $t_2 = 1.5$. The former has earliest machine available times from $\{1.0, 0.5, 1.3\}$ for $\{M1, M2, M3\}$ respectively and a set of un-executed operations including O_{12}, O_{13}, O_{22}, O_{23} plus all of the operations from the new job. Operations O_{11} and O_{21} are not included because they are already being processed at the time the new job comes in.

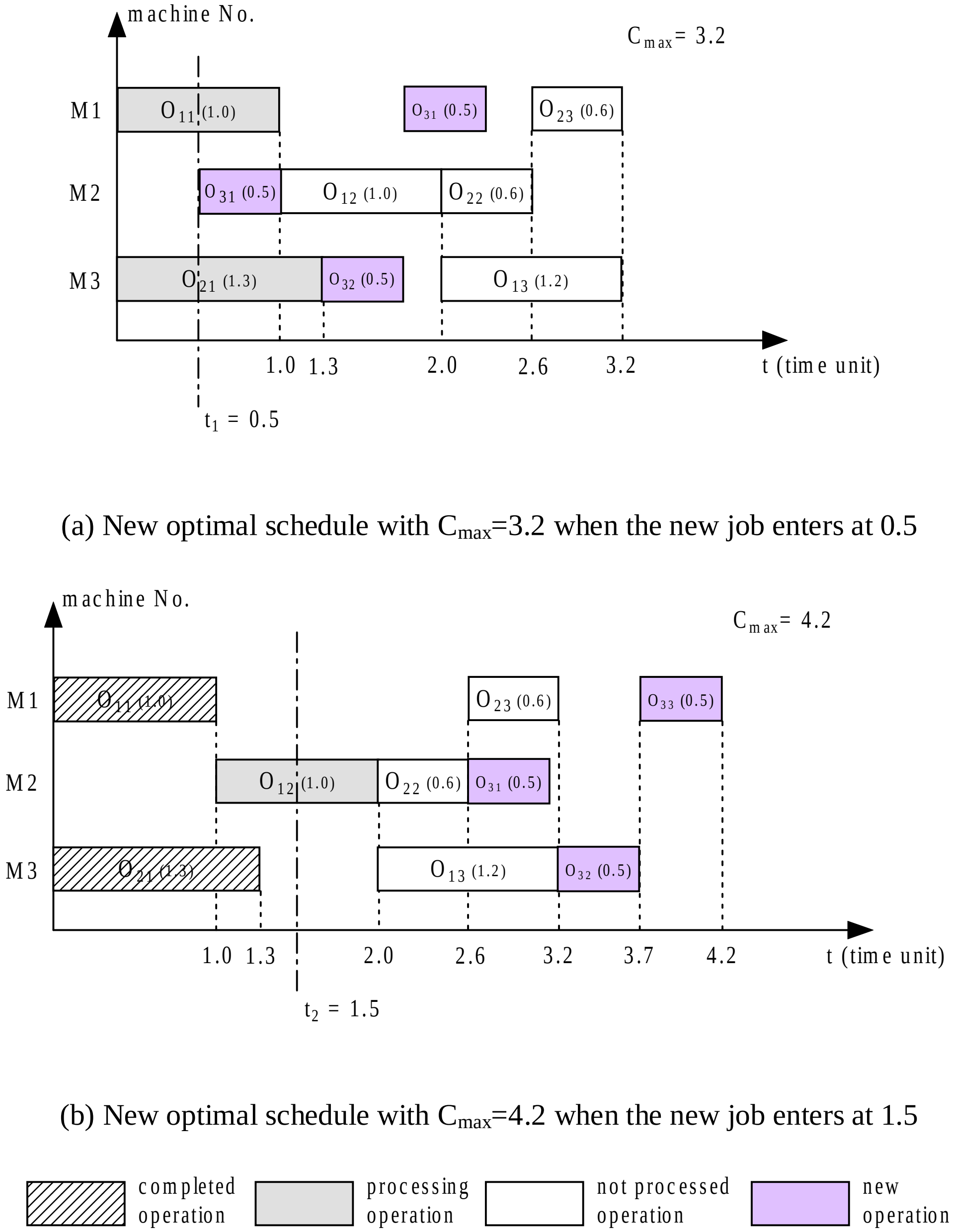
Similarly, the later problem has earliest machine available times from $\{1.5, 2.0, 1.5\}$ for $\{M1, M2, M3\}$ respectively and the set of operations including O_{13}, O_{22}, O_{23} plus those of the new job. The two different intermediate JSSPs are list in Fig. 3.2.

	operations for rescheduling	earliest machine available time
problem 1 (t_1)	O_{12}, O_{13}, O_{22}, O_{23}	1.0, 0.5, 1.3
problem 2 (t_2)	O_{13}, O_{22}, O_{23}	1.5, 2.0, 1.5

Fig. 3.2 The comparison of two intermediate problems

Two new JSSPs are different in their operations and the earliest machine available times. Thus, their complexities are different in seeking new optimal schedules. Fig.

3.3 illustrates the solutions of two different problems assuming that the technical sequence and the processing times of job J_3 are $TC_3 = \{M2, M3, M1\}$ and $PC_3 = \{0.5,0.5,0.5\}$, respectively.



(a) New optimal schedule with C_{max}=3.2 when the new job enters at 0.5

(b) New optimal schedule with C_{max}=4.2 when the new job enters at 1.5

Fig. 3.3 New optimal schedules after the same job enters at different times

3.2.1.2 The characteristics of the new job

The new problem is also affected by the characteristics of the new job, which can be described in terms of the technical order and the processing-time distribution of its operations.

- The effects of the technical order

The minimal makespan (C_{max}) could be improved from $C_{max} = 4.2$ (Fig. 3.2 (b)) to $C_{max} = 3.2$ (Fig. 3.4) if the technical order of the new job is changed to $TC_3' = \{M3, M1, M2\}$.

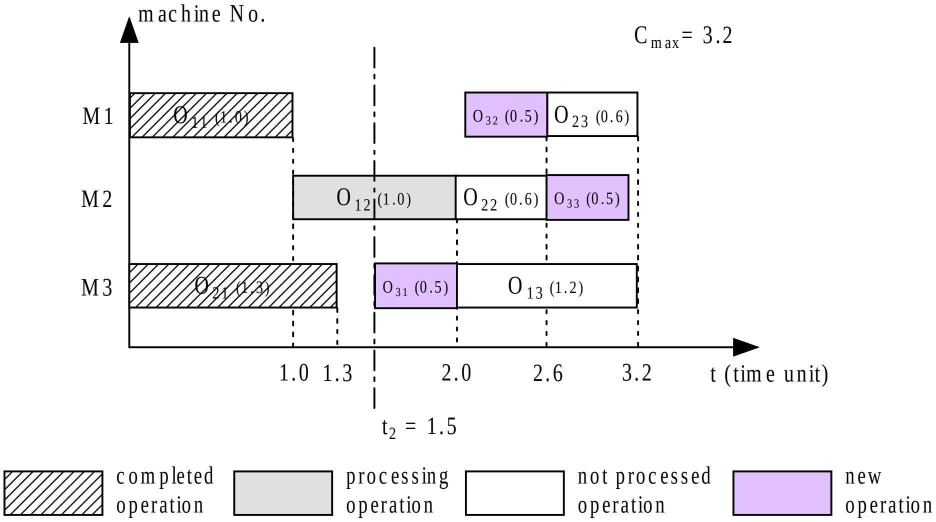


Fig. 3.4. C_{max}=3.2 after the operation order is changed

- The effects of the distribution of processing times

Similarly, the minimal makespan could be improved from $C_{max} = 4.2$ (Fig. 3.2 (b)) to $C_{max} = 4.1$ (Fig. 3.5) if the processing time of the new job is re-distributed from $PC_3 = \{0.5, 0.5, 0.5\}$ to $PC_3' = \{0.6, 0.4, 0.5\}$ while the total processing time and its arrival time remain unchanged.

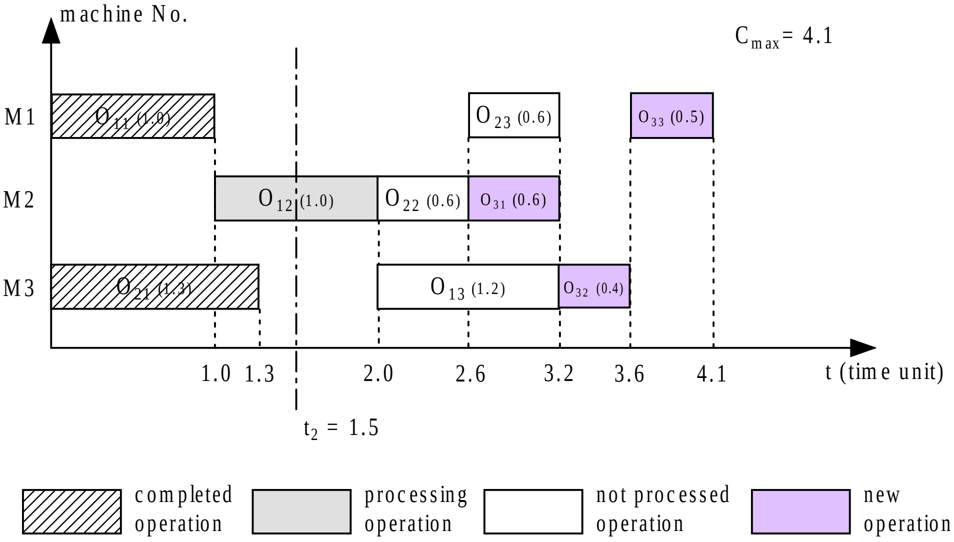


Fig. 3.5. C_{max} = 4.1 after the processing time is redistributed

3.2.2 Factors that characterize an overall dynamic JSSP

As a dynamic JSSP is the combination of all static intermediate JSSPs and each of them is determined only by the arrival time and the characteristics of the new job, it can be concluded that the distribution function of arrival times over time and the distribution function of processing times over workcenters work together to characterize an overall dynamic JSSP for the given period. The distribution function of arrival times is called ***inter-arrival function***; the distribution function of processing times over workcenters is generally determined by another two distributions within each job: the technical sequence and the processing time distribution of operations.

Furthermore, jobs can be released to the shop floor in lots, that is, several jobs can be simultaneously included in one lot. The following sections describe the effects of the above three aspects of a dynamic JSSP.

3.2.2.1 Effects of inter-arrival function

The inter-arrival function determines the moments of jobs arriving at the shop floor. In literature, this function always takes the form of a Poisson distribution, which has been shown to be a good approximation to the arrival process if the different sources generating job arrivals to the shop are statistically independent (Albin, 1982). Poisson distribution is also adopted in the current study to simulate the arrival process of incoming jobs.

This ***Poisson distribution*** is given as:

$$f(k,\lambda)=\frac{e^{-\lambda}\lambda^{k}}{k!} \tag{3.1}$$

where *e* is the base of the natural logarithm (*e* = 2.71828…); *k* is the number of occurrences of an event – the probability of which is given by the function; *k*! is the factorial of *k*; and λ is a positive real number, equal to the expected number of occurrences that take place during the given interval.

Thus the expected mean number of jobs per time unit should be: $1/\lambda$, which determines the mean workloads of all the machines over time and the dynamic level of the JSSP. For a given set of jobs, the higher the value of $1/\lambda$, the higher are the workloads of the machines and the more dynamic is the dynamic JSSP. The value of $1/\lambda$ actually determines the complexity of a dynamic JSSP as the mean size of an intermediate JSSP increases, or when the value of$1/\lambda$ increases.

3.2.2.2 Effects of the distribution of processing times

The distribution of processing times over workcenters or machines of all jobs are the collective results of the distributions of their technical orders and processing times.

- Technical order

In the literature, the order of the operations in a job is generally randomly chosen from a uniform distribution. That is, every workcenter has an equal chance to be chosen. The same mechanism is adopted in the current study.

- Values of processing times

The values of processing times are normally decided by the exponential distribution in the literature. Exponential distribution is also adopted in the current study to generate processing times. The ***exponential distribution*** has the form of:

$$F(x,\vartheta) = \begin{cases} 1 - e^{-\vartheta x}, & x \geq 0, \\ 0 \qquad\quad , & x < 0. \end{cases} \tag{3.2}$$

where $\vartheta > 0$ is a parameter of the distribution, often called the rate parameter. The distribution is supported on the interval $[0,\infty)$. The mean or expected value of an exponentially distributed random variable X with rate parameter ϑ is given by

$$E[X] = \frac{1}{\vartheta}. \tag{3.3}$$

Shannon (1979) reported that the nature of processing time distribution significantly affects the performance of the scheduling rules. An interesting observation is that the use of the exponential distribution tends to favor the SPT rule. The reason could be that SPT avoids allocating the machines to one of the very long operations, which is possible when draws are taken from an exponential distribution (Ramasesh, 1990).

3.2.2.3 Effects of job lots

Sometimes, the jobs are released in lots instead of one by one. The size of jobs in a lot determines the severity that the underlying scheduling problem is changed. For example, there are 16 unprocessed operations when a lot of new jobs are released to the shop floor. The size of the operations for the new intermediate JSSP is 22 if there is only one job, which has 6 operations, per lot. However, it becomes 28 if there is one more job (which also has 6 operations) per lot. Obviously, the underlying problem is changed more severely by the larger lot than the smaller one.

3.3 Internal problem properties determine Approaches

It is widely acknowledged that no one particular approach can perform best in all situations. Each approach has its own niches of application domains and it is important to find the appropriate application domains of a proposed scheduling algorithm. The following example shows a scenario that is best suited for FIFO dispatching rules. Some indications can be made for potential application domains of algorithmic approaches.

Figures 3.6 to 3.9 present an example where the utilizations of all machines can reach 100% with a very simple FIFO dispatching rule if dynamic jobs arrive regularly and their processing time distributions on the machines can match each other to cover all the time slots on all the machines.

Fig. 3.6 gives an initial optimal schedule, which minimizes the makespan for three types of jobs: T1, T2 and T3. Their technical orders are $TC_1 = \{M1, M2, M3\}$, $TC_2 = \{M2, M3, M1\}$ and $TC_3 = \{M3, M1, M2\}$; their respective processing times are $PC = \{2, 0.5, 0.5\}$. Jobs are assumed to arrive at the shop floor regularly in the sequence of T1, T2 and T3 per time unit. Obviously, the combination of these technical orders as well as the distributions of processing times makes the workloads on all machines identical.

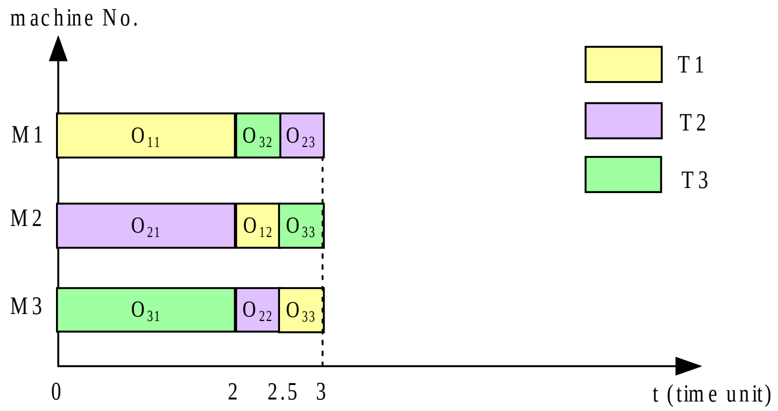


Fig. 3.6 The initial schedule

The 4th job of type 1 (T1) comes in at $t_1 = 0$ and the subsequent orders of the operations on three machines according to FIFO are given in Fig. 3.7. Those orders are changed (Fig. 3.8) where the 5th new job of type 2 (T2) at $t_2 = 1$. At the same time, the first operations of all the first three jobs are being processed. Next, the 6th job of type 3 (T3) arrives at the shop floor at $t_3 = 2$ when the operations of the first three jobs are completed. There are two possible schedules as both O_{22} and O_{61} arrive at machine 3 (M3) simultaneously. Fig. 3.9 gives both schedules when O_{22} and O_{61} are first processed respectively.

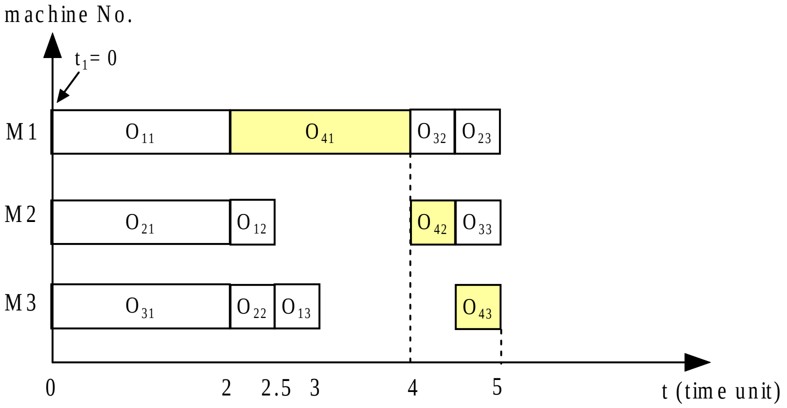


Fig. 3.7 The 4th new job of type 1 enters at t_1=0; new C_{max}=5 by FIFO

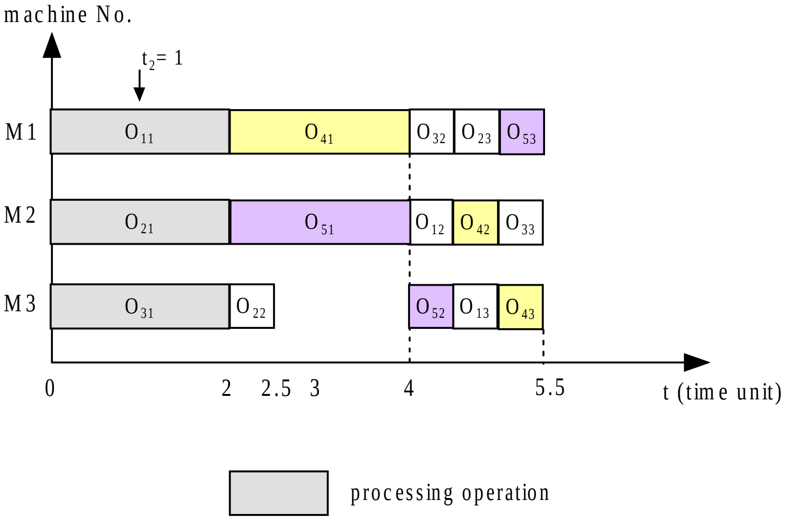
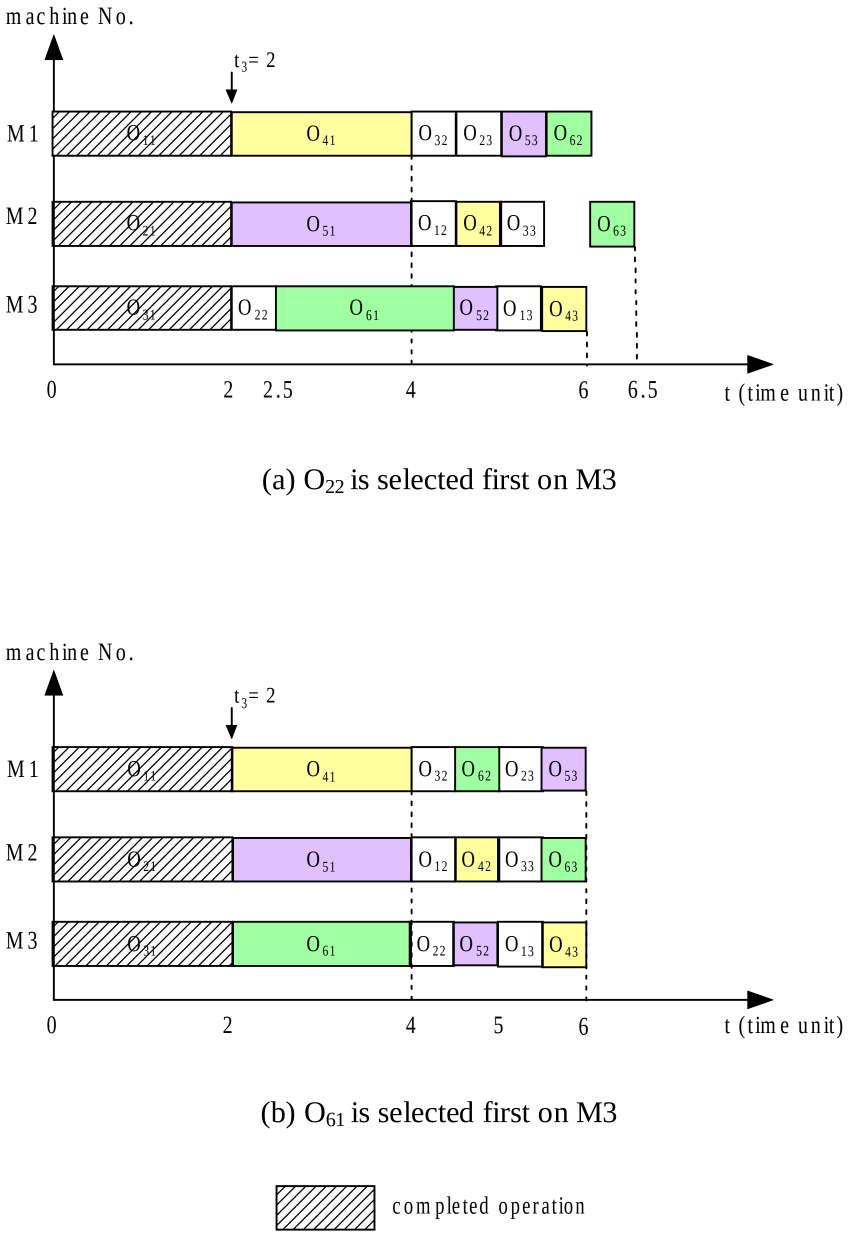


Fig. 3.8 The 5th new job of type 2 enters at t_2=1; new C_{max}=5.5 by FIFO



(a) O_{22} is selected first on M3

(b) O_{61} is selected first on M3

Fig. 3.9 The 6th new job of type 3 enters at t_2=2; new C_{max}= 6 by FIFO

The utilizations of all machines can always be optimized at 100% by FIFO if jobs continue to come in at the same inter-arrival time distribution and in the same sequence of job types. The example shows that the dynamic JSSP can be optimally solved with a very simple dispatching rule, FIFO, which takes very little computational effort. The particular combination of internal factors like the arrival frequency and the processing time distribution of dynamic jobs determine the success of this solution approach.

Furthermore, dynamic JSSPs that have no such special combination of the jobs and the inter-arrival function but have jobs coming in at a high frequency may also favor dispatching rules as many researchers have observed, which can be explained as follows. 1) The schedules found in a limited computing time may not be optimal or near optimal at all. 2) An unsatisfactory schedule may cause its following scheduling problem to be more complex. 3) Even if the schedules are optimal, their strengths may not be fully realized before they are made obsolete by dynamic events.

Thus, the dynamic JSSPs that have great potentials to be solved with high performance through a predictive-reactive approach adopting optimum seeking algorithms may have characteristics like less frequent dynamic jobs or non-uniformly distributed arrival times and processing times.

3.4 Analysis of factors affecting the evaluation of a scheduling technique

In a static JSSP, the execution of a schedule is not considered as it is assumed that the optimality predicted by a schedule can be fully realized. However, it is no longer the case for a dynamic JSSP, where the underlying scheduling problem continues to changing due to continuously arriving jobs. The performance of a scheduling technique in the predictive-reactive approach for a given period of time is thus the overall results of the realized optimality provided by many intermediate schedules.

The principle for a scheduling system adopting the rolling time horizon approach has been always trying to find a best schedule for each intermediate scheduling problem. It is also desirable to realize the optimality of intermediate schedules as early as possible since their execution is uncertain in a dynamic environment and is out of the control of a scheduling system. Thus, the performance of a scheduling technique in a dynamic environment is related not only to its ability of finding the best schedule for each static intermediate scheduling problem but also to the realization of the optimality provided by those intermediate schedules.

3.4.1 Factors that can affect the quality of an intermediate schedule

The optimality values of intermediate schedules over time in a dynamic environment can be illustrated in Fig. 3.10, with the optimality value formulated as $1/makespan$ so that a minimal makespan implies a maximal optimality. In the figure, a schedule with an optimality value of a_0 has been executed from time t_0 to t_1, when a new job J_1 arrives. The optimality of the current schedule immediately drops to a_0' if job $J_1$is simply put at the end of the schedule. Next, a reactive scheduling procedure is triggered to form a sub-problem with the backlog operations and all of the operations from J_1assuming the scheduling period allowed is $[t_1, t_1']$. A new schedule with an optimality value of a_1 is generated and executed from t_1' till the second job J_2 arrives at time t_2, where the similar procedure repeats.

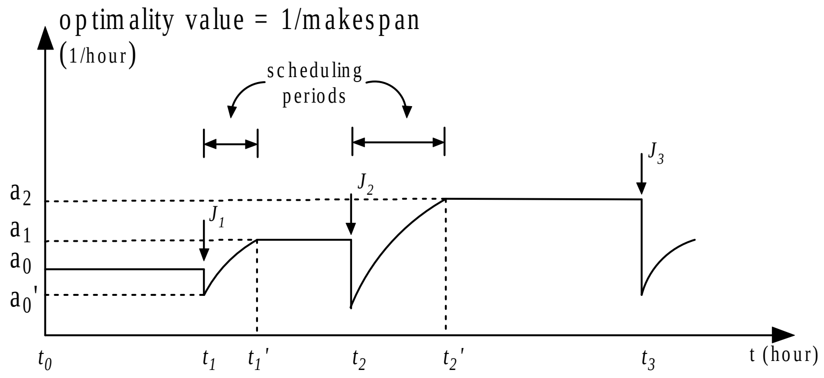


Fig. 3.10. The optimality values of schedules over time in a dynamic environment

The optimality of the intermediate schedule found in the time intervals of $[t_1, t_1']$ or $[t_2, t_2']$ is affected by the following factors: 1) the length of the time interval; 2) the operation size of the intermediate JSSP; 3) the quality of the scheduling algorithm; and 4) dynamic scheduling strategies.

3.4.1.1 The length of a computing interval

A computing interval refers to the time span that can be allowed for generating a new intermediate schedule. The length of this interval is problem-dependent, for example, the computing time for the sub-problem caused by job J_1 can be decided by its traveling time from the reception area to its first workcenter. The length can proportionally affect the optimality of the schedule.

3.4.1.2 The size of an intermediate JSSP

Given the same scheduling period, a smaller scheduling problem implies lower computational cost and better solution and *vice versa*. A schedule minimizing makespan may have better opportunity to complete more operations before an interruption occurs. Thus, the resulting intermediate sub-problem can have a smaller size and hence a better chance to find a good schedule, which facilitates the generation of another good schedule following a disturbed moment. On the contrary, a larger sub-problem may have less opportunity to find a good schedule, and a poorer schedule, in turn, can produce a larger following sub-problem. The procedure goes on and the overall performance of the scheduling system may deteriorate.

3.4.1.3 The quality of a scheduling algorithm

A good scheduling algorithm should generate a timely and satisfactory schedule to guide production. Information adaptation may help in speeding up the procedure of finding a new optimum, especially when the underlying problem is not changed severely. The idea is to generate a schedule not from scratch but to exploit the optimal information kept in the current solution and quickly find a good solution for the modified problem. This adaptation also has an advantage of maintaining similarity between two continual schedules, which is preferred in real life applications. This idea has been studied in TSP (Guntsch and Middendorf, 2001, 2002a, and 2002b) (Guntsch *et al*, 2001).

3.4.1.4 Dynamic scheduling strategies

Dynamic scheduling strategies involve choosing scheduling frequency or employing partial scheduling. ***Scheduling frequency*** refers to how often the schedule generation procedures are triggered. It can be event-driven, periodic-driven or performance-driven. The ***event-driven*** approach triggers a rescheduling procedure whenever an event occurs; the ***periodic-driven*** approach triggers the rescheduling procedure according to a pre-set time period; the ***performance-driven*** approach uses performance values of the current production system as the trigger of the rescheduling procedure. These approaches essentially solve different dynamic scheduling problems where the last two alter the original problem by postponing the reactions to interrupters.

Partial scheduling considers only a partial set of jobs from the sub-problem in one computing interval in order to cover the next estimated execution period. This approach is inspired by the fact that a schedule may not have an opportunity to be fully executed before dynamic disturbances; thus, there is no need to include the operations that may not be processed before those interruptions in order to reduce computation efforts. This approach may find a partial schedule in a short time but lacks a global view of the problem.

Dynamic scheduling strategies can change an original computing interval through different scheduling-driven approaches and alter the original size of an intermediate JSSP by partial scheduling. Keeping other experimental parameters unchanged, the adjustment of dynamic scheduling strategies can improve the performance of a proposed scheduling algorithm.

3.4.2 Problem-related properties for improving schedule optimality

Some problem-related properties, which can facilitate the realization of the optimality provided by a schedule as early as possible, should be explored. For example, given two different schedules with the same makespan for the same problem, the one with more operations at the early stage may be preferred since more operations may have been completed before the interruption and thus reduce the size of the next scheduling problem.

The identification of these properties is problem-dependent and promising in improving performance. Other potential properties may be related to the positions of time slacks and the operations with longest processing times on critical paths.

3.5 Summary

This chapter analyzes the static JSSP, the dynamic JSSP and the factors that characterize an intermediate JSSP and the overall dynamic JSSP. It also points out that internal problem properties determine appropriate approaches. Finally, it explores the factors affecting the evaluation of a scheduling technique.

Based on these analyses, the systematic approaches to test a proposed scheduling technique can be carried out in the following directions: 1) to test a scheduling technique in different experimental environments defined by different dynamic levels, dynamic severity, processing time distributions, system configurations, and performance measures; and 2) to improve the performance through adjusting the internal parameters of the scheduling algorithm if possible, and the dynamic scheduling strategies like the rescheduling-driven mechanism and partial scheduling.

4 The Test Bed

The goal of this chapter is to build a common test bed to facilitate systematic studies of the performance of scheduling algorithms in a dynamic job shop environment. The test bed simulating a generic job shop should be able to provide a realistic configuration of a shop floor, generate dynamic or stochastic events such as incoming jobs and machine breakdowns, provide necessary scheduling algorithms or dispatching rules to guide processing and control rules to react to dynamic events, track job movements and the status of the machines, workcenters and the shop floor, and provide statistical analysis for performance measures.

The structure of the chapter is as follows: in section 4.1, the related works on system modeling/test beds for dynamic scheduling are presented; in section 4.2, the definition of a generic job shop is given; in section 4.3, a generic job shop is modeled as a DES, and a prototype of the job shop is implemented in section 4.4 as an MAS. Section 4.5 is especially devoted to describe the communication of agents in the MAS and a case study is provided in section 4.6.

4.1 Background

In order to build an up-to-date test bed to study the scheduling methods in dynamic/stochastic environments, the test beds of main dynamic scheduling approaches should be reviewed. Generally, the performances of dispatching rules and predictive-reactive scheduling approaches are tested through simulation and Ramasesh (1990) gave an excellent review on the simulation research in dynamic JSSP.

Simulation is the most common method for constructing models that include the temporal dynamics of manufacturing systems, many of which can be modeled as DES (Askin and Standridge, 1993). Law and Kelton (2000) applied simulation to find the best configuration of facilities using dispatching rules. In Sabuncuoglu and Bayiz (2000), the test bed is based on a simulation model coded in the C language with ten levels of frequency of scheduling and four types of problem instances. The down time distribution follows a Gamma distribution with a shape parameter of 1.4 and a mean of 40 minutes; the number of operations for one job is drawn from a discrete uniform distribution from 5 to 15; processing times are generated from a discrete uniform distribution from 20 to 80. In Sabuncuoglu and Kizilisik (2003), six machines and three automatic guided vehicles (AGVs) comprise the flexible manufacturing system. The job inter-arrival time is exponentially distributed. Each job has either five or six operations with equal probability; operation times are drawn from a *2–Erlang* distribution. The review shows that a test bed should also have the capability for statistical analysis.

The complex nature of the dynamic scheduling problem dictates that traditional simulation experiments can only be performed on small systems. Besides, a good scheduling test bed should be able to facilitate the systematic selection of parameters and configurations. The distributed computation, which can be realized through agent technology, has the computational capacity for large problems and provides the scalable structure for many problem configurations.

Furthermore, the performance of a scheduling approach can be systematically evaluated based on statistical analysis on different dynamical levels, problem configurations and performance measures. Long-term average performance based on statistical analysis can be carried out when the generic job shop is simulated as a DES.

Combining MAS with DES enables the proposed test bed not only to meet the requirements in section 4.1 but also has the advantages of: 1) simultaneous execution of events on distributed locations, 2) distribution of event generation, state keeping, event-list managing and data recording/analyzing, 3) possible performance improvement through agent coordination or negotiation, 4) examination of long-term performance, 5) scalability of the MAS to support further extension of the test bed, and 6) a common test bed that could use the similar structure and logic between simulation and actual control of the job shop.

4.2 The generic job shop

A ***generic job shop*** in this study refers to a generalized representation of real life job shops considering not only the configuration of their floor layout but also MHS.

A generic job shop can be physically made up of several workcenters, a receiving/shipping station, and material transportation devices as shown in Fig. 4.1. A ***workcenter***, shown in Fig. 4.2, processes one type of operation using several similar machines. It has a queue to buffer incoming jobs when all the machines are not available and another queue for completed jobs to wait for transportation. The ***receiving/shipping stations*** receive new jobs and ship out all the completed jobs. All the workcenters and the receiving/shipping station are located in the shop floor according to certain layouts. The distances between every two of them are given in a layout matrix. Some *MHDs* transport jobs between workcenters.

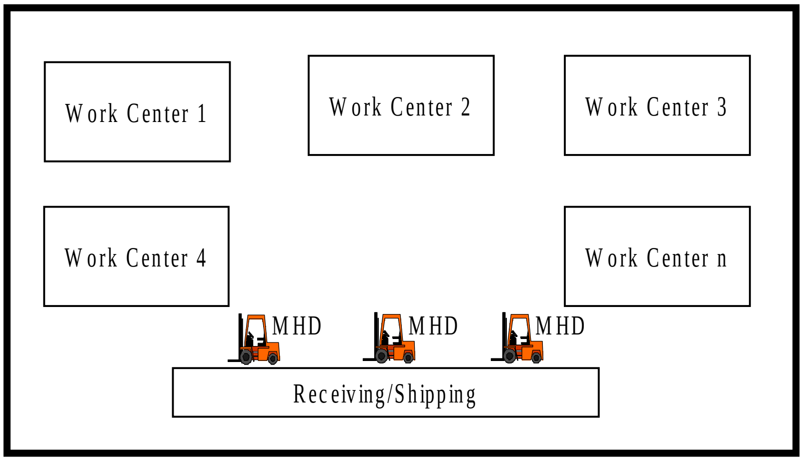


Fig. 4.1 The components of a job shop

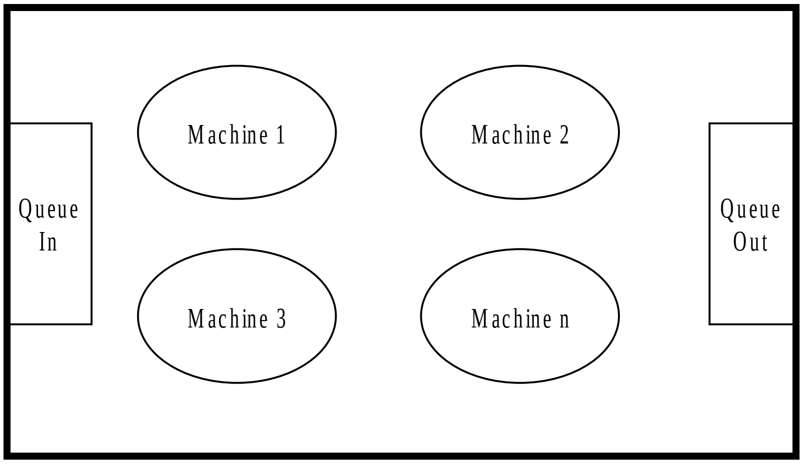


Fig. 4.2. The components of a workcenter

4.3 Discrete event simulation model

Three basic elements in the discrete event simulation include the state of the system, event actions and event lists. The overall state of a generic job shop system is determined by the status of the machines and jobs in it where machines are located in different workcenters and jobs are distributed either in workcenters or on traveling devices. According to Koestler (1967), architectures of manufacturing systems are inherently hierarchical. In section 4.3.1, entities are organized hierarchically so that the global state can be monitored by distributed entities at different levels. In section 4.3.2, the possible states for each type of entities are described. In section 4.3.3, the dynamic events and their actions are presented and finally, the mechanism to maintain distributed event lists is explained in section 4.3.4.

4.3.1 Decomposition of the global state

The hierarchical relationship in a generic job shop is illustrated in Fig. 4.3. Entities like machines or jobs can be grouped and monitored by a higher level entity, which in turn forms another group with its similar entities and is monitored by another higher level supervisor. For example, a group of machines is monitored by their workcenter manager and the state of the workcenter is monitored by the shop floor monitor. A job can be monitored by either a workcenter manager or the shop floor depending on whether it stays in a workcenter or travels on a MHD. In this manner, the global state of the job shop can be tracked through monitoring workcenters and traveling jobs.

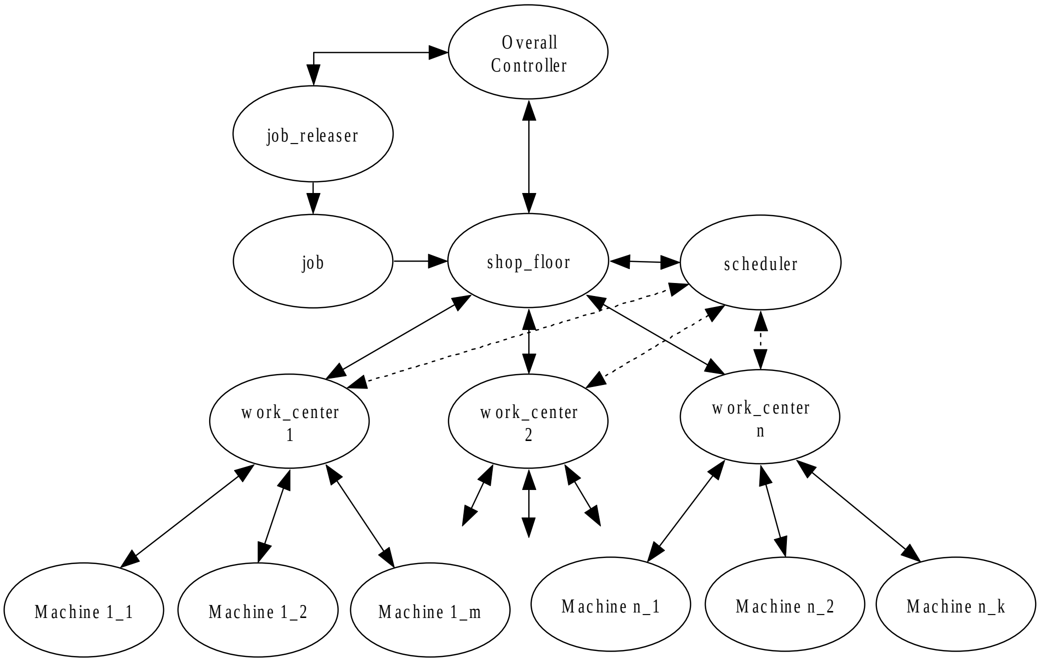


Fig. 4.3. The hierarchical relationship in a generic job shop

There are seven types of entities in the simulated job shop: machines, workcenters, shop floor, jobs, scheduler, job releaser, and controller. Generally, an entity will have a wider view of the system if it is located at a higher level in the hierarchy. A machine, a job or a scheduler can only monitor its own states while a workcenter has a wider scope by monitoring jobs, machines and buffers. Similarly, the shop floor entity can have an even wider view of monitoring the workcenters, traveling jobs and MHDs. Furthermore, the state of an entity in a higher lever does not contain the detailed state information of its supervised entities. For example, the state of the job shop does not contain the information of the buffer status in its supervised workcenters. This approach facilitates distributing data as well as their analysis to their most relevant locations.

The job releaser and the controller are not parts of a job shop but are responsible for generating new jobs and advancing the simulation time, respectively.

4.3.2 States of entities

A job can be in the states of waiting-for-process, in-process and on-traveling assuming that only one buffer is needed in a workcenter and finished jobs can be sent to the next stage immediately. It is in *waiting-for-process* when it waits at the buffer of a workcenter to be processed; it is *in-process* when being processed on a machine; and it is *on-traveling* when it is traveling between workcenters.

A machine can be in the states of busy, idle and down. It is *busy* when it processes a job and *idle* when it waits for a job. The *down* state refers to the period from machine breakdown to its recovery.

A workcenter can be in four states: idle, partial, full, and buffered. It is *idle* when there is no job in it and all available machines are idle. It is *partial* when machines are only partially used. It is *full* when all machines are busy and there are no waiting jobs. Finally, it is *buffered* when all machines are busy and there are jobs waiting.

A shop floor can be in three states according to the number of jobs in it: *idle* (no job on the floor and all workcenters are idle), *working* (at least one job is on the floor) and *completed* (simulation completed and analysis can be carried out).

4.3.3 Events and their actions

The global state of a job shop system is changed by the actions incurred by any dynamic events concerning jobs and stochastic events in the shop floor. The dynamic events related to a job entity include its arriving at or leaving resources like machines, workcenters, the shop floor and MHDs. The stochastic events include dynamic incoming of job orders, machine breakdowns/ups and processing time variation. However, it is not necessary to model all of the aforementioned events. Only five essential events are identified and they can be categorized as job-related and machine-related events.

4.3.3.1 Job-related events

Job-related events are initiated by jobs and there are two types.

- New job event

A new job event represents a new job order, which is released to the shop floor by the job releaser according to certain distribution functions. The event action for this event is illustrated in Fig. 4.4. The event is registered to the shop floor, which then increases the size of its WIP and confirms the registry. The job then heads to the next workcenter from the receiving station traveling on a MHD and another event called the incoming job event is generated immediately. The time period required to travel to the next workcenter is decided by the speed of its MHD and the distance between the two workcenters. The incoming job event is forwarded to the shop floor entity and its description is given in the following section.

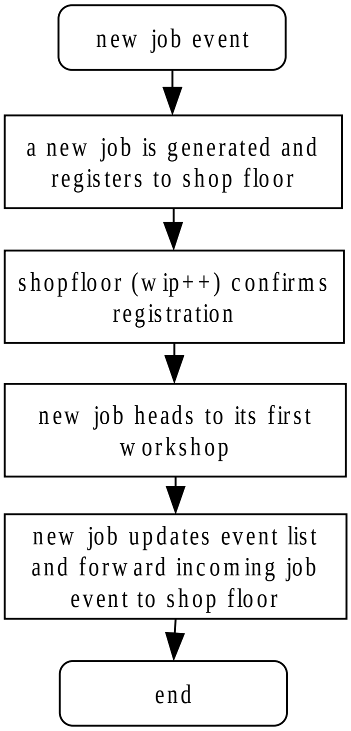

new job event
a new job is generated and registers to shop floor
shopfloor (wip++) confirms registration
new job heads to its first workshop
new job updates event list and forward incoming job event to shop floor
end

Fig. 4.4. The actions upon the new job event

- Incoming job event

An incoming job event indicates the arrival of a traveling job at a workcenter. When it is initiated, the job enters a workcenter from the shop floor and requests service; the workcenter then allocates the job according to its state and control rules or the schedule. If the job cannot be processed immediately, it will be put into the workcenter buffer, otherwise, it will be sent to one of the machines and another event, namely a “leaving job" event (from the machine), will be generated. The event actions and state changes on the related entities are represented in Fig. 4.5.

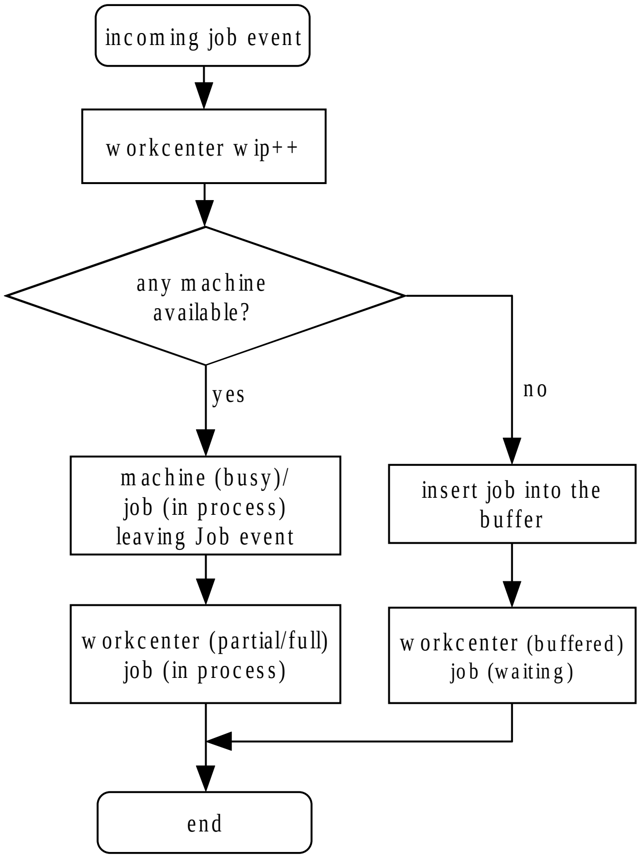

incoming job event
workcenter wip++
any machine available?
yes
no
machine (busy)/ job (in process) leaving Job event
insert job into the buffer
workcenter (partial/full) job (in process)
workcenter (buffered) job (waiting)
end

Fig. 4.5. Actions and state changes upon the incoming job event.

4.3.3.2 Machine-related events

Machine-related events are initiated by a machine and there are three types: 1) leaving jobs, 2) machine breakdowns, and 3) machine ups.

- Leaving job

A leaving job event indicates the completion of an operation by a machine. When this event is initiated, the completed job leaves its machine and workcenter, travels to the next workcenter and then generates another incoming job event. Meanwhile, the newly freed machine is available for processing the next job. If it is allocated with another job, a new leaving job event for the new job will be generated, otherwise it will be idle. Finally, the workcenter reduces the size of its WIP by one. The event actions and the state changes of the related entities are presented in Fig. 4.6.

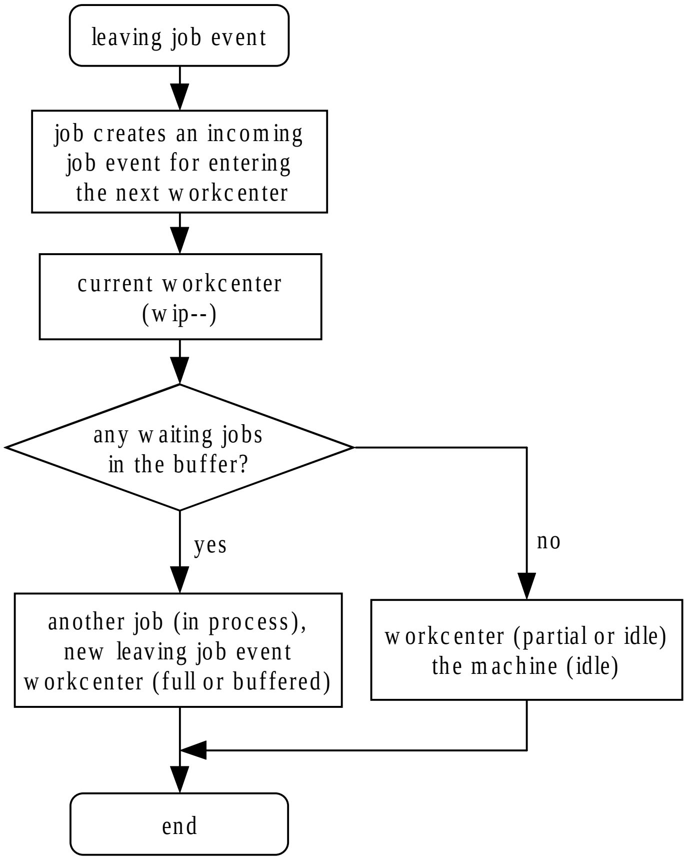

leaving job event
job creates an incoming job event for entering the next workcenter
current workcenter (wip--)
any waiting jobs in the buffer?
yes
no
another job (in process), new leaving job event workcenter (full or buffered)
workcenter (partial or idle) the machine (idle)
end

Fig. 4.6 Event actions and state changes upon a leaving job event

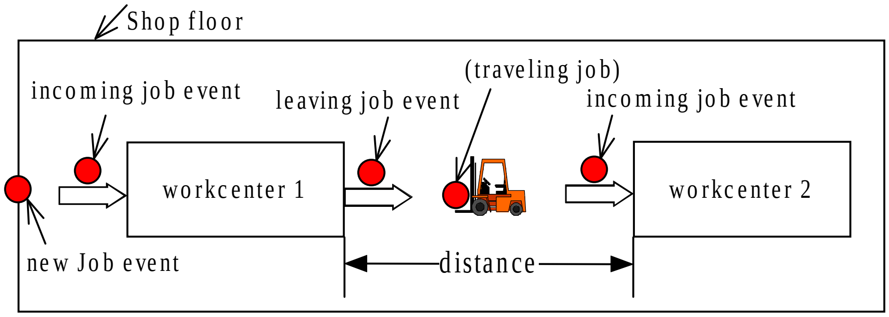


Fig. 4.7. The dynamic events incurred by a routing job

The locations of the previous events in a job shop are illustrated in Fig. 4.7. The leaving job event causes a job to leave both its machine and workcenter simultaneously, assuming that the finished job can be immediately transported to the next workcenter. The relationship between job events is shown as an event diagram in Fig. 4.8.

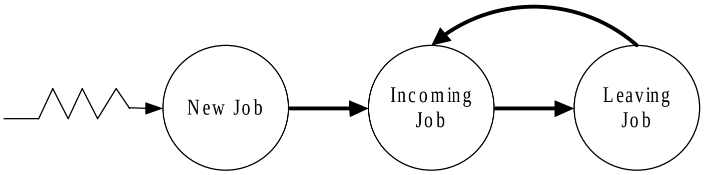


Fig. 4.8. Event graph of job related events

- Machine breakdowns/ups

A machine breakdown event is assumed in this work to occur only when a machine is busy processing jobs (Law and Kelton, 2000). The machine will change its state to *down* on a machine breakdown event and immediately create a machine-up event to represent the time that it will take to be repaired. Meanwhile, the interrupted job is sent to another available machine generating another incoming job event or it is sent to the buffer. Similarly, when a machine-up event occurs, the machine is ready to process operations. If a job is allocated to it, the machine will go to the state of busy and a new leaving job event will be generated. Otherwise, it remains idle. The action and state changes for both events are illustrated in figures 4.9 and 4.10, and their relationship is given in the event diagram in Fig. 4.11.

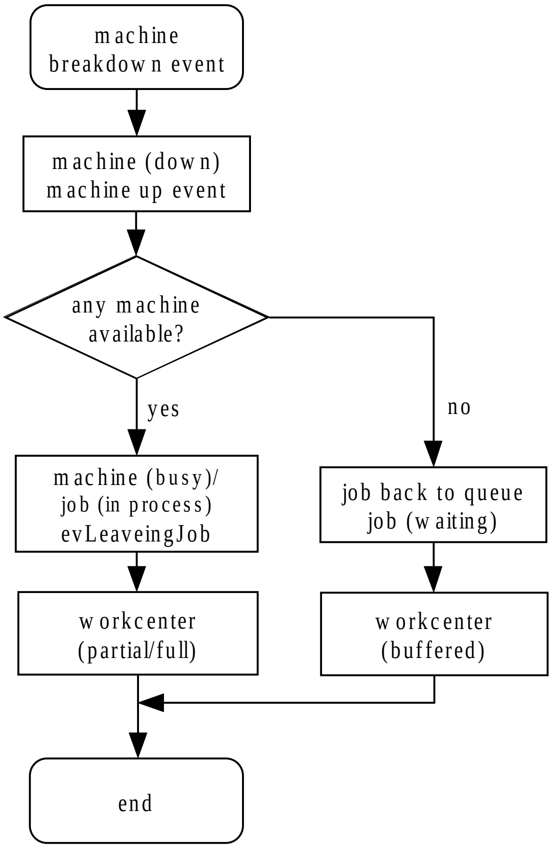


Fig. 4.9. Actions and state changes upon a machine breakdown event

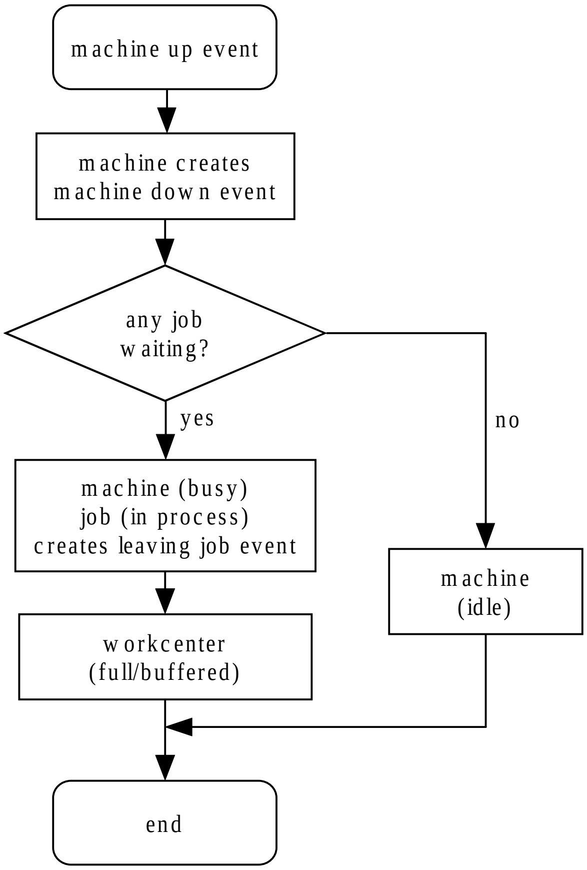


Fig. 4.10. Actions and state changes upon a machine up event

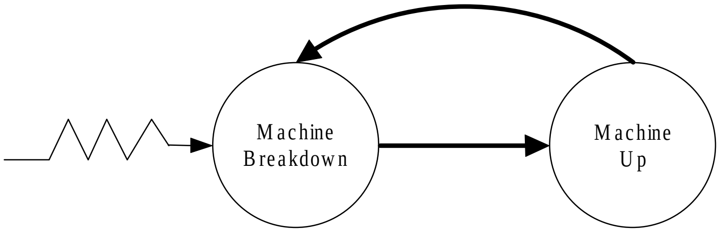


Fig. 4.11. Event graph of machine breakdown and up

4.3.4 Event lists

The global state is maintained as one entity and all events are sorted in one event list according to their occurring times in conventional approaches. However, in a system where the global state is decomposed and monitored by many distributed entities, the global event list also has to be decomposed and monitored by the respective entities. This approach can reduce the size of the list and thus the sorting time. Meanwhile, the correct simulation time should be maintained carefully since the event lists are distributed and the execution of one event may cause event changes at different entities. The analysis of the event list in each component is given in section 4.4.4.1 and the mechanism to maintain correct simulation time is presented in section 4.4.4.2.

4.3.4.1 Analysis of event lists

Each entity maintains an event list although only machines, jobs and the job releaser are the initiators of events. Other types of components only receive events from their entities supervising them and keep only the earliest ones in their own event lists.

The event list of a machine can contain at most three possible types of events: machine breakdowns, machine ups and leaving jobs. Its size can be at most two since machine breakdown and machine up events cannot co-exist. The event list of a job entity is a one-item list containing one incoming job event. Similarly, the job releaser also has a one-item list containing one new job event.

A workcenter entity keeps only the earliest events from its supervised machines; the job shop entity in turn keeps only the earliest events from all the workcenters and traveling jobs. The controller is at the top of the hierarchy and it decides the earliest event time for the next simulation round.

4.3.4.2 Mechanism to maintain correct simulation times

The mechanism to maintain correct simulation times is illustrated in Fig. 4.12, where "ev" stands for "event".

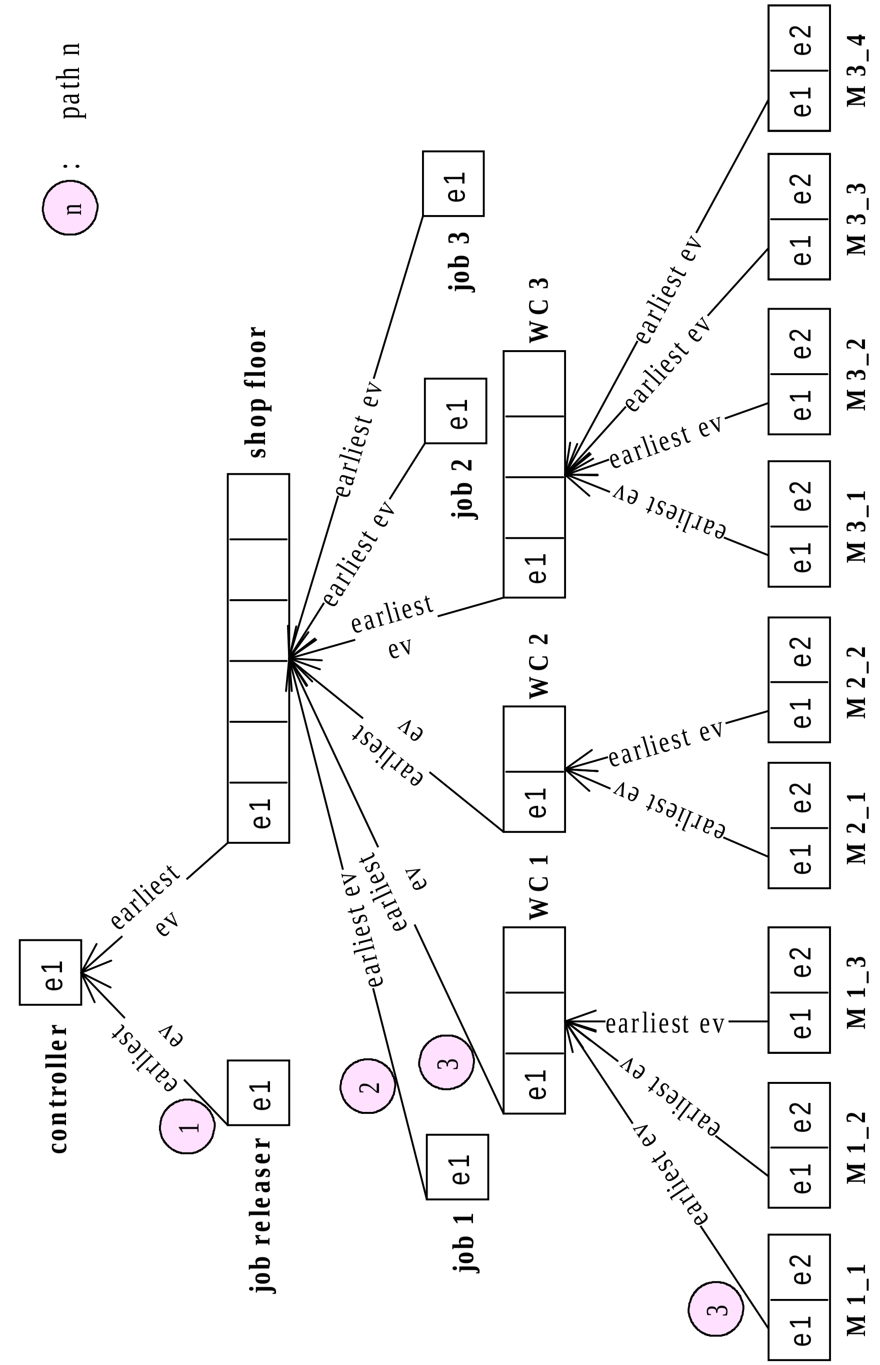


Fig. 4.12. The hierarchy of event lists

Each entity forwards its earliest events up to its respective supervisor and finally the earliest events reach the controller. There are three propagating paths: the first one is from the job releaser directly to the controller, the second one starts from the traveling jobs to the shop floor, then the controller, and the third one starts from machines, goes through the workcenters and the shop floor and finally reaches the controller.

An example for maintaining the event lists in a typical simulation round is given as follows. Workcenter 1 (WC1) has three machines, m1_1, m1_2 and m1_3. Each of them forwards its earliest event to WC1. WC1 compares the three events, identifies the earliest one and keeps it in its event list. The same procedure happens concurrently at workcenter 2 and workcenter 3. Three workcenters forward their earliest events to the shop floor, which at the same time, also keeps the events of the traveling jobs. Hence the earliest event that will occur on the whole shop floor can be found and further forwarded to the controller, which also receives the event of generating the next job from the job releaser. The controller then finds the earliest event and announces the occurring time as the next simulation time to both the job releaser and the shop floor. The shop floor forwards the new time to all the workcenters, which pass down to their machines. Each entity checks its own event list upon receiving the new time and starts to act if there are some due events; otherwise, it takes no action. It is obvious that there could be many concurrent events occurring at the different locations. The detailed messages for coordinating those single or concurrent events are illustrated in section 4.6.

It can also be seen that the size of a job shop event list is bound to the sum of both the sizes of the workcenters and the traveling jobs. In addition, the size of a workcenter event list is bound to the size of its machines. An overall long event list is thus avoided and the list-sorting time is reduced.

4.4 Implementing the simulated generic job shop as an MAS

The implementation of agent-based simulation essentially includes two steps: 1) identifying the behaviors of each individual agent, and 2) coordinating the communication among agents. The behaviors of agent and the change of its status can be expressed clearly in state charts while the coordination of communication can be illustrated in the sequential diagrams of unified modeling language (UML).

All entities of a generic job shop are modeled as autonomous agents pursuing their own interests with unique functions. The possible stable states for the main agents have been identified in section 4.4.2 and the transition between them in real time is described using UML state charts. Some transient states or actions, such as data recording, list sorting and message sending are also included in the state charts for a better illustration; the stable states are shaded. It should be noted that in the given state charts, “mg” refers to “message” and symbol C refers to a conditional gate. Finally, the mechanism of fitting an MAS to a time frame decided by DES is described.

4.4.1 Main agents

The state chart of a job agent, illustrated in Fig. 4.13, shows three stable states: 1) *waiting for process*, 2) *in processing*, and 3) *on traveling*, and four transient states: 1) *idle,* 2) *entering shop floor*, 3) *entering workcenter* and 4) *leaving shop floor*. The life cycle of a job agent involves the stable and the transient states. It starts in the *idle* state and changes to the *on traveling* state after entering the shop floor. It turns to either the *waiting for process* state or the *in process* state after entering a workcenter depending on the current state of the workcenter. If it is in the *waiting for process* state and receives an “available machine” message from its workcenter, the job will be processed on the assigned machine and its new state will be *in process*. It remains in this state until it receives either a “machine breakdown” or a “finish operation” message. It will go back to the *waiting for process* or the *in process* state if the former is received. Otherwise, its operation will be finished; it turns to the *on traveling* state, and moves to the next workcenter or the shipping bay if all the operations are completed.

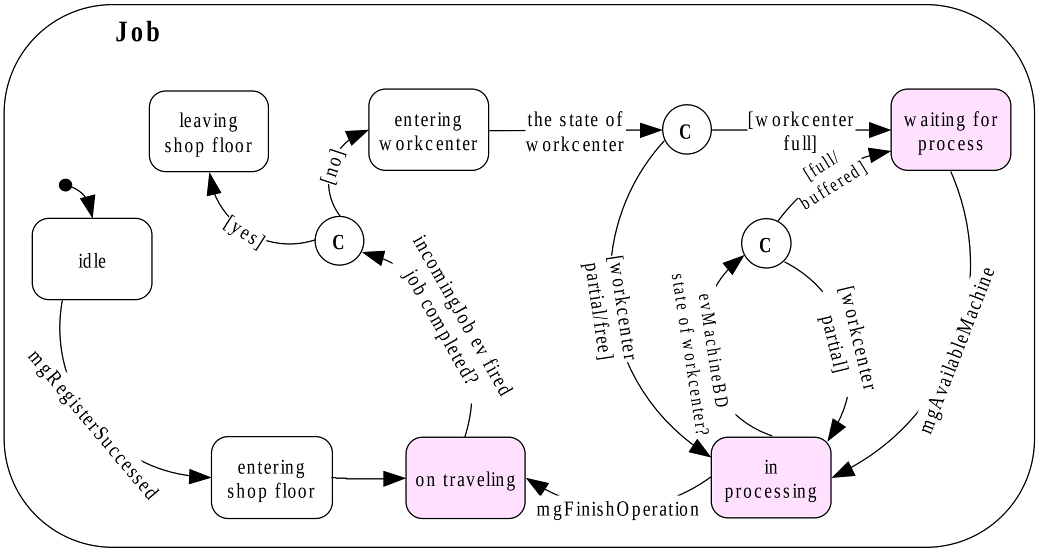


Fig. 4.13 State chart of a job agent

The flow time, waiting time and actual processing time of a job can be tracked by its own job agent, which can record the times it reaches or exits the shop floor, workcenter buffers, or machines. The state changes of the machine agent, workcenter agent, and shop floor agent are illustrated in figures 4.14 to 4.16.

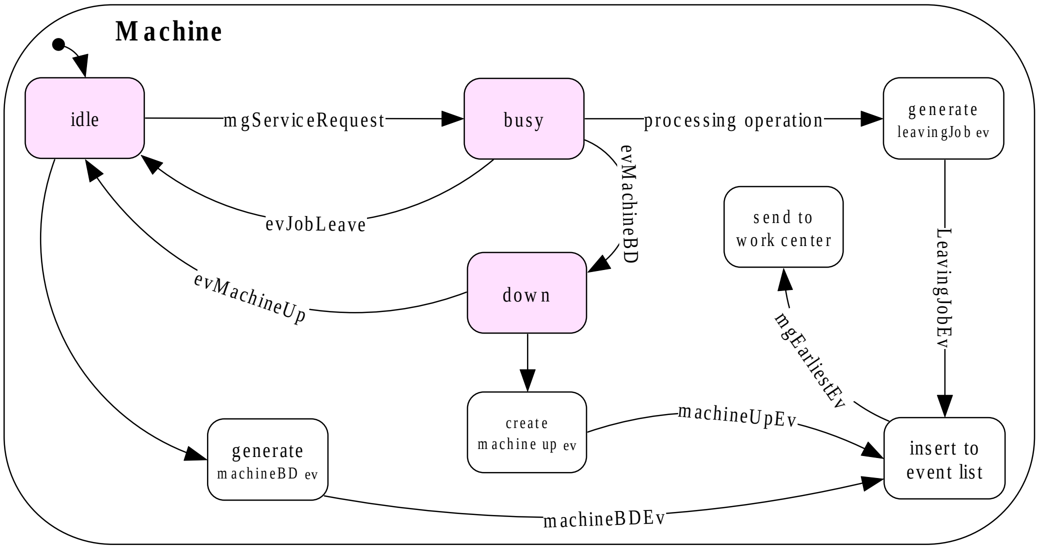


Fig. 4.14. State chart of a machine agent

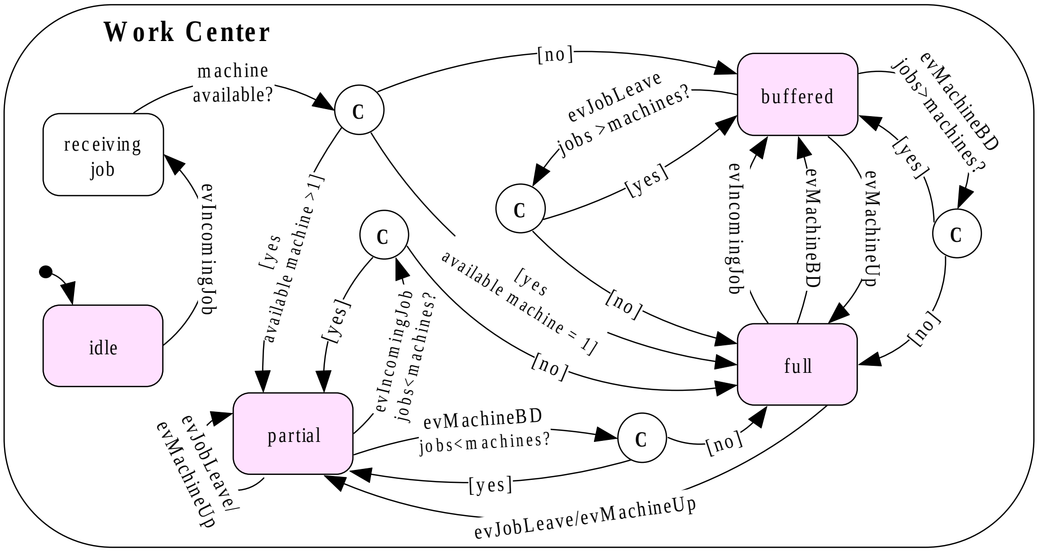


Fig. 4.15. State chart of a workcenter agent

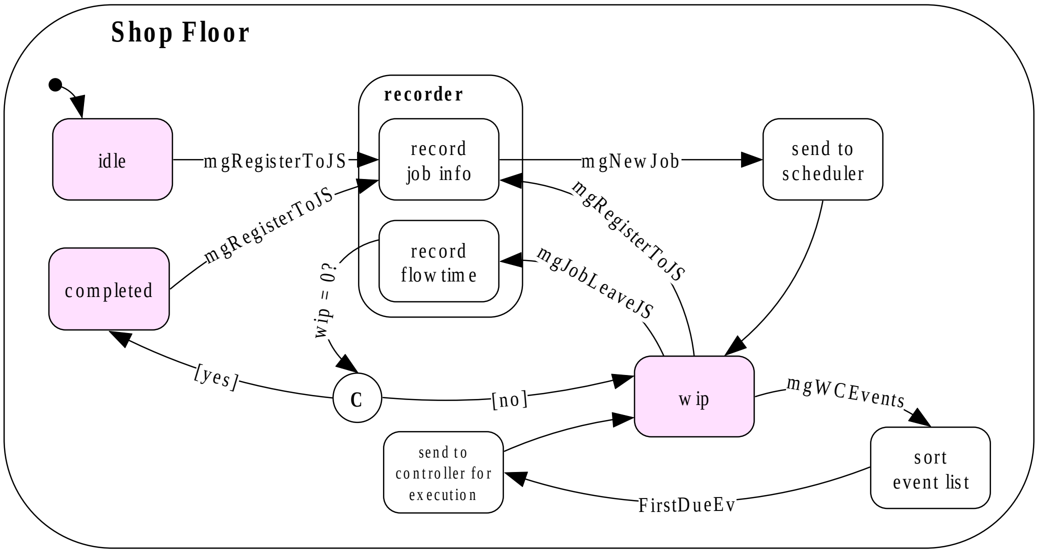


Fig. 4.16. State chart of a job shop agent

4.4.2 Other agents

The controller and the job releaser in Fig. 4.3 are also implemented as agents. The controller works to initiate the whole system and maintains the simulation clock and the job releaser generates new jobs with particular information concerning technical sequence, processing times, starting and due times, etc.

4.4.3 Fitting the MAS into the time frame of DES

There are two types of time in the system: simulation time and execution time. The simulation time is a clock time when an event starts to be executed. It is decided by DES. The execution time refers to the period of CPU time MAS takes for event execution. Their relationship is illustrated in Fig. 4.17.

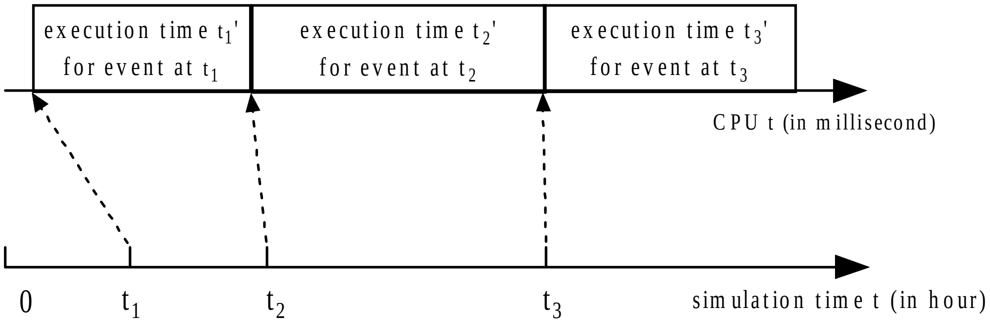


Fig. 4.17. The relationship between simulation time and execution time

The event occurred at time t_1 causes MAS to execute taking t_1' CPU time. Then the next simulation time t_2 , maybe hours after, is decided at the end of execution time t_1'. Another period of event execution then starts. The simulation proceeds in this way from t_1 to t_2 and t_3 until a predefined termination time is reached while the events are executed one after another by the agents.

4.5 Communication in the MAS

All the communication in an MAS is realized through message passing. The execution of an event always incurs a string of messages propagating to the other agents, which may react to the messages by further sending messages to other agents. Messages may be passed concurrently in many distributed locations, and it is crucial to coordinate them so that all event lists can be updated in a consistent manner and the correct simulation time can be maintained. Message passing for a single event is analyzed in section 4.6.1 and that for concurrent events in a single agent is described in section 4.6.2. The mechanism to coordinate all agents is given in section 4.6.3.

4.5.1 Message passing for a single event

- Message passing for job related events

The message passing for two job-related events, *i.e.* the new job event and the incoming job event, is illustrated in Fig. 4.18.

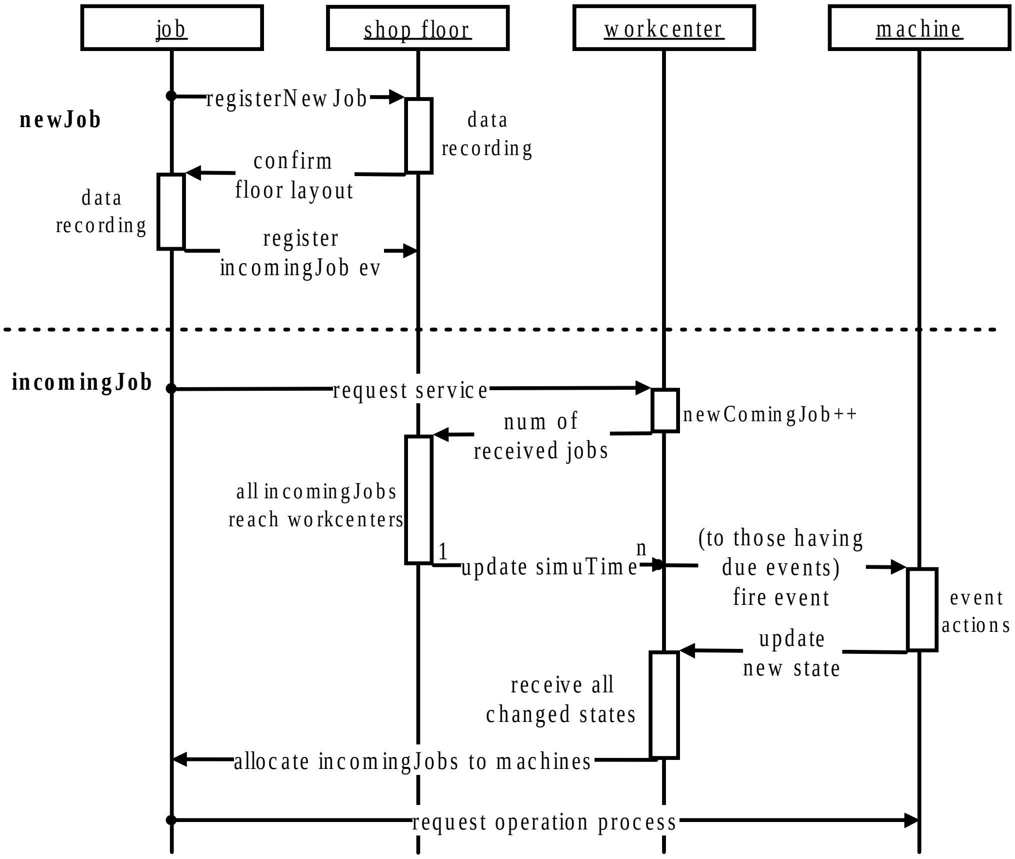


Fig. 4.18. Message passing for job-related events

- Message passing for machine related events

Message passing for three machine-related events, *i.e.*, the leaving job, the machine breakdown, and the machine up event, is illustrated in Fig. 4.19.

The machine agent sends a message to the job agent representing the job processed on the machine when a leaving job event is fired and notifies it on completion of its operation. The job agent then requests to leave the workcenter while the machine updates the workcenter about its new state. The workcenter then checks whether there are waiting jobs or interrupted jobs to be re-allocated accordingly. Finally, it permits the job agent to leave the workcenter. The job agent in turn registers with the shop floor agent with another incoming job event as shown in Fig. 4.19(a).

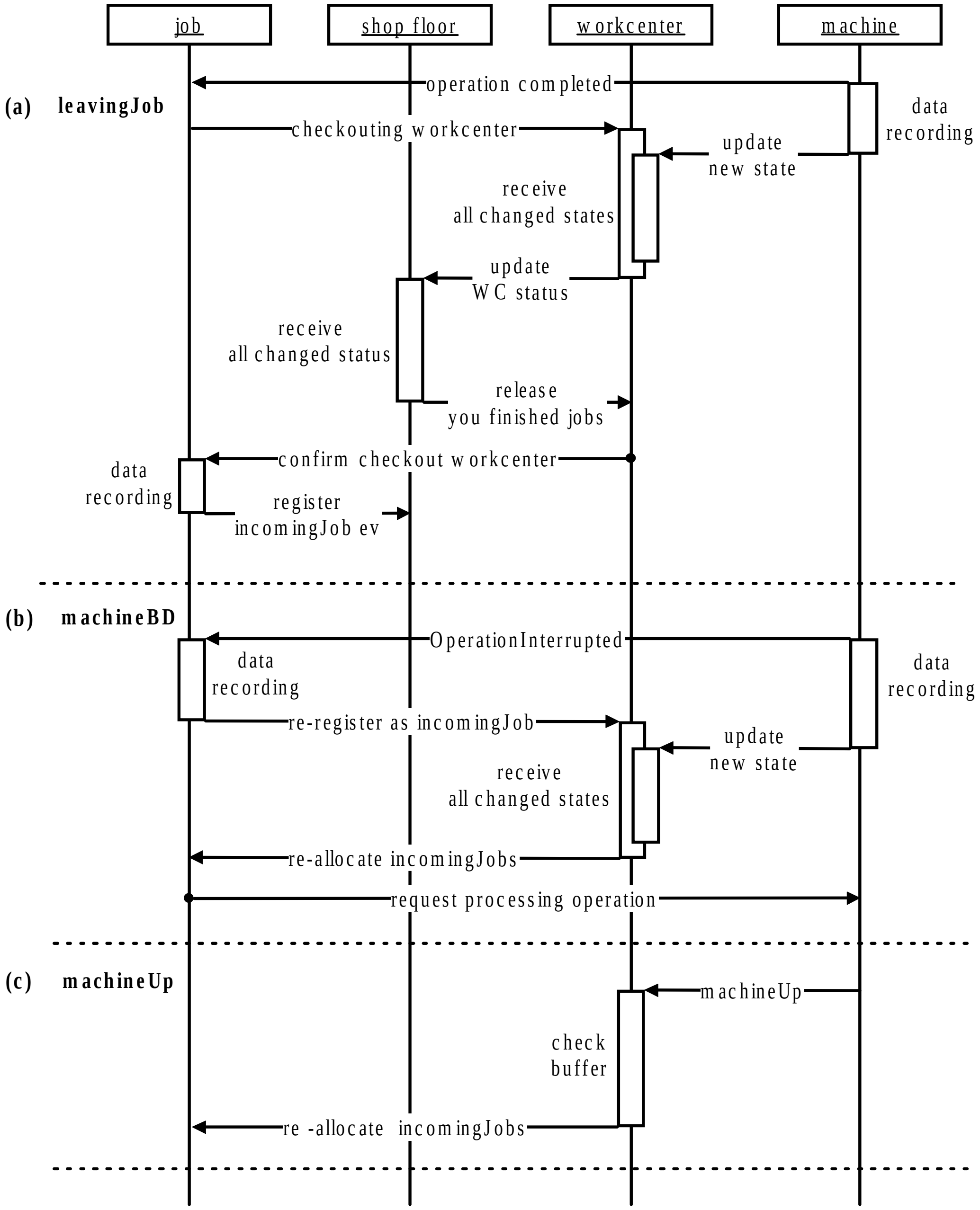


Fig. 4.19. Message passing for machine-related events

The broken down machine sends a message to its job agent announcing an interruption when a machine breakdown event is fired. The job then re-registers itself with the workcenter as a waiting job while the machine updates the workcenter about its new state. Finally, the workcenter re-allocates the interrupted job according to its new state and sends the job agent a “waiting” or a “processing” message. The job then acts accordingly as described above in Fig. 4.19(b).

The machine will notify the workcenter about its new state when a machine up event is fired and the workcenter will check its buffer to see whether there are waiting jobs. If there are, an allocation message will be sent to the appropriate jobs from the workcenter, otherwise no messages will be generated. This procedure is shown in Fig. 4.19(c).

4.5.2 Message passing upon concurrent events in a single agent

It is possible that there could be several events initiated at the same moment within one agent. The message passing for possible concurrent events is described as follows. A job agent can only have an incoming job event in its event list and thus it has no concurrent events. A machine agent can have at most two concurrent events: a leaving job and a machine breakdown event. The finished job is leaving the machine and the machine’s state turns to be *down*. The messages incurred are depicted in Fig. 4.20.

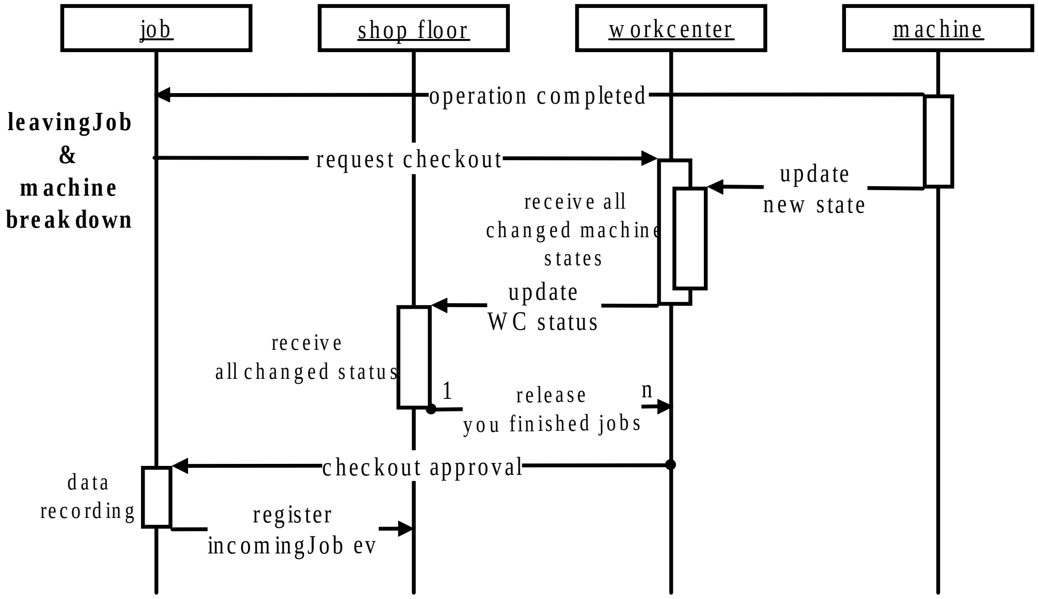


Fig. 4.20. Message passing upon concurrent events of machine breakdown and leaving job in a machine agent

The workcenter agents, the shop floor agent and the controller agent do not initiate or execute any events by themselves, but monitor the status of the agents and coordinate messages passing in their domains.

4.5.3 Agent co-ordination

The basic information flow in a simulation loop is illustrated in Fig. 4.21. It starts from sending all agents the current simulation time with messages 1 to 4. Agents from the lowest level then update their supervisors of their new status, after event actions, with messages 5 to 8. The messages contain the information on the time of their respective next events. Finally, the controller updates the simulation time to the earliest event time, and the next loop starts through messages 9 and 10.

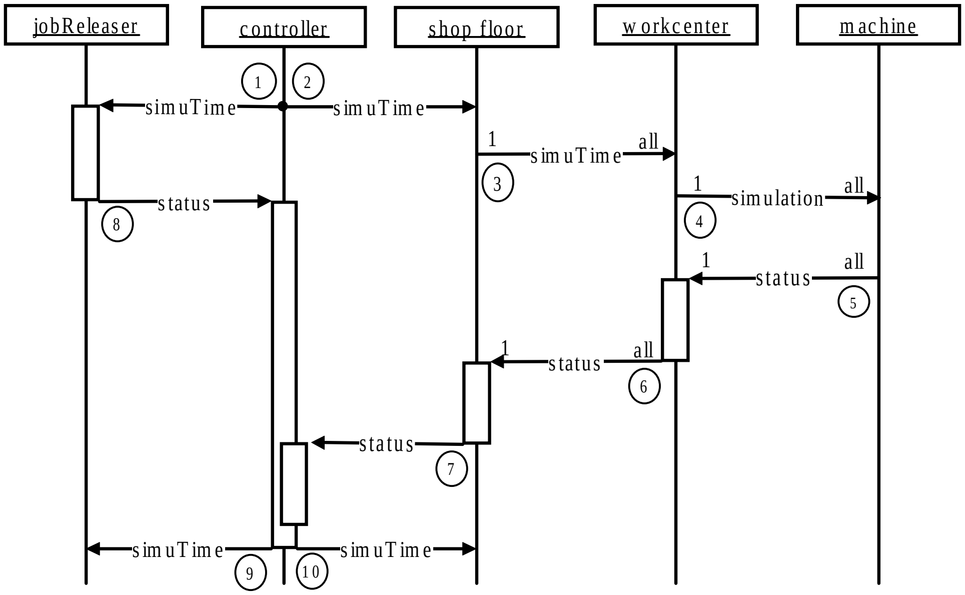


Fig. 4.21. The basic information flow in a simulation loop

A workcenter supervises several machine agents and is responsible for coordinating their messages to assure correct status updating. Thus, there are coordinating messages of a workcenter agent between messages 3 and 6. Similarly, a shop floor agent is responsible to coordinate workcenters through messages 2 to 7 in Fig. 4.21.

4.5.4 Coordination work of a workcenter

The workcenter receives a time message from the shop floor agent and then begins to coordinate all the actions in the workcenter. The most complex situation is when a workcenter has to receive new incoming jobs and all of its machines have simultaneous due events. The goals of a workcenter are thus to ensure that: 1) new incoming jobs are properly allocated, 2) the interrupted jobs are re-allocated, 3) waiting jobs are allocated when machines are available, 4) all machines update their new status, and 5) completed jobs leave the workcenter. A workcenter can only update its new status to its supervisor after all the above goals are realized.

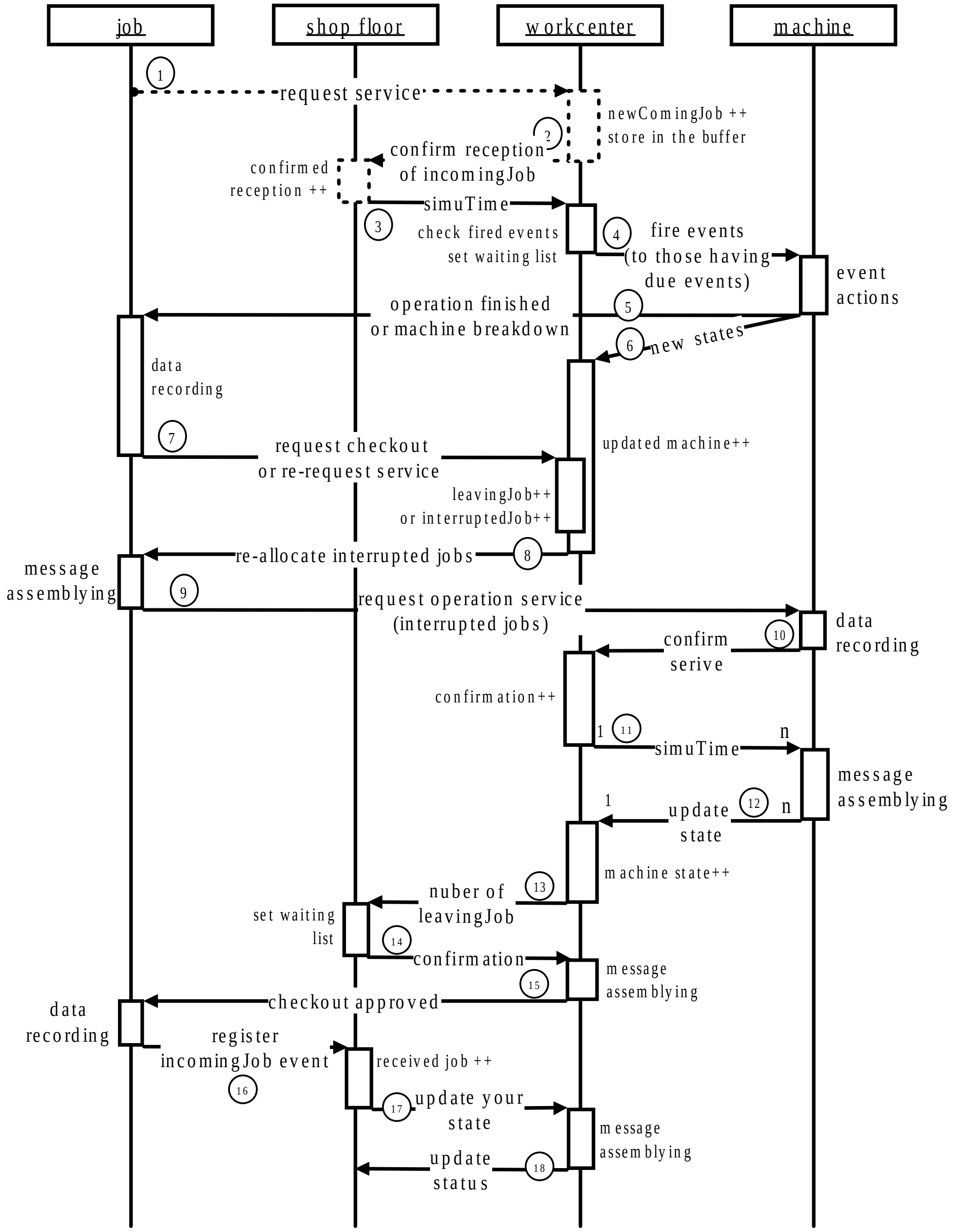


Fig. 4.22. Co-ordination work of a workcenter agent

The co-ordination messages are illustrated in Fig. 4.22 based on the most complex situation mentioned above. A workcenter receives the new incoming jobs through messages 1 and 2 before it receives a time message from message 3. It then checks the event list to determine the number and the types of fired events and a waiting list can be set up accordingly. For example, the workcenter will expect to receive both a "leave" request from the job and a new state updating it from the machine if the event for finishing a job is initiated. The workcenter then initiates all the events by message 4.

The workcenter is contacted by all the expected machines and jobs through messages 6 and 7 after the event actions are finished. The workcenter may be unbalanced at this time with newly available machines and waiting jobs in the buffer. It then allocates the waiting jobs or re-allocates the interrupted jobs to the machines through messages 8 to 10. The simulation time is forwarded through message 11 to all the machines, which immediately update their status through message 12. Finally, the completed jobs are approved to leave the workcenter through messages 13 to 16, and a workcenter can update its new status through message 18.

4.5.5 Coordination work of the shop floor

The shop floor prepares to monitor all the dynamics at the moment it receives a time message from the controller agent. The most complex situation for a shop floor agent to co-ordinate is when the following dynamics occur simultaneously: 1) new jobs come to the shop floor, 2) some traveling jobs arrive at their workcenters, and 3) some jobs in workcenters completed their operation and are ready to travel to the next stage. The shop floor needs to ensure that: 1) all new jobs are registered, 2) traveling jobs are received by their workcenters, and 3) jobs leaving their workcenters reach the shop floor. Only after all the above dynamics have been handled, can the shop floor agent update its new status to the controller agent. The co-ordination messages are given in Fig. 4.23.

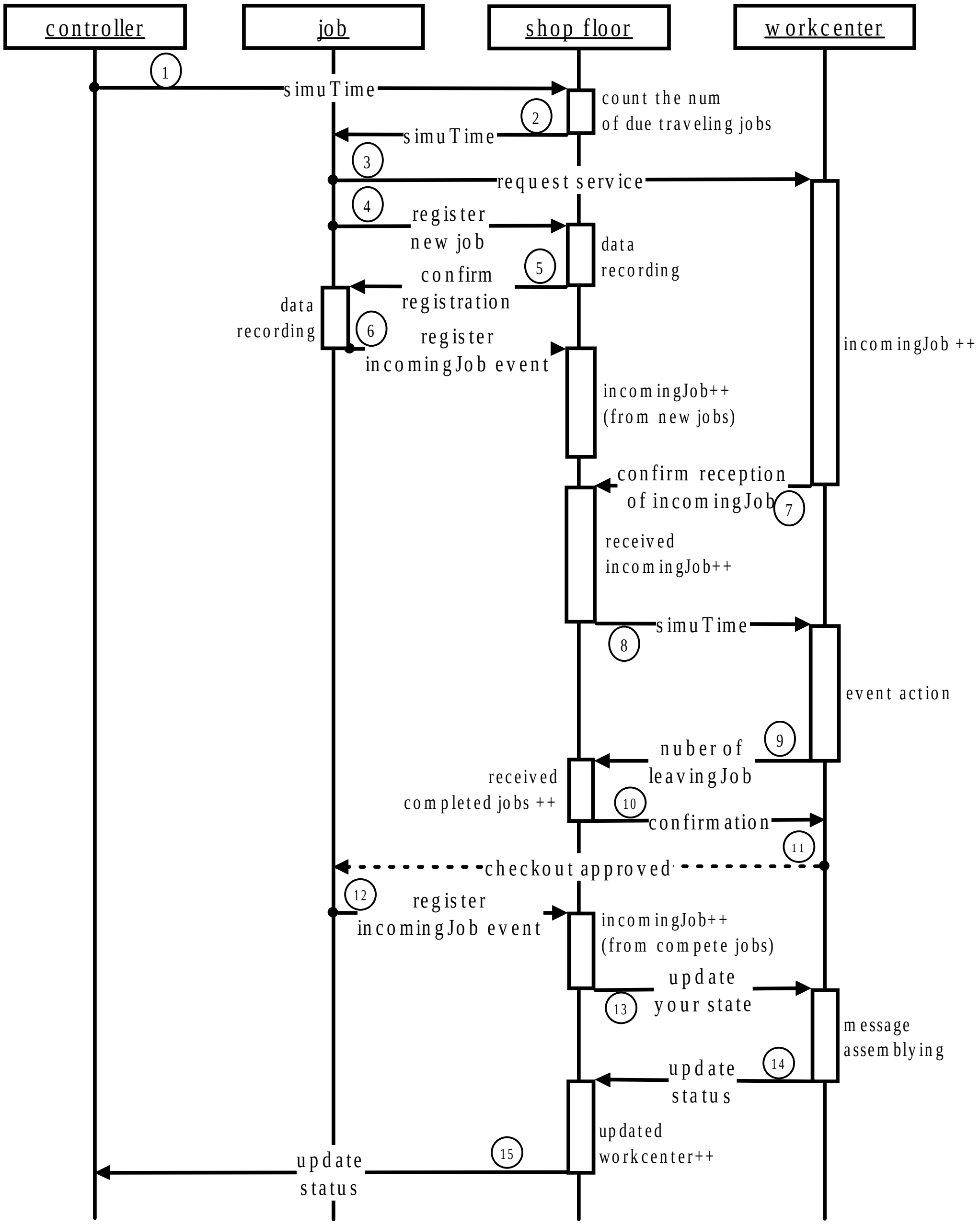


Fig. 4.23. Co-ordination work in the job shop agent

The shop floor agent also receives a time message through message 1 and initially makes sure that all the traveling jobs are received by their workcenters through messages 2, 3 and 7. It then updates all workcenters with the new simulation time through message 8. Meanwhile, there may be some new jobs entering the shop floor: they are handled through messages 4, 5 and 6. The shop floor agent is then notified of the number of leaving jobs by the workcenters through message 9 and starts to collect all the expected leaving jobs through message 12. It notifies all the workcenters to update their new states through message 13 after all the leaving jobs are collected. Finally, it updates its new status to the controller agent through message 14.

4.6 Case Study

The case study pursued here adopts the data from the example on pages 684-695 of Law and Kelton (2000). The MAS model runs on an AMD Opteron Linux Cluster with 26 nodes (2.2GHz, 4GB RAM) + 8 nodes (2.4GHz, 32GB RAM) in the Institute of High Performance Computing (IHPC). The random number generator used in simulation is proposed by L’Ecuyer *et al* (2001).

4.6.1 Inputs

The studied job shop is shown in Fig. 4.24 with five workcenters and one Receiving/Shipping station. The machines in a particular workcenter are identical while the machines in different stations are dissimilar. The distances between the six workcenters are given in Table 4.1. Jobs are transported between workcenters by MHDs assuming that there are sufficient number of them are available and the time spent on the trip is proportional to the distance between the two locations.

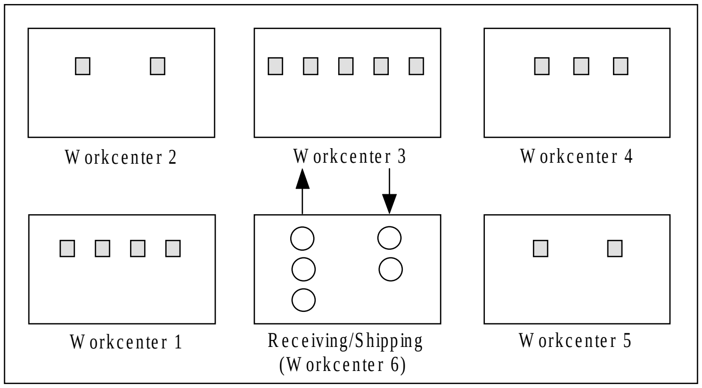


Fig. 4.24. Layout of the manufacturing system

Table 4.1. Distances between workcenters (feet)

Workcenter	1	2	3	4	5	6
1	0	150	213	336	300	150
2	150	0	150	300	336	213
3	213	150	0	150	213	150
4	336	300	150	0	150	213
5	300	336	213	150	0	150
6	150	213	150	213	150	0

Jobs arrive at the shop floor with inter-arrival times that are independent exponential random variables with a mean of 1/15 hour. There are three types of jobs: 1, 2 and 3, with respective probabilities 0.3, 0.5 and 0.2. Job types 1, 2 and 3 require 4, 3, and 5 operations to be done respectively, and each operation must be done at a specified workcenter in a prescribed routing (technical order), which is given in Table 4.2. Each job enters the shop floor at the Receiving/Shipping station (workcenter 6), travels to the workcenters on its routing and then leaves the system at the Receiving/Shipping station. All MHDs move at a constant speed of 5 feet per second.

A job joins a single FIFO buffer if all the machines in the workcenter it reaches are busy. The time to perform an operation at a particular machine is given in Table 4.3.

Table 4.2. Technical routes of jobs

Job type	Work stations in routing
1	3, 1, 2, 5
2	4, 1, 3
3	2, 5, 1, 4, 3

Table 4.3. Processing times of all operations

Job type	Mean service time for successive operations (hours)
1	0.25, 0.15, 0.10, 0.30
2	0.15, 0.20, 0.30
3	0.15, 0.10, 0.35, 0.20, 0.20

4.6.2 Simulation results

The simulation ran 10 replications of 920 hours length, which equals to 115 eight-hour days. The results for different performance measures are listed in Table 4.4. The first row shows the configuration of the job shop, which is comprised of five workcenters with four, two, five, three and two machines respectively. All performance measures except Maximum Number in Queue and Maximal Size of Working-in-Process are the average values of ten experiments.

Table 4.4. Simulation results

Number of machines: 4, 2, 5, 3, 2 Number of forklifts: enough Machine efficiency: 1					
performance measure	**1**	**2**	**3**	**4**	**5**
Proportion machines busy (workcenter)	0.806	0.450	0.795	0.570	0.825
Average number in queue (workcenter)	1.662	0.137	0.653	0.276	0.790
Maximum number in queue (workcenter)	35	9	12	10	17
Average daily throughput (shop floor)		120.075			
Average time in system (shop floor)		1.067			
Average total time in queues (shop floor)		0.240			
Maximal size of working-in-process (shop floor)		56			

4.6.3 Statistical calculation

A warming up period is first obtained using Welch's procedure [Law and Kelton, 2000] on 920 hourly throughputs in each of the 10 replications. The moving average $\overline{Y}_i(20)$ uses a window of 20, and is plotted in Fig. 4.25. A warming up period of $l = 120$ hours is obtained.

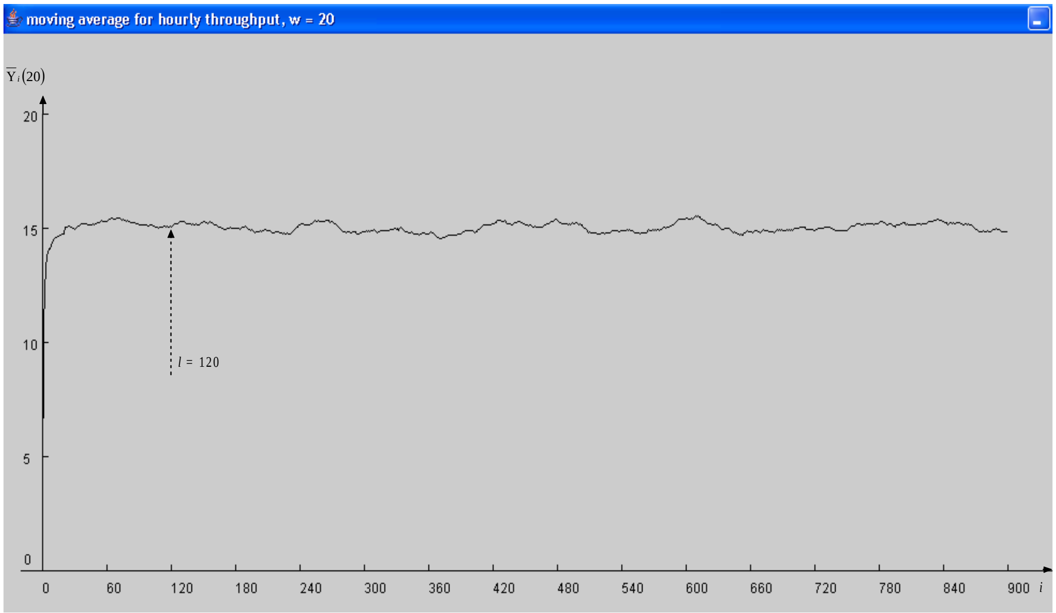


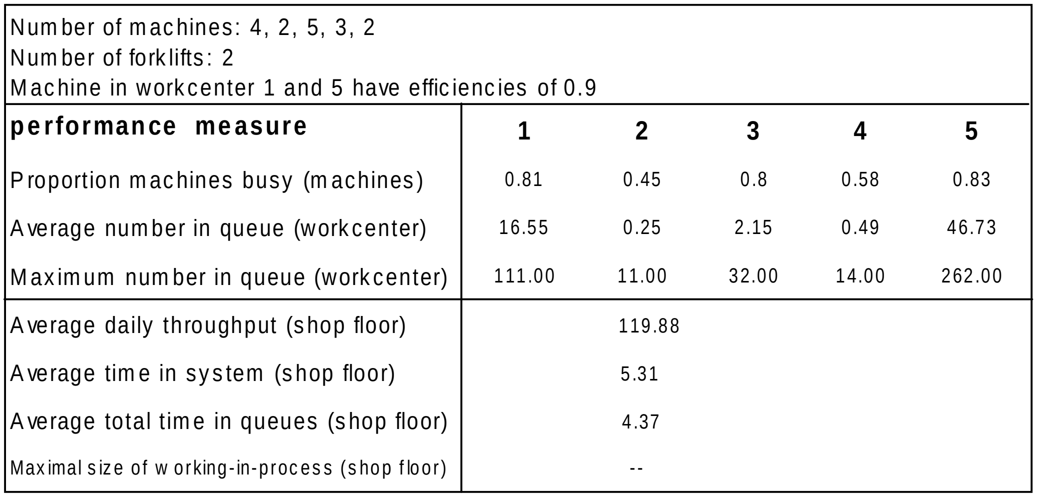
Fig. 4.25. Moving average of hourly throughputs

Then a 90 percent confidence interval for the steady-state mean daily throughput is constructed as $120.075 \pm t_{9,0.95}\sqrt{\dfrac{0.54}{10}}$ or 120.075 ± 0.23, which contains 120, which is the expected mean daily throughput.

4.6.4 Result analysis

The expected daily throughput is 120 jobs per 8-hour day, which is the maximum possible (because the inter-arrival times of jobs are independent exponential random variable with a mean of 1/15 hour). The 90% confidence interval built in the previous section demonstrates that the system can reach a daily throughput of 120 jobs.

Table 4.5. Simulation results from [Law and Kelton, 2000]



Number of machines: 4, 2, 5, 3, 2
Number of forklifts: 2
Machine in workcenter 1 and 5 have efficiencies of 0.9

performance measure	**1**	**2**	**3**	**4**	**5**
Proportion machines busy (machines)	0.81	0.45	0.8	0.58	0.83
Average number in queue (workcenter)	16.55	0.25	2.15	0.49	46.73
Maximum number in queue (workcenter)	111.00	11.00	32.00	14.00	262.00
Average daily throughput (shop floor)		119.88			
Average time in system (shop floor)		5.31			
Average total time in queues (shop floor)		4.37			
Maximal size of working-in-process (shop floor)		--			

The simulation results of a similar system built by Law and Kelton (2000) are listed in the Table 4.5 to be compared to the results in Table 4.4. Their system has more constraints such as limited MHD and machine efficiencies while the case study in this thesis assumes enough MHD and no machine breakdown. Both systems achieve 120 expected daily throughputs. This can be explained by the fact that Law and Kelton's system achieves the same level of proportion of busy machines despite of its limited resources. However, the limited resources cause both the average and the maximum number in the queues of Law and Kelton's system much larger than those in the current developed system. Subsequently, the average time of a job staying in the system is longer in Law and Kelton's system. Thus, both the statistical analysis and the comparison with the existing report have validated that the proposed DES-MAS system can correctly simulate a dynamic job shop.

The case study also takes the advantages of distributed data collection and calculation offered by MAS. All the data have been collected and maintained by their most related agents. For example, the average/maximum numbers of jobs in queues are collected by five workcenter agents and the shop floor agent keeps those data that are out of the scope of the other agents. Those data are the average daily throughput, average time in system, average total time in queues, the size work-in-process, etc. The information of machine utilization can be properly maintained by each machine itself. A workcenter can request the machine agents to provide such information when it needs to calculate the proportion of busy machines under its supervision. Thus the burden of a centralized computation can be naturally distributed to different computing entities.

4.7 Summary

An MAS simulating a real-life job shop is built in order to provide a test bed for studying approaches in a dynamic job shop environment. The essential architecture of a job shop manufacturing system is first identified, and then built as a DES, which can examine the performance of a system over a long period of time. The DES is implemented as an MAS so that the intelligent agents can be used to realize distributed computation and prompt reaction to dynamic events.

This approach requires careful coordination among event lists, which are distributed in different agents, in order to maintain a correct simulation time. The coordination involves communication among the agents. The agents in this model do not necessarily lose their autonomy. The discrete events set the time steps and the agents are autonomous within their event execution periods. In this way, a long-term performance of an MAS can be examined.

All the communication and state changes are clearly illustrated using UML sequential diagrams and state charts. A case study demonstrates the advantage of distributed data collection and analysis; it also validates the proposed system by statistical analysis and comparison to existing simulation results on a similar test case.

5 Scheduler Agent and ACO

In this chapter, the previous test bed is extended to include a scheduler which uses ACO to generate schedules. The ACO scheduler is modeled as an agent in section 5.1. The application of ACO for a dynamic JSSP and the procedure of dynamically updating the pheromone matrix are discussed in section 5.2. Finally, the implementation of ACO as an MAS is presented in section 5.3.

5.1 The scheduler agent

Implementing a scheduler agent in the MAS test bed implies not only additional coordination of the scheduler agents to the main existing agents like the job, job shop and workcenters, but also the coordination of the behaviours within the scheduler agent itself. However, a scheduler does not generate dynamic events and thus there is no change in the event management of the existing test bed.

5.1.1 Additional coordination related to the scheduler

The new agent, scheduler, can communicate with the job, shop floor and workcenter agents. A job agent contacts both the shop floor and the scheduler right after it has been generated by the job releaser agent. The scheduler agent then prepares to reschedule to include this new incoming job according to its states. A shop floor agent proactively requests the scheduler to update the schedule when necessary and suspends its actions. The scheduler then updates all the workcenters with new schedules. All workcenters confirm to the scheduler regarding to the reception of schedules; then the scheduler replies to the shop floor agent that its request has been fulfilled. At this time, the job shop resumes its work.

5.1.2 Coordination among behaviours in the scheduler agent

The scheduler agent can be in either one of two states: idle or searching. It is idle when all jobs are scheduled and the schedule is issued; otherwise, it is in a searching state. It should be able to receive new jobs and react to schedule requests anytime. These two abilities are supported by the two independent and concurrent behaviours: *receive a new job* (Fig. 5.1) and *receive schedule requests* (Fig. 5.2). The former behaviour is initiated by the arrival of a new job agent and the latter is initiated by the job shop agent. Meanwhile, solutions from the ant agents are collected through "*collect ant results*" behaviour (Fig. 5.3). The following sections present the flowcharts of those behaviours and the coordination among them.

5.1.2.1 Behaviour of receiving a new job

Fig. 5.1 presents the flowchart for the behaviour of receiving a new job, which triggers the rescheduling procedure of the scheduler when it comes to the shop floor at the reception/shipping section. The schedule should have been updated by the time a new job arrives at its first workcenter. This point of time is called ***expected due time of rescheduling*** and the operations scheduled before this moment by the previous schedule should not be considered in the new scheduling problem.

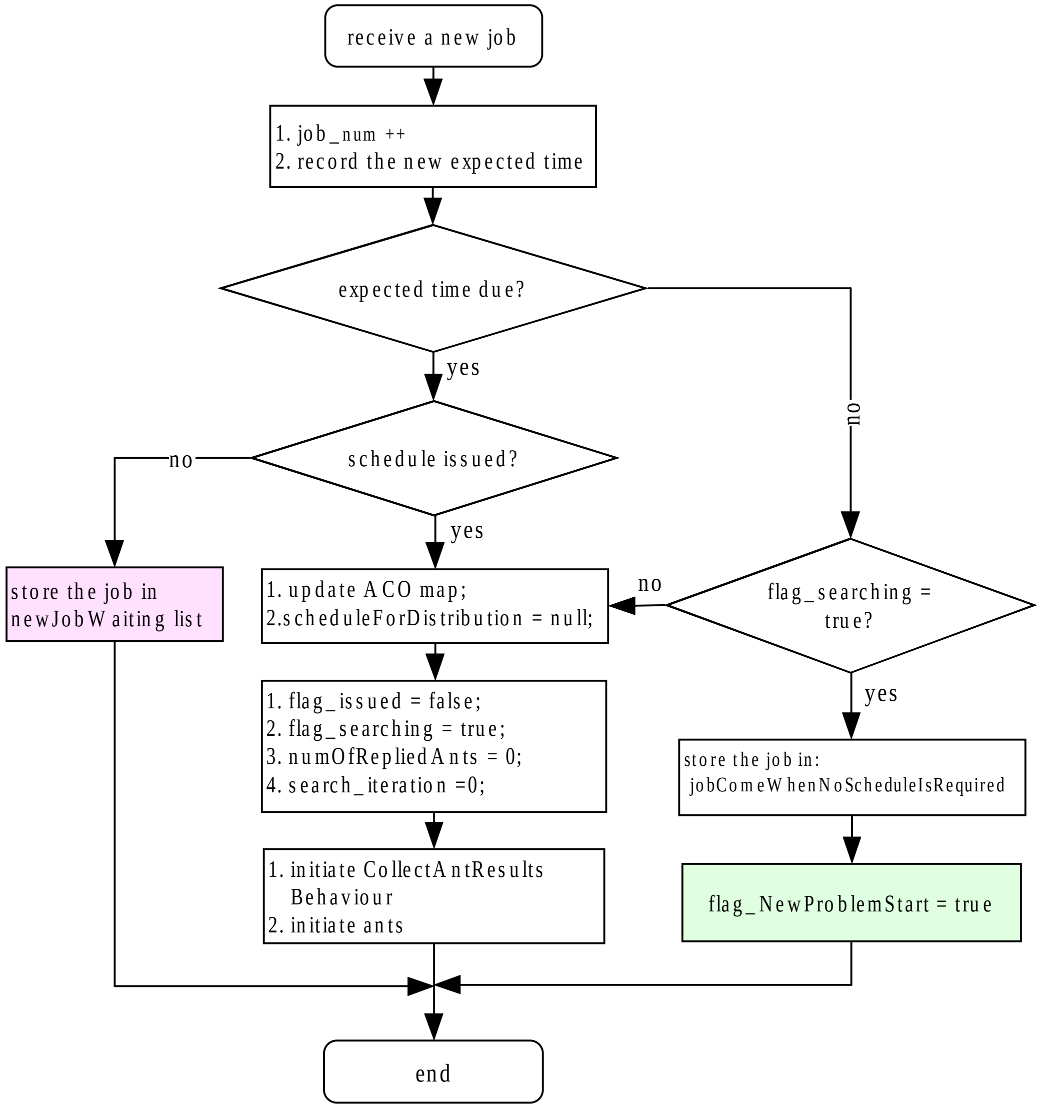

receive a new job
1. job_num ++
2. record the new expected time
expected time due?
yes
no
schedule issued?
no
yes
store the job in newJobWaiting list
1. update ACO map;
2.scheduleForDistribution = null;
no
flag_searching = true?
yes
1. flag_issued = false;
2. flag_searching = true;
3. numOfRepliedAnts = 0;
4. search_iteration =0;
store the job in: jobComeWhenNoScheduleIsRequired
1. initiate CollectAntResults Behaviour
2. initiate ants
flag_NewProblemStart = true
end

Fig. 5.1. The behaviour of receiving a new job in the scheduler agent

Upon receiving a new job, the scheduler agent increases the number of its jobs by one, records the new expected due time and checks whether a previous schedule request, if any, is due. The new job should be stored temporarily in the list called *jobComeWhenNoScheduleIsRequired* if it has not reached the expected time for releasing a schedule and the scheduler is seeking a schedule. A flag named *flag_NewProblemStart* is then raised and marked in green color in Fig. 5.1. It will be handled in Fig. 5.3 in the location with the same color. This mechanism is to synchronize the actions between two concurrent behaviours: receiving a new job and collecting ant results. However, the scheduler starts seeking schedules if it is idle at the time a new job arrives.

A new job may come at the same time that a schedule is due to be issued. The scheduler should guarantee that the due schedule is issued before the new job is considered for rescheduling. If the schedule is not issued, the new job has to be stored temporarily in the list called *newJobWaiting* and the procedure is colored pink in Fig. 5.1. It will be included to generate a new schedule right after the due schedule is issued indicated in Fig. 5.3 in the procedure highlighted with the same color. This mechanism is to synchronize the two concurrent behaviours: receiving a new job and receiving a schedule request. Otherwise, the rescheduling procedure is executed immediately if the previous schedule is issued.

5.1.2.2 Behaviour of receiving a schedule request

Fig. 5.2 presents the flowchart of the behaviour of receiving a schedule request. The scheduler basically checks whether it is in the correct state of searching a schedule and raises a flag called *flag_waitForSchedule,* which is marked in yellow color, to wait for a schedule. Then the behaviour of collecting ant results can immediately dispatch a new schedule once it is ready. The procedure is indicated in the procedure marked in the same color in Fig. 5.3. This flag synchronizes the behaviours of requesting schedule and collecting ant results.

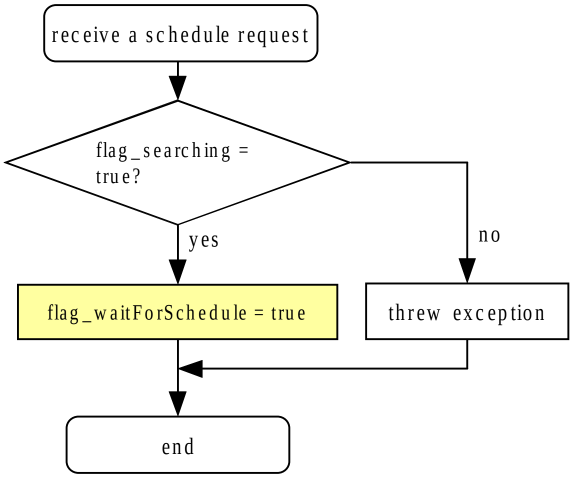


Fig. 5.2. The behaviour of the scheduler agent receiving a schedule request

5.1.2.3 Behaviour of collecting ant results

The main goal of this behaviour is to collect all ant results, update the best solution and the pheromone matrix, and initiate ants to search schedule for the next round of searching (Fig. 5.3). The behaviour checks the flag of new job coming (*flag_NewProblemStart*) when all the ant results have been collected. If it is raised, the record of the best solution is removed and the pheromone matrix/ACO map is updated. A new problem is then formed and rescheduling starts.

However, searching continues if the problem is not changed until the minimum number of iterations is met. At that time, the schedule agent checks whether a schedule request (*flag_waitForSchedule*) is waiting. It should dispatch schedules to all the workcenters if a request is made, otherwise, it will continue to search to find better solutions until a maximal number of iterations is reached. The list containing the waiting jobs (*newJobWaiting list*) is checked after the schedules are dispatched in order to synchronize the concurrency between a new job event and a schedule request.

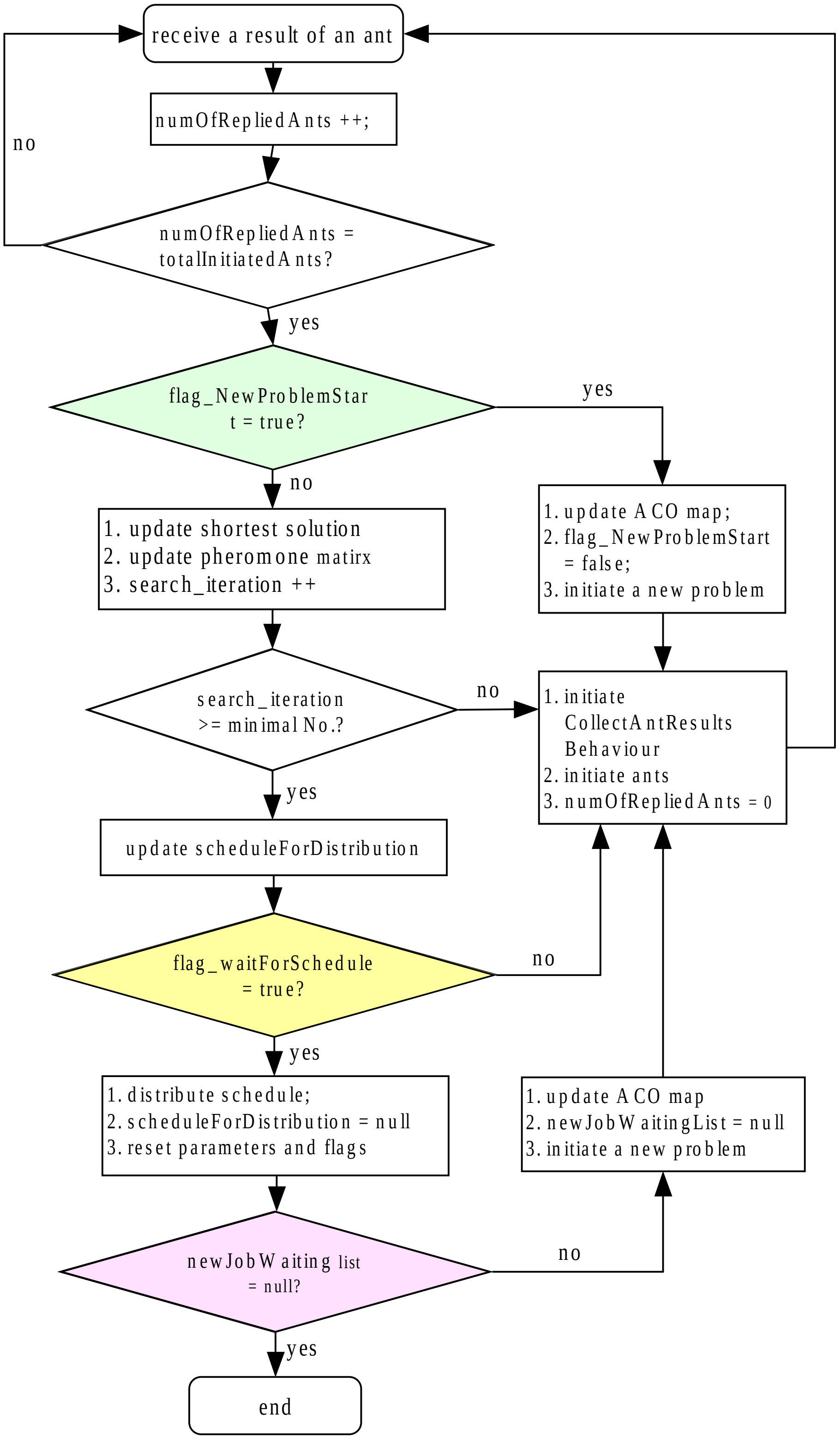

receive a result of an ant
numOfRepliedAnts ++;
numOfRepliedAnts = totalInitiatedAnts?
no
yes
flag_NewProblemStart = true?
yes
no
1. update shortest solution
2. update pheromone matirx
3. search_iteration ++
1. update ACO map;
2. flag_NewProblemStart = false;
3. initiate a new problem
search_iteration >= minimal No.?
no
yes
1. initiate CollectAntResults Behaviour
2. initiate ants
3. numOfRepliedAnts = 0
update scheduleForDistribution
flag_waitForSchedule = true?
no
yes
1. distribute schedule;
2. scheduleForDistribution = null
3. reset parameters and flags
1. update ACO map
2. newJobWaitingList = null
3. initiate a new problem
newJobWaiting list = null?
no
yes
end

Fig. 5.3. The behaviour of collecting ant results in the scheduler agent

5.2 ACO optimizer

In this section, the flowchart of the ACO algorithm is first illustrated; the representation of the JSSP as well as the application of ACO for dynamic JSSPs is then described; finally, the implementation of ACO as an MAS is described.

5.2.1 Notations

The notations used in the ACO algorithm are listed as follows.

h is the index of iteration number

p_{ij} is the probability for an ant to travel from node *i* to node *j* at h^{th} iteration

$\tau_{ij}(h)$ is the quantity of pheromone on the edge connecting nodes *i* and *j* at h^{th} iteration;

d_{ij} is the heuristic distance between nodes *i* and *j*;

ρ is the evaporation coefficient, which can be a real number between 0 and 1.0.

$\Delta\tau_{ij}(h)$ is the quantity of increased pheromone on the edge connecting nodes *i* and *j* at h^{th} iteration;

Q is a constant representing the total quality of pheromone on a route;

$f_{evaluation}(best\_so\_far)$ is the best value obtained so far optimizing the given objective.

5.2.2 ACO flowchart

The flowchart of the ACO algorithm is given in Fig. 5.4. The basic idea is to repetitively initiate a set of ants, which walk in a common environment (problem graph) comprised of all the operations in a JSSP. The operations are modeled as nodes in a graph, which is described in detail in Fig. 5.6. Each ant walks through all of those operations (nodes) one by one and thus forms a route, which can be interpreted as schedules and its length can represent the value of some performance measures like makespan, flowtime, or tardiness. The goal of each ant is to find a shortest route.

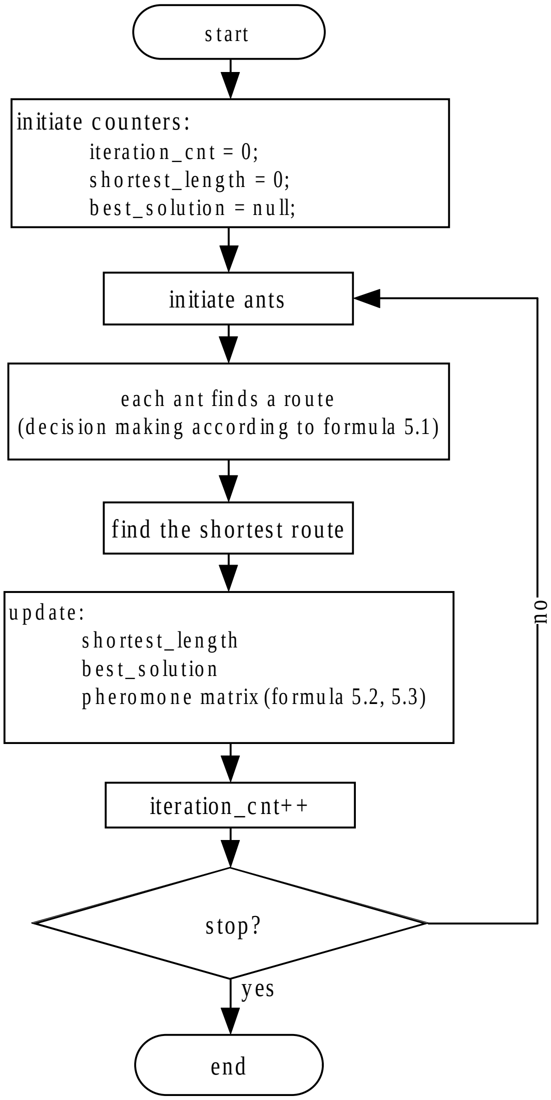


Fig. 5.4. The flow chart of the ACO algorithm

A walking ant leaves behind on its route some amount of pheromone, which changes the global environment. The probability for an ant to choose its next node is directed by both the amount of pheromone on the route and the distance from its current location to the targeted one. Ant i chooses the next node according to the State Transition Rule in formula (5.1) (Dorigo *et al*, 1996).

$$p_{ij}(h) = \frac{[\tau_{ij}(h)]^{\alpha} . \left[\frac{1}{d_{ij}}\right]^{\beta}}{\sum_{j \in allowed-nodes} [\tau_{ij}(h)]^{\alpha} . \left[\frac{1}{d_{ij}}\right]^{\beta}}$$

(5.1)

The heuristic distance d_{ij} in this study is the sum of traveling time between the current workcenter to the target workcenter and the processing time of the operation in the target workcenter. The environment is represented by a pheromone matrix, which is updated by the best solution at each iteration. The updating can be described in formulae (5.2) and (5.3) (Dorigo *et al*, 1996).

$$\tau_{ij}(h+1) = (1-\rho) \cdot \tau_{ij}(h) + \Delta\tau_{ij}(h+1)$$

(5.2)

$$\Delta\tau_{ij}(h+1) = \begin{cases} \dfrac{Q}{f_{evaluation}(best\_so\_far)} \\ 0, \quad otherwise \end{cases}$$

(5.3)

Pheromones on all edges evaporate at the rate of ρ so as to diversify the search procedure into larger solution spaces and jump out of local optima. The information of the best solution can be used to intensify certain search areas by strengthening the pheromones on all the edges of the best route by an amount of $\Delta\tau_{ij}(t+1)$ through formula (5.2).

The centralized actions include choosing and keeping the best solution, as well as deciding whether or not to continue solution seeking.

5.2.3 ACO for job shop scheduling problems

Each job in a classical JSSP is comprised of several operations to be processed on different machines. Generally, their technical orders and the processing times are represented in a technical matrix TM and a processing time matrix PM, respectively. Each row of TM indicates the order of machines that all the operations of one job will visit while each row of PM indicates the processing times that all those operations will take on their processing machines. Simple examples of these are given as follows.

$$TM = \begin{bmatrix} M1 & M2 & M3 \\ M3 & M1 & M2 \end{bmatrix} \qquad PM = \begin{bmatrix} t(O_{11}) & t(O_{12}) & t(O_{13}) \\ t(O_{21}) & t(O_{22}) & t(O_{23}) \end{bmatrix}$$

Fig. 5.5. The technical matrix TM and the processing matrix PM for a 2 x 3 JSSP

Fig. 5.5 presents a technical matrix and a processing matrix of a JSSP with two jobs and three machines. The first job has three operations O_{11}, O_{12}, and O_{13} that will be processed on machines M1, M2 and M3, in that order, and its three operations need processing times of $t(O_{11})$, $t(O_{12})$, and $t(O_{13})$ respectively.

The JSSP above can be represented as a graph (Fig. 5.6). Nodes 1 to 6 represent operations O_{11}, O_{12}, to O_{13}, and O_{21}, O_{22}, to O_{23}. They are connected by horizontal directional edges indicating the precedence constraints given in matrix TM. The bi-directional edges indicate no ordering constraints among those operations. Dummy nodes 0 and 7 representing the source and the sink of the graph are the starting and the ending points of routing. They are connected by directional edges to the first and the last operations of all jobs, respectively.

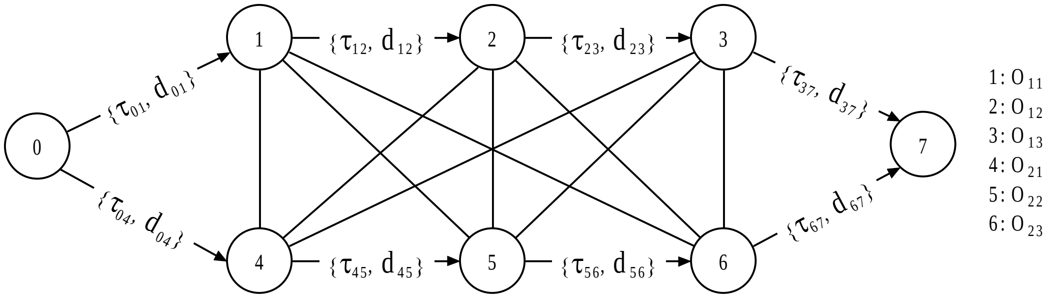


Fig. 5.6. The graph representing a 2 x 3 JSSP

Each edge is associated with a pair of values $\{\tau_{ij}, d_{ij}\}$, representing the amount of pheromone on it and the heuristic distance between the two nodes it connects. The value of d_{ij} can be easily looked up from matrix PM while the value for τ_{ij} should be found in the pheromone matrix, which is updated by the ants who found the best solutions (Fig. 5.6). An example of the pheromone matrix for the previous JSSP is shown in Fig. 5.7, which records the pheromone values of all the edges connecting every two nodes.

	N_0	N_1	N_2	N_3	N_4	N_5	N_6
N_0	0	0.1	0	0	0.1	0	0
N_1	0	0	0.16	0	0.18	0.19	0.20
N_2	0	0	0	0.18	0.19	0.20	0.21
N_3	0	0	0	0	0.20	0.21	0.22
N_4	0	0.16	0.15	0.14	0	0.22	0
N_5	0	0.17	0.16	0.15	0	0	0.24
N_6	0	0.18	0.17	0.16	0	0	0

Fig. 5.7. An example of the pheromone matrix for a 2 x 3 JSSP

The first row of Fig. 5.7 gives the pheromone values of the edges starting from node 0 to the other six nodes (The pheromones of edges that end at nodes 7 are not necessary to be included). Only $\tau_{01} = 0.1$and $\tau_{04} = 0.1$ exist since node 0 can only reach node 1 and node 4. Others are initiated to be 0. Similarly, the second row gives the pheromones of the edges starting from node 1. τ_{10}, τ_{11} and τ_{13} do not exist and are thus initiated as 0. The updating of the pheromone matrix takes the majority of the computation effort due to the dominant size of the pheromone matrix $(n \times m + 1)^2$, where n and m are the sizes of jobs and machines, respectively. As each ant walks through all the nodes in the matrix, the computational complexity is $O\left(s \times u \times (n \times m)^2\right)$, where s is the size of iterations and u is the number of ants per iteration.

Ant i cannot guarantee to find a feasible route for a JSSP before it is equipped with three lists: scheduled operation list (S_i), accessible operation list (A_i), and non-accessible operation list (NA_i). List S_i includes the nodes that are visited by ant i; A_i stores the currently accessible nodes; NA_i stores the rest of the unvisited nodes.

The size of S_i increases as ant i proceeds in the graph. Finally, the ordered nodes in list S_i form a complete route, which is a schedule for the JSSP.

5.2.4 ACO for job shop scheduling problem with parallel machines

It is assumed in a classical JSSP that there is only one machine in one workcenter. However, in the present studied problem, it is assumed that there can be an arbitrary number of machines in one workcenter. ACO demonstrates a good ability to be adjusted to this change if a list M_{ij} recording available times of all machines in workcenter j is maintained by ant i. For example, a $M_{23} = \{1.0, \quad 1.3, \quad 2.1\}$ represents the available times of all three machines in workcenter 3 kept by ant 2. Machine 1 is available from time 1.0; machine 2 is from time 1.3; and machine 3 is from time 2.1. M_{23} becomes $M_{23} = \{1.8, \quad 1.3, \quad 2.1\}$ after an operation with a processing time of 0.8 allocated to machine 1.

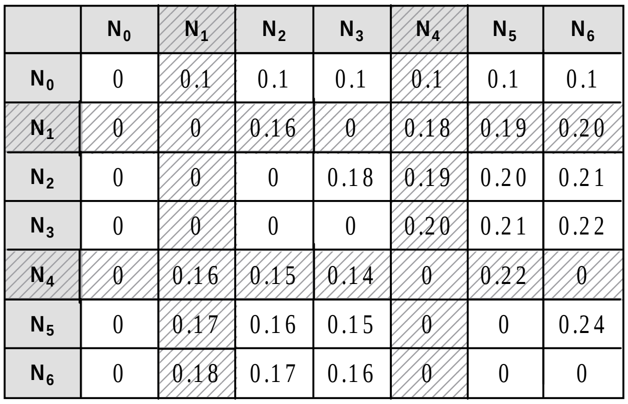
The rule to choose a machine among several available machines is based on the times that machines become available. In this study, the machine with the earliest available time has the highest priority to be chosen, assuming all the machines in one workcenter are identical. A random one will be chosen if several machines have the same earliest available times. This approach avoids the situation that some machines have been idle for too long.

5.2.5 ACO in a dynamic job shop scheduling environment

- Updating intermediate JSSP

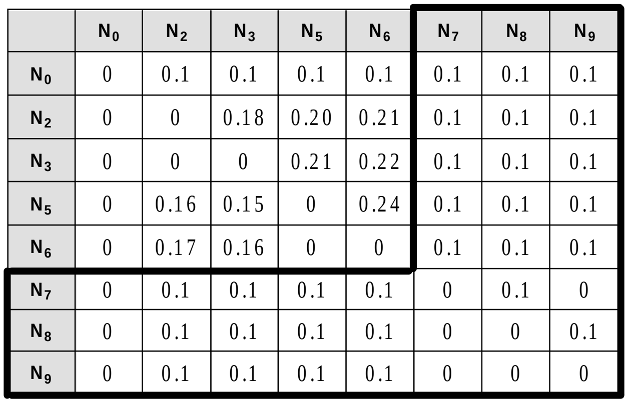
At each rescheduling moment, an intermediate JSSP has to be updated before the ACO algorithm can be executed through updating its pheromone matrix, which involves updating of nodes and pheromone values.

The updating of nodes in the pheromone matrix has two aspects: deleting the nodes that represent completed or processing operations and adding the nodes representing all the operations of the new job. For example, a new job with three operations O_{31}, O_{32} and O_{33} arrives at the job shop at the moment that node 1 is completed and node 4 is processing. The updating of nodes includes deleting all the cells related to node 1, as well as adding in the three new nodes (Fig. 5.8).



	N_0	N_1	N_2	N_3	N_4	N_5	N_6
N_0	0	0.1	0.1	0.1	0.1	0.1	0.1
N_1	0	0	0.16	0	0.18	0.19	0.20
N_2	0	0	0	0.18	0.19	0.20	0.21
N_3	0	0	0	0	0.20	0.21	0.22
N_4	0	0.16	0.15	0.14	0	0.22	0
N_5	0	0.17	0.16	0.15	0	0	0.24
N_6	0	0.18	0.17	0.16	0	0	0

(a) Deleting the cells related to nodes 1 and 4



	N_0	N_2	N_3	N_5	N_6	N_7	N_8	N_9
N_0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1
N_2	0	0	0.18	0.20	0.21	0.1	0.1	0.1
N_3	0	0	0	0.21	0.22	0.1	0.1	0.1
N_5	0	0.16	0.15	0	0.24	0.1	0.1	0.1
N_6	0	0.17	0.16	0	0	0.1	0.1	0.1
N_7	0	0.1	0.1	0.1	0.1	0	0.1	0
N_8	0	0.1	0.1	0.1	0.1	0	0	0.1
N_9	0	0.1	0.1	0.1	0.1	0	0	0

(b) Adding in three nodes 7, 8, 9

	N_0	N_1	N_2	N_3	N_4	N_5	N_6	N_7
N_0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1
N_1	0	0	0.18	0.20	0.21	0.1	0.1	0.1
N_2	0	0	0	0.21	0.22	0.1	0.1	0.1
N_3	0	0.16	0.15	0	0.24	0.1	0.1	0.1
N_4	0	0.17	0.16	0	0	0.1	0.1	0.1
N_5	0	0.1	0.1	0.1	0.1	0	0.1	0
N_6	0	0.1	0.1	0.1	0.1	0	0	0.1
N_7	0	0.1	0.1	0.1	0.1	0	0	0

(c) The updated pheromone matrix

Fig. 5.8. Update pheromone matrix

The cells related to node 1 include those from the whole third column and the third row while the cells related to node 4 include those from the whole sixth column and the sixth row. All of them are shaded in table (a) of Fig. 5.8 and need to be deleted. Three new nodes representing three operations of the new job are added to both ends of the row and the column surrounded by black borders in table (b); all the new cells are initiated with appropriate values. Finally, the nodes are re-numbered according to the updated order and a new pheromone matrix is generated in table (c).

- Parameters constrained in dynamic environment

Updating the pheromone values of the new pheromone matrix can be with or without an adaptation mechanism. In the former case, the pheromone values on all edges are re-initiated while in the latter case, only the new edges are initiated and the others remain unchanged. For example, the adaptation mechanism is presented in Fig. 5.8, where only new edges within the frame of table (b) are initiated and the others remain unchanged. In this way, some optimization information in the previous problem can be kept and a new schedule is sought based on it.

Given the computational complexity of $O\left(s \times u \times (n \times m)^2\right)$ for the ACO approach, increasing the values of the number of iterations (s) and the number of ants per iteration (u) increases both the solution quality and the computational time. Thus they are constrained in a dynamic environment where the computational timeslot for each intermediate JSSP is always limited.

The value of s can be a variable depending on the dynamism of the system in order to produce an intermediate schedule as good as possible. Thus, the minimal and maximal values of s are considered. The value of $\mathrm{s}_{\min}$ determines the minimal sets of ants that can be initiated. Its role is to guarantee a minimal computational timeslot for each intermediate JSSP. The value of $\mathrm{s}_{\max}$ determines the maximal sets of ants that can be initiated. Its role is to avoid over-enhancement of the pheromone values on some edges. A variable s within$[\mathrm{s}_{\min}, \mathrm{s}_{\max}]$ can improve the quality of an intermediate schedule as much as possible in the current test bed where the rescheduling procedure and the event of a new arrival job (*ev*) run independently on different computational threads; the rescheduling procedure is triggered only by *ev*. For example, if *ev* arrives before $\mathrm{s}_{\min}$ is satisfied in the previous intermediate JSSP, its execution will be delayed until $\mathrm{s}_{\min}$ is completed; otherwise, it can be immediately executed. Meanwhile, more iterations are allowed to initiate ants to improve the solution if the rescheduling procedure is not stopped by *ev* and s is not greater than $\mathrm{s}_{\max}$.

The values of u is also adjustable and its effects will be investigated in the experiments.

5.3 ACO implemented as an MAS

5.3.1 Implementation

ACO is inherently a distributed methodology which makes use of many individual and local procedures, and it is particularly well suited to parallelization. In this study, the ACO algorithm can be implemented as an MAS to take the advantage of parallel computation of distributed concurrent ants.

There are mainly two types of agents: environment and ant. The environment agent maintains the pheromone matrix; it initiates a set of ant agents and collects their solutions at each iteration; it also keeps the best solution and updates it during the scheduling procedure. Each ant agent seeks its own schedule independently, reports it to the environment agent, and finally kills itself and ceases its functions. The responsibilities of the environment agent in this study are fulfilled by several behaviours of the scheduler agent mentioned in section 5.1.

5.3.2 Functions of MAS in this study

Thus, in this thesis, the MAS works not only as a test bed to generate different experimental scenarios and analyze results but also as an approach to implement the ACO algorithm. A generic job shop simulated as a DES is further implemented as a MAS to be a test bed in order to systematically study the performance of control rules and algorithms in reactive scheduling under different environments. The test bed provides not only the basic entities simulating a shop floor and dynamic events, but also facilities the execution of schedules and measures the long term performance of the proposed approach for several criteria. The ACO algorithm is implemented as an

MAS taking the advantage of the concurrent computation of independent ants, which are modeled as agents. JADE is used to build the ACO algorithm as a pure MAS.

5.4 Summary

A scheduler agent using ACO as the optimizer is first combined with an existing MAS test bed to simulate the scheduling function in a real-world job shop. Next, the ACO algorithm, its application to JSSP, the representation of JSSP in a graph, and the procedure of dynamically updating the pheromone matrix have been explained. Meanwhile, the adaptation mechanism and two parameters, which are constrained in dynamic job shop environments, are also discussed. Finally, the implementation of ACO as an MAS and the functions of MAS in the current study are described.

6 Application of ACO for Dynamic Job Shop Scheduling Problems

In this chapter, ACO is applied to two dynamic job scheduling problems, which have the same mean total workload but different dynamic levels and disturbance severity. Its performances on these two problems are statistically analyzed and the effects of its adaptation mechanism are next studied. Furthermore, the effects of two important parameters in the ACO algorithm, namely the minimum number of iterations and the size of searching ants per iteration, which control the computational time and the solution quality of an intermediate scheduling problem, are also investigated. The results show that ACO can perform effectively in both cases; the adaptation mechanism can significantly improve the performance of ACO when disturbances are not severe; increasing the size of iterations and ants per iteration does not necessarily improve the overall performance of ACO.

6.1 Experimental design

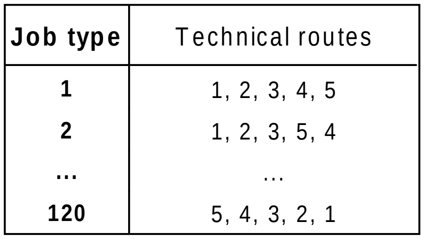
It is assumed that the reception of a new job will trigger a rescheduling procedure to find a full schedule with makespan as performance measure within the computational timeslot. The best-so-far schedule is then dispatched to be executed in all workcenters. The rescheduling procedure repeats until the preset stop criteria are met.

6.1.1 Experimental environments

- Problem configuration

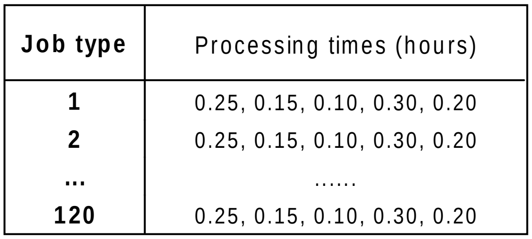
The dynamic job shop studied is shown in Fig. 4.24 with five workcenters and one receiving/shipping station. The numbers of machines in workcenters 1 to 5 are 4, 2, 5, 3, and 2, respectively. The machines in the same workcenter are assumed identical. The distances between all the workcenters are given in Table 4.1. Jobs are transported between workcenters by MHDs and the time spent on one trip is proportional to the distance between the two locations. All MHDs are assumed to be moving at a constant speed of 5 feet per second and they are assumed to be adequate.

New jobs arrive at the receiving/shipping station (workcenter 6) and travel among the workcenters according to their technical orders and finally leave the system from the receiving/shipping station. There are a total of $5! = 120$ types of jobs and each type of job occurs with a probability of $1/5!$ and the total processing time for each job is 1 hour. The technical routes and processing times of all operations of the jobs are given in Fig. 6.1.



Job type	Technical routes
1	1, 2, 3, 4, 5
2	1, 2, 3, 5, 4
...	...
120	5, 4, 3, 2, 1

(a) Technical routes



Job type	Processing times (hours)
1	0.25, 0.15, 0.10, 0.30, 0.20
2	0.25, 0.15, 0.10, 0.30, 0.20
...	......
120	0.25, 0.15, 0.10, 0.30, 0.20

(b) Processing times of all operations

Fig. 6.1. The technical routings and processing times of jobs

- ACO Parameters

The parameters of the ACO algorithm are $\alpha = 10.0$, $\beta = 10.0$, $\rho = 0.01$, $Q = 1.0$, and $\tau_0 = 0.5$ tuned by Zwaan and Marques (1999) to solve several JSSP benchmarks. They are adopted here as each intermediate JSSP is similar to those benchmarks.

It is assumed that the computation timeslot determined by $s_{\min}$ is within the time constraint in realistic applications. The following are the default values: $s_{\min} = 25$, $s_{\max} = 100$, and $u = 10$.

- Intermediate objectives

A dynamic JSSP is comprised of a series of intermediate JSSPs over time as mentioned in section 3.4. Thus the performance objective of those intermediate JSSPs has to be decided in order to yield the best total throughput, overall mean flow time, or overall mean tardiness. Three performance measures for those intermediate JSSPs are tested and they are the makespan, mean flowtime, and mean tardiness. Irrespective of the intermediate performance measure, evaluation values are recorded for all the three overall performance measures: total throughput, overall mean flowtime, and overall mean tardiness.

6.1.2 Experimental variables

Jobs arrive at the shop floor with inter-arrival times that are independent exponential random variables. The mean job inter-arrival time and the lot size are the two problem variables that decide the utilizations of workcenters. Two levels of job-arrival frequencies with the same mean size of total jobs are tested. In problem 1, jobs arrive one by one with the mean job inter-arrival time is nine jobs per hour. In problem 2, jobs are released in lots and arrive one lot per hour with nine jobs per lot. Jobs in one lot can be different types and will be processed job by job. In both problems, the type of a job is randomly decided so that each one of the 120 types has an equal chance to be chosen. Thus the mean total processing time demanded on each workcenter is the same.

The size of jobs in a lot determines the severity that an underlying scheduling problem is disturbed. For example, there are 16 unprocessed operations when a lot of new jobs are released to the shop floor. The size of operations for the new intermediate JSSP is 22 if there is only one job with 6 operations in the lot. The old operations take about 73% (16/22) of the total operations in the new problem. However, they take only 57% (16/28) of the total operations in the new problem if there is one additional job (also with 6 operations) in the lot. Obviously, the underlying problem is changed more severely by the larger lot with two jobs than the smaller one with one job.

The simulation for each problem runs five replications for 200 simulation hours (totally about 1800 jobs) and the warming-up time is 20 hours (about 180 jobs). Only steady-state performance is measured and the average values of five replications are listed for all the performance measures.

6.2 Computational results and analysis

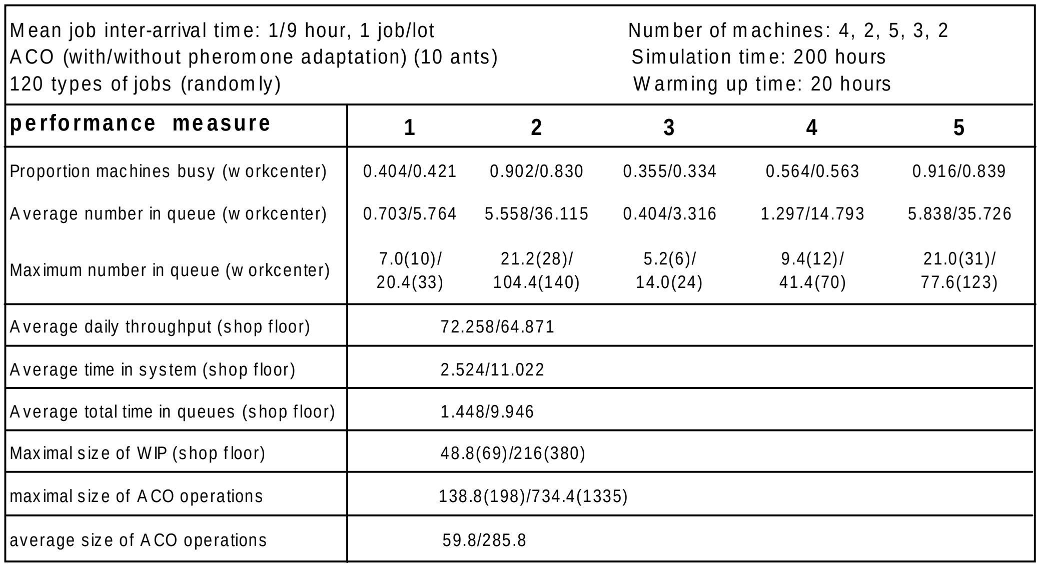
All the results are given in tables 6.1 to 6.6. Performance measures like the proportion of machine busy time, both the average and the maximum numbers of waiting jobs in queue are recorded by each workcenter agent while the average daily throughput, the average time in system, the average total time in queues, the maximum size of WIP are recorded by the shop floor agent. The maximum and the average sizes of operations in the scheduling procedure are recorded by the ACO scheduler agent.

Some general observations are as follows. Firstly, workcenters 2 and 5 are bottlenecks shown in all tables with utilizations of approximately 90%. Secondly, the machine utilization is inversely proportional to the number of the machines in its workcenter. The above two results are in accordance with the facts that the numbers of machines in both workcenters are the smallest with only 2 while having the same workload as other workcenters. Thirdly, the improvement in the average daily throughput and the machine utilization can reduce the average and maximum numbers of waiting jobs in a queue, the average time jobs spending in the system, the average total waiting times jobs spending in queues, the maximal size of WIP, and the maximal/average size of operations of intermediate problems, which reflects the overall performance of ACO as analyzed in section 3.4.1.2.

6.2.1 ACO performance analysis

The performances of ACO in two dynamic JSSPs are listed in tables 6.1 and 6.2. The 200 hourly throughputs of the five replications for both problems with the adaptation mechanism are plotted in figures 6.2 and 6.3 using moving average $\overline{\mathrm{Y}}_i(20)$ with a window of 20 (Law and Kelton, 2000) and a warming up period of $l = 20$ hours is obtained. Next, 90 percent confidence intervals for the steady-state mean daily throughputs of the two problems are constructed as $72.258 \pm t_{9,0.95}\sqrt{\frac{0.46}{5}}$ (or $[72.09, 72.43]$) for Problem 1 and $73.973 \pm t_{9,0.95}\sqrt{\frac{2.13}{5}}$ (or $[73.19, 74.75]$) for Problem 2.

Table 6.1. The effects of pheromone adaptation – Problem 1

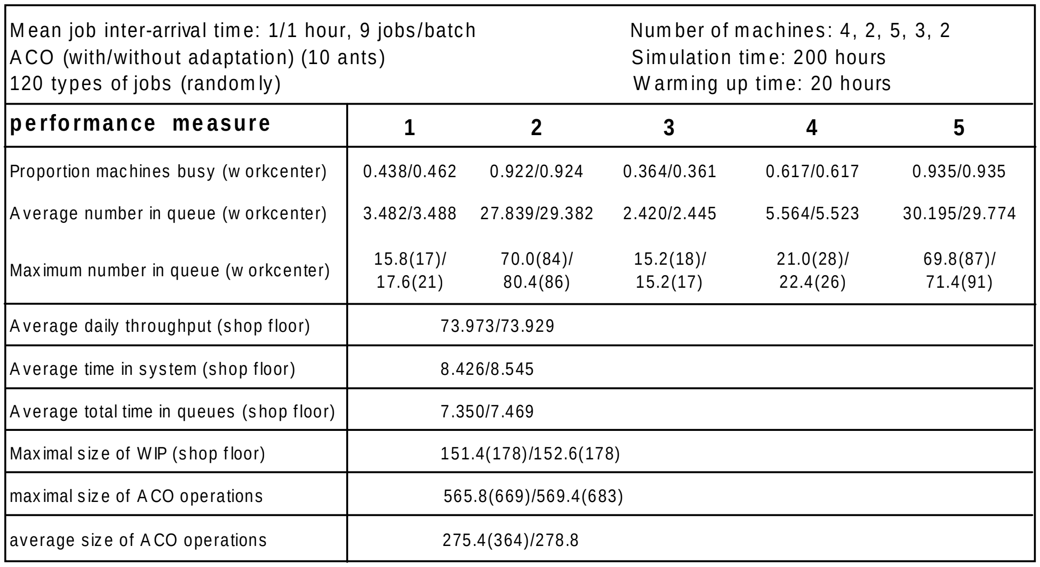


Mean job inter-arrival time: 1/9 hour, 1 job/lot
ACO (with/without pheromone adaptation) (10 ants)
120 types of jobs (randomly)

Number of machines: 4, 2, 5, 3, 2
Simulation time: 200 hours
Warming up time: 20 hours

performance measure	1	2	3	4	5
Proportion machines busy (workcenter)	0.404/0.421	0.902/0.830	0.355/0.334	0.564/0.563	0.916/0.839
Average number in queue (workcenter)	0.703/5.764	5.558/36.115	0.404/3.316	1.297/14.793	5.838/35.726
Maximum number in queue (workcenter)	7.0(10)/ 20.4(33)	21.2(28)/ 104.4(140)	5.2(6)/ 14.0(24)	9.4(12)/ 41.4(70)	21.0(31)/ 77.6(123)
Average daily throughput (shop floor)	72.258/64.871				
Average time in system (shop floor)	2.524/11.022				
Average total time in queues (shop floor)	1.448/9.946				
Maximal size of WIP (shop floor)	48.8(69)/216(380)				
maximal size of ACO operations	138.8(198)/734.4(1335)				
average size of ACO operations	59.8/285.8				

Table 6.2. The effects of pheromone adaptation – Problem 2



Mean job inter-arrival time: 1/1 hour, 9 jobs/batch
ACO (with/without adaptation) (10 ants)
120 types of jobs (randomly)

Number of machines: 4, 2, 5, 3, 2
Simulation time: 200 hours
Warming up time: 20 hours

performance measure	1	2	3	4	5
Proportion machines busy (workcenter)	0.438/0.462	0.922/0.924	0.364/0.361	0.617/0.617	0.935/0.935
Average number in queue (workcenter)	3.482/3.488	27.839/29.382	2.420/2.445	5.564/5.523	30.195/29.774
Maximum number in queue (workcenter)	15.8(17)/ 17.6(21)	70.0(84)/ 80.4(86)	15.2(18)/ 15.2(17)	21.0(28)/ 22.4(26)	69.8(87)/ 71.4(91)
Average daily throughput (shop floor)	73.973/73.929				
Average time in system (shop floor)	8.426/8.545				
Average total time in queues (shop floor)	7.350/7.469				
Maximal size of WIP (shop floor)	151.4(178)/152.6(178)				
maximal size of ACO operations	565.8(669)/569.4(683)				
average size of ACO operations	275.4(364)/278.8				

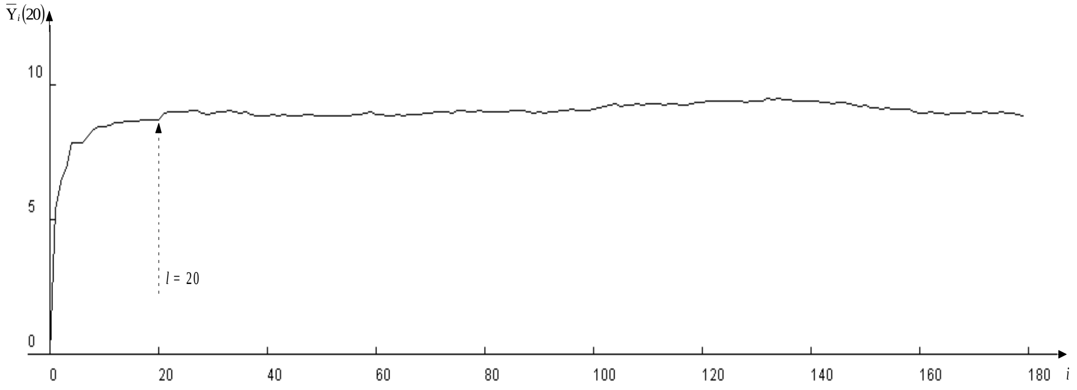


Fig. 6.2. Moving average of hourly throughputs of problem 1 with adaptation

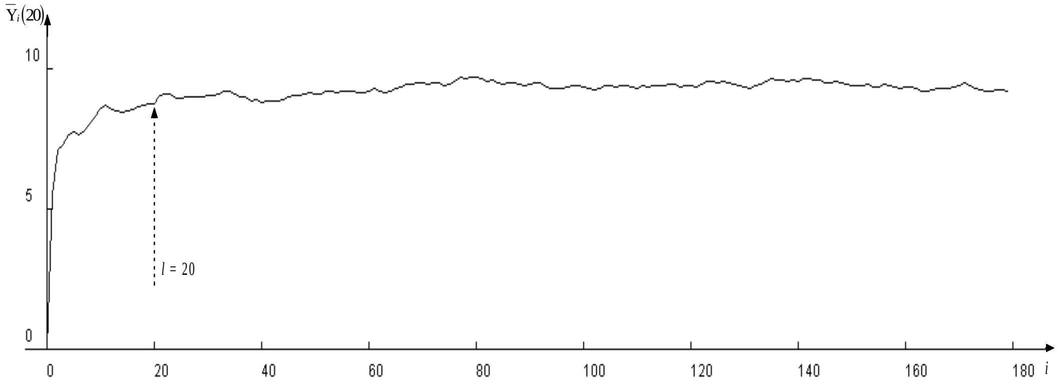


Fig. 6.3. Moving average of hourly throughputs of problem 2 with adaptation

The results show that ACO can perform well in both dynamic JSSPs to meet the expected daily throughput of 72 jobs as the mean inter-arrival time of jobs is 1/9 hour and there are 8 hours per day.

6.2.2 The effects of the ACO adaptation mechanism

The comparisons of ACO with/without adaptation in both problems are also listed in tables 6.1 and 6.2. The daily throughputs drop from 72.258 to 64.871 in Problem 1 (Table 6.1) and from 73.973 to 73.929 in Problem 2 (Table 6.2) when the adaptation mechanism is first applied and then removed. The change is significant in Problem 1 and minor in Problem 2.

The results indicate that the adaptation mechanism has greater effects in the situation where disturbances are not severe as in problem 1 and has little effect in the situation where disturbances are severe as in problem 2. The observation can be explained as follows. In problem 1, jobs arrive one by one and neighboring intermediate JSSPs are not severely different. A good solution can be found through the adaptation mechanism within a given computational timeslot. However, in problem 2, there would be not much difference between the pheromone matrices with and without the adaptation mechanism since the underlying problem can be dramatically changed by a large lot.

6.2.3 The effects of the number of minimal iterations

The results given in tables 6.3 and 6.4 show that increasing $s_{\min}$ deteriorates the performance of ACO in both problems, especially in problem 1 (72.258 for $s_{\min} = 25$ and 64.693 for $s_{\min} = 40$), when both problems adopt the adaptation mechanism. This seems to be against the initial expectation that increasing the number of minimal iterations can increase the optimality of an intermediate schedule and thus improve the overall performance of ACO.

This phenomenon could be explained as follows. The pheromone values of certain edges are increased too much as the result of increasing $s_{\min}$ and the initial amount of pheromone on the new edges introduced by new jobs becomes trivial. Thus the pheromone matrix fails to properly represent a new scheduling problem and is called too rigid to find a new good solution. Thus, the scheduler can only produce a worse intermediate schedule each time, especially in a highly dynamic environment where the computational time is limited.

Table 6.3. Increase the number of iterations – Problem 1

Mean job inter-arrival time: 1/9 hour, 1 job/lot ACO (s_{min} = 25/40 iterations) (10 ants) 120 types of jobs (randomly)			Number of machines: 4, 2, 5, 3, 2 Simulation time: 200 hours Warming up time: 20 hours		
performance measure	**1**	**2**	**3**	**4**	**5**
Machine utilization (workcenter)	0.404/0.419	0.902/0.826	0.355/0.332	0.564/0.560	0.916/0.835
Average number in queue (workcenter)	0.703/6.040	5.558/37.344	0.404/3.113	1.297/15.989	5.838/37.085
Maximum number in queue (workcenter)	7.0(10)/24.4(35)	21.2(28)/78.2(140)	5.2(6)/13.4(27)	9.4(12)/45.4(75)	21.0(31)/75.4(117)
Average daily throughput (shop floor)	72.258/64.693				
Average time in system (shop floor)	2.524/11.458				
Average total time in queues (shop floor)	1.448/10.382				
Maximal size of WIP (shop floor)	48.8(69)/222(383)				
maximal size of ACO operations	138.8(198)/778.2(1362)				
average size of ACO operations	59.8/300.2				

Table 6.4. Increase the number of iterations – Problem 2

Mean job inter-arrival time: 1/1 hour, 9 jobs/lot ACO (s_{min} = 25/40 iterations) (10 ants) 120 types of jobs (randomly)			Number of machines: 4, 2, 5, 3, 2 Simulation time: 200 hours Warming up time: 20 hours		
performance measure	**1**	**2**	**3**	**4**	**5**
Machine utilization (workcenter)	0.438/0.459	0.922/0.918	0.364/0.334	0.617/0.530	0.935/0.930
Average number in queue (workcenter)	3.482/4.640	27.839/31.667	2.420/3.436	5.564/8.295	30.195/32.961
Maximum number in queue (workcenter)	15.8(17)/19.0(30)	70.0(84)/73.6(90)	15.2(18)/16.4(24)	21.0(28.0)/25.2(37)	69.8(87)/73.6(86)
Average daily throughput (shop floor)	73.973/73.138				
Average time in system (shop floor)	8.426/9.657				
Average total time in queues (shop floor)	7.350/8.581				
Maximal size of WIP (shop floor)	151.4(178)/164(194)				
maximal size of ACO operations	565.8(669)/598.6(687)				
average size of ACO operations	275.4(364)/302.2(413)				

Table 6.5. Increase the number of ants per iteration – Problem 1

Mean job inter-arrival time: 1/9 hour, 1 job/lot ACO (*u* = 20/40) 120 types of jobs (randomly)			Number of machines: 4, 2, 5, 3, 2 Simulation time: 200 hours Warming up time: 20 hours		
performance measure	**1**	**2**	**3**	**4**	**5**
Machine utilization (workcenter)	0.421/0.423	0.873/0.838	0.348/0.335	0.550/0.566	0.884/0.848
Average number in queue (workcenter)	4.666/5.209	34.264/32.949	3.112/2.937	12.153/14.593	34.237/32.734
Maximum number in queue (workcenter)	18.2(34)/ 22.2(34)	75.2(143)/ 70(138)	13.2(22)/ 11.8(21)	36.2(79)/ 41.2(83)	72.8(122)/ 68.6(116)
Average daily throughput (shop floor)	68.364/65.502				
Average time in system (shop floor)	9.999/10.232				
Average total time in queues (shop floor)	8.923/9.156				
Maximal size of WIP (shop floor)	189.8(389)/200.2(382)				
maximal size of ACO operations	626.6(1320)/674.2(1332)				
average size of ACO operations	275.4(529)/262.6(531)				

Table 6.6. Increase the number of ants per iteration – Problem 2

Mean job inter-arrival time: 1/1 hour, 9 jobs/lot ACO (*u* = 20/40) 120 types of jobs (randomly)			Number of machines: 4, 2, 5, 3, 2 Simulation time: 200 hours Warming up time: 20 hours		
performance measure	**1**	**2**	**3**	**4**	**5**
Machine utilization (workcenter)	0.434/0.453	0.916/0.903	0.319/0.355	0.614/0.605	0.929/0.914
Average number in queue (workcenter)	4.940/9.140	31.686/37.849	3.372/6.538	8.251/16.221	33.826/37.819
Maximum number in queue (workcenter)	19.8(32)/ 30.0(41)	73.4(84)/ 77.2(97)	17.6(23)/ 27.4(36)	25.2(36)/ 38.8(45)	72.4(87)/ 75.6(94)
Average daily throughput (shop floor)	72.978/71.502				
Average time in system (shop floor)	9.759/12.349				
Average total time in queues (shop floor)	8.683/11.273				
Maximal size of WIP (shop floor)	166.6(213)/177.6(282)				
maximal size of ACO operations	599.8(679)/718.2(963)				
average size of ACO operations	305.4(443)/358.6(507)				

6.2.4 The effects of changing the number of ants per iteration

The results on the effects of changing the number of ants per iteration are given in tables 6.1, 6.2, 6.5, and 6.6, which show that the overall performance of ACO deteriorates as the size of ants per iteration increases. For example, with other problem parameters unchanged, the average daily throughput decreases from 72.258 (Table 6.3), 68.364 (Table 6.5), to 65.502 (Table 6.5) as the size of ants per iteration increases from 10, 20, to 40 in Problem 1 while the same performance measure decreases from 73.973 (Table 6.4), 72.978 (Table 6.6), to 71.502 (Table 6.6) in Problem 2.

The phenomenon can be explained as follows. A schedule with a small makespan is more likely found by more ants; subsequently, a greater pheromone value is added on the related edges. The optimality found in this schedule will be fully realized if the execution of the schedule is not disturbed by any dynamic/stochastic events. However, once the execution is disturbed, the schedule's optimality will not be able to be fully realized. Furthermore, the amount of pheromone left on edges by the optimized but obsolete schedule may over-strength the pheromone matrix, which, similar to the situation in section 6.2.3, may become rigid in capturing new information introduced by new jobs and thus fail to give good schedules for the subsequent intermediate problems. Thus, increasing the number of ant per iteration may lead to an inferior overall performance in a dynamic environment. For both cases of with and without adaptation mechanism, ten ants per iteration can provide best solutions.

6.3 Summary

A basic version of ACO has been applied to two dynamic JSSPs with the same workloads but different dynamic levels and disturbing severity. The computational results show: 1) the ACO performs effectively in both cases; 2) the adaptation mechanism of the ACO does have effects in situations where disturbances are slight but have little effects in situations where disturbances are severe; and 3) improving the optimality of immediate schedules but sacrificing the flexibility of the pheromone matrix may lead to an inferior long-term performance.

7 ACO Application Domains

The purpose of this chapter is to explore the best application domains that ACO can be applied to in the area of dynamic JSSP. The term ***domain*** describing dynamic JSSPs is comprised of three dimensions: namely the frequency of arriving jobs, the variation of processing times, and performance measures. Given the total number of jobs and performance objectives, a high frequency of arriving jobs implies a highly dynamic problem, which in turn is more difficult to be solved Sthrough algorithmic approaches. The variation of processing times refers to the range that a processing time can take. More types of performance measures are optimized for intermediate JSSPs and they are makespan, mean flowtime, and mean tardiness.

There are two series of experiments. The first series aims to find the range of dynamic levels that ACO can perform well and compares the performances of ACO with several dispatching rules in problems with different dynamic levels and performance objectives. Next, the best ACO strategy and the best dispatching rule are found and used in the second series of experiments to explore the effects of the variation of processing times. Their performances are compared and the proper ranges that ACO outperforms the best dispatching rule are identified. In this manner, a general understanding of the domains that ACO can be appropriately applied will be gained.

7.1 General experimental environment

General experimental environment and rescheduling strategies are similar to those in Chapter 6 and only some differences or important parameters are given as follows.

7.1.1 Shop floor configuration

The simulation experiments have been conducted in a job shop with five workcenters and a reception/shipping station, where new jobs are received and completed jobs are shipped. There is one machine in each workcenter. The traveling times between any of two workcenters are given in Table 7.1.

Table 7.1. Traveling times between workcenters (hours)

Workcenter	1	2	3	4	5	6
1	0	0.01	0.01	0.02	0.02	0.01
2	0.01	0	0.01	0.02	0.02	0.01
3	0.01	0.01	0	0.01	0.01	0.01
4	0.02	0.02	0.01	0	0.01	0.01
5	0.02	0.02	0.01	0.01	0	0.01
6	0.01	0.01	0.01	0.01	0.01	0

7.1.2 Job generation

Jobs have random processing times, random release dates and the routing of each job is generated randomly with every machine having an equal probability of being chosen. Each job has five operations and processing times are drawn from different ranges of the rectangular distribution. Three ranges that processing times can be drawn are: 1.0-5.0, 5.0-10.0, and 1.0-10.0 (hours). The due date of a job is decided following the total work-content method (Ramasesh, 1990). The total work-content of job i (TWK_i) refers to its total processing times and the due-date (D_i) setting follows the formula:

$$d_i = r_i + c*TWK_i$$

(7.1)

where r_i refers to the arrival time of the job i and c indicates the tightness of the due date. c equals 2 in this study to provide a tight due time so that the performance in terms of mean tardiness can be clearly shown. Jobs arrive at the shop floor with inter-arrival times that are independent exponential random variables.

The overall resource utilization of a job shop can be defined as the total processing times required on its machines. The value is affected by the mean inter-arrival time D and the mean processing time $\overline{P}$ of the incoming jobs. The desired utilization rate U can be expressed as $U = \mathrm{D} * \overline{P} / m$ where m is the number of machines. An increasing D leads to an increasing U when the values of $\overline{P}$ and m are fixed. Thus, high machine utilization means highly dynamic JSSP.

7.1.3 Experimental parameters

There are a total of 2200 tested jobs and the steady state begins from the 200th job, which is determined by the technique of the moving average of hourly throughputs (Law and Kelton, 2000). The state of the production system between the arrival times of the 201^{st} job and the 2201^{st} job are then taken as steady state and data collected during this time are collected for statistical analysis. Each simulation consists of five replications.

The parameters of the ACO in this study are $\alpha = 10.0$, $\beta = 10.0$, $\rho = 0.01$, and $\tau_0 = 0.5$ tuned by Zwaan and Marques (1999). Q is adjusted according to the mean values of the processing times in order to give a reasonable influence on the pheromone matrix. For each intermediate JSSP, the minimal and maximal numbers of iterations are 25 and 100, respectively, and the number of ants initiated per iteration is 10.

7.2 Experiments - I

7.2.1 Experimental goals

The goals of this series of experiments are designed to study the performances of the ACO optimizing three different intermediate performance measures, such as makespan, mean flowtime, and mean tardiness, in solving intermediate JSSPs under different experimental conditions. The outcomes are compared with those from FIFO, SPT, and MST for the same problems. Next, the ACO using the best intermediate performance measure, which generates the best overall performance, and the best dispatching rule are used to study the effects of different ranges of processing times in section 7.3.

Three machine-utilization levels are tested in the experiments: 70%, 80%, and 90%. It is obvious that a greater U implies a larger number of operations to be scheduled at any specified time, which implies a harder problem to address. Thus, in all, there are three ranges of processing times, three different utilization levels, and three optimization objectives, making totally 27 simulation experiment sets for the ACO approach; and total 27 simulation experiment sets for all of three dispatching rules.

7.2.2 Results

All the results are presented in tables 7.2 to 7.7. The average values of five replications for each simulation problem are recorded. Measures of the maximal WIP, total throughput, mean flowtime, and mean tardiness are listed. Furthermore, the maximal and the average numbers of operations of intermediate scheduling problems are also recorded for all instances of the ACO approach in Table 7.8.

Table 7.2. Performances of ACO - processing times ranging from 1.0-10.0 (hours)

Utilization	ACO intermediate measure	max. WIP	total TP	mean flowtime	mean tardiness
70%	makespan	23.2	**2000.2**	56.659	10.863
	mean flowtime	**19.4**	1999.6	**49.825**	**6.364**
	mean tardiness	19.8	1998.8	50.0289	6.657
80%	makespan	30.8	1997.6	74.835	25.638
	mean flowtime	**26.2**	**1998.8**	**65.552**	**18.474**
	mean tardiness	27.2	1998	66.595	19.469
90%	makespan	60.2	**2004.6**	171.982	119.319
	mean flowtime	52.2	2000.8	148.799	97.328
	mean tardiness	**51.8**	2000.2	**147.824**	**96.345**

Table 7.3. Performances of Dispatching rules - processing times ranging from 1.0-10.0 (hours)

Utilization	Rules	max WIP	total TP	mean flowtime	mean tardiness
70%	FIFO	22.8	1998.4	59.385	11.203
	SPT	**20**	**1999**	**51.723**	**6.555**
	MST	20.8	1998.8	57.363	8.261
80%	FIFO	31.4	1998.2	79.410	27.536
	SPT	**24.8**	1999.4	**63.433**	**15.135**
	MST	27.2	**2000.2**	75.325	22.683
90%	FIFO	50.6	**2003.8**	140.898	86.709
	SPT	**37.2**	1999.8	**97.571**	**45.989**
	MST	44	1999.8	138.154	83.689

Table 7.4. Performances of ACO - processing times ranging from 1.0-5.0 (hours)

Utilization	ACO intermediate measure	max. WIP	total TP	mean flowtime	mean tardiness
70%	makespan	22.6	1999	30.089	5.256
	mean flowtime	**19**	**1999.6**	**26.727**	**3.117**
	mean tardiness	20.4	1999	26.784	3.313
80%	makespan	31.2	**1998.8**	39.211	12.437
	mean flowtime	**26.6**	1998.4	**34.587**	**8.993**
	mean tardiness	26.8	**1998.8**	34.854	9.384
90%	makespan	53.4	**2004**	80.324	51.675
	mean flowtime	50.4	2001.6	74.106	46.199
	mean tardiness	**47.4**	1999.6	**72.280**	**44.387**

Table 7.5. Performances of Dispatching rules - processing times ranging from 1.0-5.0 (hours)

Utilization	Rules	max WIP	total TP	mean flowtime	mean tardiness
70%	FIFO	22	**1999.2**	30.923	4.968
	SPT	20	1998.8	**27.687**	**3.279**
	MST	**19.6**	1998	30.130	3.643
80%	FIFO	30.8	1998.4	40.843	12.771
	SPT	**25.2**	1998.6	**34.357**	**8.026**
	MST	26.8	**1999.6**	39.039	10.520
90%	FIFO	47.4	**2002.6**	70.563	41.049
	SPT	**36.8**	1999.4	**51.916**	**23.853**
	MST	40.2	2001.6	67.087	37.417

Table 7.6. Performances of ACO - processing times ranging from 5.0-10.0 (hours)

Utilization	ACO intermediate measure	max. WIP	total TP	mean flowtime	mean tardiness
70%	makespan	22.6	1999.4	72.498	11.375
	mean flowtime	**19.6**	1998.8	**64.520**	**6.848**
	mean tardiness	20.4	**1999.6**	65.032	7.361
80%	makespan	31	1997.8	92.127	26.671
	mean flowtime	**27.6**	1996.8	**81.502**	**19.158**
	mean tardiness	27.8	**1998**	83.193	20.639
90%	makespan	50.8	**2000.6**	176.617	106.093
	mean flowtime	48.2	2000.2	160.764	92.135
	mean tardiness	**47.6**	1997.8	**153.073**	**84.617**

Table 7.7. Performances of Dispatching rules - processing times ranging from 5.0-10.0 (hours)

Utilization	Rules	max WIP	total TP	mean flowtime	mean tardiness
70%	FIFO	21.6	1998.6	71.914	8.530
	SPT	19.2	1997.8	**68.433**	8.026
	MST	**19**	**1998.8**	71.152	**6.392**
80%	FIFO	29	1998	93.053	23.868
	SPT	25.8	1996.8	**84.529**	19.470
	MST	**24.4**	**1998.6**	88.847	**18.487**
90%	FIFO	44.6	1999	148.264	74.988
	SPT	38.4	1997.6	**124.880**	**55.398**
	MST	**36.4**	**1999.8**	136.226	62.350

Table 7.8. Maximal and average sizes of intermediate scheduling problems

	machine utilization	70%			80%			90%		
	processing time range	1.0~10.0	1.0~5.0	5.0~10.0	1.0~10.0	1.0~5.0	5.0~10.0	1.0~10.0	1.0~5.0	5.0~10.0
makespan	max. operations	67.6	69.6	67.4	89.8	87.6	86.4	158.8	144.6	138.0
	ave. operations	21.8	21.6	20.2	31.6	31.8	28	75	65.6	56.0
mean flow time	max. operations	67	66.8	66.6	92.2	87.6	90.2	157.2	150.8	144.6
	ave. operations	22.2	22	21.2	32.6	31.8	29.4	74.2	68.4	59.6
mean tardiness (c=2)	max. operations	69	67.8	68.6	92	90.6	90.2	153.4	147.2	139.4
	ave. operations	22	21.6	21.2	32.8	31.6	29.8	73.8	67.2	57.2

7.2.3 Discussions

First of all, it is observed that the differences of total throughputs generated by all the approaches for the same problem are very small. The greatest difference is 4.4 jobs occurring in two occasions of ACO approaches: when the processing time range is 1.0 to 10 with 90% machine utilization (Table 7.2) and when the processing time range is 1.0 to 5.0 with 90% machine utilization (Table 7.4). The size of 4.4 jobs is considered insignificant as compared to the total number of evaluated jobs, which is 2000 in this study. Thus, this performance measure will not be further considered in the following analysis.

7.2.3.1 Processing times ranging from 1.0 to 10.0 (hours)

- Identify the best ACO approach

Among the three intermediate performance measures, ACO optimizing $\overline{F}$ performs best when the machine utilizations are 70% and 80%. For example, it generates overall mean flowtimes of 49.825 and 65.552 (hours), and overall mean tardiness of 6.364 and 18.474 (hours) for machine utilizations of 70% and 80%, respectively (Table 7.2).

The results can be explained as follows. The best intermediate schedule chosen according to the minimal makespan does favor the completion of more jobs. However, this advantage is not prominent when the workload does not exceed the machine capability, especially when machine utilizations are not high. Meanwhile, the other two intermediate performance measures explicitly optimize $\overline{F}$ and $\overline{T}$. Subsequently, the values of their overall $\overline{F}$ and $\overline{T}$ are better than those from the first approach.

Furthermore, the overall values of $\overline{F}$ and $\overline{T}$ generated by minimizing $\overline{F}$ are better than those from minimizing $\overline{T}$ in all the problems where machine utilizations are 70% or 80%. The former approach considers the release times of jobs and can facilitate the jobs with earlier releasing times to be completed earlier. Thus it can improve both the performances of $\overline{F}$ and $\overline{T}$. Finally, all the ACO solutions are outperformed by the dispatching rules when the machine utilization is 90% and thus their performances are not further analyzed.

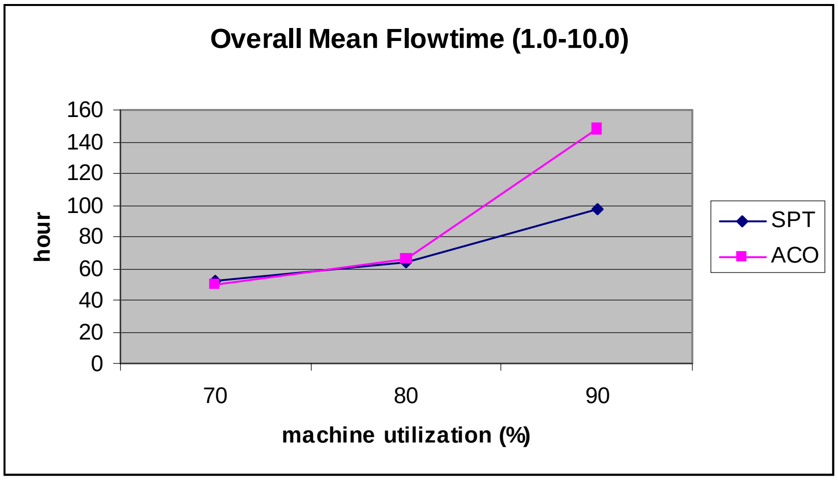
- Identify the best dispatching rule

Among the three tested dispatching rules, the dispatching rule of SPT always outperforms the other two in terms of mean flowtime and mean tardiness in most cases (tables 7.3, 7.5, and 7.7). For example, in Table 7.3 when processing times rang from 1.0 to 10.0 hours, SPT performs best for all measures when the machine utilization is 70% and it performs best for all measures except the total throughput when machine utilizations are 80% and 90%. The similar conclusion is observed in the cases when processing times rang from 1.0 to 5.0 hours (Table 7.5) and from 5.0 to 10.0 hours (Table 7.7).

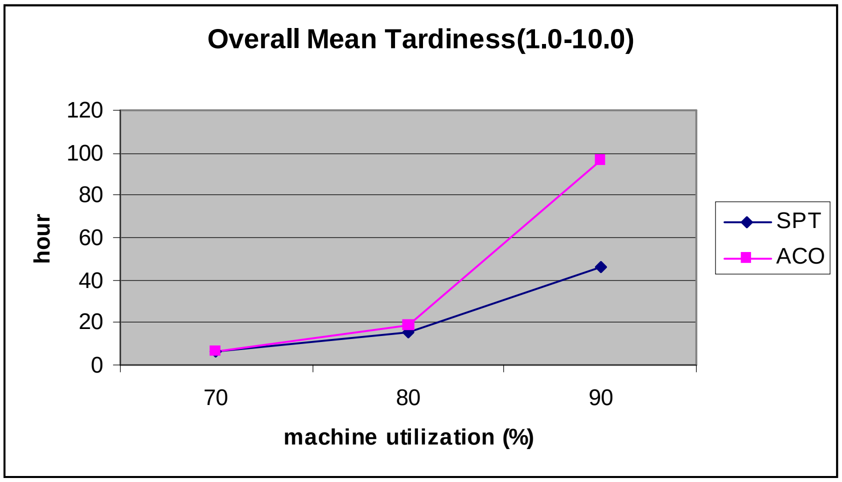
This observation shows that to reduce the total number of operations in a system is important to improve the overall performance.

- Compare the best ACO and the best dispatching rule

The comparisons of the best ACO and the best dispatching rule in terms of mean flowtime and mean tardiness are given in Fig. 7.1 according to Table 7.2 and Table 7.3.



(a) mean flowtime



(b) mean tardiness

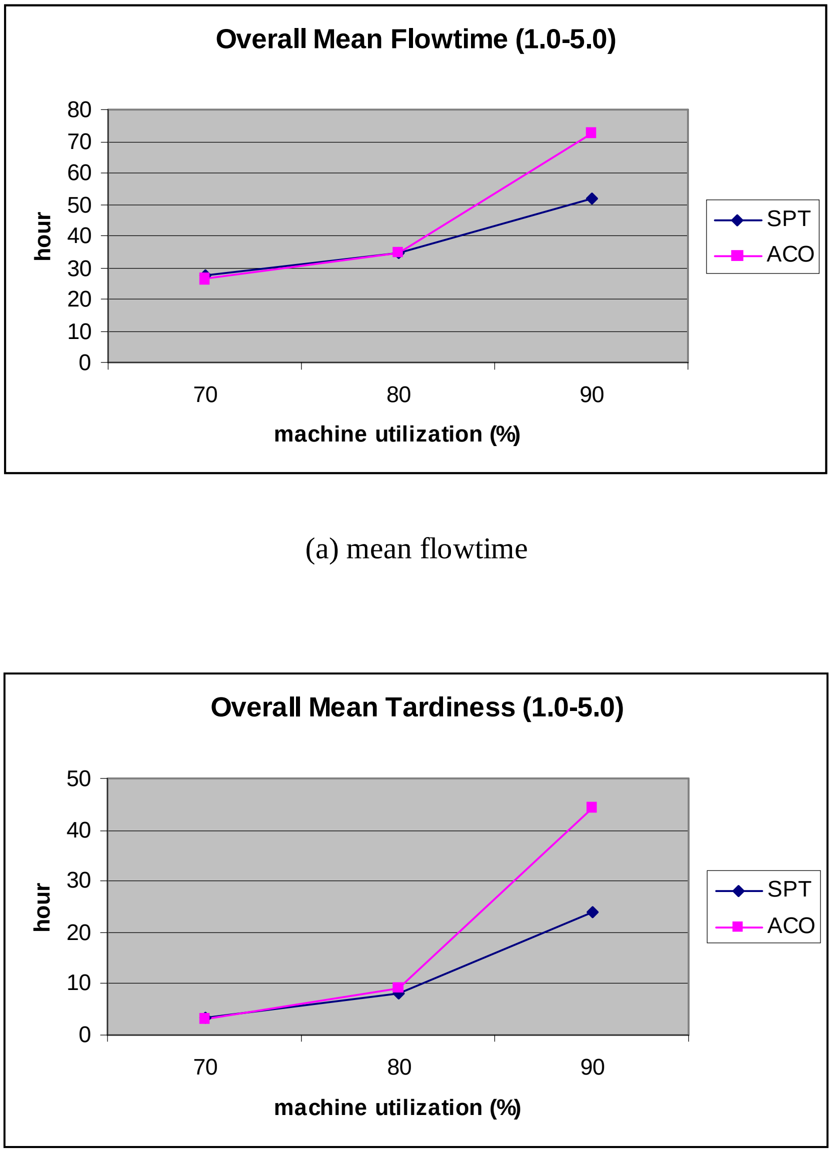
Fig. 7.1. Performance comparison when processing times ranging from 1.0 to 10.0 (hours)

In summary, the best approaches for the three levels of machine utilizations optimizing $\overline{F}$ are ACO for 70% and SPT for both 80% and 90%. The respective best values of overall $\overline{F}$ are 49,825 for 70%, 63.433 for 80%, and 97.571 for 90%. Fig. 7.1 (a) indicates that the performance of ACO deteriorates faster than SPT when the machine utilization is beyond 80%.

Similar results can also be observed in the case of optimizing $\overline{T}$. The only difference is that ACO outperforms SPT when the machine utilization is 80% (Fig. 7.1 (b)), which means that the best approaches for the three levels of machine utilizations for ACO are 70% and 80%, and SPT for 90%.

7.2.3.2 The other two ranges of processing times

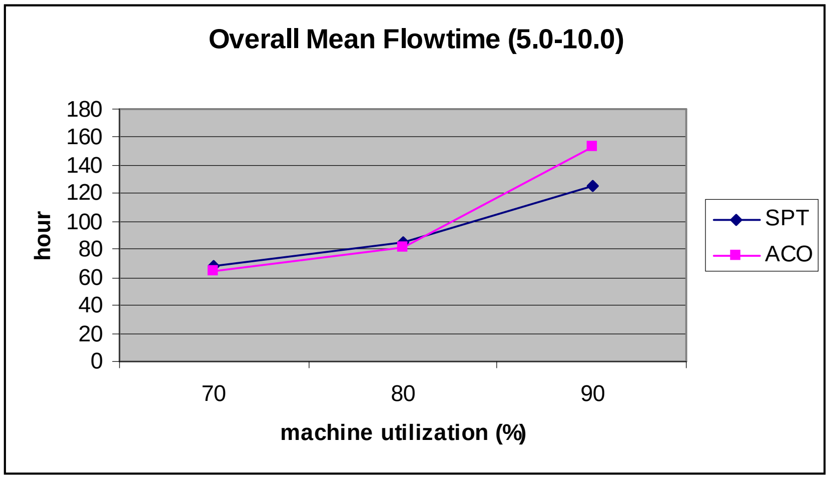
The analysis for the ranges of 1.0 to 5.0 and 5.0 to 10.0 are given in Fig. 7.2 and Fig. 7.3, which show similar results observed in the previous case in both the measures of overall $\overline{F}$ and $\overline{T}$.



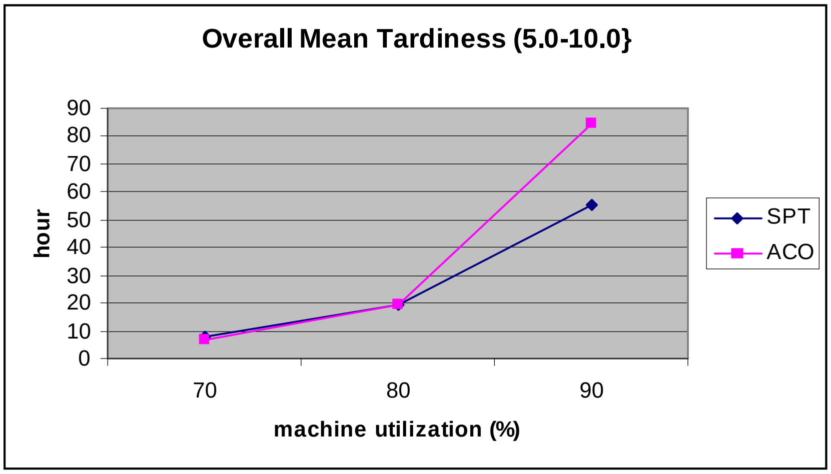
(a) mean flowtime

(b) mean tardiness

Fig. 7.2 Performance comparison when processing times range from 1.0 to 5.0 (hours)



(a) mean flowtime

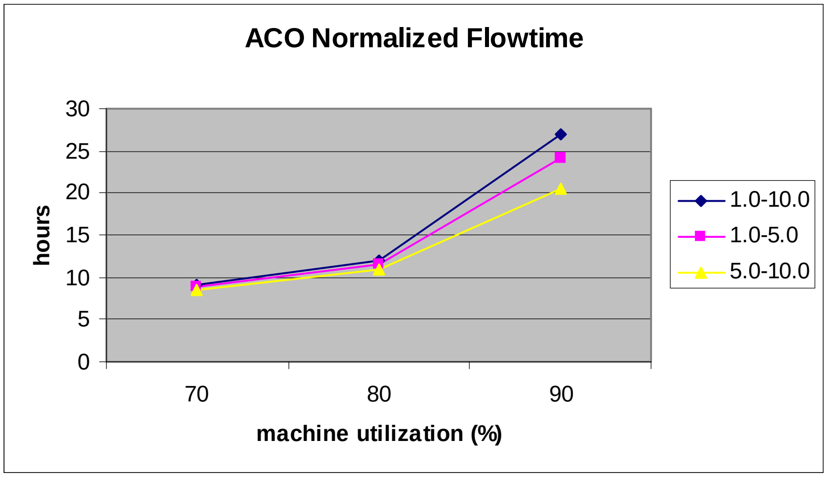


(b) mean tardiness

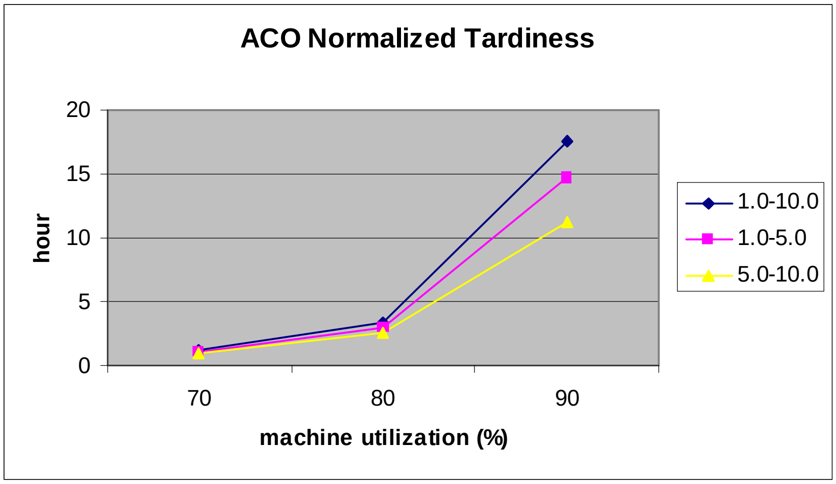
Fig. 7.3 Performance comparison when processing times range from 5.0 to 10.0 (hours)

7.2.3.3 Compare the normalized performances of ACO

The mean job processing times for the ranges of 1.0 to 10.0, 1.0 to 5.0, and 5.0 to 10.0 are 27.5, 15.0, and 37.5 (hours) respectively. In order to investigate the effect of the variation of processing times on the ACO performance, the value of a normalized performance is defined as the performance value divided by the mean job processing time. For example, the normalized mean flowtime obtained by ACO optimizing makespan for intermediate JSSPs equals to 72.498/37.5 when machine utilization is 70% and the range of processing times is 5.0 to 10.0 (hours) (Table 6). 72.498 is the mean flowtime value and the 37.5 is the mean value of the range 5.0 to 10.0. Thus, the normalized performances for the best ACO in three ranges are illustrated in Fig. 7.4 where the values of overall $\overline{F}$ and $\overline{T}$ are divided by the respective job processing times.



(a) Normalized flowtime



(b) Normalized tardiness

Fig. 7.4 Comparison of normalized performances

The comparison shows that ACO for the range of 5.0 to 10.0 performs best while ACO for the range of 1.0 to 5.0 performs worst for both mean flowtime and mean tardiness measures in all three machine utilizations. As the sizes of tested jobs for all the experiments are the same, which is 2200, the normalized performances suggest that the variation of job processing times changes either the complexity of a dynamic JSSP or the performance of ACO, or both. Further studies of the effects of the variation of processing times are presented in section 7.3.

The results also show that the performance of ACO is closely related to the average size of its intermediate JSSPs. For example, Fig. 7.5 illustrates the average sizes of intermediate JSSPs of the best ACO for three machine utilizations and three ranges of processing times, which are recorded in Table 7.8. The average operation sizes for the range of 1.0 to 10.0 are greater than the other two ranges for all three machine utilizations and the results generated by ACO for this range are the worst (Fig. 7.4). Thus, it can be concluded that the performance of ACO is inversely related to the average size of its intermediate JSSPs.

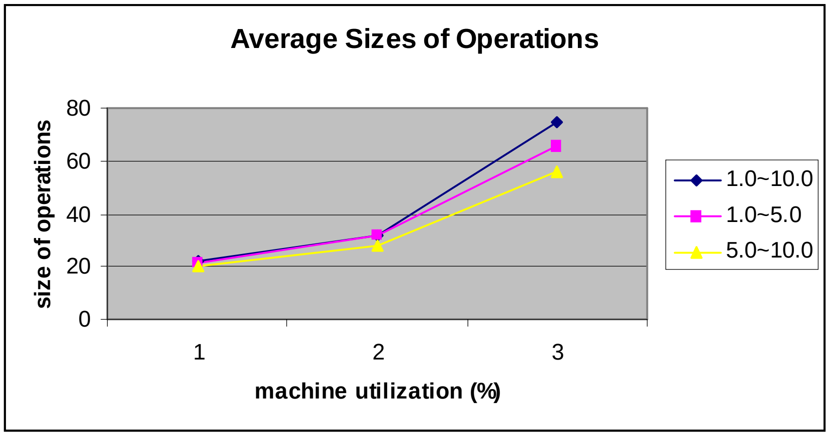


Fig. 7.5 Average sizes of operations of intermediate scheduling problems

This can be explained as follows. The optimality of the schedule generated by ACO decreases as the number of operations increases given the same numbers of iterations and ants. This inferior schedule in turn may increase the number of operations in the following JSSP, making it even harder to be solved. Thus, the approach with fewer operations averagely will always perform better in a long term.

7.2.4 Summary

In summary, the following observations can be made for the two performance measures of mean flowtime and mean tardiness, for all three different ranges of processing times.

1) ACO optimizing mean flowtime for all the intermediate scheduling problems performs better than the other two intermediate performance measures while SPT is the best one among three dispatching rules.

2) ACO performs best when the machine utilization is 70% while SPT performs best when the machine utilization is 90% in terms of mean flowtime and mean tardiness for all the three ranges of processing times; both ACO and SPT can outperform each other when the machine utilization is 80%.

3) The machine utilization is an important factor affecting the performance of ACO. ACO is outperformed by SPT quickly after the machine utilization reaches 80%. This is in accordance with the findings by Sabuncuoglu and Bayiz, (2000): a) there was not much difference between the optimum methods and heuristics when uncertainty or variability was high; and b) the performance of the off-line algorithm was affected more than the on-line method in a stochastic environment.

4) The complexity of a dynamic JSSP is also affected by the ranges of job processing times and the overall performance of ACO is affected by the average size of operations of intermediate scheduling problems.

5) The value changes for the performance measures of mean flowtime and mean tardiness in different problem settings follow similar trends.

7.3 Experiments - II

7.3.1 Experimental goals

In the current section, experiments are carried out to identify how the variation of processing times would affect the dynamic problem and the performance of ACO. Only the best ACO approach and the best dispatching rule, SPT, are compared. Three levels of processing time ranges: 7.0-8.0, 5.0-10.0, and 1.0-14.0 (hours) are chosen to represent three increasing levels of varieties while the mean operation processing times are kept unchanged at 7.5 hours. The variation of processing times is the smallest in the range of 7.0 to 8.0 (hours), followed by the ranges of 5.0 to 10.0 (hours), and then the range of 1.0-14.0 (hours). Three levels of machine utilizations are tested: 60%, 70% and 80%. $\overline{F}$ and $\overline{T}$ are also the overall performance measures. Thus, there are totally 18 simulation problems and each of them has five replications.

7.3.2 Results

The results are presented in Table 7.9 and Table 7.10 where the average values of five replications for each problem are recorded.

Table 7.9. Flowtimes generated from ACO and SPT

	machine utilization	range of processing times		
		7.0~8.0	5.0~10.0	1.0~14.0

ACO	60%	54.369	55.885	58.228
	70%	61.693	64.52	68.938
	80%	**76.156**	81.502	92.078
SPT	60%	59.37	57.052	61.126
	70%	69.557	68.433	70.964
	80%	**87.04**	84.529	87.917

Table 7.10. Tardiness generated from ACO and SPT

	machine utilization	range of processing times		
		7.0~8.0	5.0~10.0	1.0~14.0
ACO	60%	1.938	2.449	3.342
	70%	5.47	6.848	9.549
	80%	**15.476**	19.158	27.665
SPT	60%	3.6	2.644	3.879
	70%	9.136	8.026	9.216
	80%	**22.064**	19.47	21.694

7.3.3 Discussions

The results in tables 7.9 and 7.10 can be illustrated in Fig. 7.6 and Fig. 7.7 for the measure of mean flowtime and mean tardiness, respectively. The horizontal axis represents the range of processing times where 1 refers to the range of 7.0 to 8.0 (hours); 2 refers to the range of 5.0 to 10.0 (hours); 3 refers to the range of 1.0 to 14.0 (hours).

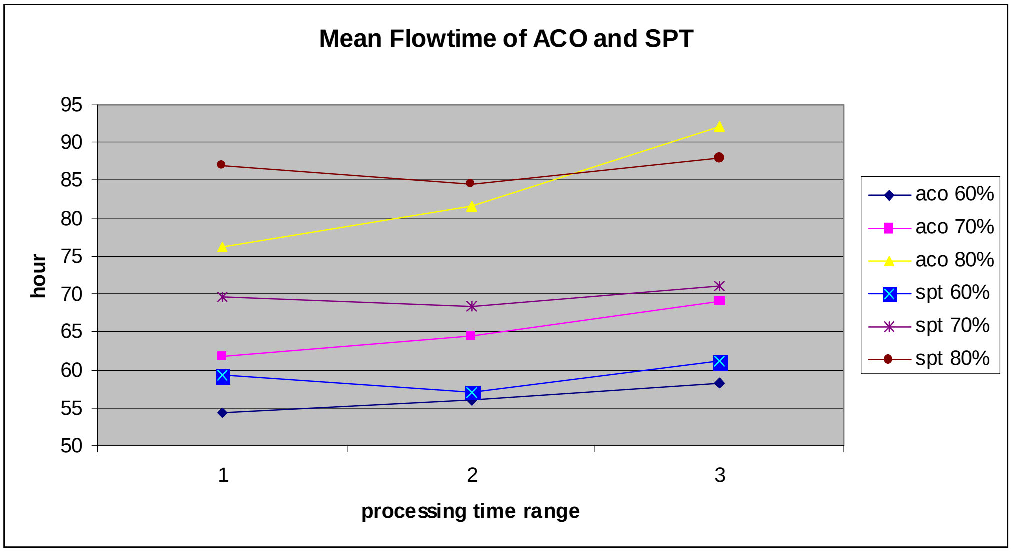


Fig. 7.6 Flowtime generated from ACO and SPT

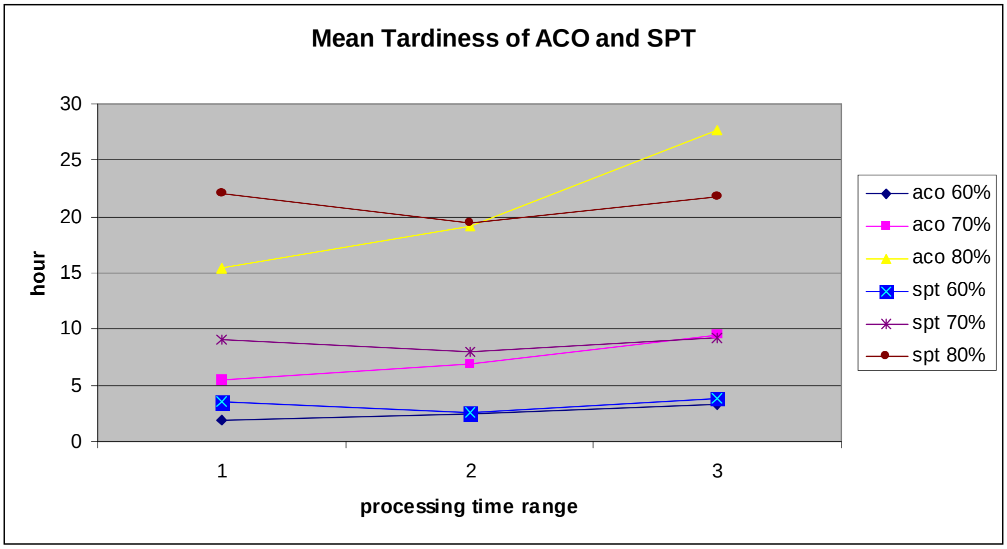


Fig. 7.7 Tardiness generated from ACO and SPT

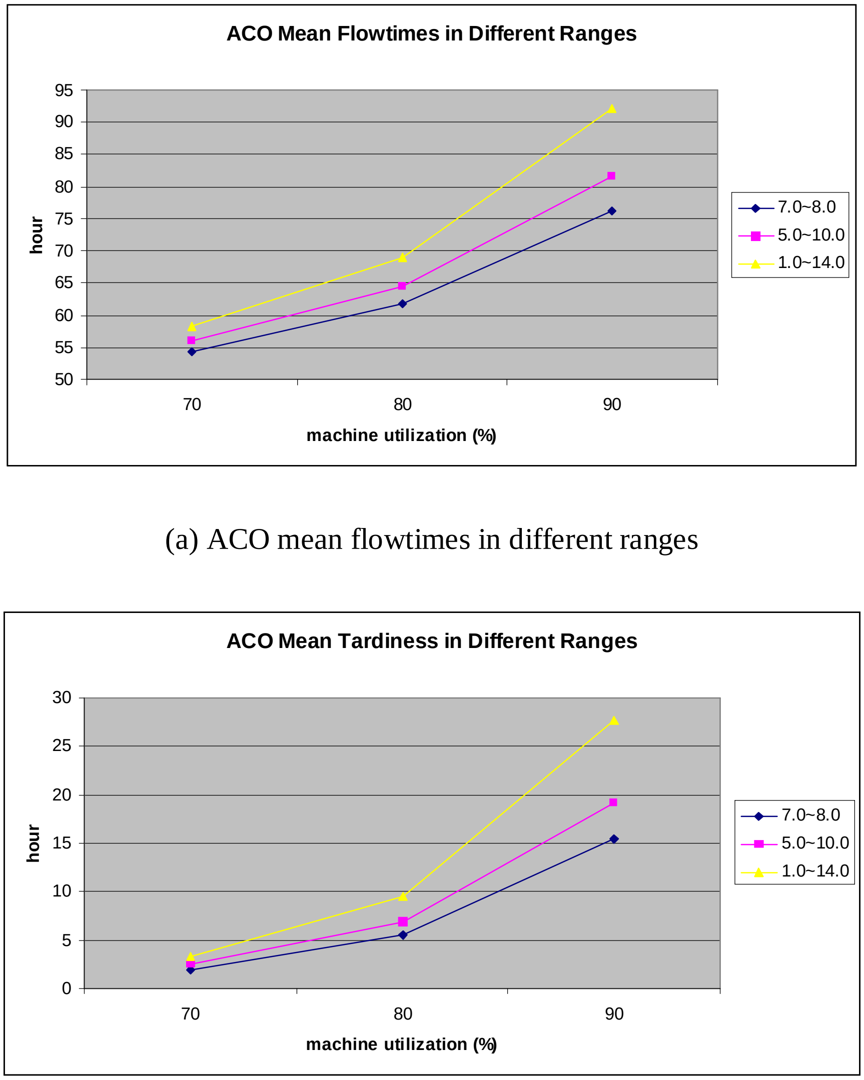
- ACO vs. SPT

In both figures, the top two lines represent the values of mean flowtimes or mean tardiness generated by ACO and SPT for the three ranges of processing times when the machine utilization is 80%. Similarly, the middle two and the lowest two represent those values when machine utilizations are 70% and 60%, respectively.

Both figures show that ACO outperforms SPT in all tested problems for both performance measures in all processing time ranges except for the only occasion when processing time range is from 1.0 to 14.0 and the machine utilization is 80%. It can be concluded that the performance of ACO decreases as the variations of processing times increase, especially, when the machine utilization is high. However, this is not the case for SPT, which performs even better for all the three machine utilizations when the processing times are in the range of 5.0 to 10.0 (hours) than in the other two ranges. The observations are more apparent in Fig. 7.6.

Furthermore, ACO increasingly outperforms SPT for all machine utilizations when the variations of processing times decrease from 1.0~10.0, to 7.0~8.0 and its superiority reaches the highest when the machine utilization (dynamic level) is 80% and processing times range from 7.0 to 8.0 (hours). The value of mean tardiness generated is 15.476 hours from ACO while it is 22.064 hours from SPT (Table 7.10 and Fig. 7.7). The difference of 6.588 hours between the two approaches is significant as the overall tardiness is obtained by multiplying the mean tardiness with the total number of jobs, which is 2000 here for the steady state analysis.

- ACO in different ranges of processing times



(a) ACO mean flowtimes in different ranges

(b) ACO mean tardiness in different ranges

Fig. 7.8 Comparison of ACO performances in different ranges of processing times

The results are further illustrated in Fig. 7.8, which also shows that the performances of ACO in terms of mean flowtime and mean tardiness are inversely affected by the variations of the processing times. That is, the performance of ACO increases when the ranges of processing times decrease from 1.0~14.0, 5.0~10.0, to 7.0~8.0 for the same machine utilization.

7.3.4 Summary

The main findings of this section are as follows. ACO can perform very well in the following situations: 1) when the machine utilization is not high, for example, below 90%, and 2) when the variation of processing times is small. In the latter case, the advantage of ACO can be further enhanced when the machine utilization increases within 90%.

8 Conclusions and Future Work

This chapter first summarizes the work reported in this thesis in section 8.1. Section 8.2 highlights the contributions and the conclusions made in the previous chapters. Finally, future research directions are outlined in section 8.3.

8.1 Research work summary

The thesis first presents a general background of dynamic JSSP. The state-of-the-art predictive-reactive scheduling, MAS scheduling, and applications of ACO on scheduling related problems are reviewed. The internal factors that characterize a dynamic JSSP as well as the factors that affect its overall performance are analyzed. Thereafter, the test bed for systematically studying dynamic JSSPs is built, validated and extended to include an ACO scheduler agent.

Extensive experiments are carried out to present the effectiveness of ACO in solving dynamic JSSPs and the effects of the adaptation mechanism of ACO in the experimental environments characterized with different dynamic levels and disturbance severity. Two important ACO parameters, namely the number of iterations and the size of ants per iteration, are tuned in order to improve the overall performance under the same problem settings. Finally, the appropriate application domains of ACO are experimentally found by testing ACO in many dynamic JSSPs defined by three dimensions of dynamic levels, processing time distributions and intermediate performance measures.

8.2 Contributions

A number of original contributions are listed in the light of the work carried out in this thesis.

8.2.1 Detailed analysis of dynamic JSSP

Detailed analyses of the internal factors that characterize a dynamic JSSP as well as the factors that affect its overall performance are given. The analyses have led to the understanding that the characteristics of a dynamic JSSP can internally determinate its solution approaches and therefore the potential scenarios that are appropriate for optimum-seeking algorithms can be predicted. Furthermore, the factors that can affect the performance of a predictive-reactive approach are analyzed. Finally, the systematic ways of testing a proposed scheduling technique are identified and the domain classification of dynamic JSSPs is introduced according to this analysis.

8.2.2 Proposal of a generic test bed combining DES and MAS

A novel test bed combining the MAS technology and DES has been built to provide scenarios for a systematic study of dynamic JSSPs. This test bed can test traditional approaches like dispatching rules, mathematical methods, or metaheuristics and pure MAS scheduling techniques on their long term performance. To the best of the author’s knowledge, this is the first implementation of MAS with DES for job shop systems.

8.2.3 Development of a simulation software prototype

A simulation software prototype was designed using UML and developed to apply ACO to many dynamic JSSPs. The software was implemented in pure JAVA and based on the JADE platform, which makes it extensible for simulating many other types of shop floor configurations, dynamic events, etc. It is also equipped with graphs to dynamically exhibit intermediate schedules in Gantt charts. Furthermore, the MAS approach makes it possible to be concurrently deployed on several computing nodes and thus the software has the potential to solve large sized problems.

8.2.4 Better understanding of ACO in dynamic JSSPs

A substantial amount of experiments have been designed according to the analyses in Chapter 3 to show the effectiveness of the adaptation mechanism of the ACO pheromone-matrix and the effectiveness of ACO for dynamic JSSPs, improve the performance through adjusting the ACO parameters, and find the appropriate application domains.

The results show that the adaptation mechanism of the ACO can facilitate the adjustment to a new good schedule when a new job interrupts, but this advantage disappears when the frequency of the dynamic events is too low or the pheromone matrix is over strengthened by too many numbers of iterations or too many ants per iteration. In general, the performance of ACO in dynamic JSSPs is affected not only by the distributions of new jobs in time and over the workcenters as well as the batch size, but also its internal important parameters such as the size of ant per iteration and the total number of iterations for one solution. ACO outperforms several main dispatching rules in domains 1) where machine utilization is not higher than 90%, 2) where the variation of processing times is small.

8.3 Further studies

8.3.1 Study other scheduling techniques using the current test bed

The most obvious direction is to study new scheduling techniques, like genetic algorithms, tabu search, simulated annealing, especially those from the MAS scheduling field, etc. to solve dynamic JSSPs taking the advantages of the current DES-MAS test bed in identifying their long-term performances.

The test bed developed can also be used to benchmark dynamic/stochastic scheduling problems so that new algorithms developed in the future can be systematically studied based on those typical scenarios.

8.3.2 Using the current scheduling technique to solve other problems

The proposed ACO can be applied to new problems generated through extending the current test bed. For example, the test bed can include more dynamic/stochastic events like machine breakdowns, processing time variations, or even job due-time settings. The increased complexity may provide new domains that ACO can have a better performance.

The test bed can also be extended to simulate other types of manufacturing systems such as Flexible Manufacturing System or flexible job shop.

8.3.3 Explore ways to improve the performance of ACO

The performance of ACO can be further improved internally and externally in a given dynamic JSSP. The internal approach is to systematically adjust its own parameters or introduce some hybrid versions of ACO, which show better performance than the basic version of ACO in static scheduling problems; the external approach is to explore other control strategies as illustrated in Fig. 2.2 through the use of partial schedules, the periodic driven rescheduling, etc.

References

Abumaizar, R. J., and Svestka, J. A., 1997. Rescheduling job shops under disruptions. International Journal of Production Research, 35, 2065-2082.

ACO, http://iridia.ulb.ac.be/dorigo/ACO/ACO.html.

Akturk, M. S. and Gorgulu, E., 1999. Match-up scheduling under a machine breakdown, European Journal of Operational Research, Vol. 112, pp. 81-97.

Albin, S.L., 1982. On Poisson approximations fro super-position of arrival processes in queues. Management Science, Vol. 28 (2), pp. 126-127.

Angus, D. and Hendtlass, T., 2002. Ant Colony Optimization Applied to a Dynamically Changing Problem. IEA/AIE 2002, LNAI 2358, pp. 618-627.

Askin, R. G. and Standridge, C. R., 1993. Modeling and Analysis of Manufacturing Systems. John Wiley & Sons, Inc.

Aytug, H., Lawley, M.A., Mckay, K., Mohan, S. and Uzsoy, R., 2005. Executing production schedules in the face of uncertainties: a review and some future directions. European Journal of Operations Research, Vol. 161, pp. 86-110.

Baker, K. R., 1974. Introduction to sequencing and scheduling, John Wiley & Sons.

Baker, A. D., 1998. A survey of factory control algorithms that can be implemented in a multi-agent heterarchy: dispatching, scheduling and pull. Journal of Manufacturing Systems, Vol. 17, pp. 297-320.

Balas, E., 1965. An additive algorithm for solving linear programs with zero-one variables, Operations Research, Vol.13, pp. 517-546.

Balas, E., 1967. Discrete programming by the filter method, Operations Research, Vol. 15, pp. 915-957.

Bean, J.C., Birge, J. R., Mittenthal, J., and Noon, C.E., 1991. Matchup scheduling with multiple resources, release dates and disruptions, Operations Research, Vol. 39, No. 3.

Bierwirth, C., 1995. A generalized permutation approach to job shop scheduling with genetic. algorithms. OR Spektrum. 17 (1995) 87–92.

Bierwirth, C., and Mattfeld, D. C., 1999. Production scheduling and rescheduling with genetic algorithms, Evolutionary Computation, Vol. 7, No. 1, pp. 1-17.

Blackstone, J.H., Phillips, D.T. and Hogg, G.L., 1982. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. International Journal of Production Research, Vol. 20, pp. 27-45.

Blazewicz, Ecker, K., Schmidt, G., and Weglarz, J., 1994 Scheduling in computer and manufacturing systems, 2nd version, Springer-Verlag.

Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G. and Weglarz, J., 1996. Scheduling computer and manufacturing processes. Springer-Verlag.

Blum, C., 2002. ACO Applied to Group Shop Scheduling: A Case Study on Intensification and Diversification. In M. Dorigo, G. Di Caro, and M. Sampels, editors, Proceedings of ANTS 2002 - Third International Workshop on Ant Algorithms, volume 2463 of Lecture Notes in Computer Science, pages 14-27. Springer Verlag, Berlin, Germany.

Bonabeau, E., Dorigo, M. and Theraulaz, G., 1999. Swarm Intelligence: From natural to Artificial Systems. Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press.

Bongaerts, L., 1998, Integration of scheduling and control in holonic manufacturing system, Ph.D thesis, Katholieke University, Leuven.

Bongaerts, L., Monostori, L., McFarlane, D. and Kadar, B., 2000. Hierarchy in distributed shop floor control. Computers in Industry, Vol. 43, pp. 123-137.

Branke, J., 2002. Evolutionary optimization in dynamic environments, Kluwer Academic Publishers, Boston.

Bullnheimer, B., Hartl, R. F., and Strauss, C. 1999. A new rank-based version of the Ant System: A computational study. Central European Journal for Operations Research and Economics, 7(1), pp. 25-38.

Buxey, G., 1989. Production Scheduling: Practice and Theory. European Journal of Operational Research, 39, 17-31.

Cavalieri, S., Garetti, M., Macchi, M. and Taisch, M., 2000. An experimental benchmarking of two multi-agent architectures for production scheduling and control. Computers in Industry, Vol. 43, pp. 139-152.

Chryssolouris, G., 2006. Manufacturing systems, Theory and Practice, Springer-Verlag, New York.

Church, L. K., and Uzsoy, R., 1992. Analysis of periodic and event-driven rescheduling policies in dynamic shops. International Journal of Computer Integrated Manufacturing, 5, 153-163.

Cicirello, V. A. and Smith, S. F., 2001. Ant colony control for autonomous decentralized shop floor routing. In ISADS-2001, fifth International Symposium on Autonomous Decentralized Systems. IEEE Computer Society Press, pp. 383–390.

Cicirello, V. A. and Smith, S. F., 2001a. Insect societies and manufacturing, in IJCAI-01 Workshop on Artificial Intelligence and Manufacturing: New AI Paradigms for Manufacturing.

Cicirello, V. and Smith, S. F., 2004. Wasp-based agents for distributed factory coordination, Journal of Autonomous Agents and Multi-Agent Systems, Vol. 8 (3), pp. 237-266, May.

Colorni, A., Dorigo, M., Maniezzo, V., and Trubian, M., 1994. Ant system for Job-shop scheduling. JORBEL-Belgian Journal of Operations Research, Statistics and Computer Science, 34(1), pp. 39-53.

Conway, R.W., Maxwell, W.L. and Miller, L. W., 1967. Theory of scheduling,

Dorigo, M. and Di Caro, G., 1999. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, New Ideas in Optimization, pp. 11–32. McGraw-Hill.

Dorigo, M., Di Caro, G., and Gambardella, L. M., 1999. Ant algorithms for discrete optimization, Artificial Life, Vol.5, No.3, pp. 137-172.

Dorigo, M. and Gambardella, L. M. 1997a. Ant colonies for the traveling salesman problem. BioSystems, 43(2), pp. 73-81.

Dorigo, M. and Gambardella, L. M. 1997b. Ant colony System: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, 6(4), pp. 317-365.

Dorigo, M. and Stützle, T., 2004. Ant Colony Optimization. A Bradford Book, The MIT press, Cambridge, Massachusetts, London, England.

Dorigo, M., Maniezzo, V. and Colorni, A., 1991. Distributed optimization by ant colonies, Proceedings of ECAL91 – European Conference on Artificial Life, Elsevier Publishing, 134-142.

Dorigo, M., Maniezzo, V. and Colorni, A., 1996. Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, Part B, 26(1).

Durfee, E.H., 1988. Coordination of Distributed Problem Solvers, Kluwer Academic Publishers.

Durfee, E.H. and Lesser, V., 1989. Negotiating task decomposition and allocation using partial global planning. In L. Gasser and M. Huhns, editors, Distributed Artificial Intelligence, Volume II, pp229-244. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA.

French, S., 1982. Sequencing and scheduling: an introduction to the mathematics of the job shop, Ellis Horwood Limited and John Wiley & Sons.

Froeschl, K., 1993. Two paradigms of combinatorial production scheduling operations re-search and articial intelligence. In Scheduling of Production Processes. Dorn, J. and Froeschl, K., 1993, England: Ellis Horwood.

Fox, M.S. and Smith, S.F., 1984. ISIS: a knowledge-based system for factory scheduling. Expert Systems 1, 25-49.

Garey, M.R. and Johnson, D.S., 1979, Computers and intractability: a guide to the theory of NP-completeness, San Francisco: W. H. Freeman.

Glover, F., 1989. Tabu search-Part I, ORSA Journal on Computing, 1(3), pp. 190-206.

Glover, F., 1990. Tabu search-Part II, ORSA Journal on Computing, 2(1), pp. 4-32.

Glover, F., and Laguna, M., 1997. Tabu Search, Boston, Kluwer Academic Publishers.

Goldberg, D.E., 1989. Genetic algorithms in search, optimization and machine learning, MA, Addison-Wesley.

Grave, S.C. 1981. A review of production scheduling, Operations Research, Vol. 29, No. 4, Operations Management/Research, pp. 646-675.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics 5, 287--326.

Guntsch, M. and Middendorf, M., 2001. Pheromone modification strategies for ant algorithms applied to dynamic TSP. In Boers, E.J.W., Gottlieb, J., Lanzi, P.L., Smith, R.E., Cagnoni, S., Hart, E., Raidl, G.R. and Tijink, H., (eds.): Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2001, number 2037 in Lecture Notes in Computer Science, pages 213–222. Springer Verlag.

Guntsch, M. and Middendorf, M., 2002a. A Population Based Approach for ACO, In Cagnoni, S., Gottlieb, J., Hart, E. Middendorf , M., and Raidl, G.R. (eds.): Applications of Evolutionary Computing - Evo Workshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN, volume 2279 of LNCS, pp. 72-81. Springer.

Guntsch, M. and Middendorf, M., 2002b. Applying Population Based ACO to Dynamic Optimization Problems. In Ant Algorithms, Proceedings of Third International Workshop ANTS 2002, volume 2463 of LNCS, pp. 111-122.

Guntsch, M. and Middendorf, M., and Schmeck, H., 2001. An ant colony optimization approach to dynamic TSP. In Spector, L, Goodman, E.D., Wu, A., Langdon, W.B., and Voigt, H.M., editors: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), pages 860–867. Morgan Kaufmann Publishers.

Hall, N. G. and Posner, M. E., 2001. Generating experimental data for computational testing with machine scheduling applications, Operations Research, Vol. 49, No. 7, pp.854-865.

Haupt, R., 1989. A survey of priority rule-based scheduling, Operations Research, Spektrum, Vol.11, No.1, pp.3-36.

Holloway, C.A., and Nelson, R. T., 1974. Job shop scheduling with due dates and variable processing times, Management Science 20 (9), pp. 1264-1275.

Holthaus, O., 1999. Scheduling in job shops with machine breakdowns: an experimental study, Computer & Industrial Engineering, Vol. 36, pp. 137-162.

Holthaus, O. and Rajendran, C., 1997. Efficient dispatching rules for scheduling in a job shop. International Journal of Production Economics, Vol. 48, pp. 87-105.

Hopp, W. J. and Spearman, M. L., 2000. Factory physics: Foundations of manufacturing management, 2nd Edition, Irwin McGraw-Hill.

IHPC, Institute Of High Performance Computing, Singapore. http://www.ihpc.a-star.edu.sg (in April. 2007).

JADE, http://jade.tilab.com/ (in April. 2007).

Jain, A. K., and ElMaraghy, H. A., 1997. Production scheduling/rescheduling in flexible manufacturing. International Journal of Production Research, 35, 281-309.

Jain, A. S. and Meeran, S., 1998. A state-of-the-art review of job shop scheduling techniques. In: http://citeseer.nj.nec.com (in April. 2007).

Jain, A. S. and Meeran, S., 1999. Deterministic job-shop scheduling: Past, present and future. European Journal of Operational Research, 113 (2), 390-434.

Jennings N. R., Sycara K. and Wooldridge M., 1998. A Roadmap of Agent Research and Development, Autonomous Agents and Multi-Agent Systems, Vol. 1, pp.275-306, Kluwer Academic Publishers.

Jennings, N. R. and Wooldridge, M., 1998. Agent Technology: Foundations, Applications and Markets, Springer, pp. 3-28.

Jones, A. and Rabelo, L. C., 1998. Survey of job shop scheduling techniques. In: http://citeseer.nj.nec.com (in April. 2007).

Johnson, S.M., 1954. Optimal two-and-three-stage production schedules with set-up times included. Naval Research Logistic Quarterly, Vol. 1, pp. 61-68.

Kirkpatrick, S., Gelatt, C.D., Jr., and Vecchi, M.P., 1983. Optimization by simulated annealing, Science, Vol. 220, pp. 671-680.

Koestler, A., 1967. The Ghost in the Machine, Arkana Books, London.

Kusiak, A., 2000. Computational Intelligence in Design and Manufacturing, John Wiley & sons, Inc.

Law, A.M. and Kelton, W.D., 2000. Simulation Modeling and Analysis. McGraw Hill.

Lawrence, S. R. and Sewell, E. C., 1997. Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. Journal of Operations Management Vol. 15, pp. 71-82.

L’Ecuyer, P., Simard, R., Chen, E.J., Kelton, W.D., 2001. An Object-Oriented Random-Number Package with Many Long Streams and Substreams, Operations Research, Vol. 50 (6), pp. 1073-1075.

Li, R.K., Shyu, Y.T., and Adiga, S., 1993. A heuristic rescheduling algorithm for computer-based production scheduling systems. International Journal of Production Research, 31, 1815-1826.

Lin, S., Goodman, E.D., and Punch, W.F., 1997. A Genetic Algorithm Approach to Dynamic Job Shop Scheduling Problems. . In Back, T., editor, Proceedings of the Seventh International Conference on Genetic Algorithms, pages 481-489, Morgan Kaufmann, San Mateo, California.

Liu, J., 1996. Coordination of multiple agents in distributed manufacturing scheduling. PhD thesis, Carnegie Mellon University.

MacCarthy, B. L. and Liu, J. 1993. Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. International Journal of Production Research, 31 (1), 59-79.

Maniezzo, V., 1999. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. INFORMS Journal on Computing, 11(4), pp. 358-369.

Mehta, S. V., and Uzsoy, R. M., 1998. Predictable scheduling of a job shop subject to breakdowns. IEEE Transactions on Robotics and Automation, 14, 365-378.

Miyashita, K., 1995. CABINS: A framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair, AI Vo. 76, pp. 377-426.

Morley, D, 1996. Painting trucks at general motors: The effectiveness of a complexity-based approach, in Embracing Complexity: Exploring the Application of Complex Adaptive Systems to Business, pp.53-58. The Ernst and Young Center for Business Innovation.

Morley, D. and Schelberg, C., 1993. An analysis of a plant-specific dynamic scheduler. In Final Report, Intelligent Dynamic Scheduling for manufacturing System, pp. 115-122.

Morton, T. e. and Pentico, D. W., 1993. Heuristic Scheduling Systems: With Applications to Production Systems and Project Management. John Wiley and Sons.

Muhleman, A.P., Lockett, A.G., and Fan, C.K., 1982. Job shop scheduling heuristics and frequency of scheduling, International Journal of Production Research, Vol. 20, No. 2. pp.227-241.

Nelson, R., Holloway, D., and Wong, R.M., 1977. Centralized scheduling and priority implementation heuristics for a dynamic job-shop model with due dates and variable processing time. AIIE Transactions, Vol. 9, No. 1, pp. 95-102.

Nowicki, E. and Smutnicki, C., 1995. A fast taboo search algorithm for the job shop problem. Management Science, 42, 797-813.

Okino, N., 1993. Bionic manufacturing systems. Proceedings of the CIRP Seminar on Flexible Manufacturing Systems Past-Present-Future, edited by J. Peklenik, Bled, Slovenia, pp. 73-95.

OR-Library, www.ms.ic.ac.uk/info.html (in April. 2007).

Ovacik, I.M., and Uzsoy, R., 1994. Exploiting shop floor status information to schedule complex job shop. Journal of Manufacturing Systems, 13 (2), pp. 73-84.

Ovacik, I.M., and Uzsoy, R., 1997. Decomposition methods for complex factory scheduling problem, Kluwer, Dordrecht.

Panwalkar, S. S., and Iskander, W., 1977. A survey of scheduling rules, Operations Research, Vol. 25, pp. 45-61.

Park, J., Kang, M., and Lee, K, 1996. Intelligent operations scheduling system in a job shop. International Journal of Advanced Manufacturing Technology, Vol. 11, pp. 111-119.

Parunak, H. V.D., 1987. Manufacturing Experience with the Contract Net. Distributed Artificial Intelligence, Huhns, M.N. ed., Pitman, pp. 285-310.

Parunak, H. V. D., 1991. Characterizing the manufacturing scheduling problem. Journal of Manufacturing Systems, Vol. 10, pp. 241--259.

Parunak, V. 1994. Applications of Distributed Artificial Intelligence in Industry. O'Hare and Jennings, eds. 1996, Foundations of Distributed Artificial Intelligence. Wiley Inter-Science.

Parunak, H.V.D., 1997. “Go to the Ant”: Engineering Principles from Natural Multi-Agent Systems, Annals of Operations Research, Vol. 75, pp. 69-101 (Special Issue on Artificial Intelligence and Management Science).

Peeters, P., Brussel, H. V., Valckenaers, P., Wyns, J., Bongaerts, L., Kollingbaum, M., and Heikkilä, T., 2001. Pheromone based emergent shop floor control system for flexible flow shops, Artificial Intelligence in Engineering, Vol. 15, pp. 343-352.

Pendharkar, P.C., 1999. A computational study on design and performance issues of multi-agent intelligent systems for dynamic scheduling environments, Expert Systems with Application, Vol.16, pp. 121-133.

Pinedo, M., 2002. Scheduling theory, algorithms and systems. Second edition, Prentice Hall, Upper Saddle River, New Jersey.

Raheja, A.S. and Subramaniam, V., 2002. Reactive recovery of job shop schedules - a review. International Journal of Advanced Manufacturing Technology, Vol. 19, pp.756-763.

Raman, N. and Talbot, F.B., 1993. The job shop tardiness problem: a decomposition approach, European Journal of Operational Research, vol. 69, pp. 187-199.

Ramasesh, R., 1990. Dynamic job shop scheduling: a survey of simulation research, OMEGA International Journal of Management Science, Vol. 18, No. 1, pp. 43-57, 1990.

Randhawa, S.U. and McDowell, E.D., 1990. An investigation of the applicability of expert systems to job shop scheduling, International Journal of Man-Machine Studies, Vol. 32, pp. 203-13.

Ryu, K. and Jung, M. 2003. Agent-based fractal architecture and modelling for developing distributed manufacturing systems, International Journal of Production Research, 2003, Vol. 41, No. 17, 4233-4255.

Sabuncuoglu, I. and Bayiz, M., 2000. Analysis of reactive scheduling problems in a job shop environment, European Journal of Operational Research, Vol. 126, pp.567-586.

Sabuncuoglu, I. and Kizilisik, O.B., 2003. Reactive scheduling in a dynamic and stochastic FMS environment, International Journal of Production research, Vol. 41, No. 17, 4211-4231.

Sadeh, N, 1991. Look-Ahead Techniques for Micro Opportunistic Job Shop Scheduling PhD thesis, Carnegie Mellon University.

Schoonderwoerd, R., Holland, O., Bruten, J. And Rothkrantz, L., 1996. Ant-based Load Balancing in Telecommunications Networks. Adaptive Behavior Vol. 5, pp. 168-207.

Sellers, D. W., 1996. A survey of approaches to the job shop scheduling problem, Proceedings of the 28th Southeastern Symposium on System Theory (SSST '96), IEEE, pp. 396 – 400.

Shannon, J.B., 1979. An analysis of the distributions of the job shop scheduling rules. AIDS Proceedings, 11th Annual Meeting, November 1979.

Shen, W. and Norrie, D.H., 1999, Agent-based systems for intelligent manufacturing: a state-of-the-art survey. International Journal Knowledge and Information Systems, 1, 129–156.

Smith, R.G., 1980. The contract net protocol: high-level communication and control in a distributed problem solver. IEEE Transactions on Computers, C-29 (12), pp.1104-1113.

Smith, S.F., Ow, P.S., Potvin, J.-Y, Muscotella, N., and Matthys, D., 1990. An integrated framework for generating and revising factory schedules. Journal of Operational Research Society, Vol. 41, (6), pp. 539-552.

Smith, S. F., 1995. Reactive scheduling systems, in *Intelligent Scheduling Systems*, edited by Brown, D. E, and Scherer, W. T. Boston : Kluwer Academic Publishers, pp. 155-192.

Smith, S. F., 2003 (August). Is Scheduling a Solved Problem? Invited Keynote Talk, *Proceedings First Multi-Disciplinary International Conference on* Scheduling: Theory and Applications (MISTA 03), Nottingham, UK.

Stoop, P.P.M., Wiers, V.C.S., 1996. The complexity of scheduling in practice. International Journal of Operational and Production Management 16 (10), 37–53.

Stützle, T., 1998. An Ant Approach to the Flow Shop Problem, Proceedings of EUFIT'98, Aachen, pp. 1560-1564.

Stützle, T. and Dorigo, M., 1999. ACO Algorithms for the Traveling Salesman Problem, In Miettinen, K., Mäkälä, M. M., Neittaanmäki, P. and Périaux, J. (Eds.), Evolutionary Algorithms in Engineering and Computer Science (pp. 163-183). Chichester, UK, John Wiley & Sons.

Stützle, T. and Hoos, H. H., 1997. The MAX-MIN Ant System and local search for the traveling salesman problem. In Bäck, T., Michalewicz, Z. and Yao, X. (Eds.), Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC’97) (pp. 309-314). Piscataway, NJ, IEEE Press.

Stützle, T. and Hoos, H. H., 2000. MAX-MIN Ant System. Future Generation computer Systems, 16 (8), pp. 889-914.

Sule, D.R., 1997. Industrial Scheduling, PWS Publishing Company.

Suresh, V., and Chaudhari, D., 1993. Dynamic scheduling – a survey of research, International Journal of Production Economics, Vol. 32, pp. 53-63.

Szelke, E. and Kerr, R., 1994. Knowledge-based reactive scheduling. Production Planning and Control, Vol. 5, pp. 124-145.

UMBC. http://agents.umbc.edu/Institutes_and_labs/Academic/index.shtml

UML, http://www.omg.org/technology/documents/formal/uml.htm (in April. 2007).

Valckenaers, P., Bonneville, F., Brussel, H.V. and Wyns, J., 1994. Results of the holonic system benchmark at KU Leuven, Proceedings of the Fourth International Conference on Computer Integrated Manufacturing and Automation Technology, Oct. 10-12, Troy, New York, pp. 128-133.

Valckenaers, P., Brussel, H. V., Kollingbaum, M, and Bochmann, O., 2001. Multi-agent coordination and control using stigmergy applied to manufacturing control, Mutli-Agents Systems and Applications, pp. 317 – 334.

Vieira, G.E., Herrmann, J.W. and Lin, E., 2003. Rescheduling manufacturing systems: A framework of strategies, policies, and methods. Journal of Scheduling, Vol. 6, No. 1, pp. 39-62.

Vogel, A., Fischer, M., Jaehn, H. And Teich, T., 2002. Real-world shop floor scheduling by Ant Colony Optimization, In Dorigo, M., Di Caro, G, and Sampels, M. *al* (Eds.): ANTS 2002, LNCS 2463, pp. 268-273.

Vollmann, T.E., Berry, W.L. and Whybark, D.C., 1992. Manufacturing Planning and Control Systems, Irwin, Homewood, IL.

Voß, S., Martello, S., Osman, I. H., and Roucariol, C., editors, 1999. Meta-heuristics: advances and trends in local search paradigms for optimization. Kluwer, Boston.

Voß, S., 2001. Meta-heuristics: The State of the Art, in (ed. Nareyek, A.) Local Search for Planning and Scheduling, ECAI 2000 Workshop, Springer Lecture Notes in AI, Vol. 2148.

Warnecke, H.J., 1993. The Fractal Company, a Revolution in Corporate Culture, Springer-Verlag, Berlin.

Weirs, V. C. S., 1997. A review of the applicability of OR and AI scheduling techniques in practice. Omega International Journal of Management Science, 25(2), 145-153.

Weiss G., 1999. Multiagent systems – a modern approach to distributed artificial intelligence. Cambridge: MIT Press.

Wong, T.N., Leung, C.W., Mak, K.L., and Fung R.Y.K., 2006a. An agent-based negotiation approach to integrate process planning and scheduling, International Journal of Production Research, Vol. 44, No. 7, pp. 1331-1351.

Wong, T.N., Leung, C.W., Mak, K.L., and Fung R.Y.K., 2006b. Integrated process planning and scheduling/rescheduling-and agent-based approach, International Journal of Production Research, Vol. 44, Nos. 18-19, pp. 3627-3655.

Wooldridge, M., 2001. An Introduction to Multiagent Systems, John Wiley & Sons, Ltd.

Wyns, J., 1999. Reference architecture for holonic manufacturing systems: the key to support evolution and reconfiguration. Ph.D thesis, Katholieke University, Leuven.

Xiang, W., Fox B. and Lee, H. P., 2005. "Ant Colony Optimization for the Job Shop Scheduling Problem using Multi-Agent Systems", Proceedings of the 2005 International Conference on Artificial Intelligence (ICAI'05), Las Vegas, June 27-30, 2005, pp. 898-904, 2005.

Zhou, R., Fox, B., Lee, H. P., and Nee, A. Y. C., 2004. Bus maintenance scheduling using multi-agent systems, Engineering Applications of Artificial Intelligence, Vol. 17, pp. 623-630.

Zhou, R., Lee, H. P., and Nee, A. Y. C., 2006. Simulating the Generic Job Shop as A Multi-Agent System, International Journal of Intelligent Systems Technologies and Applications (IJISTA): Special Issue on: “Advanced Evolutionary Computational Techniques for Design, Manufacturing, Logistics and Supply Chain Problems” (in press).

Zwaan, d.v.S. and Marques, C., 1999. Ant colony optimisation for job shop scheduling. In Proceedings of the ’99 Workshop on Genetic Algorithms and Artificial Life GAAL’99.

Zweben, M., Duan, B., Deale, M., 1994. Scheduling and rescheduling with iterative repair. In: Fox, M.S. (Ed.), Intelligent Scheduling. Wiley, New York, pp. 241–255.

Publications arising from this Thesis

Zhou, R., Fox, B., Lee, H.P., and Nee, A.Y.C., 2004. Bus Maintenance Scheduling using Multi-Agent Systems, Engineering Application of Artificial Intelligence, Vol. 17, pp. 623-630.

Zhou, R., Lee, H.P., and Nee, A.Y.C., 2008. Simulating the Generic Job Shop as A Multi-Agent System, Special Issue of International Journal of Intelligent Systems Technologies and Applications (IJISTA), Vol. 4, Nos. 1/2, pp.5-33.

Zhou, R., Lee, H.P., and Nee, A.Y.C., 2007. Application of Ant Colony Optimization Algorithm for Dynamic Job Shop Scheduling Problems, International Journal of Manufacturing Research (in press).

Zhou, R., Nee, A.Y.C., and Lee, H.P., 2007. The Performance of Ant Colony Optimization Algorithm in Dynamic Job Shop Scheduling Problems, International Journal of Production Research (in press).