The Crane Problem: Scheduling with Sequence-Dependent Set-up and Processing Times

David Dominic Clark

A research project submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Master of Science.

Johannesburg, 1998

DECLARATION

I declare that this research project is my own, unaided work. It is being submitted for the Degree of Master of Science in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

David Dominic Clark

18TH day of NOVEMBER 1998

ABSTRACT

The problem of scheduling with sequence-dependent set-up times in a dynamic environment is investigated by studying how various dispatching rules perform when used to schedule two cranes. Motivated by a practical scheduling problem, the effect on production by delays due to the conflicts that result between cranes is examined. The problem is formalized, and it is shown that it can be classified as a problem of scheduling with both sequence-dependent set-up and processing times. The effectiveness of simple dispatching procedures that are used in machine scheduling and for the control of automated guided vehicles is studied, using a simulation of a crane aisle with jobs arriving dynamically. In addition, a dispatching rule, which explicitly uses information regarding the state of the second crane, is examined. The simulation results confirm the non-dominance of certain dispatching procedures, and show how performance is improved as the rules are provided more information regarding the state of the scheduling environment. It is shown that when there are sequence-dependent processing times, a scheduling heuristic that uses global information does significantly better than more commonly used local heuristics.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The control of two overhead cranes, that share a common track, is presented as a scheduling problem that involves sequence-dependent set-up and processing times. Set-up times are prevalent in many scheduling environments. In manufacturing the time spent preparing a machine for a specific task can be defined as the set-up time for that task. If the set-up period is dependent not only on the present task to be carried out, but also on what jobs were performed previously, then the set-up times are sequence-dependent.

Manufacturing processes are usually geared towards transforming some physical attribute of an object, however the crane problem is concerned with changes in the spatial state of an object. A crane's function is to move objects from a *pickup* location to some *drop-off* point. The set-up time can therefore be considered as the time taken for the crane to move from its initial position to the *pickup* location.

When only one crane is on the track, the scheduling problem involves only set-up times. However, when a second crane is introduced, not only are set-up times present but also sequence-dependent process times. This dependence is not only on the previous job of the crane, but also on the position and state of the second crane. The possible interference of the cranes results in set-up and processing times that are contingent on the sequencing of jobs on both cranes.

The rest of this report is organized as follows. The remainder of the introduction introduces some of the salient characteristics of machine scheduling problems, along with terminology and definitions relating to the measurement of scheduling performance. An introduction to Automated Guided Vehicles (AGVs) follows, along with a brief comparison of the AGV problem to the crane problem. Concluding this section is a formal description of the crane system based on a model presented in [Lieberman and Turksen, 81] along with definitions of the set-up and processing time of jobs, and how crane interference affects these times.

Section 2 presents a survey of relevant literature. The emphasis is on dispatching rules and scheduling problems that deal with set-up times. A selection of papers that study the dispatching of AGVs is provided, as well as literature that is related to various types of crane problems.

A description of the real-world crane problem, and the assumptions made regarding the model used are presented in Section 3. Section 4, outlines the six dispatching heuristics that are examined. Their performance with regard to the output of the system is reported in Section 5, along with various other measurements such as the slaved distances of the cranes.

An analysis of the results is presented in Section 6. The results of two statistical procedures, ANOVA and Duncan's multiple range test, are presented with respect to the production output values. Section 7 concludes the paper with a discussion of the results.

1.1 Scheduling: Basic definitions and terminology

The problem of allocating resources over time to perform tasks is known as the scheduling problem. This general definition of scheduling can be characterized using the terminology of a manufacturing environment as follows. Given a number of *tasks*, each consisting of one or more *operations* that must be completed in some order, and that require a certain amount of *processing-time* on one or more *machines*, the scheduling problem involves determining the sequence, timing, and machine assignment of each operation to optimize some performance criterion.

The complexity of real-world scheduling problems [Rinnooy Kan, 76], [Graves, 81], [McKay, *et al*. 88], [Ramesh, 90] and [Chen and Yih, 96], has meant that sequencing and scheduling theory has been mostly occupied by developing a substantial body of knowledge on the analysis and optimization of simplified problems [Rinnooy Kan, 76].

Scheduling problems belong to the broader class of combinatorial problems [Blazewicz, *et al.* 88]. An analysis of the scheduling problem is shown in Figure 1.1. An *easy* problem is referred to as a problem for which there exists an efficient polynomial-time algorithm. When the problem is NP-hard, three general approaches can be taken. The problem may be *relaxed* e.g. allow jobs to be pre-empted, *approximation* algorithms may be used to find efficient but non-optimal solutions to the problem, or in the case of small problem instances, *complete enumeration* algorithms may be used.

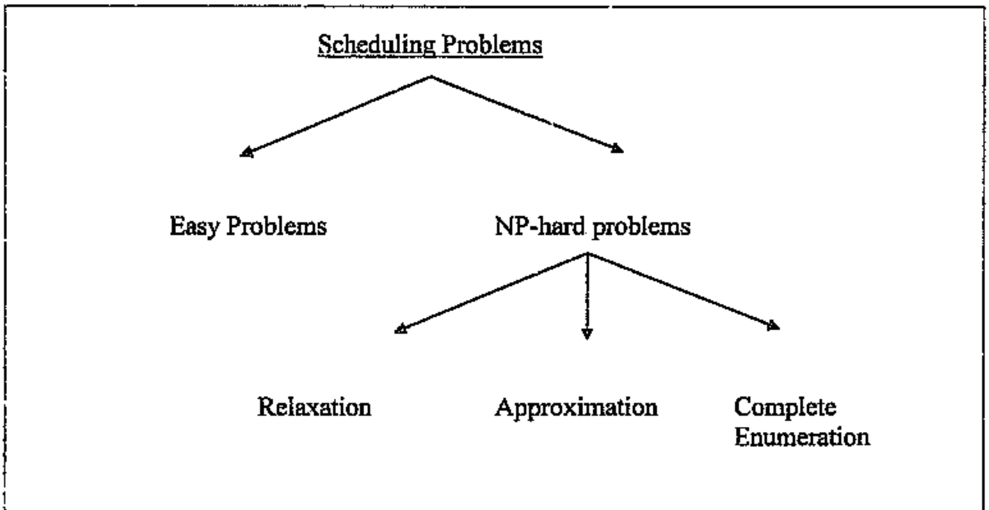


Figure 1.1: Analysis of the scheduling problem.

A number of methods have been employed to solve various types of scheduling problems. Figure 1.2 shows classes of the better known methods that have been used.

Types of Solutions

- Complete enumeration
- Mixed Integer and Non-linear programming
- Branch and Bound
- Dynamic Programming
- Heuristic Methods
  - Priority Rules
  - Bayesian Analysis

Figure 1.2: Types of solutions to scheduling problems.

Machine scheduling problems can be classified according to how the jobs are routed between the machines and various simplifying assumptions regarding the attributes of jobs and machines.

Let $T = \{T_1, T_2, \ldots T_n\}$ represent the set of *n tasks*, and $M = \{M_1, M_2, \ldots M_m\}$ the set of *m machines* to carry out the tasks.

Some of the relevant attributes of each task T_j can be denoted as follows:

1. An *arrival time* (ready time) – r_j, the time at which T_j is ready to begin processing. If all the arrival times for T are equal then it is assumed, without loss of generality, that $r_j = 0$ for $j = 1,2,\ldots,n$.
2. A *processing time* – p_{ij} which is the time needed by machine M_i to complete task T_j, or simply p_j if the processing time is equal on all machines.
3. A *due-date* – d_j. The time when T_j should be completed.
4. The *slacktime* – σ_j. The extra time available to process a task and still meets its due date. $\sigma_j = d_j - (r_j + p_j)$.
5. A *weight* (priority) – w_j, which reflects the urgency or importance of T_j.

Machines can be characterized by the tasks they can complete and the speed at which they can complete them, as shown in Figure 1.3. If they all perform the same functions, they are described as *parallel*. When they are specialized to perform certain specific tasks, they are referred to as *dedicated*. Parallel machines may be further classified into three groups according to their speeds. Firstly, when all the machines have equal processing speeds such that $p_{ij} = p_j$ for $i = 1,2, \ldots ,m$ the machines are referred to as *identical*. Secondly, if their speeds differ by a constant amount that is independent of the task such that $p_{ij} = p_j/b_i$ for $i = 1,2,\ldots,m$ where p_j is the *standard processing time* and b_i is the processing speed factor of machine M_i then the machines are *uniform*. Finally, if the speed of the machine depends on the particular task to be processed then they are called *unrelated*.

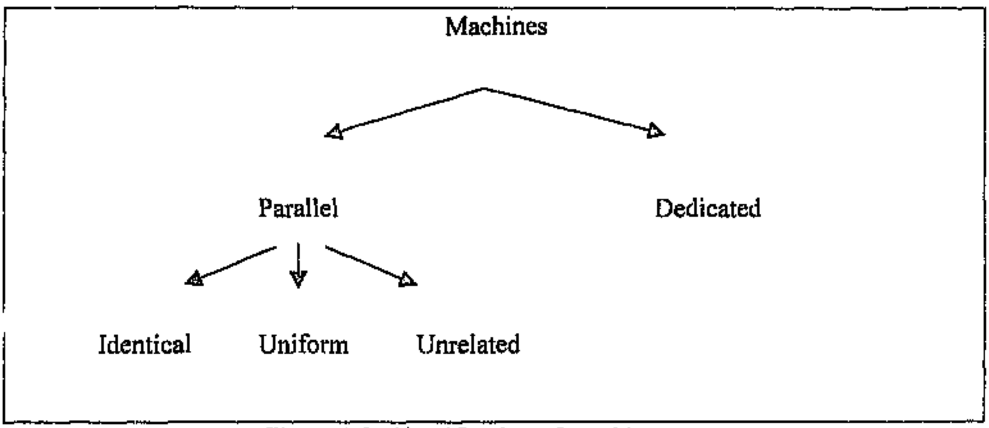


Figure 1.3: Classification of machine types.

When the schedule involves dedicated machines, a multistage task T_j can be decomposed into operations $O_{j1}, O_{j2}, \ldots , O_{jk}$. Each operation may require a different machine, if the flow of work is unidirectional then the environment is called a flow shop. In other words, if each operation in T_j is linearly ordered by an ordering relation $\prec$, then it is possible to number the machines such that if $O_{jn} \prec O_{jn+1}$, then the machine required by O_{jn} has a lower number than the machine required by the O_{jn+1} operation. Figure 1.4 shows the flow of work in a general flow shop.

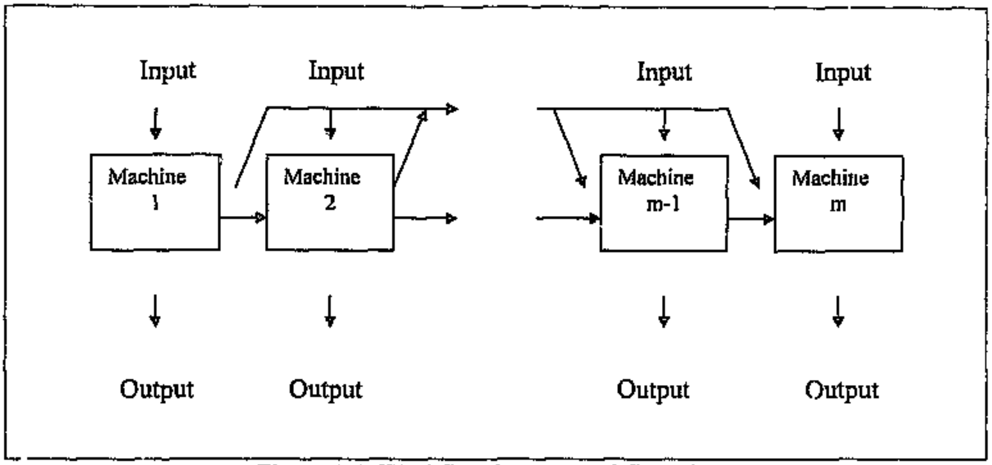


Figure 1.4: Workflow in a general flow shop.

By comparison, the job-shop scheduling problem differs from a flow shop in that the flow of work is not unidirectional. The number of operations per task, their assignment to machines, and the route or order that they take through the system is arbitrary but known in advance. An operation in the job-shop case is described as a triplet *(i, j, k)* in order to denote that operation *j* of task *i* is required on machine *k*.

The most common assumptions and their implications to the complexity and type of scheduling problems are briefly outlined below.

1. The sets of tasks T and machines M are known in advance and fixed.

This assumption distinguishes the static/deterministic problem from the dynamic/stochastic one. When the set of tasks to be scheduled is not known in advance then the problem must be studied by probabilistic methods. Instead of dealing with information regarding the attributes of tasks, the problem is now characterized by the process that generates the tasks. The stochastic problem is concerned with the distributions that characterize the task attributes and makes use of queuing and uncertainty theory to solve the scheduling problem in terms of distributions.

2. All tasks are independent and available for processing at time zero.
   ($r_i = 0$ for $i = 1,2,\ldots,n$).

This assumption may be tightened in two ways. Firstly, the jobs can become available at non-equal integer *ready times*. Secondly, there may be non-simultaneous availability of jobs and mutual dependency between them. This occurs when *precedence constraints* exist between jobs, in other words that set T is partially ordered by $\prec$, such that $T_i \prec T_j$ implies that the processing of T_j cannot start before the completion of T_i. When at least two tasks in T are ordered by the precedence relationship, the tasks are denoted as dependent. The precedence structure of the tasks is usually shown as a directed acyclic graph $H=(V, A)$ with the vertex set $V=\{1,2,\ldots,n\}$ and arc set $A = \{(i,j)|T_i \prec T_j\}$.

3. All machines are available at the same time instant, and remain available during an unlimited period.

Under real world conditions, it is likely that some of the machines will not be available due to breakdowns, stoppages or labour shortages. This assumption regarding the availability of machines is rarely removed in analytical scheduling literature.

4. All jobs remain available during an unlimited period.

In most cases this is an unrealistic assumption, since deadlines for jobs are common in most scheduling environments. The concept of *due-dates* makes provision for this in various scheduling models.

5. Each job can be in one of three states: waiting for the next machine, being processed or finished.

In many scheduling cases, there is little or no storage available for work-in-progress. For instance, a computer operating system has limited buffer space. In some types of manufacturing, there may be no waiting allowed between jobs, for example in the steel industry where the temperature of the metal must be maintained throughout the production process.

6. Each job is processed by all the machines assigned to it, and similarly each machine processes all the jobs assigned to it.

This assumption enforces the deterministic character of the scheduling model. If jobs may be left unfinished or rejected under certain conditions then the scheduling problem falls into a dynamic/stochastic scheduling framework.

7. Each job is processed by one machine at a time.

This assumption can be relaxed under certain conditions, for instance *assembly-type* production can easily be incorporated in the standard job-shop model without any complication. Allowing a job to start on a second machine before being finished on the previous machine is another relaxation of this assumption that has been studied.

8. Each machine processes one job at a time.

Increasing the capacity of a machine M_i from 1 to some number k relaxes this assumption, and is equivalent to assuming that there are k identical machines of type M_i. A simpler assumption that one or more of the M_i are *non-bottleneck* machines is sometimes made. Such a machine is capable of processing all jobs simultaneously, i.e. its capacity is greater than or equal to the number of jobs in the system.

9. All processing times are fixed and sequence-independent.

While the minimum processing time may be a fixed amount, delays may occur that result in the processing time being represented by a random variable with some known distribution. Sequence-dependent processing times occur if the delays are caused by a resource that in turn can be scheduled to minimize delays on other machines.

Sequence-dependent set-up times for a job occur when the set-up time of a job cannot be absorbed into its processing time. The time interval during which a job *j* occupies a machine can be expressed as $S_{ij} + t_j$, where *i* is the job that precedes *j* in sequence. S_{ij} is the set-up time required for job j once job i has been completed, and t_j is the amount of direct processing time required to complete job j.

10. Each operation once started must be completed without interruption.

Dropping this assumption by allowing *job splitting* or *pre-emption* may actually simplify the scheduling problem. A task that can be stopped at any time, and restarted later with no additional cost and perhaps on another machine is said to be pre-emptible. When the above assumption holds the scheduling model is called *non-preemptive*.

11. The tasks have linear processing functions, i.e. the amount a task has been processed depends linearly on the amount of time it has been assigned to a machine.

The above assumptions form the basis for the analysis of scheduling problems and allow the models to be characterized by which assumptions hold and which are relaxed.

The effectiveness of a particular schedule is typically determined by *regular measures* of performance. A function $f(C_1, C_2, \ldots, C_n)$ is regular if it is non-decreasing in every variable. If f is regular

$$f(C_1, \ldots, C_n) < f(C_1', \ldots, C_n')$$

implies that $C_i < C_i'$ for at least one i.

Once a set of tasks T has been scheduled, the following scheduling information becomes available:

Starting time – S_j, which denotes the time task T_j starts processing.

Completion time – C_j, which is the time at which T_j is finished, such that $C_j = S_j + p_j$ (when there is no pre-emption allowed).

Flowtime (F_j). The amount of time T_j spends in the system : $F_j = C_j - r_j$.

Lateness (L_j). The amount of time that T_j exceeds its due date: $L_j = C_j - d_j$

Aggregate quantities from the above information can be used as one-dimensional performance measures. The flowtime measures the interval that a job waits between its arrival and departure, and is thus an indicator of how responsive the schedule is when individual jobs demand competing resources. Three performance measures based on flowtime are:

Mean flowtime: $$\bar{F} = \frac{1}{n}\sum_{j=1}^{n} F_j$$

Weighted Mean flowtime: $$\bar{F}_w = \frac{\sum_{j=1}^{n} w_j F}{\sum_{j=1}^{n} w_j}$$

Maximum Flowtime: $$F_{max} = \max_{1 \le j \le n} F_j$$

Clearly if all the ready times of the tasks in T are 0 then the flowtime is s mply the completion time. The *make-span* or *total production time* corresponds to the maximum completion time $C_{max} = \max_{1,\dots,n} C_i$.

The lateness measure reflects how well the schedule can meet due date demands. Performance measures such as the *maximum lateness* L_{max}, and the *average lateness* $\bar{L}$ are generally used. Often negative lateness can simply be ignored since there may not be any penalty involved in completing a task earlier than necessary. The tardiness (T_i) expresses the positive lateness of a job and is defined as $T_i = \max(0, L_i)$. Aggregate measures of tardiness such as the *maximum tardiness* T_{max}, the *mean tardiness* $\bar{T}$ and the number of tardy jobs can be used.

The above performance measures can be expressed as functions of the set of completion times, which belongs to the class of regular measures. As such it can easily be verified that the above performance measures are also regular. A number of useful relationships between these performance measures exist. The criteria $\overline{F}$, $\overline{C}$, and $\overline{T}$ can be shown to be equivalent, in that, if a schedule is optimal with respect to one of them, then it is optimal with respect to the others. However, a schedule that is optimal with respect to F_{max}, does not imply anything about L_{max} [Conway, *et al.* 67].

A convenient classification of scheduling problems is provided by [Rinnooy Kan, 76]. The classification has the following format: $\alpha \mid \beta \mid \gamma, \Gamma \mid \delta$, where

- α represents the number of jobs, with n representing the general case.
- β represents the number of machines, with m representing the general case.
- γ the type of machine ordering, g = {F flow shop, P a permutation schedule and G indicating the general job-shop problem}.
- Γ indicates dropped assumptions, and will be explained as the notation is used.
- δ indicates which optimality criterion is used.

Schedules can also be categorized into four types, that are useful when dealing with procedures that generate schedules, such as dispatching rules. These are the set of all schedules, the set of semi-active schedules, active schedules and non-delay schedules. Their relation can be seen in Figure 1.5.

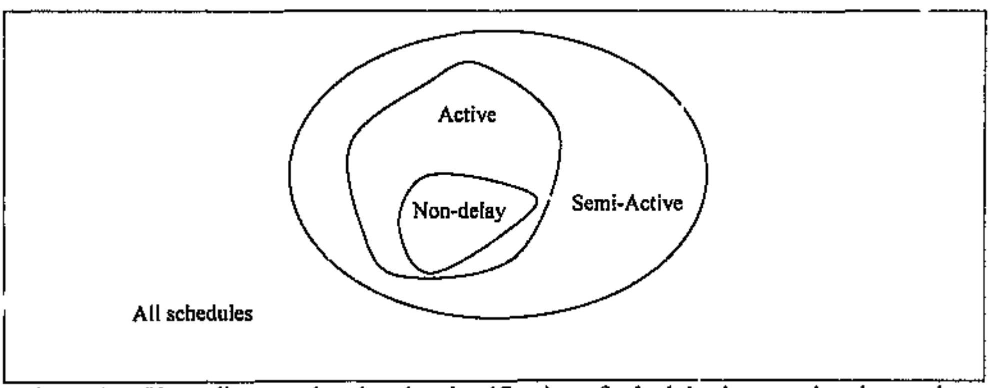


Figure 1.5: Venn diagram showing the classification of schedules into semi-active, active and non-delay. Optimal solutions are always found in the active set.

In theory there are an infinite number of schedules since an arbitrary amount of idle time can always be inserted between jobs to create a new schedule. *Semi-active* schedules are those schedules such that the starting time of no operation can be decreased without altering the processing order on some machine. The cardinality of the semi-active set is finite, since if each job has exactly one operation on each machine, and each machine must process n operations then there are n! possible sequences for each machine. If the sequences on the machines are independent then there can be at most $(n!)^m$ semi-active schedules. This set dominates the set of all schedules, and hence in order to optimize regular measures of performance it is only necessary to consider the semi-active schedules.

Active schedules in turn are those semi-active schedules in which it is not possible to decrease the starting time of any operation without increasing the starting time of a least one other operation. The set of active schedules dominates the semi-active set and must contain an optimal schedule with respect to every regular measure. The number of active schedules still tends to be too large for the set to be effectively enumerated, a smaller subset of active schedules called *non-delay* schedules may be considered. In a non-delay schedule, no machine is kept idle at a time when it could be processing some operation. While the non-delay set is smaller, there is no guarantee that it contains the optimum. However, it has been shown in [Conway, *et al.* 67] that the random generation of non-delay schedules seems to provide better schedules on average than a similar generation of active schedules.

The above terminology and definitions of scheduling environments and its characteristics provide a conceptual framework for the problems and research of scheduling solutions, more of which is provided in the literature review.

1.2 Automated Guided Vehicles (AGVs)

Many manufacturing environments require a material handling system (MHS) to transport raw material, work-in-progress and finished goods between various locations.

Automated Guided Vehicles (AGVs) are being increasingly used as a flexible and more efficient mechanism to reduce MHS times. These vehicles are driverless, and can be programmed from a system controller to travel along a predetermined route. The control system dispatches idle vehicles to carry materials from one processing or storage station to another. Research into the design of efficient on-line control algorithms for AGVs has a far shorter history in operations research than general scheduling, with most significant papers related to this field being written from 1980 onwards. In general, this control problem has three main issues: dispatching, routing and scheduling. The last being the amalgamation of dispatching and routing with time constraints.

Most AGV studies have been focused on design issues, such as determining the number of vehicles required, flow path design, and route planning [Choi, *et al.* 94] and [Taghaboni-Dutta and Tanchoco, 95]. Operational studies on vehicle dispatching and traffic management have not been studied to the same extent. The performance and behavior of various dispatching rules, under differing environmental conditions, have been the predominant studies in this field.

The crane problem can be expressed as an AGV problem. The cranes can be considered as two bi-directional vehicles, both running on a shared acyclic route. The route has no detours at pickup and delivery points and the vehicles are required to service overlapping stations. Since the number of tasks at any instance will generally be greater than the number of vehicles available, the problem can be classified as a vehicle initiated assignment problem [Egbelu and Tanchoco, 84].

1.3 Definition of the crane problem

The following definitions are provided as a framework for representing the crane scheduling problem. A descriptive model of a crane system by [Lieberman and Turksen, 81] is presented below.

The crane system is defined as $\mu = \langle \mathbf{C}, \mathbf{L}, \mathbf{J}, \mathbf{R}, \mathbf{T} \rangle$ where

C is the set of m cranes $\{C_i \mid i = 1, 2, \ldots, m\}$.

L is the set of N job locations $\{l_i \mid i = 1, 2, \ldots, N\}$.

J is the set of n jobs available $\{J_i \mid i = 1, 2, \ldots, n\}$.

R is the set of job ready times $\{r_i \mid r_i \geq 0,\ i = 1, \ldots, n\}$

T is the set of operation process times.

A representation of the sets **C** and **L** is shown in Figure 1.6 below.

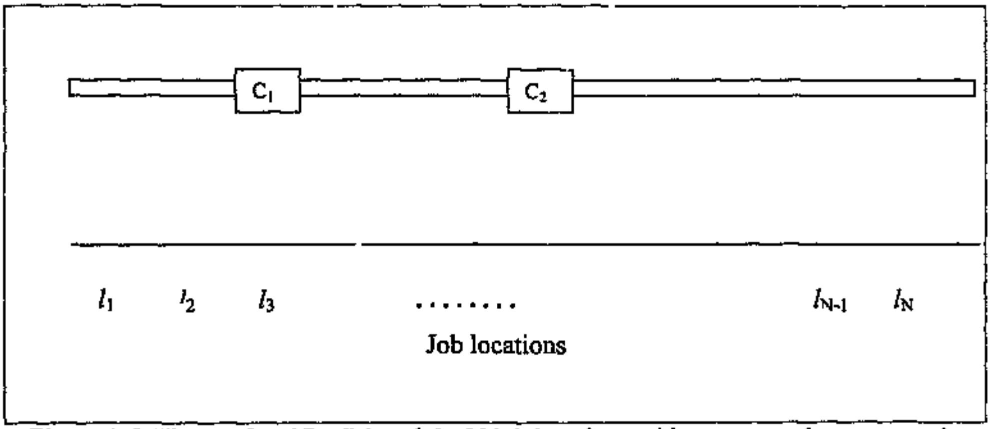


Figure 1.6: The set **C** = $\{C_1, C_2\}$ and the N job locations with respect to the crane track.

The job locations l_i are defined as the physical distance of the location from the leftmost side of the crane track. Each job J_i is defined by the locations that the crane must move to in order for the job to be completed. Thus, a job consists of one or more ordered job locations that the crane must traverse. J_i can be defined as:

$$J_i = \{l_{i1}, l_{i2}, \ldots, l_{iq_i}\} \text{ where } l_{ij} \in \mathbf{L} \text{ and } q_i \text{ is the number of operations for job } J_i$$

Given that l_{ij} is the job location of the jth operation of the ith job, then τ_{ij} is defined as the processing time associated with the operation l_{ij}. The set of processing times is

$$\mathbf{T} = \{\tau_{ij} \mid \tau_{ij} \geq 0,\ i = 1, \ldots, n,\ j = 1, \ldots, q_i\}$$

Further definitions and notation regarding characteristics of crane problems follow.

- The set-up time S_{ij} of J_j following J_i is defined as:

$$S_{ij} = k \cdot \left| l_{iq_i} - l_{j1} \right|$$

where k is some constant related to the speed of the crane, see Figure 1.7. This definition is appropriate when considering set-up times created by job sequencing, however the following set-up definition may also be used:

$$S_{ij} = k \cdot \left| l(t) - l_{j1} \right|$$

where *l(t)* refers to the location of crane *c* at time *t*. In most cases $l(t) = l_{iq_i}$, i.e. the crane position will be at its last drop-off point.

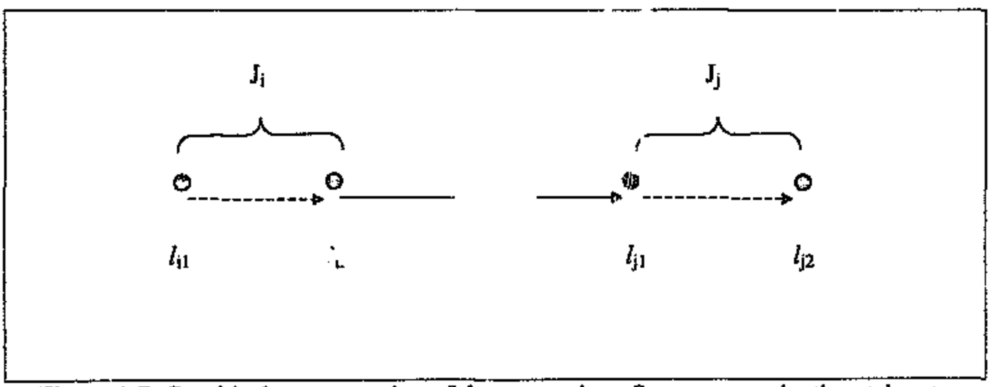


Figure 1.7: Graphical representation of the set-up time. S_{ij} represents the time taken to move from the last job location of J_i to the first job location of J_j.

- The process time P_i of job J_i is defined as:

$$P_i = \sum_{j=1}^{q_i-1}\left(\tau_{ij} + k\cdot\left| l_{ij} - l_{ij+1} \right|\right) + \tau_{iq_i}$$

The above definitions of the processing time and the set-up time are local, in that they do not take into consideration how they may be influenced by the state of the other cranes in the system. The following global definitions of set-up and processing time take all cranes into account.

Let S'_{tj} be the expected set-up time for job j at time t given the current location of the crane and global state of the system as

$$S'_{tj} = S_{tj} + \chi_s(t)$$

where $\chi_s(t)$ is extra time due to crane conflicts during set-up, i.e. when two or more cranes both require the same section of track at a particular time.

Similarly let P'_{it} be the expected time to process job i at time t given the global state of the system as

$$P'_{it_p} = P_i + \chi_p(t_p)$$

where t_p refers to the time we expect to begin processing job i and $\chi_p(t)$ represents the additional time needed to resolve any conflicts while performing the task, given the state of the system at time t.

The values of $\chi_s(t)$ and $\chi_p(t)$ depend on how the crane movements are scheduled. These values are a result of the dependence of the set-up and processing times on the state of the other crane in the system, but have no bearing on the spatial dependence of the set-up movement between jobs.

The current state of the second crane can be classified as idle, moving, or busy, where busy denotes a crane that is not moving, but completing some operation. The tasks that the cranes have to perform can be classified into those that consist of two jobs, and those that consist of three jobs. A state diagram of a task consisting of two jobs is shown in Figure 1.8.

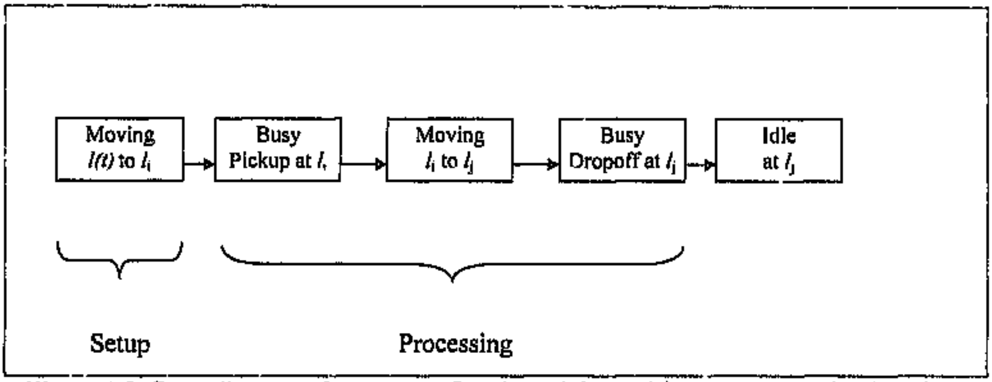


Figure 1.8: State diagram of a crane performing a job requiring two processing locations at l_i and l_j.

Let $L_{ij} = \{l_x \mid i \le x \le j\}$, be the set of all locations between l_i and l_j. Then the range of job locations for job J_i, given the crane is currently at $l(t)$ can be defined as

$$R_i = L_{lr} \text{ where } l_l = \min(\, l(t), \min(\, l_{ij}\,, j = 1 \ldots q_i)) \text{ and } l_r = \max(\, l(t), \max\, (l_{ij}, j = 1 \ldots q_i\,))$$

The range is thus the set of all locations between the minimum and maximum locations that the crane will travel to while processing a job.

If c is the crane currently being scheduled and c' is the other crane in the system, then no conflicts will occur between the cranes under the following conditions:

- c' is idle.

- R_i and R_j are disjoint, where one crane is processing job J_i and the other job J_j.

- c' is moving in the same direction as c and will not impede c at any time, i.e. it will either move beyond l_j or become idle before c reaches it.

- c' is moving in the opposite direction but will become idle before c reaches it.

When crane interference does take place, the resulting conflict can be classified into two types, namely, *busy* and *slaved*.

A *busy* conflict occurs when one crane has to wait for another crane to finish its current operation before it can continue to its destination. For example, crane c' may be busy with a pickup or drop-off operation that it must complete before it can move out of the way of crane c.

Let β, the time taken by a *busy* conflict, equal the difference between the time taken to complete an operation τ_{ij} by crane c' and the time of arrival of crane c at location l_{ij} such that

$$\beta = \max \{ 0, \phi_{ij} - t(l_j)\}$$

where ϕ_{ij} represents the time that operation τ_{ij} is completed by crane c', and $t(l_{ij})$ the time at which crane c arrives at location l_{ij}.

A *slaved* conflict occurs when a crane is *pushed* out of the way by another crane. A *slaved* conflict always incorporates a *busy* conflict, since the crane that is pushing the other crane will only do so if it needs to complete some operation. Thus once the crane reaches its location, the *slaved* crane will have to wait for that operation to be completed before it can move back to its original position.

The *slaved* conflict time μ, represents the time that crane c must wait for c' to reach its location and finish the required operation, in addition to the time that it takes for crane c to move back to its original position.

$$\mu = 2k \cdot \left| l_s - l_{ij} \right| + \tau_{ij}$$

where l_s is the location at which crane c is slaved to crane c', l_{ij} is the location that c' is moving to, and τ_{ij} is the length of the operation that c' must complete at l_{ij}.

Various combinations of *busy* and *slaved* conflicts can occur, depending on the number of locations that must be visited in order to finish a job, and the control strategy that determines which crane is slaved during a *slaved* conflict.

The following example illustrates the types of conflicts that occur when crane c is trying to complete a set-up movement from location $l(t)$ to l_x and c' is busy with a two operation job $J_1 = \{l_{11}, l_{12}\}$.

- A single busy conflict occurs if crane c reaches l_{11} (or l_{12}) before c' is finished, see Figure 1.9.

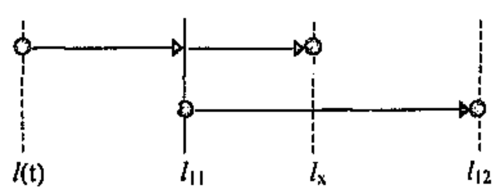


Figure 1.9: Single *busy* conflict at l_{11}.

- Two busy conflicts. Crane c has to wait for c' to finish its operations at both l_{11} and l_{12}, see Figure 1.10.

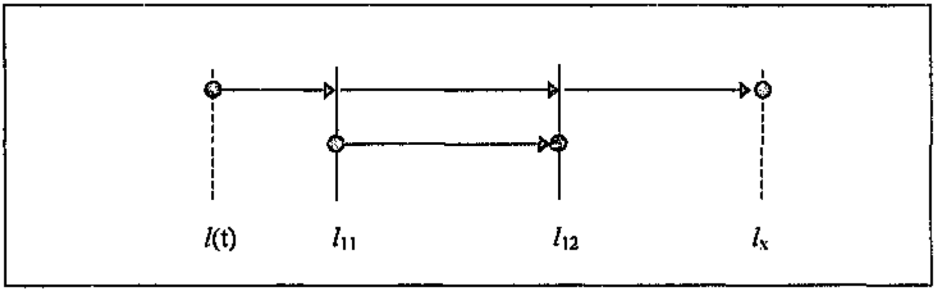


Figure 1.10: Two *busy* conflicts at l_{11} and l_{12}.

- A single slaved conflict. Crane c is slaved by c' from location l_s to l_{12}, and must wait for c' to complete its operation at l_{12} before it can move again. Figure 1.11 shows the slaved movement of c as a dashed line.

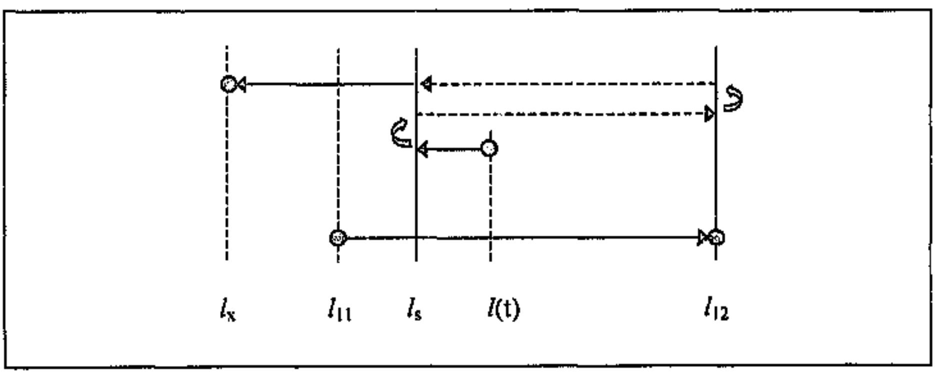


Figure 1.11: Single *slaved* conflict from l_s to l_{12}.

- A busy and slaved conflict. This case is similar to the above example in Figure 1.11, however, this time $l_s = l_{11}$. Crane c thus experiences a busy conflict first, and is then slaved to l_{12}, as Figure 1.12 shows.

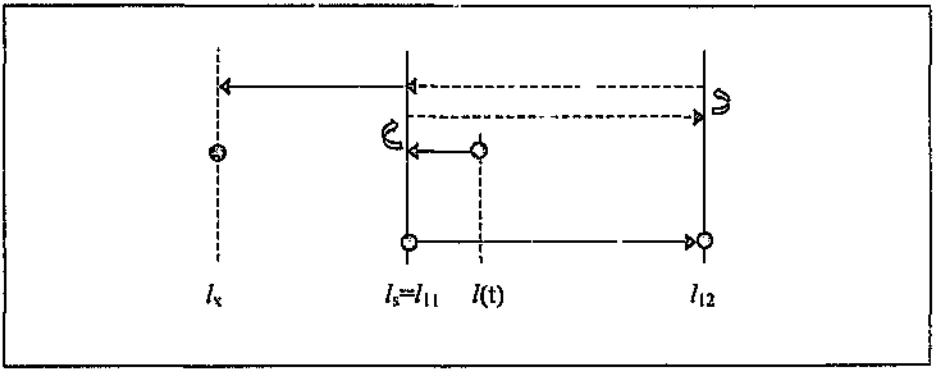


Figure 1.12: A *busy* conflict at l_{11} and a *slave* conflict from l_{11} to l_{12}.

The value of $\chi_s(t)$ can now be formulated for the conditions given in the above example:

Let b be the number of *busy* conflicts and β_i the time taken by the ith *busy* conflict, similarly let s be the number of *slaved* conflicts and μ_j the time for the jth *slaved* conflict. Then

$$\chi_s(t) = \begin{cases} 0 \text{ when no conflict occurs} \\ \sum_{i=1}^{b} \beta_i + \sum_{j=1}^{s} \mu_j \end{cases}$$

where b ≤ 2 and s ≤ 1.

Clearly, the delays can only be calculated once the control strategy of the cranes is known, as this will affect which cranes become *slaved*. This strategy may be static or dynamic in nature. Under static conditions, the scheduling system is unable to control the behavior of the cranes once a job as been assigned. The dynamic situation gives the scheduler more control over the cranes' movement, in effect scheduling both inter-job and intra-job crane movements.

The above definitions of the crane problem show that it can be considered as a dynamic two machine problem, with sequence-dependent set-ups, and processing times that are affected by delays. The delays are caused by interference between the two cranes, and hence can be minimized by sequencing the movements of the cranes. The uncertainty regarding the actual processing time of a crane task is therefore sequence-dependent on previous jobs and the state of the other cranes.

2 LITERATURE REVIEW

2.1 Introduction – Development of Scheduling

Sequencing and scheduling problems occur whenever an efficient allocation of resources to tasks is required. The environments in which schedules are required cover a wide variety of situations, from railway timetabling through to production scheduling. Computers have proved not only to be tools for the construction of schedules, but have also provided new environments for scheduling applications. Multiprocessor scheduling, robot activity scheduling, large scale network scheduling and hard real-time scheduling applications have introduced a number of new problems.

Due to the complexity of the scheduling problem, analytical studies have been confined to solving scheduling problems with various restrictive assumptions. Several theoretical results for a number of instances of machine scheduling are provided in [Conway, *et al.* 67], [Baker, 74] and [Rinnooy Kan, 76]. However, the assumptions used to find optimal solutions to certain problems, have resulted in a number of discrepancies between practice and theory in production scheduling, which [Graves, 81] highlights. McKay contends that many theoretical formulations may be irrelevant, as they do not capture the intricacies of real-world scheduling problems [McKay, *et al.* 88]. The need to schedule in environments that do not conform to theoretical worlds has resulted in many different approaches being used, from combinatorial analysis through to control theory, uncertainty theory, and artificial intelligence. The use of machine learning to act as a knowledge acquisition tool for dynamic scheduling systems [Nakasuka and Yoshida, 92] and various other knowledge-based approaches have also been studied [Noronha and Sarma, 91]. A survey regarding the various methods being applied to production scheduling in recent years is provided in [Rodammer and White, 88].

2.2 Simple Deterministic Static Problems

Most production environments are stochastic and dynamic. Scheduling models on the other hand have generally been deterministic and static. In a static environment the requirements are finite and fully specified in advance, with the assumption that no additional requirements will be added and none of the existing ones will be altered. An overview of the major work done in this area can be found in [Graves, 81] and [Sen and Gupta, 84]. These problems can be broadly classified into four main areas: one-machine problems, the parallel machine model, the flow shop problem and the job-shop problem.

The one-machine problem or one-stage, one-processor problem is the most popular scheduling model. Two procedures which determine the optimal task sequence by a simple ordering procedure are the *shortest-processing time* (SPT) rule, and the *earliest due-date* (EDD) rule or *Jackson's Rule* [Baker, 74]. The SPT rule solves the $n/1//\overline{F}$ problem sequencing the jobs in order of nondecreasing processing-time and EDD solves the $n/1//L_{max}$, by sequencing jobs in order of nondecreasing due-dates requiring O(n log n) steps [Rinnooy Kan, 76]. However, simply changing the ready times of the jobs creates the $n/1/r_n \geq 0/L_{max}$ problem. This problem has been shown to be NP-complete by a reduction of the KNAPSACK problem [Rinnooy Kan, 76].

The one-machine problem is generally too simple for any practical shop floor use, however an important generalization of it, the parallel processor problem often occurs in industry.

The simplest environment for multistage jobs is the flow-shop, as shown in Figure 1.4. The most general and difficult scheduling problems are associated with the job-shop and open-shop environments.

2.3 Heuristics and Dispatching Rules

The complexity of many scheduling models imposes computational requirements that are too severe for large problems, and even for relatively small problems there is often no guarantee of finding a solution quickly enough to suit the environment. Heuristic algorithms are able to provide solutions with limited computational effort, but they do not guarantee optimality, and it may be difficult to judge their effectiveness. While the restrictive assumptions made in scheduling theory may over simplify the real world conditions, the results gained from the models provide useful insights when developing heuristic rules. A survey of dispatching rules for manufacturing environments is presented in [Blackstone, *et al.* 82].

Two methods that have been extensively studied for their general applicability are *priority rules* and *Bayesian analysis*. An overview of these approaches is discussed in [Rinnooy Kan, 76].

[Gere, 66] defines a priority rule as a function that assigns a value to each waiting job, and schedules the job with the lowest value first. An early study by [Jeremiah, *et al.* 64] examined a number of factors influencing priority rules. Some of the information that can be used effectively is:

SPT (Shortest Processing Time): Select the operation with the shortest processing time.

FCFS (First come, First Serve): Select the operation that first becomes available for further processing.

MWKR (Most Work Remaining): Select the operation which has the most work remaining.

MOPNR (Most Operations Remaining): Select the job that has the highest number of operations remaining.

LWKR (Least Work Remaining): Select the operation that has the least work remaining.

RANDOM : Select the operation at random.

The study is summarized by [Conway, *et al.* 67] and [Baker, 74]. The results showed that no one rule dominated all the others. The most significant result is that non-delay dispatching provided a better basis for heuristic schedule generation than active scheduling. The MWKR rule and some of its derivatives often produce better schedules than the other rules in terms of minimizing the makespan of the schedule. SPT and LWKR tend to be superior to the others when the criterion is minimizing the mean flow time. However the RANDOM rule was also found to perform well under this measure of performance.

[Gere, 66] considers a rule to be random if it does not take into account any information regarding the state of the jobs or machines. For example, the first in first served rule (FIFS) is considered a random rule since it does not take into account any information regarding the attributes of the job. He also differentiates between priority and heuristic rules. A heuristic rule by Gere's definition is a rule that can take into consideration other aspects of the environment, and take exception to what the priority rule suggests as being the best choice of operation. Heuristic rules generally require more complex considerations of the environment and rely on anticipating future conditions, the effects of alternate operations, or qualitative reasoning [Panwalker and Iskander, 77]. In a study on the effect of various heuristic rules Gere made two interesting conclusions. Firstly, when the goal is to meet due dates, the he[illegible]hat pass specific knowledge to the priority rule are more important than the priority rule itself. Secondly, that there is little difference in the effectiveness of priority rules after they are combined with one or more heuristics, and hence a simple priority rule could be used.

The heuristics that Gere found to improve the schedules were the alternate operation and look-ahead heuristic combined with the insert rule. The alternate operation checks to see if the application of the priority rule makes another job *critical* (if the slack of any other job has become negative or reached a certain predefined critical level). If so, the last operation is revoked, the next best operation is scheduled according to the rule, and a check for critical jobs is once again made. The look-ahead heuristic tests to see if a critical job will reach a machine at some future given time, yet before the scheduled operation is completed. If this is the case, then the critical job is scheduled, and the effect on other jobs is checked. The new schedule either remains, or is replaced with the previous operation suggested by the priority rule, depending on how the lateness of the jobs have been affected. The insert heuristic can be used in conjunction with the look-ahead rule. If an idle gap exists between the job to be scheduled by the look-ahead rule and the present time, then the longest operation that can be fitted into this gap is inserted.

These results seem to show, that by themselves, priority rules are insufficient for the scheduling of complex problems. This is demonstrated by the fact that no rule dominated any other, including the RANDOM rule. Gere also confirms this by concluding that the heuristic rule is more important than the priority rule.

However dispatching procedures (decisions that are taken in the order of implementation and never revoked) still have numerous advantages in dynamic/stochastic environments. They are able to provide real-time response and can be tuned with the use of heuristic rules for specific scheduling environments. For this reason a great deal of research is still being done on various combinations of rules of this type.

There are a number of ways that dispatching rules can be classified. Rules can be static, with job priority values that do not change as a function of time, or dynamic, reflecting the status of jobs from time to time as the schedule progresses. They can also be classed as general rules which cover a broader range of scheduling problems but may trade this flexibility with loss of performance, or they can be specifically formulated for the characteristics of the given environment.

Over a hundred scheduling rules are classified in [Panwalker and Iskander, 77], who use the following broad definition of the types of priority rules. *Simple*; these rules require information related to a specific job, e due date, processing time, etc. Consideration of the queue length at the next machine that the job will visit is also considered as a simple rule. *Combination*; applying different dispatching rules as the environment changes, or applying different rules to different jobs in the queue depending on their attributes. Finally, *Weighted Priority Indexes*, are weighted combinations of the above rules.

An extended dispatching rule (EDR) approach is presented in [Ishii and Muraki, 96]. The EDR method applies the best dispatching rule depending on the process states. A simple procedure, the EDR search algorithm, is used to find an appropriate dispatching rule combination. It consists of two stages; in the first, the best single dispatching rule is selected as the initial combination for the search process, called the current best dispatching rule combination (CBRC). The next stage attempts to improve the CBRC sequence by replacing dispatching rules in the CBRC with alternative rules. The new combination is tested using a dispatching-simulation mechanism. If it improves the performance the CBRC is updated. One dispatching rule is replaced at a time, and terminated when replacement reaches the last rule in the CBRC.

Whether the assumption that actual processing times for jobs are deterministic or stochastic has any significant bearing on the performance of dispatching rules is discussed in [Elvers and Taube, 83].

Dispatching rules for flexible manufacturing are discussed in [Chandra and Talavage, 91] and a transient-based real-time scheduling algorithm, that selects dispatching rules dynamically for short time periods in order to respond to changes in the state of the system, is given by [Ishii and Talavage, 91].

2.4 Scheduling Problems with Set-up Times

The usual assumption for job-shop scheduling research is that the jobs are sequence independent. In many cases however, set-up times of jobs are sequence dependent, and the time taken to perform the set-up or change over is a function of the preceding job. This occurs for instance when die or tool changing is needed in a metal processing shop, or a particular program must be loaded into memory to perform a task. The majority of research in this field has concentrated on static job arrival patterns, a summary of past research can be found in [Kim and Bobrowski, 94].

The problem is formulated by [Bruno and Downey, 78] as follows. Given *n* disjoint classes of tasks $C_1, \ldots, C_n$ such that each C_i has a *set-up task* S_i with *set-up time* $\tau(S_i)$, and some *set-up cost* (*change-over cost*) $c(S_i)$. Then assuming the schedule is non-preemptive, that every task has a deadline and that there is only one machine, does there exits a schedule for all the tasks in C such that all the non-setup tasks finish before their due dates? This problem is called the *feasibility problem*. An example of a Gantt representation of this type of schedule is shown in Figure 2.1.

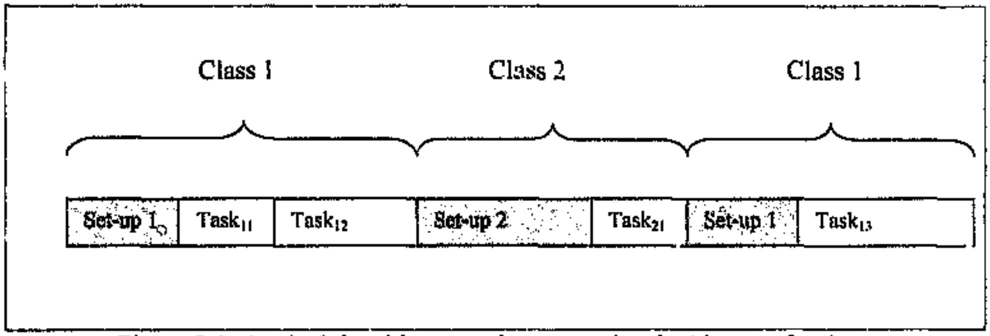


Figure 2.1: A schedule with set-up times associated with sets of tasks.

Bruno and Downey show that the since the Knapsack Problem can be reduced to the feasibility problem that it is NP-hard [Bruno and Downey, 78]. Even when the set-up times are equal the problem remains NP-complete. When the number of identical machines is increased to two then all the above scheduling problems can be shown to be NP-hard, by reduction to the Partition Problem.

In the above problem the set-up time is a function of the class only. When the set-up time is a function of both the current class and the previous class, then the set-up time is sequence-dependent. Baker shows that when all due dates are identical, that the single machine case of this problem can be shown to be equivalent to the Travelling Salesman Problem (TSP) which is NP-hard [Baker, 74]. When the set-up times have arbitrary values, and are equated with arbitrary inter-city distances in the TSP, then Sahni and Gonzalez show that no polynomial-time algorithm can yield a fixed data-independent worst-case error bound [Sahni and Gonzalez, 76].

Ovacik and Uzsoy, however, are able to provide tight data independent worst case bounds for list scheduling heuristics, when set-up times are bound by the processing times [Ovacik and Uzsoy, 93]. List scheduling algorithms generate non-delay schedules, where no machine is kept idle if there is a job available for processing. List schedules are based on some permutation of the jobs, such that whenever a machine becomes idle, the unscheduled job at the head of the list is scheduled on that machine. Clearly such a procedure is restricted to identical parallel machine problems. However, when set-up times are introduced, non-delay schedules are no longer dominant and the optimal schedule need not be a list schedule. The worst case performance of a list schedule can be quantified when the set-up times have some special structure, as [Ovacik and Uzsoy, 93] show using the assumption that the set-up time is always smaller than the processing time, i.e. $s_{ij} \leq p_j$.

The effect of sequence-dependent set-up times is examined by [Kim and Bobrowski, 94]. They show that ordinary sequencing rules do not perform as well as set-up orientated rules, and that sequence-dependence has a significant impact on shop performance. The sequencing rules are classified as *set-up orientated* sequencing rules if they use information regarding set-up procedures. The JCR (Job of smallest Critical Ratio) and SIMSET (Similar Set-up) rules represent set-up orientated rules. The JCR heuristic looks for a job that is ready for processing and that is identical to the job that it has just processed. When no such job exists then the job with the smallest critical ratio is scheduled. The SIMSET procedure simply selects the next job based on minimizing the set-up time. The experimental results using these rules showed that rules using set-up information provided increased throughput, better machine utilization and showed less variation when meeting due-dates.

2.5 AGVs and Material Handling Devices

Most Automated Guided Vehicle studies have been focused on design issues, such a determining the number of vehicles required, flow path design and route planning. There have been far fewer operational studies on vehicle dispatching and traffic management. Most have been with regard to how various dispatching rules perform under certain conditions.

Three of the most popular dispatching rules are the SDT (Shortest Travel Time) rule, MQS (Maximum Queue Size) rule, and the LWT(Longest Waiting Time) rule. The SDT rule aims at minimizing the travel time of empty vehicles, and was found by [Egbula and Tanchoca, 84] to be a very powerful and robust heuristic because, the efficiency of a material handling system is usually determined by the speed at which components can be moved to the next destination. However the performance of the SDT rule is highly dependent on the layout of the departments that must be serviced as well as the locations of pickup and delivery points.

A hierarchical on-line dispatching algorithm for scheduling jobs on machines and AGVs is proposed in [Sabuncuoglu and Hommertzheim, 92a]. The algorithm takes into account interactions between machines and AGVs during the scheduling process. Most dispatching rules consider the machines and AGVs as independent sets, and do not use any information regarding the state of the other system when making a scheduling decision. The AGV algorithm assigns the next task to an awaiting vehicle using a hierarchical decision process. The first level checks critical workstations that are blocked or have full queues. A number of criteria are then applied to determine which workstation should be serviced first if a number of workstations are classified as critical. The second level checks if there are any jobs waiting in central buffers that can be moved to a workstation queue. Next, if any idle stations are found, then an AGV will be dispatched to bring work to it, using either a SDT or LWT rule. Finally, if there are no idle or critical machines, and the buffers are empty then the AGVs are dispatched to the workstations that are most likely to finish first.

The hierarchical algorithm discussed above was compared to various dispatching combinations for machines and AGVs. The SPT/LQS (Shortest Processing Time / Largest Queue Size) and SPT/SDT (Shortest Processing Time/Smallest Distance Traveled) were the main rules used as a comparison. These were previously found by [Sabuncuoglu and Hommertzheim, 92b] to be the best rule combinations against the mean-flow time criterion. The results showed the hierarchical algorithm's ability to use information about both machines and AGVs did improve performance, especially at high load level (utilization rates), but was only slightly better when the load level was low.

A control and scheduling mechanism for AGVs is presented in [Aktruk and Yilmaz, 96] that considers the interaction of an AGV system with the rest of the decision making hierarchy found in a manufacturing environment. The automated manufacturing research facility (AMRF) model, a five level decision making hierarchy, is relaxed to allow the AGV system to provide feedback regarding scheduling decisions made at the shop and cell levels. A micro-opportunistic scheduling algorithm (MOSA) is proposed by Aktruck and Yilmaz that can simultaneously consider both critical jobs from a job-based scheduling perspective, and the unloaded-travel times of the AGVs from a vehicle-based viewpoint. On average MOSA outperformed the simpler dispatching rules such as shortest travel time first (STTF), earliest due-date (EDD), earliest release-time (ER), and the Rachamadagu-Morton (RM) rules. However, there was no dominant rule since each rule performed the best in at least one run.

Based on earlier work on finding conflict-free shortest time routes for bidirectional vehicles in [Kim and Tanchoco, 91], a myopic strategy for working in dynamic environments is presented in [Kim and Tanchoco, 93]. The method is myopic in that it considers only one vehicle at a time, and adheres strictly to all previously made scheduling decisions. The study looks at whether the benefit of a bidirectional AGV system over its unidirectional counterpart is significant in terms of throughput and flowtime. [Kim and Tanchoco, 93] found that the bidirectional system outperformed the unidirectional system in terms of throughput, but an upper bound is reached as the number of vehicles is increased. The difference in throughput also decreases as the P/T ratio increases, where the P/T ratio is the ratio of the average processing time per operation to the average transport time per transfer. This is mainly as the impact of the AGV system becomes less as the P/T ratio increases.

AGV dispatching in a Just-In-Time environment is studied by [Occeña and Yokota, 91]. They propose a heuristic that is sensitive to different degrees of demand in the JIT environment. The maximum demand (MD) dispatching rule prioritizes which stations should be serviced first by using threshold values on the input and output queues of each workstation. Departments that have no work (i.e. are starving) are given first priority followed by departments with the most number of service requests. The rule produces higher performance than AGV dispatching rules in terms of both transport performance measured by throughput and logistic performance measure by the total average inventory level.

Four vehicle-initiated AGV rules are examined by [Lee, 96]. These include three composite rules, which combine the concepts of shortest distance and maximum outgoing queue size. Results show that the Nearest-station/Stay-in-Same-Station rule is the best on average in terms of throughput and flowtime. This rule works by sending the AGV to the nearest station with work, and if there is no work it moves to the nearest pickup station on its route.

Klien and Kim, generalise the composite rule approach and examine the performance of multi-attribute dispatching rules [Klien and Kim, 96]. Since AGV dispatching is a multi-attribute decision-making problem, it is argued that a multi-criteria decision should be superior or at least equal in performance to a single-criterion solution process. A simple additive weighting method (SAWM) heuristic as well as more complex fuzzy logic based decision making procedures were examined. The STD rule is superior to other single-attribute dispatching rules and comparable to multi-attribute methods when the workstation lay-out is such that no department is being ignored [Klien and Lee, 96]. Even though the multi-attribute methods are unable to find optimal solutions, they outperform the single-attribute rules.

Numerous other techniques have been used to schedule and control AGV systems. A LISP driven controller for scheduling free-ranging AGVs is described in [Taghaboni and Tanchoco, 88]. A shortest travel time dispatching rule is used, with a subroutine incorporated in their routing procedure to check if more than one vehicle can pass an intersection simultaneously without crossing each other's paths. [Krishnamurthy, *et al.*

93] propose a column-generation based heuristic to find conflict free routes for multiple AGVs to minimise the makespan. Studies using neural-networks have also been conducted. [Hao and Lai, 96] propose a new methodology for the quasi real-time control of an AGV system using a self-organizing network, and show encouraging results, supporting the potential of such a technique.

2.6 Rolling Horizon Methods

While most practical job-shop and AGV scheduling problems use dispatching rules, their rather myopic view of the environment may lead to poor long-term performance. At the other extreme, if all jobs in the system, both current and future, are considered as a single problem, an exact solution could be found. However the computational burden of such a procedure renders it impractical for problems of a realistic size. Rolling horizon procedures (RHP) allow for the explicit trade-off between solution quality and computational time, through the choice of parameter values that define the size and number of subproblems.

Ovacik and Uzsoy, use RHP to decompose the dynamic scheduling of machines with sequence-dependent set-up times, into a subproblem that consists of the jobs on hand and a subset of the jobs that will arrive in the near future [Ovacik and Uzsoy, 94]. The overall solution of the problem is approximated by the solutions to the successive subproblems. This decompositional approach allows some degree of forward scheduling visibility combined with optimization procedures that can focus on smaller problems and thus make explicit use of due dates and set-up times. The RHP algorithm used by [Ovacik and Uzsoy, 94] consists of three decision parameters. The length of the forecast window, which denotes the period within which the arrival times of future jobs can be predicted, the maximum number of jobs that will be considered at any decision point, and finally the maximum number of jobs that are scheduled at a decision point. The subproblems are then solved using a hybrid of a depth-first and best-bound search. The results indicated that the RHP algorithm was able to produce results substantially better than dispatching rules, with the added flexibility of being able to explicitly trade off solution time and quality by the parameter choices.

The above approach is extended to dynamic parallel machines in [Ovacik and Uzsoy, 95]. However a branch and bound approach is not used to solve the subproblems, since it is hard to find efficient branching schemes due to the non-dominance of list schedules, and it is also difficult to find effective lower bounds due to the presence of sequence-dependent setup-times.

2.7 Crane Related Problems

Research regarding crane scheduling is mostly limited to a single crane operating in a flow-shop type production environment. A typical example of such a system occurs when electronic circuit boards are chemically treated in a sequence of tanks, see Figure 2.2, [Phillips and Unger, 76] and [Ge and Yih, 95]. The cranes perform inter-tank transfers of the jobs where each move consists of lifting the unit from a tank, moving to the next tank and finally submerging the unit in the new tank. Tanks can only process one unit at a time and no inter-tank buffers exist. An additional constraint is often imposed on the minimum and maximum time that a unit can remain in a tank. These intervals are called *time window constraints*.

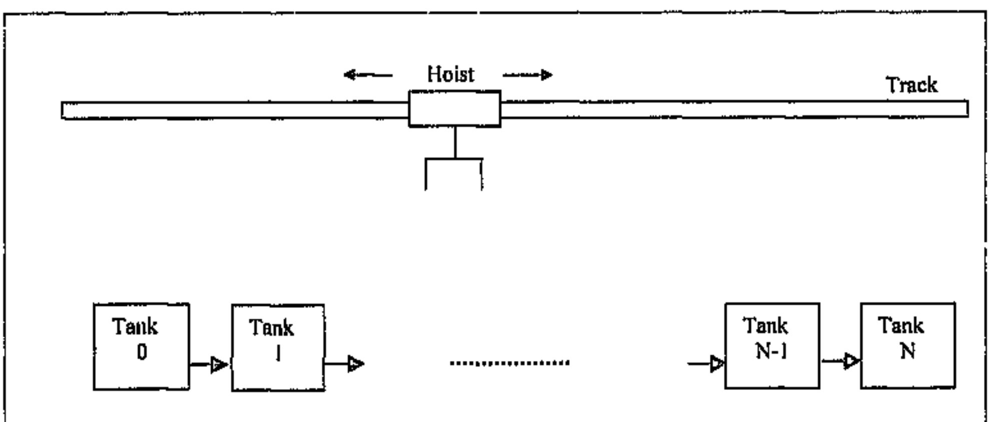


Figure 2.2: Flow of work in an electroplating line. The hoist moves work-in-progress from one tank to the next.

Several studies have focused on the creation of cyclic-schedules for flow-type environments. The cranes are assigned a fixed sequence of moves, which are performed repeatedly. Each repetition of the move sequence is called a *cycle* and the time to complete a cycle is called the *cycle time*. Cycles may be distinguished by how many units are introduced into the system during each period. In an *n-cycle*, n units are introduced each period. [Phillips and Unger 76], [Shapiro and Nuttle, 88], [Ge and Yih, 94], and [Lei and Wang 91] consider *simple* cyclic schedules (n=1), where exactly one job enters and one job leaves the system. The objective of the schedule is to minimize the cyclic time to increase throughput.

[Phillips and Unger, 76] use a mixed integer programming model to minimize the cycle time. The length of a cycle is considered to be from the time a unit departs tank 0 to the time that the next unit leaves tank 0. Since the cycles are assumed to be identical, the configuration of tanks (whether they are in use or empty) at the end of a cycle must be the same as when the cycle began. The model involves n+1 continuous variables, $(n^2 + n)/2$ zero-one variables and $(n+1)^2$ constraints. Real data from a system of 13 tanks is used as a numerical example. The model was solved using the IBM MPSX/MIP package of mixed integer programming algorithms, which uses a branch and bound approach. Additional constraints were added based on experience to reduce the problem size. A schedule with a cycle time of 580 time units was found. The authors also note that the tightness of the time window constraints probably contributed to the short run times by enabling efficient pruning of the solution tree.

[Shapiro and Nuttle, 88] demonstrate a cyclic-schedule of 521 time units for the same problem that is studied in [Phillips and Unger, 76]. They also employ a branch and bound approach, but use [illegible] programming to bound the search space. A heuristic, which attempts to introduce units as soon as feasible and stopping as soon as a simple cycle is obtained, characterizes the essential idea behind the branch and bound procedure used. The algorithm can also be used to generate cycles of a specified duration (if such a schedule is possible) by adding a lower bound on the cycle length.

The Minimum Common-Cycle algorithm (MCC) is proposed by [Lei and Wang, 91] to solve the *cyclic two-hoist* problem. The number of alternate schedules that exists for a system of N+1 tanks is shown by [Lei and Wang 89] to be $N!2^{N-1}$ where N! represents the number of circular permutations that exits for N+1 elements and the 2^{N-1} crane assignments are due to each tank being serviceable by either crane, excluding the first and last tanks. The MCC procedure partitions the system into two sets of contiguous stations and assigns a crane to each set. The creation of non-overlapping subsets eliminates the need to consider interference between the cranes but sacrifices global optimality. Each sub-problem is optimally solved and then the common-cycle time that is acceptable to both subsystems is determined through an iterative process. The partitioning approach reduces the number of possible combinations to O(N(N-1)!) where N is an upper bound on the number of possible partitions and (N-1)! is an upper bound for the total number of circular permutations of the subproblems. However, certain properties of the problem can be used to reduce the number of partitions investigated without losing the minimal common-cycle. The following inequality is shown to hold:

$$X_{optimal} \leq X_{mcc} \leq x(N-1) \leq x(N)$$

where $X_{optimal}$ is the minimum global cycle time, X_{mcc} is the minimum common-cycle time among all the partitions examined, and x(N-1) and x(N) stand for the optimal cycle time for a single crane system with N-1 and N tanks respectively. Increases in the variation of job processing times and crane travelling times, where the cranes can perform more efficiently if they are not constrained to partitions, have a negative effect on the solution provided by the MCC algorithm.

Real-time scheduling is an alternative approach to deal with the crane problem. Instead of creating a fixed cyclic schedule, a real-time system determines what should be scheduled using the current state of the environment. A semi-Markov decision model for real-time scheduling is proposed by [Yih and Thesen, 91] and applied to a material-handling robot running along a single track. The model, however, requires a large amount of data that is difficult to obtain, and the resulting state is usually too large for analytical study. An optimal solution is derived by first eliminating those states that did not occur when the system was scheduled by an expert scheduler, and shown to be significantly better than the one used by the observed expert.

An incomplete branch and bound approach is presented by [Ge and Yih, 95] to schedule one crane in a flow-shop type production environment. Each branch of the tree corresponds to a moving sequence of operations, and is said to be feasible if the corresponding moving sequence is feasible. A depth first search is used with heuristics to determine which node to start examining and a linear programming formulation is used to test the feasibility of the sub-branch found so far. Real-time control is achieved by implementing the algorithm at each decision point in time, for example when the crane unloads a job. Based on the state of the system at that moment, the algorithm will give an efficient schedule for one or more succeeding operations.

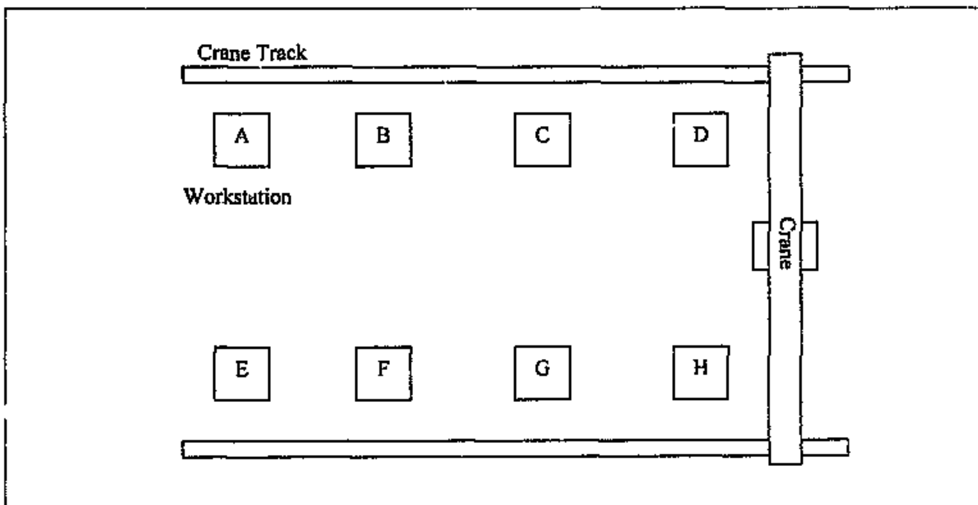
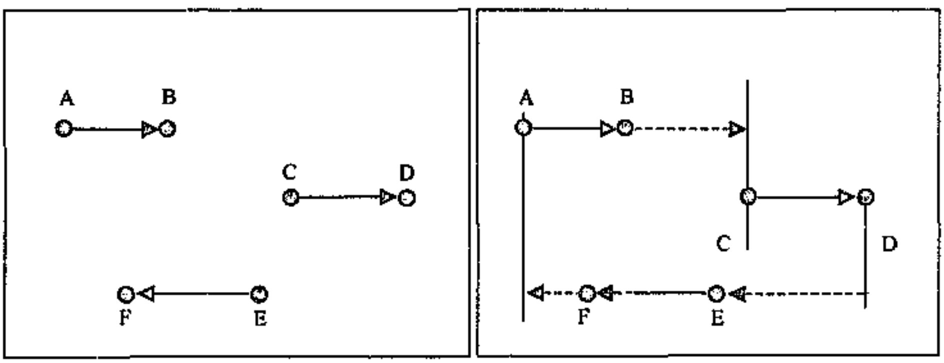


Figure 2.3: Example of a stacker crane layout. The crane is used to pick up, transport, and drop-off work-in-progress between various workstations.

The stacker crane-scheduling problem can be viewed as a relaxation of the flow-type environment discussed above. The crane is used to move units of work from one workstation to another fc processing. Figure 2.3 shows that, unlike the flow-type environment, the movement of work is not necessarily linear.

The stacker crane problem is a generalization of the travelling salesman problem (TSP) and can be described as follows. Given a set V of vertices and a set A of directed edges[1], such that each edge is an ordered pair of vertices. The goal is to find the minimum length tour which traverses each element of A in the specified direction at least once. Figure 2.4 shows the movement of a crane that has to complete three jobs, $J_1=\{A,B\}$, $J_2=\{C,D\}$ and $J_3=\{E,F\}$, with the dashed lines indicating set-up movements between the jobs. The problem can be shown to be NP-complete by a reduction to the problem of finding a Hamiltonian circuit, and remains NP-complete even if all the edge lengths equal 1 [Garey and Johnson, 79]. The *Equalizing Interval Heuristic*, to produce a non-stop cyclic schedule for a stacker crane, which is shown to be superior to dispatching rules in a deterministic and repetitive environment is presented in [Matsuo, *et al*, 91].



(a) (b)

Figure 2.4: The stacker crane problem. (a) An instance of the stacker crane problem A={(A,B),(C,D),(E,F)}. (b) A feasible solution of instance A.

Lieberman and Turksen, study the problem of using *m* cranes that process jobs of one operation only, that is, the crane only has to move to one position to complete a job

[1] The undirected version of the stacker-crane problem is called the rural postman problem and is also NP-complete.

[Lieberman and Turksen, 81]. The scheduling problem is stated as "Given the conflicting demands, assign cranes to tasks so as to minimize the delays in processing due to crane interference." By removing the single-track constraint, the system is treated as an *m*-parallel server system (mps).

The simplest case of the crane problem occurs when the crane only needs to perform an operation at a single location, and does not transport material between locations. When all the job ready times are the same (i.e. $r_j = 0$, for all j) and deadlines are not enforced then this problem is trivial since each job is processed as it occurs along the crane track. The problem reduces to a traveling salesman that has to visit cities that lie along a straight road.

The static single-operation crane scheduling problem is defined by [Lieberman and Turksen, 81] as a system of *m* cranes and *n* jobs such that $m < n$, all the jobs are ready for processing simultaneously, the jobs have equal processing times τ, and they consist of only one operation. The lower bound on the makespan of such a system is $\lceil n / m \rceil \cdot \tau$. A simple O(n) batching algorithm is presented which provides an optimal solution.

The concept of batching the jobs is extended to the case when there are constant inter-arrival times Δr for the jobs. The system can be modeled as a D/D/m queuing system with arrival rate $\lambda = 1 / \Delta r$. An O(n) optimal algorithm can be devised to yield a schedule with no interference, because the jobs have only one operation.

When the restriction of requiring equal processing times is relaxed, the problem becomes NP-complete [Lieberman and Turksen, 81]. In this case a lower bound on the makespan can be shown to be

$$M = \mathrm{Max}\left\{ \left\lceil (1/m)\sum_{i=1}^{n} \tau_i \right\rceil, \underset{1 \le i \le n}{\mathrm{Max}} \lceil \tau_i \rceil \right\}$$

The model is extended to two-operation crane problems in [Lieberman and Turksen, 82], since most tasks involving a crane require picking up material at one location and transporting it to another position. The algorithms for one-operation problems cannot be directly applied to this situation, as they are not able to take into account the precedence structure that exists with a two-operation job, and are therefore unable to eliminate the resulting crane interference.

Under certain conditions however, [Lieberman and Turksen, 82] show that interference-free schedules can be obtained when all the jobs are simultaneously available and the processing time of the jobs are equal. Using the concept of a minimum ordered partition (MOP), a job set can be tested in $O(n^2$ for a necessary condition as to whether or not the set can be partitioned in such a way as to batch the jobs so that no crane interference will occur. If such a solution is possible the optimal time for the makespan of the problem is $2 \cdot \lceil n/m \rceil \tau$. In some cases however complete enumeration of all possible MOP's is necessary, making the problem NP-complete in general.

[Lieberman and Turksen, 82] also provide an alternative procedure that can be used when the necessary conditions for the above method cannot be met, known as a *mesh* procedure with complexity $O(n^2)$. It can only be applied to a system containing two cranes, such that given both are busy, one crane is processing the first location of its job, while the other is processing the second location of its job. This forces a crane to be idle for the first and last τ time units. The procedure yields schedules whose makespans are at worst 4/3 of the lower bound value.

3 PROBLEM DESCRIPTION

3.1 The Crane Problem

The crane problem, can be generalised to any problem where a machine requires some contiguous spatial resource for some period of time, in order to complete a task. In the case of cranes on a single track, the space required is the area over which the crane must move in order to perform its operation. The area over which it must move may be contested by the presence of other cranes processing tasks in the same region, leading to conflicts in movement.

Using the notation presented in the introduction the above crane system is defined as $\mu = \langle \mathbf{C}, \mathbf{L}, \mathbf{J}, \mathbf{R}, \mathbf{T} \rangle$ where:

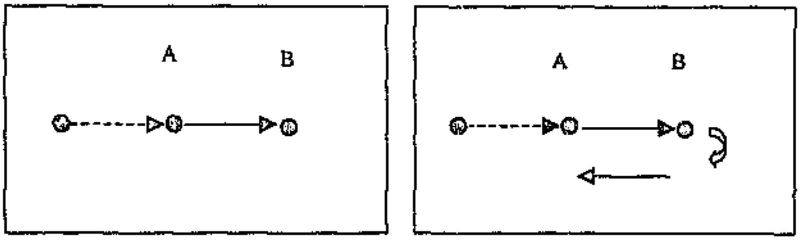
C is the set of 2 cranes $\{C_i \mid i = 1, 2\}$. Two cranes are responsible for the movement of material between various locations along the aisle.

L is the set of N job locations $\{l \mid l = 1, 2, ..., N\}$. The job locations occur at each piece of equipment along the aisle. As these are overhead bridge cranes, job locations occur at either side of the aisle.

J is the set of n jobs available $\{J_i \mid i = 1, 2, \ldots, n\}$. Two classes of jobs occur in this environment, $J_i = \{l_{i1}, l_{i2}\}$ and $J_i = \{l_{i1}, l_{i2}, l_{i1}\}$, see Figure 3.1.

R is the set of job ready times $\{r_i \mid r_i \geq 0, i = 1, \ldots n\}$.

T is the set of operation process times.



(a) (b)

Figure 3.1: The two types of jobs that occur in this scheduling environment. (a) Shows a two location job $J_i = \{A,B\}$, (b) depicts $J_i = \{A,B,A\}$. The dashed arrow indicates the setup movement.

This study uses a computer simulation model of a smelter aisle developed by [Lubinsky, *et al.* 96]. The model was developed in G2, an object orientated simulation language. See the appendix for further details regarding the simulation.

The assumptions together with operational policies are as follows:

1. The cranes have identical capability (identical processors).
2. There is no job pre-emption.
3. Each crane can process at most one location at a time.
4. Each job can be processed by at most one crane.
5. Crane speeds are constant.
6. Cranes are continuously operational.
7. A crane can only transfer material required for one job at a time.

3.2 Crane Jobs in a Smelter Aisle

To study the crane problem a real-world smelter environment was simulated. A schematic of the environment on which the simulation is based is presented in Figure 3.2. Converting is the process of removing impurities from molten metal by blowing air through the liquid. The impurities are changed either into gaseous compounds or into liquids that are removed as slag. The plant consists of six main areas between which material has to be transported. These are: furnaces, slow cooling bays, cold matte and silica feed chutes, slag heap, and the converters, each having a silica and cold matte hopper. Material enters the system through the furnaces and feed chutes and exits via the slow cooling bays and slag heap.

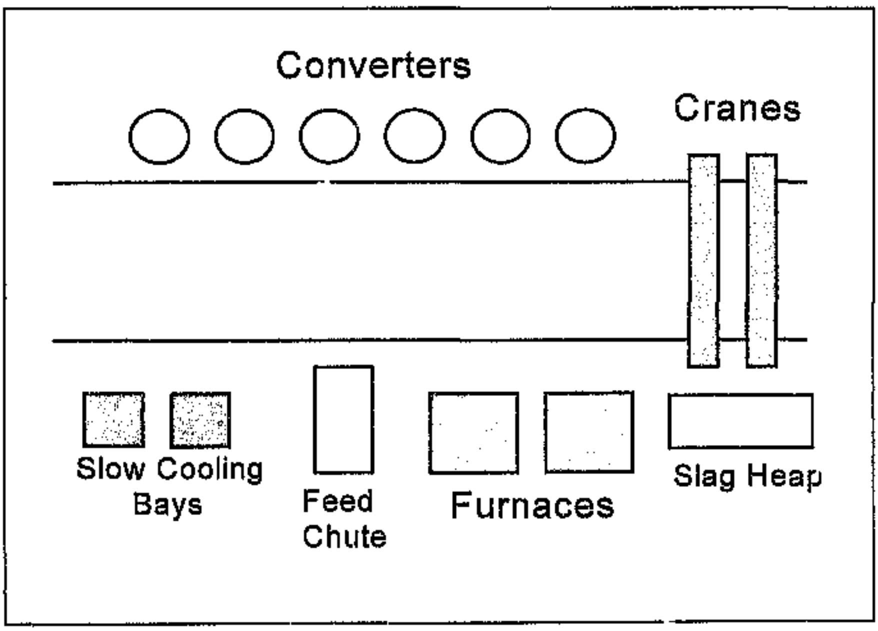


Figure 3 2: Schematic of the converter aisle.

The main activity of the cranes is to transfer matte (a molten mixture of metallic sulphides) from the furnaces to the converters, so that the converting process of the mattes can take place. Once the converter has been charged (filled) with matte, cold matte, and silica for fluxing purposes, a sustained blast of air is introduced. Slag accumulates while the blow is in progress. After a certain amount of slag has formed the air is shut off, and the slag is removed and returned to the furnaces. Another charge of matte and flux can then be added and the blow started again. This process continues until enough converter matte has been produced and can be transferred to the slow cooling bays. A list of the jobs that must be performed are as follows:

Jobs of the form $J_i = (l_{i1}, l_{i2})$: 2 operations and 1 crane movement.

- Filling the cold matte and silica hoppers that feed the converters. Although the hopper must be returned to the converter, the job can be split into moving the hopper to the feed chute, and vice versa. This is possible because the crane can leave the hopper at the chute and perform another job while the hopper is being filled. The example in Figure 3.3 shows the movement needed to transfer a hopper from converter number 5 to the feed chute.

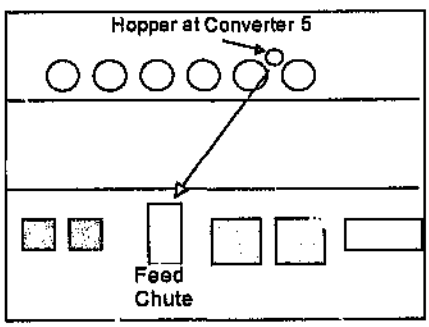


Figure 3.3: Moving a hopper to the feed chute.

- Cleaning the mouth of a converter - moving the service platform to the converter.

- Returning a slag ladle – moving a slag ladle to a converter.

Jobs of the form $J_l = (l_{l1}, l_{l2}, l_{l1})$: 3 operations and 2 crane movements.

- Charging the converters - transferring matte from the furnaces to a converter. Requires moving a ladle of matte from a furnace to a converter, pouring the matte into the converter and returning the ladle to the furnace. Figure 3.4 shows the movements needed to charge converter number 3.

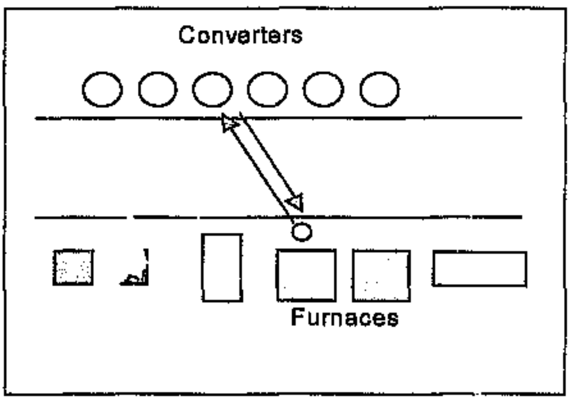


Figure 3.4: Charging converter three.

- Decanting slag - transferring slag from a converter to either a furnace for resmelting or to the slag heap, and returning the slag ladle to the converter, see figure 3.5

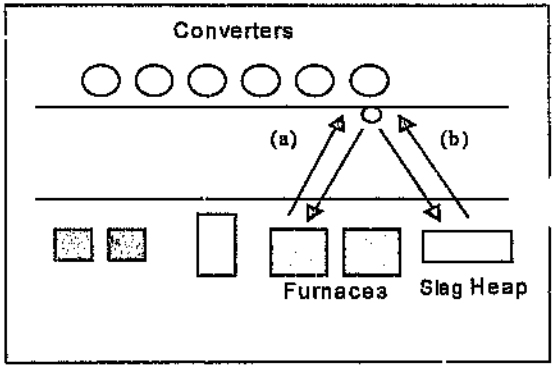


Figure 3.5: Decanting slag. (a) Movement of slag to a furnace. (b) Movement of slag to the slag heap

- Decanting matte – Fetching a ladle from a slow cooling bay, moving to the converter to get the matte, returning to the slow cooling bay and emptying the ladle, see figure 3.6.

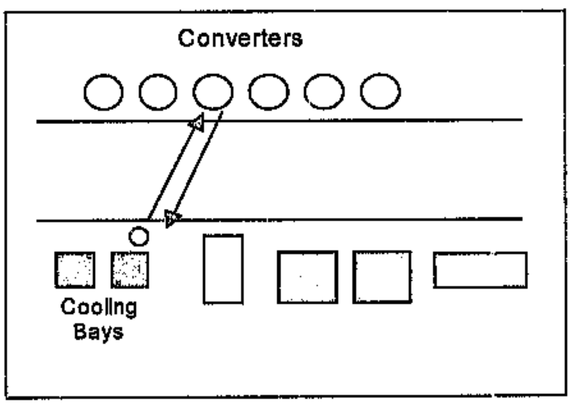


Figure 3.6: Moving matte to the slow cooling bays.

4 DISPATCHING HEURISTICS EXAMINED

Six heuristics have been tested using the simulation of the smelter plant. Due to the dynamic nature of the environment and a limited control horizon, the most obvious scheduling technique is the use of simple dispatching rules. However the effect of sequence-dependent set-up times and more importantly the problem of interference with other cranes has a significant impact on their performance.

The first four dispatching rules use very little information regarding the state of the scheduling environment. As a result they suffer from inherent myopic deficiencies common to dispatching rules. A random heuristic is used as a benchmark to determine whether the dispatching rules provide any performance gains whatsoever.

The final dispatching technique examines the effects the other crane may have on the chosen job. This is a type of rolling horizon procedure with a look ahead of only one job, and forms the basis of an intermediary procedure between simple dispatching rules and a longer term scheduling formulation.

The state of the crane system at any time consists of the present state and locations of the cranes, the future commitments of the cranes (i.e. a crane that is committed to a job must process it to completion since there is no job pre-emption) and pending jobs.

Let pending jobs at time t be defined as:

$$\mathrm{Pend}(t) = \{J_i \mid r_i \leq t\ ,\ J_i \not\subseteq C(t)\}$$

were C(t) represents those jobs that have been completed or are currently being processed at time t. Let I(t) be defined as those cranes that are idle at time t.

The following initialization constants are used:

Maxdist = length of the crane track + 1.

LongTime >> Length of time for the longest job possible.

Using the above crane system information, the dispatching rules discussed below assign jobs to cranes.

4.1 Random Dispatching Rule

As a basis for evaluating the rest of the methods described in this section, the random dispatching rule is used to determine which job should be carried out next.

Random Dispatching Algorithm

```
For each crane C_i in I(t) do
        Assign C_i to a random job in Pend(t)
End
```

4.2 Shortest Distance (SDist) Dispatching Rule

The shortest distance rule is closely related to the greedy algorithm used to solve the travelling salesman problem. It is the easiest rule to practically apply in this type of scheduling environment. The criterion for choosing a job J_i is based on the distance the crane must travel to reach l_{i1}, the first processing location of J_i. S_{ti} represents the set-up distance between the crane at time t and l_{i1}, as discussed in the introduction.

Shortest Distance Dispatching Algorithm

```
Best_Job, AssignedCrane = {null,null}
Nearest_Found = Maxdist
For each crane C_i in I(t) do
        For each Job J_i ∈ Pend(t) do
                If S_ij ≤ Nearest_Found then
                        Nearest_Found = S_ij
                        Best_Job, AssignedCrane = {J_i , C_i}
                End
        End
End
Dispatch AssignedCrane to Best_Job
```

4.3 Local Shortest Processing Time (LSPT) Dispatching Rule

Shortest distance algorithms choose jobs only on the basis of minimizing the set-up time. The LSPT dispatching rule combines the set-up and processing time, and uses the resulting time as the criterion to judge which job should be dispatched next. The set-up and processing time are calculated assuming that there is no interference with the other crane. For this reason it is termed a local dispatching rule, as it does not take into account the state of the other cranes in the environment.

LSPT Dispatching Algorithm

```
Best_Found, AssignedCrane = {null,null}
Shortest_Found = LongTime
For each crane C_i in I(t) do
                For each Job J_i ∈ Pend(t) do
                If S_ij + P_i ≤ Shortest_Found then
```

Shortest_Found = $S_{tj} + P_j$

Best_Job, AssignedCrane = $\{J_j, C_t\}$

End

End

End

Dispatch AssignedCrane to Best_Job

4.4 Shortest Distance with Priority (SDist + P) Dispatching Rule

The shortest distance rule can be rendered useless if low priority jobs are always being serviced first. The SDist+P rule first selects the candidate jobs based on a static priority that is assigned to each job. The SDist dispatching rule then chooses from this subset of pending jobs. Let Priority(J_j) = the priority of job J_j represented as an integer value between 1 and 9, with 9 being the most urgent jobs.

SDist + P Algorithm

PriorityJobs = $\{J_j \mid J_j \in$ Pend(t) , Priority(J_j) = maximum priority of Pend(t)$\}$

Best_Job, AssignedCrane = {null,null}

Nearest_Found = Maxdist

For each crane C_t in I(t) do

For each Job $J_j \in$ PriorityJobs do

If $S_{tj} \leq$ Nearest_Found then

Nearest_Found = S_{tj}

Best_Job, AssignedCrane = $\{J_j, C_t\}$

End

End

End

Dispatch AssignedCrane to Best_Job

4.5 LSPT with Priority (LSPT + P) Dispatching Rule

This rule is similar to the SDist + P dispatching rule, however this time the LSPT rule is used to choose the next job from the candidate jobs.

LSPT + P Algorithm

```
PriorityJobs = {Ji | Ji ∈ Pend(t) , Priority(Ji) = maximum priority of Pend(t)}
Best_Job, AssignedCrane = {null,null}
Shortest_Found = LongTime
For each crane Ci in I(t) do
                For each Job Ji ∈ PriorityJobs do
                If Sij + Pi ≤ Shortest_Found then
                        Shortest_Found = Sij + Pi
                        Best_Job, AssignedCrane = {Ji,Ci}
                End
        End
End
Dispatch AssignedCrane to Best_Job
```

4.6 Global Shortest Processing Time (GSPT) Dispatching

The GSPT algorithm is similar to the LSPT rule, except that it takes into account possible se. up and processing delays caused by the interactions of the two cranes. When a crane becomes idle, it examines each job in the pending list, and calculates the time it takes to complete each job given the state of the second crane. Clearly, if the second crane is idle then the GSPT time equals the LPST time.

In order to calculate the GPST time, a look-ahead simulation of the job is made. This allows all interactions with the other crane to be taken into account when calculating the time it will take to complete its assigned job. The total time taken to complete the job can be expressed as $J_i = S_{ij} + \chi_s(t) + P_i + \chi_p(t_p)$.

The delays, $\chi_s(t) + \chi_p(t_p)$, are implicitly taken into account when performing the simulation.

The look-ahead simulation uses the same control strategy that occurs when the cranes actually perform the jobs. Let Look-head(J_i,c,t) return the look-ahead simulation time for J_i performed by crane c at time t , then the delays caused by *busy* or *slaved* conflicts can be shown by

$$\text{Delays}(J_i,c,t) = \text{Look-ahead}(J_i,c,t) - (S_{ti} + P_i).$$

GSPT Algorithm

```
Best_Found, AssignedCrane = {null,null}
Shortest_Found = LongTime
For each crane Ci in I(t) do
        For each Job Ji ∈ Pend(t) do
                If Look-ahead(Ji,Ci,t) ≤ Shortest_Found then
                        Shortest_Found = Look-ahead(Ji,Ci,t).
                        Best_Found, AssignedCrane = {Ji,Ci}
                End
        End
End
Dispatch AssignedCrane to Best_Found
```

5 RESULTS

The aggregate results of 6 simulation runs for each dispatching rule are presented in the tables below. Each run consists of six, 31-day months, representing 186 simulated days per run.

5.1 Total Output of System

The total output is a measure reflecting the productivity of the system. It represents the total amount of material that leaves the system after 6 months. To show the increase in production the output values have been normalized with respect to the Random heuristic's average. See Table 5.1.

	Average	*Normalized*	*Standard Deviation*
Random	23665	1	416
LSPT	24010	1.01	421
SDist	23923	1.01	490
SDist + P	24132	1.02	257
LSPT + P	24525	1.04	112
GSPT	25791	1.09	110

Table 5.1: Output of system.

5.2 Total Blocked, Idle, Busy and Moving Percentages

Table 5.2 shows the percentage time that the two cranes spend in one of four states. A blocked state occurs when the crane is unable to move to its destination due to the interference of the other crane on the aisle. Busy states occur when a crane is processing a job. A moving state is the result of a crane moving to perform some job. A crane is idle when it is not in any of the other three states.

	Blocked	*Idle*	*Busy*	*Moving*
Random	13.8	11	61.6	13.6
LSPT	15.4	10	62.2	12.4
SDist	15.8	10	61.8	12.4
SDist + P	15.9	9.1	62.1	12.9
LSPT + P	11.9	8.8	67.3	12.0
GSPT	9.6	9.2	68.8	12.4

Table 5.2: Percentage times that the cranes spent in blocked, idle, busy and moving states.

5.3 Total distance, slaved distance, and percentage slaved distance.

Table 5.3 shows the travelling distance of both cranes over the simulated period. The slaved distance represents how far each crane has had to move to stay out of the way of the other crane.

	Total Distance	*Slaved Distance*	*% Slaved Distance*
Random	4608140	881035	19
LSPT	4282240	785979	18
SDist	4429277	817081	18
SDist + P	4443204	759322	17
LSPT + P	4329444	678074	16
GSPT	4022567	604922	15

Table 5.3: Average distances that the cranes traveled during a six month period.

5.4 Actual time to Optimum time for all jobs

Table 5.4 shows the additional time needed to complete a job, as a percentage increase of the best possible time that it could have taken. The results have been obtained using the following formulas.

$$\text{Processing} \quad \frac{(\text{Total time taken to process all jobs}) - (\text{Optimum time to process all jobs})}{\text{Optimum time to process all jobs}} \; 100$$

$$\text{Set - up} \quad \frac{(\text{Total set - up time for all jobs}) - (\text{Optimum set - up time for all jobs})}{\text{Optimum set - up time for all jobs}} \; 100$$

Note that these times refer to the sum of the individual jobs actual and best times, given the state of the system at the time of measurement.

	Processing	*Setup*	*Combined*
Random	17	24	20
LSPT	15	21	18
SDist	16	20	17
SDist + P	16	21	18
LSPT + P	13	18	15
GSPT	11	16	13

Table 5.4: Operational times for all jobs as a percentage of the optimum time.

5.5 Actual time to Optimum time were the Actual time > Optimum time

The results in Table 5.5 are calculated in the same way as in Table 5.4 but in this case, only those instances in which there was a difference between the actual time and optimum time were used.

	Processing	*Setup*	*Combined*
Random	61	79	67
LSPT	59	78	66
SDist	60	79	67
SDist + P	59	77	65
LSPT + P	53	70	59
GSPT	48	67	54

Table 5.5: Operational times for jobs where delays occurred, as a percentage of the optimum time.

6 ANALYSIS OF RESULTS

6.1 Statistical Procedures

Two types of statistical procedures are used to analyse the simulation results. The analysis of variance (ANOVA), and the Duncan multiple range test. The purpose of ANOVA is to test the differences in means for statistical significance. The Duncan multiple range test, groups the methods according to statistical differences between the means.

6.2 Analysis

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	17524470	5	3504894	30.84	5.78E-11	3.69
Within Groups	3409470	30	113649			
Total	20933940	35				

Table 6.1: ANOVA results on production output of heuristics.

The effect of the heuristics on the mean output values is statistically significant at $\alpha \leq 0.01$ as seen in Table 6.1.

Figure 6.1 shows how production output increases, as the heuristics have more information regarding the state of the environment.

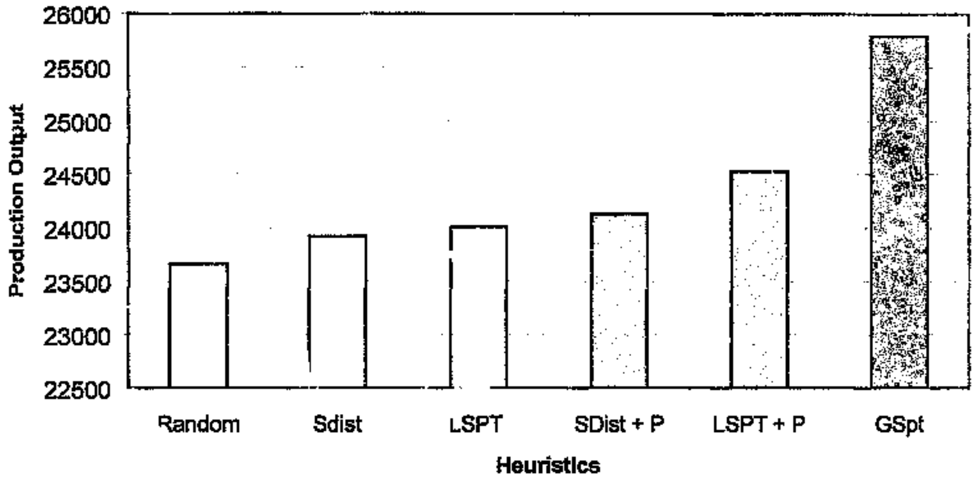


Figure 6.1: Production Output. The bars are shaded according to the heuristic's Duncan Grouping shown in Table 6.2. Group A - ▨ , Group B -□ , Group C -□ .

The results of the Duncan Multiple Range Test on the production outputs are shown in Table 6.2.

Duncan Grouping	Mean	Heuristic
A	25791	GSPT
B	24525	LSPT + P
B	24132	SDist + P
C	24010	LSPT
C	23923	SDist
C	23665	Random

Table 6.2: Duncan Groupings based on production output.

The groupings show that simple heuristics such as SDist and LSPT do not perform well when sequence-dependent job times are present. Only when information that is specific to the scheduling environment is introduced, in the form of priority rules, do the heuristics show some improvement.

The fixed priority scheme used in SDist + P and LSPT + P works in this case, since the characteristics of the pending list and job precedence structure ensure that high priority rules will not always dominate the scheduling system. On average 77% of all scheduling decisions for the SDist + P and LSPT + P were determined by priority only.

The results of the GSPT show how important it is, in environments where set-up and processing delays are caused by the state of other machines in the system, to take into account knowledge that is not limited to only to the machine and job being scheduled, as is the case in most dispatching heuristics.

Figure 6.2 shows the percentage improvement of the production means, over the Random heuristic.

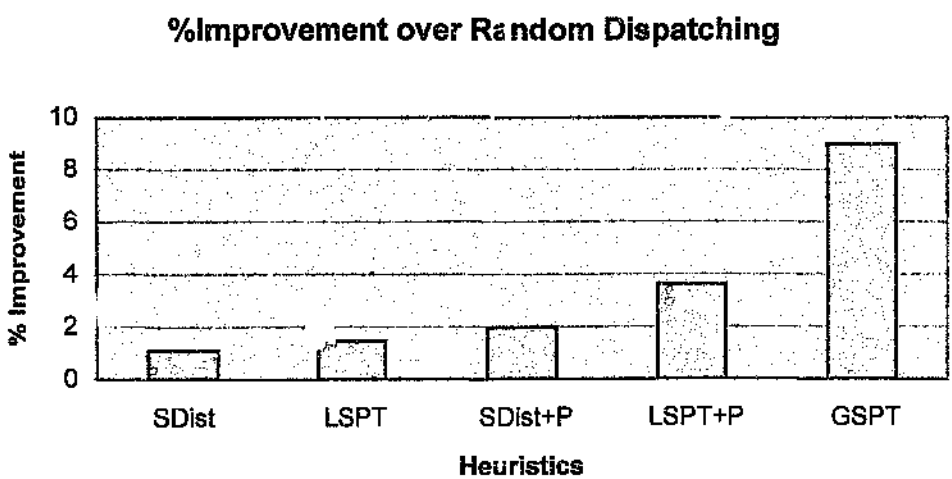


Figure 6.2. Percentage improvement over Random dispatching heuristic.

The GSPT rule shows a 9% improvement over the random case and a 5% improvement over the next best rule tested LSPT + P. The LSPT rule outperforms the SDist rule by 33%, and by 84 % when priority scheduling decisions are included (i.e. LSPT + P).

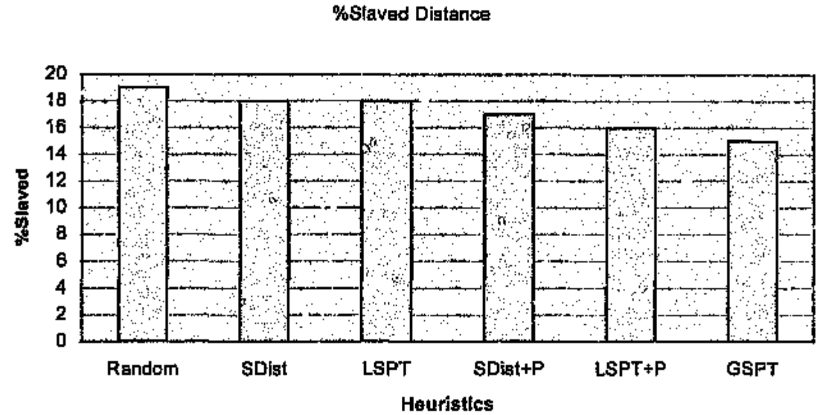


Figure 6.3: Percentage slaved distance to total distance moved by both cranes.

GSPT also shows a 4% improvement compared to the Random rule, in respect to the percentage of the distance the cranes had to travel to move out of each others' way, and the total distance they moved during the simulated period, see Figure 6.3.

Figure 6.4 shows the improvement in the percentage distances moved with respect to the Random dispatching rule. When using the GSPT rule, the slaved distance moved is 31% less than the Random rule, and the total distance moved is 13% less. The SDist heuristic actually performed better than the LSPT algorithm with 11% and 7% decreases in the slaved and total movement of the cranes as opposed to 7% and 4% respectively for the LSPT method. This suggests that the SDist rule which always attempts to make the smallest move to the next job, succeeded in reducing crane movement, but that this in itself was unable to increase throughput.

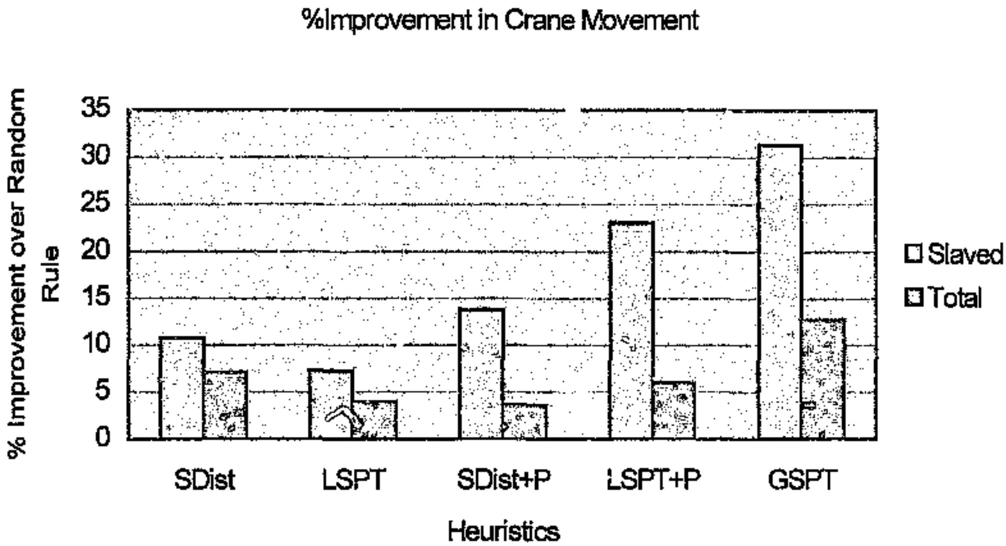


Figure 6.4: Decrease in crane movements as a percentage of the distances moved under the Random dispatching rule.

Given the order of jobs assigned to the cranes by the various heuristics, Figure 6.5 shows how the jobs total completion times compare with their individual optimum completion times. These results show the effects of delays on the individual jobs, and do not indicate the total makespan of the jobs, which would be dependent on the sequencing of the jobs as well.

The Random rule shows that the jobs take 20% longer than if no conflict delays had occurred. For jobs where conflicts are present the percentage increase in the job times is 67%. The GSPT rule is able to reduce the delays to 13% for the overall time, and 54% for conflicted job times.

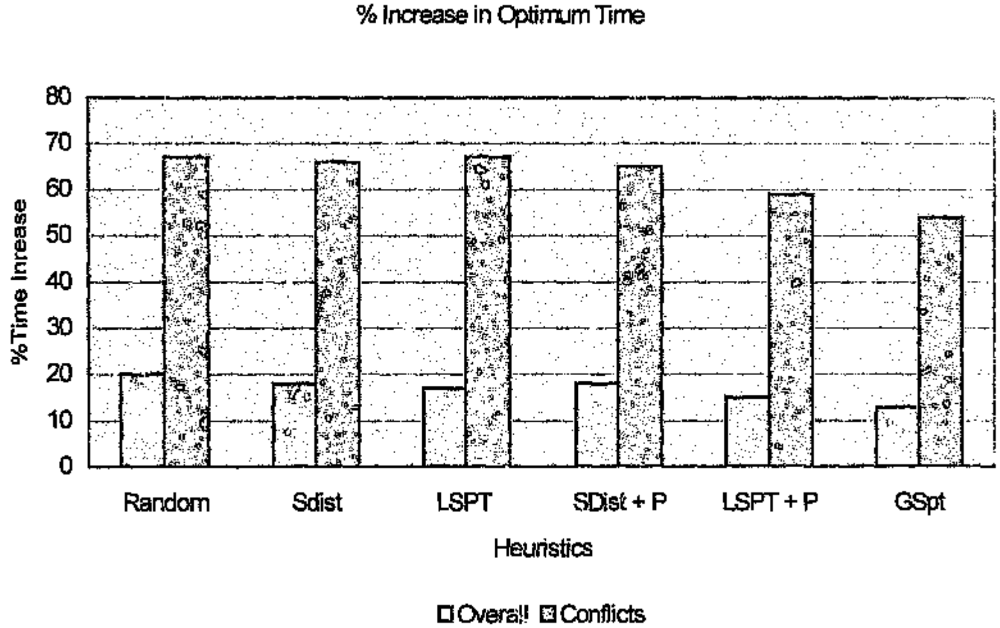


Figure 6.5: The increase in the overall processing time, and time when conflicts did occur, as compared to the optimal time for those jobs.

The time taken for the jobs which took longer than the optimum time can be split into their set-up and processing components respectively. Figure 6.6. shows the effects of delays on each component as a percentage increase in the optimum time.

The LSPT + P and GSPT heuristics stand out from the others. The set-up and processing increases are 53% and 73% respectively for the LSPT+ P, and 48% and 67% for the GSPT method. While the improvement in times due to a decrease in delays is present, the corresponding increases in output for the different heuristics must also be attributed to the sequencing of the jobs, and not only the decrease in conflicts.

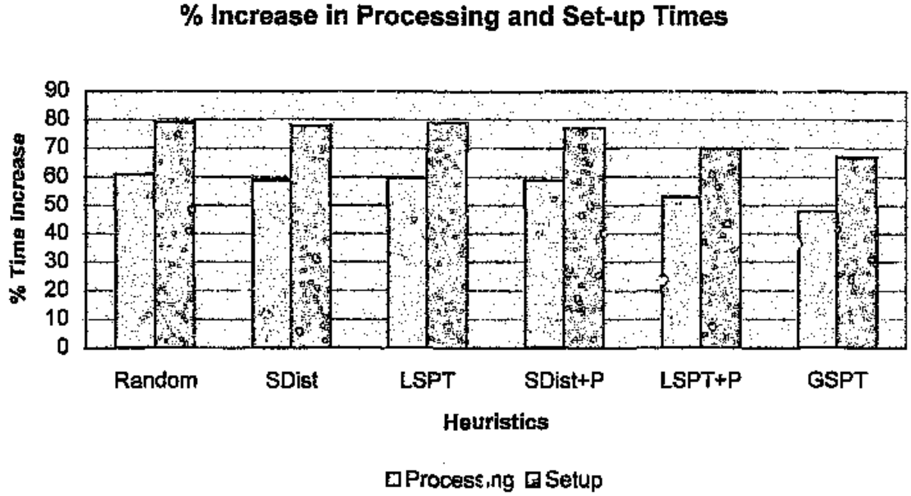


Figure 6.6 Increase in processing and set-up times as a percentage of the optimum time.

7 CONCLUSION

The results of the heuristics used in this report, suggest that the GSPT dispatching rule yields a higher throughput than those rules that do not take into account interactions with the other crane. When the SDist and LSPT rules were supplemented with a priority decision making process, they both showed an improvement in output.

The set-up time of any environment, which involves spatial movement, is inherently sequence-dependent. The notion of sequence-dependent process times, however, is not often presented. In this case, the delays caused by the interactions of the cranes while performing their respective jobs are considered to be part of the processing time, as opposed to a separate aspect of the scheduling environment.

This idea is carried forward in the definitions of local and global processing times. The local processing time does not consider delays, and indicates the expected time a job would take if no crane interaction took place. The global processing time, is the time it takes when the states of the other machines are taken into account. Processing time is generally considered in scheduling literature to be independent of other machines in the environment. In the case of cranes or AGVs a spatial resource is required to perform a job, and this resource may be contested by other vehicles in the area. This results in the processing time being dependent on the sequencing of those vehicles so that the resource in question can be used efficiently, thus minimizing the processing time.

The resource need not be spatial. Other examples could be specific memory areas that programs need to perform certain tasks. If the memory region needed is already in use then the process will be delayed. The delay however is a function of the sequence-dependence of the processing times. A similar situation may occur when labour shortages arise. In this situation the contested resource is the labour, as the processing time for job J_i on machine M_j may increase due to delays caused by the shortage of operators. In this case however the resource is discrete as opposed to continuous.

The delays also affect set-up times, and thus the set-up time is a function of distance and crane interactions in this environment. Thus, both the set-up and processing times are sequence dependent.

These dispatching heuristics have focused on the effects of the delays caused within jobs, and since they look only one job ahead, they are not effective as sequencing rules for the overall job order. The results show however, that dispatching rules should attempt to take into account the current state of any machine that may cause delays within the jobs being considered.

References

AKTURK, M.S., and YILMAZ, H., 1996, Scheduling of Automated Guided Vehicles in a Decision Making Hierarchy. *Internaticnal Journal of Production Research*, vol. 34, no. 2, pp. 577-591.

BAKER, K., 1974, *Introduction to Sequencing and Scheduling*. New York, Wiley.

BLACKSTONE, J.H. JR, PHILLIPS, D.T., and HOGG, G.L., 1982, State-of-art survey of dispatching rules for manufacturing job-shop operations. *International Journal of Production Research*, vol. 20, no. 1, pp. 27-45.

BLAZEWICZ, J., FINKE, G., HAUPT, R., and SCHMIDT, G., 1988. New Trends in machine scheduling. *European Journal of Operational Research*, vol. 37, pp. 303-317.

BRUNO, J., and DOWNEY, P., 1978, Complexity of Task Sequencing with Deadlines, Set-up Times and Changeover Costs. *SIAM Journal of Computing*, vol. 7, no. 4, pp 393-404.

CHANDRA, J. and TALAVAGE, J., 1991, Intelligent dispatching for flexible manufacturing. *International Journal of Production Research*, vol. 29, no. 11, pp. 2259-2278.

CHEN, C.C., and YIH, Y., 1996, Identifying attributes for knowledge-based development in dynamic scheduling environments. *International Journal of Production Research*, vol. 34, no. 6, pp. 1739-1755.

CHOI, H-G., KWON, H-J., and LEE, J., 1994, Traditional and Tandem AGV System Layouts: A Simulation Study. *Simulation*, vol. 63, no. 2, pp. 85-93.

CONWAY, R.W., MAXWELL W.L., and MILLER, L.W., 1967, *Theory of Scheduling*. Addison-Wesley, Reading, Massachusetts.

EGBELU, P.J., and TANCHOCO, J.M.A, 1984, Characterisation of Automated Guided Vehicle dispatching rules. *International Journal of Production Research*, vol. 22, pp. 359-374.

ELVERS, D.A., and TAUBE, L.R., 1983, Time Completion for Various Dispatching Rules in Job Shops. *Omega*, vol. 11, no. 1, pp. 81-89.

GAREY, M.R., and JOHNSON, D.S., 1979, *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H.Freeman and Company, New York.

GE, Y., and YIH, Y., 1995, Crane Scheduling with time windows in circuit board production lines. *International Journal of Production Research*, vol. 33, no.5, pp 1187-1199.

GERE, W.S. JR, 1966, Heuristics in Job Shop Scheduling. *Management Science*, vol. 13, no. 3, pp. 167-190.

GRAVES, S.C., 1981, A Review of Production Scheduling. *Operations Research*, vol. 29, no. 4, pp. 646-675.

HAO, G., and LAI, K.K., 1996, Solving the AGV problem via a Self-Organizing Neural Network. *Journal of the Operational Research Society*, vol. 47, no. 12, pp. 1477-1493.

ISHII, N., and MURAKI, M., 1996. An extended dispatching rule approach in an on-line scheduling framework for batch process management. *International Journal of Production Research*, vol. 34, no. 2, pp. 329-348.

ISHII, N., and TALAVAGE, J.J., 1991, A transient-based real-time scheduling algorithm in FMS. *Internation Journal of Production Research*, vol. 29, no. 12, pp. 2501-2520.

JEREMIA, B., LALCHANDANI, A., and SCHRAGE, L., 1964, Heuristic rules toward optimal scheduling. *Research Report*, Department of Industrial Engineering, Cornell University.

KIM, S.C., and BOBROWSKI, P.M., 1994, Impact of sequence-dependent setup time on job shop scheduling performance. *International Journal of Production Research*, vol. 32, no. 7, pp. 1503-1520.

KIM, C.W., and TANCHOCO, J.M.A, 1991, Conflict-free shortest-time bidirectional AGV routing. *International Journal of Production Research*, vol. 28, no. 6, pp. 2377-2391.

KIM, C.W., and TANCHOCO, J.M.A, 1993, Operational control of a bidirectional automated guided vehicle system. *International Journal of Production Research*, vol. 31, no. 9, pp. 2123-2138.

KLIEN, C.M. and KIM, J., 1996, AGV dispatching. *International Journal of Production Research*, vol. 34, no. 1, pp. 95-110.

KRISHNAMURTHY, N.N., BATTA, R., and KARWAN, M.H., 1993, Developing conflict-free routes for automated guided vehicles. *Operations Research*, vol. 41, no. 6, pp. 1077-1090.

LEE, J., 1996, Composite Dispatching Rules for Multiple-Vehicle AGV Systems. *Simulation*, vol. 66, no. 2, pp. 121-130.

LEI, L., and WANG, T., 1991, The Minimum Common-Cycle Algorithm for Cyclic Scheduling of Two Material Handling Hoists with Time Window Constraints. *Management Science*, vol. 37, no. 12, pp. 1629-1639.

LIEBERMAN, R.W., and TURKSEN, I.B., 1981, Crane Scheduling Problems. *AIIE Transactions*, vol. 13, no. 4, pp. 304-311.

LIEBERMAN, R.W., and TURKSEN, I.B., 1982, Two Operation Crane Scheduling Problems. *AIIE Transactions*, vol. 14, no. 3, pp. 147-155.

LUBINSKY, D.S., NAGY, P. and PERRY, K., 1996, Smelter Aisle Simulation Model. *Proceedings of the G2 Users Group 1996*, Idle Winds Conference Center.

MATSUO, H., SHANG, J.S., and SULLIVAN, R.S., 1991, A Crane Scheduling Problem in a Computer-Integrated Manufacturing Environment. *Management Science*, vol. 37, no. 5, pp. 587-606.

MCKAY, K.N., SAFAYENI, F.R., and BUZACOTT, J.A., 1988, Job-Shop Scheduling Theory: What is Relevant? *Interfaces*, vol. 18, no. 4, pp. 84-90.

NAKASUKA, S., and YOSHIDA, T., 1992, Dynamic scheduling system utilising machine learning as a knowledge acquisition tool. *International Journal of Production Research*, vol. 30, no. 2, pp. 411-431.

NORONHA, S.J., and SARMA, V.V.S., 1991, Knowledge-Based Approaches for Scheduling Problems: A Survey. *IEEE Transactions and Data Engineering*, vol. 3, no. 2, pp. 160-171.

OCCEÑA, L.G., and YOKOTA, T., 1991, Modeling of an automated guided vehicle system (AGVS) in a just-in-time (JIT) environment. *International Journal of Production Research*, vol. 29, no. 3, pp. 495-511.

OVACIK, I.M., and UZSOY, R., 1993, Worst-case error bounds for parallel machine scheduling problems with bounded sequence-dependent setup times. *Operations Research Letters*, no. 14, pp. 251-256.

OVACIK, I.M., and UZSOY, R., 1994, Rolling horizon algorithms for a single-machine dynamic scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, vol. 32, no. 6, pp. 1243-1263

OVACIK, I.M., and UZSOY, R., 1995, Rolling horizon procedures for dynamic parallel machine scheduling with sequence-dependent setup times. *International Journal of Production Research*, vol. 33, no. 11, pp. 3173-3192.

PANWALKER, S.S., and ISKANDER, W., 1977, A survey of Scheduling Rules. *Operations Research*, vol. 25, no. 1, pp. 45-61.

PHILLIPS, L.W., and UNGER, P.S., 1975, Mathematical Programming Solution of a Hoist Scheduling Program. *AIIE Transactions*, vol. 8, no. 2, pp. 219-225.

RAMASESH, R., 1990, Dynamic Job Shop Scheduling: A Survey of Simulation Research. *Omega*, vol. 18, no. 1, pp. 43-57.

RINNOOY KAN, A.H.G., 76, *Machine Scheduling Problems: Classification, complexity and computations*. Martinus Nijhoff, The Hague.

RODAMMER, F.R., and WHITE, K.P., JR., 1988, A Recent Survey of Production Scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 6, pp. 841-851.

SABUNCUOGLU, I., and HOMMERTZHEIM, D.L., 1992a, Experimental investigations of FMS machine and AGV scheduling rules against mean flow-time criterion. *International Journal of Production Research*, vol. 26, no. 2, pp 173-188.

SABUNCUOGLU and HOMMERTZHEIM, D.L, I., 1992b, Dynamic dispatching algorithm for scheduling machines and automated guided vehicles in a flexible manufacturing system. *International Journal of Production Research*, vol. 30, no. 5, pp. 1059-1079.

Sahni, S., and Gonzalez, T., 1976, P-Complete approximation problems. *Journal of the ACM*, no. 23, pp. 555-565.

Sen, T., and Gupta, S.K., 1984, A State-of-Art Survey of Static Scheduling Research Involving Due Dates. *Omega*, vol. 12, no. 1, pp. 63-76.

Shapiro, G.W., and Nuttle, H.L.W., 1988, Hoist Scheduling for a PCB Electroplating Facility. *IIE Transactions*, vol. 20, no. 2, pp. 157-167.

Taghaboni-Dutta, F., and Tanchoco, J.M.A., 1988, A LISP based controller for free-ranging automated guided vehicle systems. *International Journal of Production Research*, vol. 26, no. 2, pp. 173-188.

Taghaboni-Dutta, F., and Tanchoco, J.M.A., 1995, Comparison of dynamic routing techniques for automated guided vehicle system. *International Journal of Production Research*, vol. 33, no. 10, pp. 2653-2669.

Yih, Y., and Thesen, A., 1991, Semi-Markov decision models for real-time scheduling. *International Journal of Production Research*, vol. 29, no, 11, pp. 2331-2346.