Improving Production Schedules Without Scheduling
Improving Production Schedules Without Scheduling

Proefschrift

ter verkrijging van de graad van doctor aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus,
prof. ir. K.F. Wakker,
in het openbaar te verdedigen
ten overstaan van een commissie
aangewezen door het College van Dekanen
op 27 september 1993 te 14.00 uur

door

Eric Paul Jansen

geboren te Vlissingen,

wiskundig ingenieur.
Dit proefschrift is goedgekeurd door de promotoren:

Prof. dr. H. Koppelaar
Prof. dr. C.B. Besant
Technische Universiteit Delft
Imperial College of Science, Technology and Medicine

en door de promotiecommissie:

Prof. dr. ir. J.J.M. Evers
Prof. dr. ir. F.W. Jansen
Prof. dr. ir. J. van Katwijk
Dr. ir. H. de Swaan Arons
Technische Universiteit Delft
Technische Universiteit Delft
Technische Universiteit Delft
Erasmus Universiteit Rotterdam
for my wife Petra

to the trains
Typeset using Word for Windows.
Cover, layout and illustrations, except illustrations in appendix-C,
are by the author and produced using Corel Draw and Microsoft Graph.
Illustrations in appendix-C are by the ESPRIT Project 2415
Distributed Manufacturing Planning and Control.
Printed by Universiteitsdrukkerij

ISBN 90-9006355-2

© Copyright 1993 Eric-Paul Jansen

All rights reserved. No part of this publication may be reproduced or
transmitted, in any form or by any means, electronic, mechanical, photocopying,
recording or otherwise, or stored in any retrieval system of any nature, without
the written permission of the author.
Acknowledgements

To satisfy the urge to extend my knowledge and to be involved in technology in a broad sense, I gratefully took the opportunity to work on the ESPRIT Project 2415. During this project, various project and thesis related subjects have been tackled.

From 1983 until 1985, Henk de Swaan Arons and I have worked on various projects. In February 1989, Henk and I started working together on the ESPRIT Project. I like to thank Henk for his support during the ESPRIT Project and for his support concerning this thesis.

I specially thank my promotor professor Henk Koppelaar for giving me the opportunity to work on this thesis and for his criticism during the past four years.

The past four years have been years to cherish, especially concerning the co-operation with Imperial College of Science, Technology and Medicine. I am very grateful to professor Colin Besant, who is my co-promotor and who has supported me throughout the project and afterwards.

I have been in London many times, and most of the time I have worked together with Leon Hatzikonstantis. As a team, we have solved most problems that have occurred during the project. Further, we have had various discussions on the topics of our theses, for which I like to thank Leon.

I also like to thank Peter Champ, Mike Ristic, Sharam Etemad, Jeya, Rolf Slatter, Sabine Bohlen and Erhard Schültchen for various happenings during the ESPRIT Project. Further, I thank the students Jan Groeneveld and Michael Heinrich for performing various tests.

Being involved in writing a thesis, one tends to fail spotting mistakes. Therefore, proof reading forms an important stage. I am grateful that, besides my promotors, Henk de Swaan Arons, Bob Goedhart and Jan
Improving Production Schedules Without Scheduling

Koning have found the time to read this thesis and to supply me with hints.

I further thank Bob Goedhart for his support concerning Delfi3 developments, porting and debugging. As roommates we have had various discussions on our theses and football. Fortunately, the comments on the latest football matches can be exchanged through electronic mail.

I like to thank Koos Schonewille and Dennis van Loon for their support reproducing and binding copies of various versions produced for reviewing. Further, I like to thank students Jan Groeneveld and Michael Heinrich for performing various tests.

Working on a thesis takes a lot of (free) time. I like to thank Petra, my wife, for her patience, understanding and time.
# Contents

1 Background and Thesis ................................................................. 1  
   1.1 The Domain: Manufacturing ...................................................... 2  
      1.1.1 Philosophies ................................................................. 2  
      1.1.2 Flexible Manufacturing Systems ....................................... 4  
      1.1.3 Drawbacks of modern manufacturing systems ......................... 4  
   1.2 Project environments ........................................................... 6  
      1.2.1 The ESPRIT Project 2415 ................................................. 7  
      1.2.2 The Delfi Project .......................................................... 7  
   1.3 The Ph.D. research topic ....................................................... 9  
   1.4 Outline of the thesis ............................................................ 10  

2 Scheduling and Optimisation .......................................................... 13  
   2.1 Automated scheduling and planning: recent research ...................... 14  
   2.2 Scheduling ............................................................................. 16  
      2.2.1 Job-shop scheduling .......................................................... 17  
      2.2.2 Dispatching rules .............................................................. 18  
      2.2.3 Heuristics ........................................................................ 20  
   2.3 Evaluating schedules .............................................................. 20  
      2.3.1 Evaluation based on due dates .......................................... 21  
      2.3.2 Evaluation based on completion times ................................ 22  
   2.4 Improving schedules ............................................................... 23  
      2.4.1 Using different dispatching rules ..................................... 23  
      2.4.2 Using heuristics ............................................................... 24  
      2.4.3 Other optimisation techniques .......................................... 24  
      2.4.4 Concluding ...................................................................... 27  

3 Schedule Fine Tuning Techniques ..................................................... 29  
   3.1 Overlapping manufacture .......................................................... 30  
   3.2 Batch Splitting ....................................................................... 32
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1</td>
<td>Vertical Splitting</td>
<td>33</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Horizontal Splitting</td>
<td>35</td>
</tr>
<tr>
<td>3.3</td>
<td>Alternative Manufacture</td>
<td>36</td>
</tr>
<tr>
<td>3.4</td>
<td>Using and combining fine tuning techniques</td>
<td>38</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Why fine tuning?</td>
<td>38</td>
</tr>
<tr>
<td>3.4.2</td>
<td>When fine tuning?</td>
<td>38</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Improving one job using one technique</td>
<td>39</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Improving one job using several techniques</td>
<td>40</td>
</tr>
<tr>
<td>3.4.5</td>
<td>Improving one job using techniques on other jobs</td>
<td>40</td>
</tr>
<tr>
<td>3.4.6</td>
<td>Consequences of improving some job</td>
<td>41</td>
</tr>
<tr>
<td>3.4.7</td>
<td>Improving several jobs</td>
<td>41</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Fine Tuning Schedules Through Overlapping Manufacture</td>
<td>43</td>
</tr>
<tr>
<td>4.1</td>
<td>Characteristics</td>
<td>44</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Equal operation times per part</td>
<td>44</td>
</tr>
<tr>
<td>4.1.2</td>
<td>OVL-operation time per part &lt; NXT-operation time per part</td>
<td>45</td>
</tr>
<tr>
<td>4.1.3</td>
<td>OVL-operation time per part &gt; NXT-operation time per part</td>
<td>46</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Remarks</td>
<td>46</td>
</tr>
<tr>
<td>4.2</td>
<td>Mathematical problem description</td>
<td>48</td>
</tr>
<tr>
<td>4.2.1</td>
<td>A straightforward algorithm</td>
<td>51</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Linear programming (LP) approach</td>
<td>53</td>
</tr>
<tr>
<td>4.3</td>
<td>Optimal solution by search space reduction</td>
<td>56</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Equal operation time per part: ( p_{OVL} = p_{NXT} )</td>
<td>56</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Operation times per part not equal: ( p_{OVL} &lt; p_{NXT} )</td>
<td>58</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Operation times per part not equal: ( p_{OVL} &gt; p_{NXT} )</td>
<td>61</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Fine Tuning Schedules Through Batch Splitting</td>
<td>63</td>
</tr>
<tr>
<td>5.1</td>
<td>Vertical Splitting</td>
<td>63</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Mathematical problem description</td>
<td>64</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Optimal solution by search space reduction</td>
<td>66</td>
</tr>
<tr>
<td>5.2</td>
<td>Horizontal Splitting</td>
<td>76</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Mathematical problem description</td>
<td>76</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Optimal solution by search space reduction</td>
<td>77</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Using the Fine Tuning Techniques</td>
<td>79</td>
</tr>
<tr>
<td>6.1</td>
<td>Implementation concept</td>
<td>80</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Communication and DSR</td>
<td>80</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Implementing communication and DSR</td>
<td>81</td>
</tr>
</tbody>
</table>
Background and Thesis

Various manufacturing companies use computers for production planning and control, some of them fully exploit the possibilities and features of available software, others only use a small set of features. Problems that occur on the shop floor are mostly handled using this software. However, looking at metal manufacturing companies that produce a high variety of products in small to medium sized batches, one recognises the desire for software that fits the needs of this very substantial and specific branch of manufacturing.

The main property of the desired software is to handle the diversity of products and batch sizes and to offer solutions for handling the increased possibility of disturbances. Apart from short term scheduling, handling problems, such as late jobs, should be a feature of this new software.

The research performed concerns the development of techniques to handle problems, such as late jobs, by investigating schedules and using idle time to improve the lead time of these problem jobs.
In this introductory chapter we describe the manufacturing domain in a
general manner, visiting some of the philosophies being used. Further we
discuss drawbacks of modern manufacturing systems and some attempts
that have been made to improve the production planning and control for
manufacturing companies.

After this discussion, the attention focuses on small to medium sized
batch manufacturing. For this area we show some shortcomings regarding
support for production planning and control. This leads to the drafting of
some research questions, after which we present an approach to perform
research on the essence of various techniques and how to use them.

1.1 The Domain: Manufacturing

Manufacturing companies are involved in the large scale production of
products using machinery. The automation of a manufacturing company,
the Factory Automation, can be depicted using a six level reference model
as developed by the National Bureau of Standards in the U.S.A. and the
International Organisation of Standards. The six levels, from top to bot-
tom, are

- the facility or plant level
- the factory level
- the area level
- the cell level
- the workstation level
- the process level

In the following sections we refer to this model.

1.1.1 Philosophies

Manufacturing industries operate mostly according to some philosophy of
manufacturing management. One of the most well known and often re-
ferred to philosophies is the Japanese approach towards manufacturing:
just-in-time (JIT). The success of Japanese companies, using the JIT phi-
losophy and reporting levels of performance in terms that were previ-
ously thought to be unobtainable such as lead times, quality and so on
(Pegler, 1990), urged and inspired the Western world to pursue the same
success and to investigate this philosophy. The cornerstones of JIT are
elimination of waste and continual improvement of processes.
Another philosophy is Group Technology (GT). The basic principle of GT is to take advantage of similarities that exist amongst related or similar parts and processes after identifying and bringing them together as such. The advantages of these similarities can be exploited during all stages of design and manufacture (Gallagher, 1986). They mainly lead to a reduction in time and costs of manufacture. By producing similar parts or part families in a particular cell, GT can reduce setup time (Black, 1983). Other advantages of GT that are frequently claimed include an increase in the speed of through-put, a reduction in the rate of scrap and easier control over production (Edwards, 1971).

An application of GT can be found in the cellular manufacturing where a portion of a firm’s manufacturing system has been converted to cells. A manufacturing cell comprises machines or processes that are located in proximity and which manufacture specific part families. The primary target of cellular manufacturing is the reduction of times involved in the process such as set-up and flow times and consequently the reduction of inventories and (market) response times.

A survey by Wemmerlöv and Hyer (1989) shows that a majority of the manufacturing companies that use cellular layouts, also use machine tools in other types of layouts. Such layouts can be, for instance, functional layouts in which the machine tools are grouped according to function (turning, milling).

Manufacturing Resource Planning (MRP II) is another philosophy of manufacturing management. However, this philosophy is often referred to in the context of manufacturing-control software together with MRP (Material Requirements Planning).

Although the JIT and GT philosophies are widely spread, environmental properties and constraints play an important role in the use of such a philosophy as the following example shows. Viewing the Philips Machinery Factory Singapore (De Swaan Arons, 1990) the management aim is to extensively utilise the available machines and almost every customer order will be considered assuming the factory has 'infinite' capacity. This assumption is based on the fact that machines are available 24-hour each day and, due to the low wages, overtime and thus extra capacity can be easily arranged.
1.1.2 Flexible Manufacturing Systems
The advance in technology plays a significant role in today's world. Within the manufacturing world, these advances have already led to numerical control (NC), computer numerical control (CNC), robots, automated guided vehicles (AGVs) and so forth. Bringing these techniques into action within a manufacturing environment leads to a better solution to manufacturing problems constituting systems that are known as Flexible Manufacturing Systems (FMS) (Besant, 1986).

Improving techniques is an ongoing process that causes definitions and the meaning of some terminology to be adjusted. Ranky (1990) defines FMS as *a reprogrammable manufacturing system in its broadest sense, dealing with high level distributed data processing and automated material flow, using highly flexible, computer controlled (and in some cases manually operated) material and information processors within an integrated, multi-layered feedback control architecture.*

We have already mentioned CNC. One of the advantages of CNC comprises flexibility and productivity, which, within the concept of FMS, can be adapted and applied to the manufacture of small to medium volume quantities. Further the ability to have several operations at the same location can be exploited by FMS and the utilisation of the machines can be increased.

The decrease of lead time, inventory, labour and special tooling, together with an increase of productivity and flexibility constitute some of the benefits of FMS.

1.1.3 Drawbacks of modern manufacturing systems
Developments in manufacturing and technology lead to the development of manufacturing systems that reduce the amount of decision making by those responsible for operating and supervising the manufacturing process. The reduction of decision making can especially be witnessed at the cell and workstation levels. Moreover, such manufacturing systems have often been developed to centralise authority higher up the management: the plant and factory levels.

The factory level incorporates a large computer, which holds the main database for computer aided design (CAD) and computer aided planning (CAP) applications. Often the computer is also being used for running a system that is fundamental to the planning and operation of the work in
the plant. Such a system, an MRP II system, is the materialisation of the centralisation of authority and decision making.

MRP II systems, however, generate plans projected on a long time scale, which means that such plans are only valid if nothing 'unplanned' happens at the cell and workstation levels such as machine breakdowns and rush orders. So when everything is running according to plan, the MRP II systems are capable of being effective.

This centralised approach with the MRP II system at the factory level presents us with some problems.

Firstly, it is more common than rare that production is not running according to plan. In practice this causes inefficiency due to the lack of appropriate information to overcome disruptions at the time they occur and the 'prohibited' intervention by the shop floor worker.

Secondly, the conventionally designed man-machine systems are usually oriented towards the highest degree of automation that is feasible. In the design process, people are only considered at an advanced stage in the system design, whilst the system or the machine has priority (Besant, 1990). As a result, left-over tasks that could not be easily automated constitute the work-places in such environments. The operator then serves as a trouble-shooter or machine minder.

Obviously both situations lead to deskilling and demotivation of the employees together with sub-optimal performance and a lack of flexibility and reliability of the technical system.

This deskilling and demotivation of operators due to the automation is a well-known presumed negative side-effect of applying new techniques. Often the introduction of, for instance, robots is being used as a synonym of deskilling, demotivation and loss of jobs, whilst it only appears to shift the level of applying the skills and knowledge.

Technological improvements, however, do not always turn operators into remotely controlled working humanoids. Cavestro (1990) opposes the negative effects of the technological improvements and shows that operators become involved in the work preparation and programming, and in supervision and control of automated systems. Their knowledge is then being used to adjust program parameters such as sequencing and tool length during testing phases and due to changes in the environment and circumstances. Still the centralisation of the decision making limits the intellectual and creative motivation of the operator.
Decentralisation of the planning and scheduling benefits both the required flexibility in small to medium sized batch manufacturing and the possibility to involve the human operators into the decision making at various levels. Decentralisation implies that instead of the long term oriented MRP II plans, plans and schedules for one day to one or two weeks will be used at the shop floor. These plans and schedules will be generated by computer systems dedicated to the management of a specific cell. In fact, such systems use the MRP II plans as input and refine those parts of the plans that are specific to the cell the system is used to manage.

The control systems at factory level perform complex tasks. Van der Drift (1990) describes an open architecture for a Factory controller in which a first production plan is generated by a standard MRP system. As expected in an environment of small to medium sized batch manufacturing, such a plan will often cause overloading of certain resources during certain periods due to the MRP system not knowing the capacity usage of the resources. Through negotiation with the cell or line controllers, the system tries to eliminate these overload situations. Since finite resource capacities are now taken into account, the system can produce more realistic production plans.

1.2 Project environments

In the next sub-sections two projects are described that play a role in this thesis. The first sub-section concerns the ESPRIT ESPRIT Project 2415\(^1\), titled "Distributed Manufacturing, Planning and Control". The ideas within this project have served as the main incentive for the work described in this thesis.

For the implementation of the ideas, the Delfi3 knowledge engineering environment has been used. The experiences based on using Delfi3, have been used for further development of Delfi3 as part of the Delfi project. The second sub-section briefly describes this project.

---

\(^1\) ESPRIT: European Strategic Programme of Research in Information Technology.
After the exit of RDP Technology (London, UK), Imperial College of Science Technology and Medicine (London, UK) became the main contractor of the ESPRIT Project 2415. The consortium consists of Imperial College, Atlas Datensysteme (Essen, Germany), Delft University of Technology (Delft, The Netherlands) and Harmonic Drive Antriebstechnik (Limburg a.d. Lahn, Germany).
1.2.1 The ESPRIT Project 2415

Within manufacturing companies there still exists the urge to improve the efficiency and accuracy of the manufacturing process and of the usage of resources during the manufacturing process. The allocation of resources for the various steps of the production process is done either manually or for instance by using an MRP II system. In both cases the decision making is at a high level within the management hierarchy. This implies that the experiences on specific circumstances and products, which are present at lower levels, do not have the impact that is desired. Besides, reacting to incidents is difficult, especially concerning small or medium sized batches. This problem has led to the ESPRIT Project 2415, which primarily focuses on having decisions taken at the appropriate levels.

Working with the long term plans provided by an MRP II system, the Production Management System (PMS) operates at the area level and assigns the orders to the various cells on the shop floor. Besides the data received from the MRP II, the production planner may feed the PMS with the proper data. The PMS is able to value the capacity of the various resources such as cells and machine type, and offers a better schedule than the MRP II system.

Working at the cell level, the cell operator uses the Operator Management System (OMS) to generate more refined schedules using the capacity of the individual machines. The OMS uses either the PMS schedule as input or the data as entered by the operator.

At both levels, capacity problems may lead to orders being late. To investigate possibilities to minimise the tardiness of the orders, the Expert Management System (EMS) developed in the sequel should act on problem solving requests by either OMS or PMS using the schedule information stored in a common database. The EMS is not noticeable to the user nor accessible by the user, it runs as a background process serving the OMS or PMS.

Figure 1-1 shows the various systems that constitute the Integrated System.

1.2.2 The Delfi Project

At Delft University of Technology the application of theory and practical techniques is inherent to the research being performed at the various facilities. One field of research is applicability of AI, knowledge-based systems in particular, within a variety of problem domains.
Research concentrates on building tools for knowledge-based system applications. The research is being performed within the Delfi Project (De Swaan Arons, 1991), which resulted in the first Delft expert system shell Delfi1 in 1983. This simple expert system shell was based on comparing facts represented by strings.

On the basis of the needs of the Dutch industry, a new expert system shell Delfi2 (backward chaining inference) was developed. This shell was similar to the well-known expert system shell Emycin and had its first prototype working in 1984. Delfi2 was, for instance, used for building applications for the Dutch Railways (Thunissen, 1987). In 1987, further development and a reorganisation of the software lead to the Delfi2+ system, which included an integrated model of backward and forward chaining.

During development and application of the expert system shells it became clear that just using rule based systems limited the use of knowledge-based systems to well-defined and small problem areas where the knowledge had to fit the structure imposed by the rule based nature. The need for using several different techniques within a problem area to solve the
emerging problems initiated the start of the development of the hybrid knowledge-based system Delfi3 (1986). The initial research that resulted in the Delfi3 system has been completed (Jonker, 1990). The development of this system however is ongoing and the system has already been successfully used for building some applications (Goedhart, 1991a).

Similar to the development of the previous Delfi systems, to evaluate the functioning of Delfi3 and to assist further developments, Delfi3 is being used in various projects to implement knowledge-based systems and to gather information on the shortcomings of Delfi3 as to improve them. The ESPRIT Project 2415 is one of these projects where the Delfi3 system is being used to implement knowledge systems and the findings, on how to use Delfi3 and on existing bugs, are used to improve the Delfi3 system. Also Delfi3 is being used by students performing practical work and the findings from these are used as well.

1.3 The Ph.D. research topic

Metal manufacturing companies that produce parts of high variety and in small to medium sized batches face flexibility problems concerning production and control. One of the problems concerns generating the best possible schedule for handling a number of jobs during some period. Using various strategies, different schedules are generated from which the best one is selected.

Having the best one selected from a set of schedules, does not mean that there exists no better schedule. Furthermore, the characteristics of the schedule may be of such an interest that, in the operator’s opinion, it will not pay off generating several other schedules in a search for a better one with similar characteristics. Such characteristics comprise resource utilisation, cell loading and reduced set up times.

However, being satisfied with the schedule the operator may want to improve this schedule, without re-scheduling, using the idle periods of the resources. Possible benefits of such improvements are

- only a few disturbances with respect to the operation sequences in the original schedule, when deriving a better schedule
- reduced idle periods
- reduced lead times
- reduced number of tardy jobs
Unfortunately, the operators are currently not able to pursue this kind of schedule improvement due to the

- complexity of the schedules
- enormous number of ways to improve a schedule
- lack of computerised support

Now, we have sketched the advantages of improving schedules without scheduling and we have shown some reasons why, until now, it is difficult to improve schedules without scheduling. This directs us to the main question for the present research:

Is it possible to devise and to build a knowledge-based support system, that
- is able to counsel on how to improve the lead time of a job,
- uses the idle periods of the machines,
- and does not schedule all over again.

In this thesis we will show that it is sometimes possible to improve an existing schedule without scheduling by intelligent usage of the idle periods of the machines. As a result of the techniques used, job lead times will be improved. Also, the schedule that results from implementing the advice will not be much different from the original schedule concerning the sequences of operations.

1.4 Outline of the thesis

In this section we summarise the contents of this thesis, chapter by chapter.

Drawbacks and thesis
This chapter briefly discusses some philosophies of manufacturing, flexible manufacturing systems and drawbacks of modern manufacturing systems. Before deliberating the present research topic, we will view the project environments that have had their impact on the work done for this thesis.

Scheduling and optimisation
As a prelude to a discussion on job-shop scheduling and how to optimise schedules, we present a survey of various attempts made to automate scheduling and planning in chapter 2. Further, we briefly discuss scheduling, and job-shop scheduling in particular.
How a schedule is generated will be influenced by dispatching rules and heuristics and following this discussion, we deal with evaluating schedules and we focus on evaluation based on due dates. To find the best schedule within a given period, one can obtain a set of possibly different candidate schedules. For instance, different candidate schedules can be obtained by using some heuristics or by varying the dispatching rules that are used. Also other techniques can be used such as overlapping operations, invoking overtime, setting priorities and splitting batches.

From these techniques we select some to be used on existing schedules to improve these schedules. During this thesis, these techniques will be referred to as fine tuning techniques, since we try to fine tune the schedules, for instance to eliminate problems, with minimised disturbance to the schedule.

Schedule fine tuning techniques
In chapter 3 we discuss the essence of three fine tuning techniques. What situations each of the techniques covers is explained. Why and when fine tuning is envisaged to be used, will be discussed in section 3.3. This section also discusses the various combinations of using the fine tuning techniques.

Fine tuning schedules through overlapping manufacture
The Overlapping Manufacture fine tuning technique is reviewed in chapter 4. First we present some characteristics of Overlapping Manufacture before describing a mathematical model. Consecutively, the chapter contains strategies for reducing the search space to swiftly derive the optimal usage of the fine tuning technique.

Most of the chapter has been presented in a paper, which has been accepted for publication in the International Journal of Advanced Manufacturing Technology (Jansen, 1993).

Fine tuning schedules through batch splitting
Both the Vertical Splitting fine tuning technique and the Horizontal Splitting fine tuning technique are reviewed in chapter 5. For each fine tuning technique we present some characteristics and a mathematical model. Consecutively, the chapter contains strategies for reducing the search space to swiftly derive the optimal usage of the fine tuning technique.

Using the fine tuning techniques
In chapter 6, before viewing the implementation of these fine tuning techniques within the context of the ESPRIT Project 2415, we will discuss the
issues that have appeared considering the architecture of the Integrated System and the communication between the various separate systems.

In the implementation part, we will go into more detail of the implementation of the fine tuning techniques, including the main structure of the EMS and the manners of posting a request, invoking EMS and returning the advice.

*Test results EMS*

The results achieved during the testing of the implementation of fine tuning techniques in EMS are described in chapter 7.

*Evaluation and conclusions*

In chapter 8, the ESPRIT Project 2415 and the research will be evaluated. We will also show how the main research question has been answered.
Scheduling and Optimisation

Shop floor scheduling is a difficult problem to solve, because of large computation times in search for an almost optimal solution. Applying heuristics in this search and using knowledge-based systems to support the scheduling can be very fruitful concerning bounding the search space for instance. Various examples of such applications exist, such as the Dynamic Expert Scheduler (DES) (De Swaan Arons, 1989) and the Expert System Scheduler (ESS) (Jain, 1990).

The next section summarises several different application areas for automated scheduling and planning and knowledge-based support. The chapter continues with a discussion on job-shop scheduling, using dispatching rules and heuristics, evaluating schedules and optimising schedules. We conclude this chapter with a section on various optimisation techniques.
2.1 Automated scheduling and planning: recent research

Verbraeck (1990) discusses a scheduling algorithm that uses heuristics. A heuristic order function is used to measure expected difficulties concerning lessons that have to be scheduled. Further, a heuristic planning function is used to give a measure of the quality of a partial schedule. Important is the remark that the decision support system, which makes use of this scheduling algorithm, aids the human planner. However, human interaction remains necessary to give the 'finishing touch' to the schedule.

Scheduling is attended with combinatorial explosion, so 'clever' approaches may limit this. An appropriate approach for time tabling is by using AI techniques, specifically heuristics aimed at finding an acceptable solution. However, these heuristics often come to a halt when finding one solution and do not take into account any criteria for measuring the quality of a schedule. Falcão et al. (1990) have developed an interactive time-tabling system based on the concepts of Decision Support and Expert Systems. Their system is highly interactive, allowing the user to test and compare alternative scenarios based on local modifications. For example by adjusting the weighing of criteria used to qualify a proposed timetable, the user can achieve a different timetable.

Fujiwara and Sakaguchi (1986) use knowledge-based systems to support the planning process concerning power systems. Using conventional techniques the planner generates a plan to deal with a summer peak in power consumption for instance. Occasionally solutions cause other problems, such as overloads and node voltage problems, for which an experienced planner is supposed to know how to solve them. In case of problem situations, the knowledge-based system infers an adjustment plan, which is the basis for a new run by the conventional planning software. This process can be repeated to enhance the plan.

CNC-machines produce by machining materials according to plans. Such a plan describes the steps that have to be taken by the machine. Obviously, errors in such a plan can be very costly and time-consuming since a new plan has to be generated. The generation of a process plan for complex and/or large parts may take a highly qualified expert weeks of intensive effort. Eliyahu, Zaidenberg and Ben-Bassat (1987) indicate that some degree of automation or the use of decision support tools is necessary because of the costs and non-recoverability of mistakes in the design. Their expert system is designed to plan machining processes. Geometry descriptions from a CAD system can be interpreted, albeit a certain
amount of interaction with the user is needed. Based on available knowledge and rules, entered by the user in English and translated by the system, process plans can be generated.

Prevalently, products are composed of several parts that are machined separately. After production they have to be assembled to create the requested product. Obviously, the sequence in which the parts are used to assemble the product is important. Waarts, Boneschanscher and Bronsvoort (1991) describe a semi-automatic assembly sequence planner which input consists of product data derived from a commercial CAD system.

The advantages of their semi-automatic approach encompass the utilisation of the user’s knowledge and experience, and the user’s ability always to be in control because outcomes of previous checks can be overruled. A disadvantage they discovered is the need of an elaborate user interface, which is not needed by a completely automatic sequence planner.

Evidently, the knowledge and experience of experts are a necessity to perform such planning. Feldmann and Kleineidam (1989) incorporate the experts’ knowledge and experience by using expert systems to assist the assembly planning. They point out the convenience of decision explanation and altering of the solution path during the evaluation process, by using expert systems. Within minutes the planner can receive a variety of proposals for an assembly problem.

The survey on automated scheduling and planning as depicted in this section clearly shows the involvement of various techniques to enhance the planning and scheduling processes. A survey by Kusiak and Chen (1988) supports this conclusion. They have reviewed more than twenty planning and scheduling systems, discussing the relationship between AI and operations research approaches. Some conclusions drawn are:

- rule-based expert systems are being more frequently applied in manufacturing planning and scheduling
- heuristics are widely used to assign priorities for jobs, operations and machines
- simulation is widely used in the development of expert systems
- expert systems and operations research approaches can be combined for mutual benefits

These conclusions are not contradicted by the ESPRIT Project 2415. Within this project, knowledge-based systems are included for planning and scheduling support and heuristics are used by the schedulers which have
been developed. After initially approaching some problems using operations research techniques, the knowledge-based systems' strategies were derived using problem domain characteristics to bound the search spaces.

However, critics may refer to the paper by Parrello (1988). In this paper, Parrello clearly explains the cataclysm of using expert systems for car sequencing in a large automobile manufacturing concern. After various attempts it became clear that expert systems were worthless in this domain. Concluding, Parrello states that they vastly overestimated the capabilities of expert system technology and that essentially their approach was wrong; they attacked a procedural problem with non-procedural tools.

These conclusions can be understood, knowing these attempts started in 1984. In the early 1980's the euphoria concerning expert systems was enormous. Everything remotely difficult had to be solved using expert system technology, mostly generating a stand-alone system. Nowadays, expert system technology or knowledge-based system technology is often used to build knowledge-based systems as integral parts of information systems. Within the ESPRIT Project 2415 this direction has been followed and has presented us an Integrated System in which some functions are supported by knowledge-based systems.

2.2 Scheduling

Having a set of available processing units, we can perform tasks that need one or more of these processing units. Before we actually perform a selected number of these tasks, we can devise a schema in which the tasks have been assigned to the various processing units and for each task, the start and finish times for each processing unit have been set. The process of allocating processing units and setting times is called scheduling. The schema, which is the result of a scheduling session, is called a schedule.

Scheduling appears in many different disciplines and concerns various tasks, such as continuous processes in chemical plants and the discrete manufacturing of metal parts. However, a common denominator among the disciplines is the striving towards a schedule that is good enough. What schedule is good enough depends on various aspects within the application environment.

For instance, in a hard real-time system, every time-critical task must meet its timing constraint: the deadline. If such a task fails to complete and to
produce its results by this deadline, a timing fault occurs and the task’s result is of little or no use (Liu, 1991). Such a deadline may be very strict and any violation may be disastrous. French (1982) mentions a typical example of such an occasion: if an aircraft is scheduled to land at a time after which it will have exhausted its fuel, it is really of no importance to the outcome whether it is scheduled to land only one minute too late or one hour. Yet, exceeding the deadline or due date for some tasks may not be that important when the tasks concern the manufacturing of some parts to increase the inventory.

Within the scope of this thesis, we look only at the manufacturing of small or medium quantities of a large variety of products.

2.2.1 Job-shop scheduling

We refer to scheduling as job-shop scheduling when we have to deal with \( n \) jobs that have to be processed by \( m \) machines and the route the jobs follow to visit some of the machines may be different for each job. Visiting a machine means that the job undergoes some processing on that machine. The processing of a job on a machine is called an operation. The various operations that have to be performed on a job, are ordered depending on technological constraints.

Reviewing the literature (Baker, 1974; French, 1982), one finds that several assumptions are made about the structure of the scheduling problems, hence limiting the structure of these problems. Although in general these assumptions do not represent the actual situations, they have been helpful in setting up the mathematical theory of scheduling. Common assumptions are

- each job has \( m \) distinct operations, one on each machine
- no cancellation, that is each job must be processed to completion
- there is only one of each type of machine
- machines never breakdown and are available throughout the scheduling period
- no pre-emption, that is each operation must be completed before another operation may be started on that machine

Obviously, in real life situations machines break down, for some type of machine more than one machine is available, jobs are cancelled and so on. These situations have to be dealt with either by the human scheduler or by the automated scheduler.
Independent of the assumptions that were upheld in the miscellaneous schedulers which have been developed over the years, the schedulers have to select jobs to be processed next on a certain machine. Apparently, at a given moment zero or more jobs may be waiting to be processed by a certain machine and as the machine becomes idle, some job has to be selected from the waiting queue. One way of selecting a job is at random, which can be easily implemented since pseudo random number generators are available to the programmer.

"Select at random", though, is a simple dispatching rule that does not ensure creating a good schedule since the jobs are selected in a higgledy-piggledy manner. A dispatching rule is used to select the next job to be processed from a set of jobs waiting to be serviced. Using more elaborate dispatching rules or heuristics, schedules can be generated that reflect or approach the objectives set by the operator. The next two sections discuss various dispatching rules and heuristics.

2.2.2 Dispatching rules

Dispatching jobs at a machine is a difficult problem since there are $n!$ ways to sequence these jobs if $n$ jobs are waiting. Further, some shop condition at another machine might influence the optimal sequence of jobs at the present machine.

A well-known and often referred to survey on dispatching rules for manufacturing job shop operations was performed by Blackstone, Phillips and Hogg, (1982). In their appendix they have formally defined 34 dispatching rules, divided into four categories. The first three categories concern dispatching rules primarily based on processing time and due dates, and rules based on neither processing time nor due dates. The fourth category, which will not be discussed here, concerns those rules that are formed by combining rules from two or three of the first three categories.

Rules involving processing time

The decision which job to process next on some machine may depend on several characteristics of the jobs waiting to be processed. Regarding the processing time for the jobs' operations, various dispatching rules are in use. Some dispatching rules for a single machine based on processing time are:
• SPT — Shortest Processing Time
  The job with the shortest processing time will be selected. This rule is also referred to as SI or SIO for shortest imminent operation.
• SRPT — Shortest Remaining Processing Time
  Selects the job with the shortest remaining processing time for this operation and the succeeding operations together.
• MRO — Most Remaining Operations
  Selects the job for which the most operations remain to be processed.

Rules involving due dates
Because a job's completion time is often important, various dispatching rules are based on the due dates of the jobs. The following list contains several rules based on due dates. However, some of them do involve processing times:

• EDD — Earliest Due Date
  The job that has to be completed earlier than all other jobs will be selected to be processed first.
• MDD — Earliest Modified Due Date
  Instead of regarding only the due date of a job, also the remaining processing time is taken into account. The job with the smallest combination will be selected.
• MST — Minimum Slack Time
  A job's slack time, for the processing still to be done, at some time $t$ is the difference between time available until the job's due date and the total remaining processing time for this job. Using this MST rule, the job with the earliest due date and the most remaining processing time is selected.

Rules involving other characteristics than processing time or due date
Apart from selecting jobs based on processing time or due date, other characteristics can be employed as well. In section 2.2.1 we introduced dispatching rules using random selection. Besides selecting at random, other dispatching rules exist that belong to this category:

• MQT — Maximum Queuing Time
  The job that has been waiting to be processed the longest, will be selected.
Improving Production Schedules Without Scheduling

- **FIFO — First In First Out**
  The job that has entered the machine’s waiting queue first will also be selected first, and so on. This dispatching rule is also known as FCFS (First Come First Served).

### 2.2.3 Heuristics

Using heuristics, that is by evaluating experience and moving by trial and error to a solution, situations can be identified in which the used dispatching rule should be violated. For instance, normally as a machine becomes idle, the corresponding dispatching rule, that is the dispatching rule used for this machine, is activated to select a job from the waiting queue. The only possibility for the machine to remain idle is when this queue is empty. If the dispatching rule is invoked, apparently only jobs in the queue are considered, which is awkward if some high priority job will arrive in the queue in due time and should be handled first.

*Look Ahead* is a heuristic that also takes into consideration those jobs that will soon arrive in the machine’s waiting queue and that are more critical (expected to be tardy: negative slack time) than the job that would normally be selected by the dispatching rule in use. This means that in case such a job-to-come is selected, idle time will be artificially introduced for the machine. However, if the queue contains a job that fits into this idle period, this job can be selected without consequences for the job-to-come.

Another contemplation using dispatching rules concerns the implications of some selection from the waiting queue. If selecting some job causes another job (in the queue) to become critical, investigate the consequences of selecting that would-be critical job. If selecting this job does not cause any waiting job to become critical, this would-be critical job will be selected. Otherwise, just select the job originally selected by the dispatching rule. This heuristic is called the *Alternate Operation* heuristic.

### 2.3 Evaluating schedules

Clearly, there exist numerous ways to generate a schedule. How the schedule is generated depends on various factors such as the dispatching rules that are used or the heuristics involved. Since for each machine a separate dispatching rule and heuristic can be defined, evidently the number ways to come up with a schedule is enormous. Nevertheless, the operator has to choose which schedule to use at the shop floor, given a set
of possible or candidate schedules. His choice of schedule depends on various constraints, some of them implied by the company's objectives or due to his own objectives as a cell manager. To make a well-founded decision, he has to be able to evaluate the various candidate schedules.

2.3.1 Evaluation based on due dates

Evaluating schedules, various performance measures (French, 1982) can be used. Several of these performance measures involve the due date of the orders or jobs. If we denote the due date of job $i$ by $d_i$ and the finish or completion time by $C_i$, the lateness $L_i$ is defined as $L_i = C_i - d_i$ and the tardiness $T_i$ is defined as $T_i = \max(L_i, 0)$. Based on the assumption that usually the cost of a schedule is related to how much the target dates are missed by, obvious performance measures are the mean lateness $\overline{L}$ ($\overline{L} = \frac{1}{n} \sum_{i=1}^{n} L_i$), the maximum lateness $L_{\max}$ ($L_{\max} = \max\{L_1, L_2, \ldots, L_n\}$), the mean tardiness $\overline{T}$ and the maximum tardiness $T_{\max}$. So, for instance, being interested in minimising the maximum tardiness $T_{\max}$, essentially means that one relates the schedule's 'goodness' or costs to the job that has the largest due date violation.

Rewards or penalties for early jobs

Minimising either $\overline{L}$ or $L_{\max}$ is an appropriate goal whenever there is a positive award for having jobs finishing earlier, possibly proportional to the amount of time. Also such an award may depend on an agreed amount of time.

Although this seems very reasonable, situations exist for which the early completion of a job incurs penalties of the form of inefficient use of resources or inventory holding costs (Kanet, 1989). Such situations often occur in connection with the 'just-in-time' approach.

Penalties for tardy jobs

A job finishing beyond its due date may jeopardise the customer's planning and incur additional costs. These additional costs include penalties as agreed within the contract, loss of goodwill, damages to the company's reputation and costs of intensive communication with the customer. Under these circumstances, minimising either $\overline{T}$ or $T_{\max}$ is appropriate (Sen, 1984).
However, for some jobs it does not matter how late they are, just the fact of being late incurs some penalty. A simple example of a situation for which it does not matter how late one is, can be found in our daily life. If one is supposed to deposit a certain amount at the bank before the end of the day, it does not matter whether the bank has been closed for just a minute or for two hours. So if we deal with jobs of this kind, an objective may be to minimise the number of tardy jobs.

**Considerations**

Mostly we value the time needed to process a job. On the one hand, the early finishing of a job can be rewarded either depending on the amount of time gained or it can be rewarded based on meeting a target date. On the other hand, not finishing a job in time may incur penalties, either for just being late or proportional to the time the job is overdue.

Different schedule optimisation goals exist involving the due dates. One can adopt one of these optimisation goals, for instance minimising $L_{\text{max}}$, when most of the jobs are valued in the same way. However, for small to medium sized batch manufacturing with a large variety of products, the number of customers may also be large and hence the jobs are likely to be valued in different ways. Nevertheless, the operator selects the optimisation goal.

Other rewards or penalties may be defined more fuzzy. For instance, the appreciation by the customer for jobs being earlier or the customer’s dissatisfaction for jobs being too late. Further, some customers are seen to be more important than others. Such contemplations complicate evaluation.

### 2.3.2 Evaluation based on completion times

The completion time $C_i$ of a job $i$ is the time at which the processing of the job finishes. The flow time $F_i$ or lead time for that job is then the completion time minus the time the job became available for the scheduler. Performance measures involving completion times are $C_{\text{max}}$, $\bar{C}$, $F_{\text{max}}$ and $\bar{F}$.

Minimising the maximum flow time $F_{\text{max}}$ actually says that the schedule’s costs directly relate to the length of the longest job. Minimising the maximum completion time $C_{\text{max}}$ however, is essentially saying that the schedule’s costs are directly related to how long the complete set of machines needs for processing the set of jobs.
2.4 Improving schedules

Starting from the principle that some dispatching rules are in use for the machines on the shop floor, a candidate schedule can be generated for the set of jobs that has to be processed. Using the performance measures, the operator is able to evaluate the candidate schedule and verify the results regarding the objectives that have been set. Obviously, the operator seeks a schedule that best satisfies the objectives.

If the operator is not pleased with the candidate schedule and given some time, he is bound to improve on the performance of the schedule. This can be accomplished by generating another candidate schedule (e.g. through another set of dispatching rules), evaluating it and choosing the best one. Given even more time, the operator may be able to generate more candidate schedules.

2.4.1 Using different dispatching rules

As the first candidate schedule has been generated using some dispatching rules, the next schedule might be generated using some other dispatching rules.

Although this seems applicable, just changing the dispatching rules that will be used for the next scheduling run, no operator will have time to try all possible combinations. If there are $m$ machines and $q$ dispatching rules, the number of possible different assignments of the dispatching rules to the machines is $m^q$.

A simple sum in arithmetic learns that it is not feasible for the operator to try all possible combinations. Suppose there are 6 machines and 6 dispatching rules, then the operator has to set up the dispatching rules, generate the candidate schedule and evaluate the results, 46656 times. Being optimistic in using one minute to set up the dispatching rules, generate the schedule and evaluate the results, the whole process would last more than 32 days.

Evidently, only a few candidate schedules will actually be generated and some dispatching rules will be changed by the operator according to his experience and due to the circumstances and characteristics of the previous candidate schedule.
2.4.2 Using heuristics

Another possibility to create a different schedule, is to apply heuristics to the selection process for some machine. Again, for some machine the operator may choose the Look Ahead heuristic, the Alternate Operation heuristic for another machine or just no additional heuristic. The number of heuristics that can be used together with the possible combinations of dispatching rules, causes even more possible combinations to be investigated.

Nevertheless, from the operator’s experience it may show that for some machine, bottleneck or no bottleneck, using a specific heuristic will always enhance the outcome of a scheduling run.

2.4.3 Other optimisation techniques

However, just scheduling again is not enough, since the scheduler has no knowledge of the operator’s intentions to generate a different schedule. In section 2.2.1 we have listed some of the assumptions made in literature for the mathematical theory of scheduling. Combinations of these assumptions or its derivatives are incorporated in some schedulers, whilst other schedulers do allow actions that are not allowed according to these assumptions.

In this section we view several possibilities to optimise the schedules. The improvements, however, relate to the actual situation at the shop floor and to the operator’s objectives.

Exploiting unexpected circumstances

On the shop floor, various events may occur that influence the execution of the schedule currently being used. For instance, machines that break down often cause jobs to finish later than scheduled. If such an event occurs, the expected duration of the disturbance and its effect on the schedule can be assessed. The operator can either stick to the current schedule or he can generate a new schedule. Besides preferring jobs that do not use the broken machine over jobs that do when selecting a job from the queue, no real advantages can be taken from this event.

However, some events that disturb the shop floor can be used to an advantage. For instance, due to the customer’s cancellation of an order, jobs may have to be cancelled. The resulting additional capacity is then available to other jobs either by shifting operations forward or by re-
scheduling with the reduced job set. This may cause jobs to finish less late or even in time.

**Splitting the job's batch**
It is common that more than one machine of a type is available. A derivative of the latter is that for a critical job, the operation that has to be performed on a machine type for which, let say, two machines are available, the batch may be split into two sub-batches to be processed independently. Hence, this operation may finish earlier and the job may lose its criticality.

**Pre-empting machines**
Generally, all parts of a job have to be finished on some machine before the next job can be processed on that machine. Intermediately stopping some job to allow another job to be processed on the same machine is called pre-emption. One of the assumptions (section 2.2.1) when dealing with job-shop scheduling is that no pre-emption is used.

Not allowing pre-emption simplifies scheduling computations. Further, implementing pre-emption on the shop floor may imply setting up a machine with different tooling, which costs time and money. Therefore, allowing pre-emption to occur as a rule will probably complicate scheduling as well as the implementation on the shop floor. However, just for the occasion of optimising the schedule for a small set of jobs, pre-emption is an alternative especially if the revenues are large enough considering the costs of using pre-emption.

For instance, if a more important or critical job arrives at a machine, one might stop the processing of the current job to start processing on this critical job. As with the Look Ahead heuristic, the current job should not become critical itself. Further, in the discussion on the Look Ahead heuristic (section 2.2.3) we mentioned that idle time was intentionally introduced. In case some job from the queue fits in this idle period, this job will be selected. Similarly, when allowing pre-emption, the job originally selected by the dispatching rule could be selected. Of course this job has to be stopped as soon as the critical job arrives and will be continued later.

**Invoking overtime**
One of the limiting factors on the available resource capacity is the availability of manual labour. If steps in the production process have to be performed by humans, production planning has to consider the workers.
The working periods are defined as shifts and schedulers, human or automated, mostly regard only the machine capacity during the workable time as defined by these shifts.

An easy way out when having tardy jobs is to invoke overtime or to extend the shifts, which enlarges the total capacity that can be used. Obviously, this is only true if the available machine capacity relates to the shifts. Machines that will autonomously keep running until the complete batch has been processed, do not regard the shifts. In this case, overtime can be used to set up the machine and start the batch processing after the previous batch has been finished.

**Changing maintenance times**

Machines have to undergo regular and occasional maintenance. During maintenance periods the machines are not available, which has its influence on the available capacity and consequently on the schedule. If necessary, the availability of the machines can be enhanced by shifting the maintenance to another period. Note that if the maintenance period is outside the shift periods this option will only be considered if overtime is required during this maintenance period. Further, we should beware of shifting the maintenance period too far or too often, because machine breakdown due to lack of maintenance and the resulting delays for various jobs may be worse than allowing some job to finish too late.

**Setting job priorities**

Dispatching jobs at a machine, some properties of the jobs waiting to be handled are examined. The dispatching rules mentioned in section 2.2.2 examine, among others, processing times and due dates. However, the job selection may also consider the importance or priority of the jobs, that is, a rush job may go out of turn due to its higher priority. Job priorities (e.g. hot, red hot) influence the outcome of the scheduling process and can be set just to do that.

**Similar machines**

After scheduling, one can get an indication on the loading or utilisation rate of the machines, either by calculation or by reviewing the scheduler's evaluation report. Due to dispatching rules used and the product mix involved, the machine loading differs from time to time. Nevertheless, some machines may appear to be heavily loaded, causing the overall performance of the schedule to be poor.
Within a shop floor, or in a cell, machines may be available that perform similar operations on similar parts. For instance, the difference between similar machines may be the size of the parts they can handle. Using such a machine that handles larger parts as an alternative to the machines that handle the smaller parts, the loading of the machines may be more balanced if the loading of the machines for the smaller parts is higher than the loading for its alternatives, with its possible positive results concerning the new schedule.

**Overlapping operations**

Typically, after finishing an operation on all parts in the batch of a job, the whole batch moves to another machine for the next operation. In the best case, the lead time for the job is the sum of the set up, operation and tear down times for all operations involved and each operation starts directly after the previous operation finishes.

The lead time can be shortened if one allows to overlap operations. That is, to allow a number of already finished parts to go on to the next operation, whilst the remainder of the batch still has to be processed by the current operation. In this way, several sub-batches of parts can be moved to the next operation when finished.

### 2.4.4 Concluding

By performing various scheduling runs using different dispatching rules and heuristics, the operator generates a set of candidate schedules. From this set for instance, the operator selects the best schedule by evaluating the schedules using some performance measure. Also other criteria to appoint some candidate schedule such as the amount of tool changes, may influence the decision by the operator.

Apart from varying the dispatching rules and heuristics, other possibilities exist to influence the appearance of a schedule. Some of them involve creating additional capacity by employing overtime or changing maintenance periods, or using similar or alternative machines. Changing the priority of a job constitutes yet another way to influence the appearance of a schedule.

Other optimisation techniques shorten the lead time of a job, either by splitting the operation or the batch over two or more identical machines or by overlapping consecutive operations. These optimisation techniques have no or little influence on the job and operation sequences in the
schedule. Hence, when the operator favours some schedule and likes to improve it, he can employ such technique leaving most of the sequences intact instead of changing the settings, re-scheduling and possibly disrupting some of the characteristics of the previous schedule.

Also, in case some event occurs on the shop floor the operator may want to generate a new schedule to minimise the burden of the event's consequences. Obviously, he does not want a schedule that leads to rearranging all the jobs on the shop floor, but likes to make use of the actual schedule and improvements to this schedule given the circumstances.

Applying these optimisation techniques for all jobs and all operations, increases the amount of administration and management on the shop floor. Therefore it should be seriously considered to apply these techniques sporadically.

The next chapters discuss the various optimisation techniques that refine the schedule for some jobs. These optimisation techniques will be referred to as schedule fine tuning techniques. Fine tuning techniques, such as overlapping operations, splitting batches or using similar machines, assess and use the machines' idle periods in the schedule. It should be noted that no scheduling takes place, but operations may be shifted somewhat and most of the job sequences in the schedule remain intact.
Schedule Fine Tuning Techniques

"So you're splitting and overlapping some batches" I say.
"Sure," he says. "I know we're not really supposed to do
that, but you need the parts, right?"

The GOAL — Eliyahu M. Goldratt

In chapter 2 “Scheduling and Optimisation” we presented techniques for optimising schedules. Independent of the way a schedule has been generated, some of these techniques can be used to refine or fine tune such a schedule without scheduling all over again.

For instance, the operator may be able to assign the bottle-neck operation of the job to an alternative machine or he may decide to split the batch to make use of the idle time periods to process these sub-batches. Such solutions cause the schedule to be upgraded or fine tuned. However, due to their complexity several fine tuning techniques are rarely used. For these techniques to be applied, an intelligent computerised approach using
knowledge-based systems to represent the knowledge on the subject efficiently and adequately, may be envisaged.

This chapter discusses various schedule fine tuning techniques in more detail.

3.1 Overlapping manufacture

The typical handling of a job is one in which the batch of raw material is transported to the first machine as indicated in the schedule. First the machine has to be installed with the proper fixtures, work handlers and tools. Then, one by one, the raw material parts are fetched from, for instance, a pallet, they are machined and subsequently they are placed onto another pallet. If more than one pallet is necessary, then the pallets are handled one by one. So after machining all the raw material, the intermediate result is transported to the next machine to undergo the scheduled operation.

The machine to which the complete batch is transported is either still busy processing another job, has just finished one or has been idle for some time. Suppose the machine has been idle for some time and the batch is distributed over three pallets. The idea that may come into one's mind is that as soon as one pallet has been filled at the previous machine, this pallet could already have been transported to the current machine, resulting in the current machine not being idle. Yet the idle time of this machine may shift to a later time or idle time may be introduced for other machines or the complete batch has to wait at another machine for a longer time. Also the control of the complete batch is more difficult since the pallets at different locations logically belong to each other and the transportation means might be able to move several pallets at once, where it now has to move one pallet at a time. The result of these thoughts is that the idle time of a machine at a certain moment may be reduced, but idle time at other moments or for other machine may be introduced and the gain in lead time for this job may be zero.

This process of prematurely moving a sub-batch of a job to the next machine as soon as the sub-batch is finished on the current machine is called Overlapping Manufacture. The example shows that overlapping manufacture as a stand-alone goal is difficult to pursue. Nevertheless overlapping manufacture is an option to fine tune a schedule by intelligent use of the idle times of the machines. The option should only be applied for jobs
violating their due dates and jobs for which any improvement of the lead
time is worthwhile. Further, invoking overlapping manufacture causes
additional administrative work to trail the parts in the sub-batches. Such
accounting should be performed by the operator's planning and control
system.

It should be noticed that, when applying overlapping manufacture, no
additional machines will be used since overlapping manufacture focuses
on releasing sub-batches of finished parts and allowing them to arrive
earlier at the next foreseen machine. This means that only those machines
will be used that were originally scheduled for this job.

**figure 3-1** Gantt-chart of part of schedule with a job's two operations
without overlapping

**figure 3-2** Gantt-chart of part of schedule with a job's two operations
after overlapping

In this and the following chapters we refer to the operation that will be
overlapped and the corresponding machine by adding the prefix OVL.
Similarly, we add the prefix NXT for the next operation or machine. So, the operation being overlapped is called the OVL-operation, the next machine is called the NXT-machine.

Figure 3-1 shows a Gantt-chart representation of a part of a schedule, with two operations shown for the ‘striped’ job. Notice that at the time the batch is released by the OVL-machine, the NXT-machine has been idle for some time. Figure 3-2 shows the same schedule after overlapping the operations on the OVL-machine and the NXT-machine.

3.2 Batch Splitting

Depending on the batch size, the operations and times involved and of course the other jobs present, a job will have a particular lead time. If everything is optimal, each of the job’s operations starts immediately after its predecessor has finished. Commonly, a machine is capable of processing one part at a time, which can be seen as a limiting factor for decreasing the job’s lead time. For instance, if the machine was able to process two parts at the same time, the operation time would be half the original operation time (set up and tear down times excluded). Such a situation can be achieved, however in a different way.

\[\text{figure 3-3} \quad \text{two different ways of splitting: horizontally and vertically}\]

In this section we discuss two different ways of splitting a batch: vertically and horizontally. Figure 3-3 depicts the two types of splitting.
3.2.1 Vertical Splitting

Machines that can be found within a cell do not necessarily have to be different from each other. Some machines may be identical and hence capable of handling the same operations. Typically, two machines (or more) of the same type may be present to offer sufficient capacity. However, only one of these machines will be selected during scheduling to perform an operation for a certain job, another one of these machines may be selected to handle another job.

These two identical machines can also be viewed as one 'super'-machine with double capacity or the possibility to process two parts at the same time. Consecutively, to reduce the job's lead time the corresponding operation has to exploit this feature. Since one machine processes a batch, the batch has to be split into two sub-batches to realise the idea of processing two parts simultaneously.

Well, simultaneous processing is the ideal situation. However, due to the other jobs it is likely that the two identical machines are not both available at the time they are needed. More often one machine may start on a sub-batch, whereas the other sub-batch remains in the other machine's queue. Evidently, the first sub-batch may then be larger than the second one.

\[\text{Figure 3-4} \quad \text{time allocation for the last operation of a job on one of the identical machines with all previous operations of the job already finished}\]
This batch splitting will be referred to as Vertical Splitting. The term vertical, as opposed to horizontal, refers to the direction in which the Gantt-chart representation is being investigated, namely vertically. A machine for which splitting is contemplated, will be called an SPL-machine. All identical machines of the type of the SPL-machine will be referred to as SPL*-machine.

Inspecting the Gantt-chart representation of the schedule, we may discover that during the period in which the SPL-machine is handling some job, the SPL*-machines show some idle time. Utilising this idle time, by removing parts of the original batch from the SPL-machine and having them processed as sub-batches on the SPL*-machines, may shorten the lead time for this job. If the job was tardy, the result may be that the job will not be tardy after all.

To depict the application of vertical splitting, we consider a simple example (set up and tear down times are set to zero) in which a tardy job is considered. First we view the last operation of the job, for which two identical machines exist to perform this operation. Figure 3-4 shows the last operation starting later than the latest start time for that operation. Hence, the last operation finishes too late and so does the complete job.

![Diagram](attachment:image.png)

**Figure 3-5** Vertical splitting: possible solution if an operation causes a due date violation and idle time is available on other identical machines.
The other machine of the same type has some idle time between the end of the previous operation and the due date. Using this idle time, it is possible to have the tardy job finishing in time (figure 3-5) by splitting the batch for the last operation and consecutively distributing the sub-batches over the SPL*-machines.

### 3.2.2 Horizontal Splitting

One of the commonly used assumptions is that no pre-emption may occur during scheduling. Suppose pre-emption is allowed to happen, then some operation will start but stopped before completion. Consecutively, another operation will be performed and after finishing that operation the pre-empted operation can continue. Actually, the batch has been split into two sub-batches to be processed sequentially but not in a contiguous period. Since this concerns only one machine, we may refer to this kind of splitting as Horizontal Splitting. But this is using the optimisation technique during scheduling. What about applying this as a fine tuning technique on an existing schedule?

#### Figure 3-6

*The last operation on the SPL-machine for some job and some idle time on that machine*

Suppose we want to improve the lead time for some job using the Horizontal Splitting fine tuning technique on a particular operation, the SPL-operation, splitting the batch into two sub-batches. To be able to move one of the sub-batches, the SPL-machine has to be idle for some time. Obviously, we will not move either of the sub-batches to a later point because this will not decrease the job’s lead time. Hence, the idle period of the SPL-machine must occur between the end of the previous operation of this job and the beginning of the original SPL-operation. Figure 3-6 shows the last operation of a job as the SPL-operation and some idle time that can
be used. Figure 3-7 shows the situation after horizontal splitting has been used.

It should be noted that under normal circumstances idle time as shown in figure 3-7 does not occur. At the moment the SPL-machine becomes idle, a job will be selected from its waiting queue using one of the dispatching rules. Since the previous operation of the job has already been finished, at least this job should be selected. However, using the Look Ahead heuristic, this situation may occur. The machine is waiting for the critical job to arrive. Also for actual schedules, this situation may occur whenever a job is cancelled.

![Diagram showing horizontal split and scheduling](image)

*figure 3-7  applying horizontal splitting shortens the lead time of the job*

3.3 Alternative Manufacture

Generally several machines are available in a factory. Independent of the layout of the shop floor we find machines of the same type, machines that are similar and machines of different types. For manufacturing processes, a number of these machines are used. The selection of the proper machines depends, among other things, on

- the operations that have to be performed on the raw material,
- the dimensions of the (final) product and
- the accuracy of the operations and final measurements.
For example, a turning machine, a milling machine and a sawing machine may be necessary for the production of some gear box. However, which turning machine will be used depends on the size of the gear box.

Although capable of performing the same or almost the same operations, some machines are selected over others for the production of a specific part. Still, the machines that are not selected may be used for the production of that part, but these machines may be more expensive regarding this part's production.

Suppose that after scheduling, some job for the production of part A, will finish too late. Envisage that using a machine of a similar type as an alternative machine causes the job to finish earlier than originally scheduled or even being in time. Depending on the financial benefits of having the job finish earlier or in time and the costs of using another machine, the operator may choose to use the alternative machine.

Refining a schedule using machines of similar type or alternative types is a technique we refer to as Alternative Manufacture. Figure 3-8 displays some characteristics of this technique.

![Figure 3-8](image)

**Figure 3-8** shortening a job's lead time by applying alternative manufacture
3.4 Using and combining fine tuning techniques

Numerous ways exist to use fine tuning techniques. Not only because of the many jobs and operations on which they can be applied, but also because of the various combinations of different fine tuning techniques applied to various operations of the same job and of different jobs.

3.4.1 Why fine tuning?

As stipulated earlier, the operator generates schedules and tries to find an optimal schedule among them regarding some goal, such as minimising the mean lateness. One of the properties of the manufacturing domain we are discussing, is the variety of customers and products. Therefore, the optimisation using a goal such as the mean lateness works well for the majority of the jobs, but not for other jobs.

Because of the diversity of jobs, currently the operator has to be satisfied with the best schedule he can get using the standard scheduling system, but it is possible that even the best schedule may be improved using fine tuning techniques.

3.4.2 When fine tuning?

The operator is not likely to fine tune all jobs since applying one of the fine tuning techniques to a job incurs additional administrative work and costs. Hence only for particular cases, such as for jobs with high rewards or high penalties, overlapping operations or splitting batches may prove to be interesting.

The additional administrative work incurred by fine tuning techniques include

- keeping track of sub-batches
- accounting additional transportation
- knowing when to start the first sub-batch on the next operation when overlapping
- knowing when to split the batch and when to join the sub-batches

The additional costs include costs for

- performing the additional administrative work
- setting up and tearing down tools on the various identical machines when splitting between machines
• setting up and tearing down tools on one machine when splitting on that machine
• transporting the sub-batches to the different machines (splitting) or to the next machine (overlapping).

So if the additional costs are less than the penalty for being late or if the additional costs are less than the reward for being early, then using one or more of the fine tuning techniques may be worthwhile.

3.4.3 Improving one job using one technique

If we look at a typical Gantt-chart we can distinguish various operations belonging to various jobs, whilst each job will consist of at least one operation. Except for the Overlapping Manufacture fine tuning technique, which involves two operations of a job, the fine tuning techniques operate on one operation. To shorten a job's lead time by applying fine tuning on an operation of this job, two possibilities exist.

Firstly, if it is possible to apply fine tuning to the last operation of the job, or the last two operations for Overlapping Manufacture, there will be an improvement of the lead time. Secondly, if it is possible to apply fine tuning to an operation, but not the last one, of the job, there will only be an improvement of the lead time if all the succeeding operations of that job can start earlier. This implies that originally the machines involved are idle before processing these succeeding operations.

![Gantt Chart](image)

**figure 3-9** the actual improvement on the lead time depends on the idle time directly preceding all operations involved
Consecutively, in this case the improvement of the lead time depends upon the idle times and the effect of the fine tuning. For instance, if the fine tuning itself may save 230 minutes, the overall improvement of the lead time will only be 185 minutes if the maximum idle time for the succeeding operations is 185 minutes (see figure 3-9).

3.4.4 Improving one job using several techniques

In the previous section we discussed the improvement of a job's lead time by applying fine tuning techniques to one operation (two in case of Overlapping Manufacture). It became clear that applying such a technique puts restrictions on the succeeding operations, if any. Namely, to be able to profit from an earlier finish of an operation, the succeeding operations should be able to start earlier, thus the corresponding machines should initially be idle for some time.

Suppose a job has four operations and the second and third operation can be overlapped such that the third operation finishes 45 minutes earlier. If the machine processing the fourth operation is idle for 35 minutes before processing that operation, than the fourth operation will finish 35 minutes earlier. If the machine is not idle before processing the operation, there will not be any improvement of the lead time just by overlapping the second and third operation. Still the fourth operation can finish earlier if we can use fine tuning for this operation too.

For instance, if the fourth operation is on a machine for which there are several identical machines available, invoking Vertical Splits may result in the fourth operation being 30 minutes early. Although the overall gain for the job is 30 minutes caused by the fourth operation, it may be the overlapping the second and third operation, and consecutively the early finish of the third operation, that allowed the splitting of the fourth operation.

Concluding it may be obvious that applying just one fine tuning technique to a job will not necessarily lead to an improvement of the lead time, but a combination of fine tuning techniques might.

3.4.5 Improving one job using techniques on other jobs

Apparently, one first investigates the possibilities of fine tuning concerning the operations of the job for which an improvement of the lead time is necessary. However, looking only at that job may result in having no improvement at all due to the inability to shift the last operation of that job, caused by some operation of another job. Therefore, that 'blocking'
operation has to be moved somehow, for instance by applying fine tuning to the corresponding job. Again, this may be dependent on yet another job.

Each application of fine tuning to some operation of some job will increase the costs involved in improving the lead time for that one job we started to look at in the beginning. Clearly, if the costs become larger than the benefits involved, improving the job through improvement of other jobs is not worthwhile.

3.4.6 Consequences of improving some job
Improving the lead time of some job involves shifting the finish times of some of that job's operations to an earlier point, possibly also shifting the finish times of other jobs' operations (see section 3.4.5). The side effect of this forward shifting of finish times is the possibility for other operations that are not involved in the fine tuning process, to start and finish earlier as well.

So evaluating the resulting schedule, the operator may find this schedule to be even better than originally expected after fine tuning the operations.

3.4.7 Improving several jobs
In the previous sections we discussed how to improve the lead time for one particular job by fine tuning one or more operations of that job and, if necessary, even one or more operations of other jobs. Further the side effect of this process may include the improvement of the lead time for other jobs as well.

Improving the lead time of some job using fine tuning is, of course, not restricted to that one job only. Other jobs may be investigated as well and their operations may also be fine tuned.
Fine Tuning Schedules Through Overlapping Manufacture

Numerous possibilities exist for overlapping an operation. The number of sub-batches can vary as well as the volume distribution of the sub-batches. Additionally, several combinations can lead to the maximum forward shift in time of the next operation's finish time, but some combinations involve idle time between the operating on the individual sub-batches. Such idle time cannot be allocated for other operations, so we call this *not allocatable idle time* (NAIT).

Objectives about the Overlapping Manufacture fine tuning technique include:

- to forward shift the end time as far as possible
- to have a small number of sub-batches
- to distribute the number of parts "smoothly" among the various sub-batches
4.1 Characteristics

The various aspects involved when applying overlap need further investigation to obtain a better insight. The time involved in handling a batch of parts comprises the set-up and tear-down times and the operation times per part. All these times are dependent on the part and the machine, the total operation time being dependent on the number of parts in the batch:

\[ \text{total time} = \text{set-up time} + \text{tear-down time} + \text{batch size} \times \text{processing time per part} \]

For the investigation only the operation time per part is important so set-up and tear-down times are set to zero. The operation which is overlapped and the following operation will be referred to as the OVL-operation and the NXT-operation respectively.

Evidently the OVL-operation time per part will be equal to, or less than or greater than the NXT-operation time per part. In this section we will examine these situations by using the following example.

The job concerns the manufacture of 20 identical parts. For overlapping the batch is divided into four sub-batches of which each sub-batch contains 5 parts. The operation times per part will either be equal (one unit per part) or the one will be twice the other (two / one units per part).

4.1.1 Equal operation times per part

The case in which the OVL-operation time equals the NXT-operation time is first investigated. Figure 4-1 (I) shows both operations when no overlapping is applied. If we apply overlapping (figure 4-1 (II)), after five time units the first sub-batch is ready on the OVL-machine and machining on the NXT-machine may take place. Meanwhile the OVL-machine starts machining on the second sub-batch.

When both machines have finished the machining of the current sub-batches — the OVL-machine the second sub-batch and the NXT-machine the first sub-batch — they can start machining the next sub-batch. Since machining on both machines takes the same time per sub-batch, at the end of the OVL-operation only one sub-batch still needs to be handled by the
NXT-machine. Thus the gain in time is the time needed to operate on the first three sub-batches, which is equivalent to 15 time units.

If each sub-batch contains only one part, there is a continuous flow of parts from the OVL-machine to the NXT-machine.

### 4.1.2 OVL-operation time per part < NXT-operation time per part

In this particular case, having a shorter operation time per part on the OVL-machine than on the NXT-machine leads to sub-batches arriving and waiting at the NXT-machine (figure 4-2). This should not pose a problem, since under normal circumstances the complete batch arrives at the NXT-machine.

<table>
<thead>
<tr>
<th>operation</th>
<th>1. without overlapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVL</td>
<td></td>
</tr>
<tr>
<td>NXT</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>operation</th>
<th>II. with overlapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVL</td>
<td></td>
</tr>
<tr>
<td>NXT</td>
<td>forward shift</td>
</tr>
</tbody>
</table>

![figure 4-1](result of overlapping with both operation times set to one time unit)

In this situation also the NXT-machine is continuously operating. The gain in time again is 15 time units, because the NXT-machine starts after the first sub-batch arrives, instead of starting after the fourth sub-batch (which completes the batch) arrives. The difference is 3 sub-batches of 5 parts each. The OVL-operation time is 1 time unit per part, which leads to the forward shift of 15 time units.
4.1.3 OVL-operation time per part > NXT-operation time per part

The situation in which the OVL-machine takes a longer time to machine a part than the NXT-machine presents various possibilities depending on the distribution of parts over the sub-batches. In this example, however, an equal distribution is taken namely "5-5-5-5". Again the situation without overlapping is shown in figure 4-3 (I).

<table>
<thead>
<tr>
<th>Operation</th>
<th>I. without overlapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVL</td>
<td></td>
</tr>
<tr>
<td>NXT</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>II. with overlapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVL</td>
<td></td>
</tr>
<tr>
<td>NXT</td>
<td></td>
</tr>
</tbody>
</table>

**figure 4-2** result of overlapping with OVL-operation time set to one time unit and the NXT-operation time set to two time units

That the OVL-operation takes longer than the NXT-operation implies (in this example) that the NXT-machine has to wait after finishing each sub-batch (figure 4-3 (II)). This idle time cannot be used for other purposes. Again the gain is 15 time units, because the machining by the NXT-machine of the fourth and final sub-batch starts immediately after the finishing of this sub-batch by the OVL-machine.

4.1.4 Remarks

It seems that for each situation the forward shift is the same, but this is because of the settings used in the example. One major factor governing the forward shift is the distribution of the parts over the sub-batches.

For the situation where the OVL-operation time is equal to the NXT-operation time, the size of the first sub-batch defines the forward shift if each
succeeding sub-batch contains, at most, the same number of parts as the directly preceding sub-batch. Otherwise, after finishing a sub-batch, the NXT-machine has to wait until the OVL-machine releases another sub-batch.

![Diagram](image)

Figure 4-3: Result of overlapping with OVL-operation time set to two time units and the NXT-operation time set to one time unit

A similar approach exists when the OVL-operation time is smaller than the NXT-operation time.

If, however, the OVL-operation time per part is larger than the NXT-operation time per part, the forward shift does not explicitly depend on the size of the first sub-batch. The key factors here are the distribution of the parts over the sub-batches and the size of the last sub-batch.

It may be obvious that, if the NAIT is not zero or "just" zero, the NXT-machine can operate on the last sub-batch as soon as it arrives. The forward shift is then the total NXT-operation time for the batch minus the NXT-operation time for the last sub-batch.

Two possibilities exist concerning the last sub-batch if a minimum batch size (e.g. owing to pallet size) has been defined. First, each sub-batch, including the last sub-batch, has to obey this minimum, or only the last
sub-batch may be the exception. In the following sections we consider only the first possibility.

4.2 Mathematical problem description

The previous sub-section described various possibilities involved in the Overlapping Manufacture fine tuning technique. When applying this fine tuning technique the aim is to obtain the largest forward shift possible. As with the scheduling itself, this may be an exhausting task to perform. The problem of finding such a maximum forward shift has been translated into the following problem.

For the mathematical description we define the following parameters:

- \( p_{ovl} \) processing time per unit on the OVL-machine
- \( p_{nxt} \) processing time per unit on the NXT-machine
- \( m \) number of sub-batches
- \( s_i \) number of parts in sub-batch \( i \), \( 1 \leq i \leq m, \ s_i = 0 \) otherwise
- \( S \) total number of parts in the batch
- \( S_{min} \) minimum number of parts in a sub-batch after overlapping
- \( D \) set of all possible distributions of \( S \) parts over \( m \) sub-batches
- \( t_0 \) start of operating the first part at the OVL-machine
- \( ts_{ovl,i} \) start time for machining sub-batch \( i \) at the OVL-machine
- \( ts_{nxt,i} \) start time for machining sub-batch \( i \) at the NXT-machine
- \( te_{ovl} \) end time of machining the batch at the OVL-machine
- \( te_{nxt} \) end time of machining the batch at the NXT-machine

With these parameter definitions we formalise the problem. The maximisation of the forward shift is equivalent to the minimisation of the lead time for both operations, i.e. the difference in time between the start of the machining at the OVL-machine for the first sub-batch and the end of the machining at the NXT-machine for the final sub-batch. First, we define \( ts_{ovl,i} \) and \( ts_{nxt,i} \).

For the OVL-operation \( ts_{ovl,1} = t_0 \) and each succeeding sub-batch \( i \) will start \( s_{i-1} \cdot p_{ovl} \) time units later than sub-batch \( i-1 \). If the end time of the OVL-operation is the start of sub-batch \( m+1 \), we find

\[
\begin{align*}
    ts_{ovl,1} &= t_0 \\
    ts_{ovl,i} &= ts_{ovl,i-1} + s_{i-1} \cdot p_{ovl} \\
    te_{ovl} &= ts_{ovl,m+1}
\end{align*}
\]

Clearly
\[ ts_{\text{OVL},i} = t_0 + \sum_{k=1}^{i-1} s_k p_{\text{OVL}} \quad i = 2, 3, \ldots, m+1 \]

and

\[ te_{\text{OVL}} = t_0 + S p_{\text{OVL}} \]

The NXT-operation is somewhat more complicated. On the NXT-machine the start time \( ts_{\text{NXT},i} \) for operating on sub-batch \( i \) depends on two facts. First, sub-batch \( i \) should be available at the NXT-machine, which is at time \( ts_{\text{OVL},i+1} \) when the OVL-machine has finished machining this sub-batch \( i \). Also the preceding sub-batch \( i-1 \) at the NXT-machine should be finished, which is at time \( ts_{\text{NXT},i-1} + s_{i-1} p_{\text{NXT}} \). Hence, \( ts_{\text{NXT},i} \) is equal to the maximum of these finish times:

\[ ts_{\text{NXT},i} = \max \{ ts_{\text{OVL},i+1}, ts_{\text{NXT},i-1} + s_{i-1} p_{\text{NXT}} \} \]

Regarding the end time of the NXT-operation as the start of sub-batch \( m+1 \), we find:

\[ ts_{\text{NXT},i} = t_0 + s_1 p_{\text{OVL}} \]

\[ ts_{\text{NXT},i} = \max \{ ts_{\text{OVL},i+1}, ts_{\text{NXT},i-1} + s_{i-1} p_{\text{NXT}} \} \quad i = 2, 3, \ldots, m+1 \]

\[ te_{\text{NXT}} = ts_{\text{NXT},m+1} \]

By induction we shall prove that \( ts_{\text{NXT},i} \) can be computed through the formula

\[ ts_{\text{NXT},i} = \max_{1 \leq j \leq i} \{ t_0 + \sum_{k=1}^{j} s_k p_{\text{OVL}} + \sum_{k=j}^{i} s_k p_{\text{NXT}} - s_j p_{\text{NXT}} \} \quad i = 2, 3, \ldots, m+1 \]

Clearly, the basis of induction is

\[ ts_{\text{NXT},2} = \max \{ ts_{\text{OVL},3}, ts_{\text{NXT},1} + s_1 p_{\text{NXT}} \} \]

\[ = \max \{ t_0 + s_1 p_{\text{OVL}} + s_2 p_{\text{OVL}}, t_0 + s_1 p_{\text{OVL}} + s_1 p_{\text{NXT}} \} \]

\[ = \max \{ t_0 + \sum_{k=1}^{2} s_k p_{\text{OVL}} + \sum_{k=2}^{i} s_k p_{\text{NXT}} - s_2 p_{\text{NXT}} \} \]

As the induction hypothesis, suppose that

\[ ts_{\text{NXT},n} = \max_{1 \leq j \leq n} \{ t_0 + \sum_{k=1}^{j} s_k p_{\text{OVL}} + \sum_{k=j}^{n} s_k p_{\text{NXT}} - s_j p_{\text{NXT}} \} \]
Now, NXT-operation \((n + 1)\) can start if both the \((n + 1)\)-st sub-batch has been released by the OVL-machine (at \(ts_{\text{OVL}, n+2}\)) and the \(n\)-th sub-batch at the NXT-machine has been finished (at \(ts_{\text{NXT}, n} + s_n \cdot p_{\text{NXT}}\)). Hence

\[
ts_{\text{NXT}, n+1} = \max \{ ts_{\text{OVL}, n+2}, ts_{\text{NXT}, n} + s_n \cdot p_{\text{NXT}} \}
\]

\[
= \max \{ t_0 + \sum_{k=1}^{n+1} s_k \cdot p_{\text{OVL}} ,
\]

\[
\max \{ t_0 + \sum_{k=1}^{j} s_k \cdot p_{\text{OVL}} + \sum_{k=j+1}^{n} s_k \cdot p_{\text{NXT}} - s_n \cdot p_{\text{NXT}} \} + s_n \cdot p_{\text{NXT}}
\]

\[
= \max \{ t_0 + \sum_{k=1}^{j} s_k \cdot p_{\text{OVL}} + \sum_{k=n+1}^{n+1} s_k \cdot p_{\text{NXT}} ,
\]

\[
\max \{ t_0 + \sum_{k=1}^{j} s_k \cdot p_{\text{OVL}} + \sum_{k=j}^{n} s_k \cdot p_{\text{NXT}} \}
\]

\[
= \max \{ t_0 + \sum_{k=1}^{j} s_k \cdot p_{\text{OVL}} + \sum_{k=j}^{n+1} s_k \cdot p_{\text{NXT}} - s_n \cdot p_{\text{NXT}} \}
\]

\[
= \max \{ t_0 + \sum_{k=1}^{j} s_k \cdot p_{\text{OVL}} + \sum_{k=j}^{n+1} s_k \cdot p_{\text{NXT}} - s_n \cdot p_{\text{NXT}} \}
\]

\[
\]

\[\blacksquare\]

With \(te_{\text{NXT}} = ts_{\text{NXT}, n+1}\) and \(s_{n+1} = 0\), the lead time (LT) for both operations, that is the time span between the start of the OVL-operation and the end of the NXT-operation, is equal to \(te_{\text{NXT}} - t_0\):

\[
LT = te_{\text{NXT}} - t_0 = \max \{ \sum_{k=1}^{j} s_k \cdot p_{\text{OVL}} + \sum_{k=j}^{m} s_k \cdot p_{\text{NXT}} \} \tag{4.1}
\]

The Overlapping Manufacture fine tuning technique causes the batch to be split into several sub-batches. The size of each sub-batch has to be at least equal to a specified minimum \(S_{\text{min}}\).

The problem of finding the maximum forward shift for the NXT-operation, or finding the minimum lead time (MLT) for both operations together, is

\[
\text{minimise} \quad \max \{ \sum_{k=1}^{j} s_k \cdot p_{\text{OVL}} + \sum_{k=j}^{m} s_k \cdot p_{\text{NXT}} \} \tag{MLT-0}
\]

50
subject to \[ \sum_{k=1}^{m} s_k = S \quad \text{(MLT-1)} \]
\[ 1 \leq S_{\text{min}} \leq s_k \leq S \quad k = 1, 2, \ldots, m \quad \text{(MLT-2)} \]
\[ 2 \leq m \leq \left\lfloor \frac{S}{S_{\text{min}}} \right\rfloor \quad \text{(MLT-3)} \]

For this problem several parameters can change. The number of sub-batches \( m \) (for \( m = 1 \) no overlapping exists) varies from 2 to \( \left\lfloor \frac{S}{S_{\text{min}}} \right\rfloor \), a number that is defined by the batch size \( S \) and by the minimum batch size \( S_{\text{min}} \) when applying the Overlapping Manufacture fine tuning technique.

### 4.2.1 A straightforward algorithm

The complete MLT-problem consists of \( \left\lfloor \frac{S}{S_{\text{min}}} \right\rfloor - 1 \) different sub-problems.

Therefore for a given \( m \ (2 \leq m \leq \left\lfloor \frac{S}{S_{\text{min}}} \right\rfloor) \) the problem will be referred to as an MLT\(_m\)-problem.

Just being superficial, one can attempt to solve the MLT-problem using a straightforward algorithm. Algorithm 4-1, written in pseudo-code, is such a straightforward algorithm for solving the MLT-problem and we show that such an algorithm will not solve the MLT-problem in a reasonable amount of time.

When investigating only the second for-loop, which in fact represents an MLT\(_m\)-problem, and discarding the complexity of the body of this loop, we find that the complexity of this loop alone is of \( O(C(S - m \cdot S_{\text{min}} + m - 1, m - 1)) \).

The number of \( r \)-combinations of \( n \) objects is \( C(n, r) = \frac{n!}{r!(n - r)!} \ (n \geq r) \).

Based on this, the number of possibilities to distribute \( S \) non-distinct parts into \( m \) distinct sub-batches with at least \( S_{\text{min}} \) parts in each sub-batch is equal to

\[ \frac{(S - m \cdot S_{\text{min}} + m - 1)!}{(S - m \cdot S_{\text{min}})(m - 1)!} = C(S - m \cdot S_{\text{min}} + m - 1, m - 1), \]

which can be derived as follows:
Each sub-batch will be filled with $S_{\text{min}}$ parts. Distributing the remaining $S - m \cdot S_{\text{min}}$ parts into the $m$ sub-batches, where each sub-batch can receive zero or more parts, can then be seen as arranging $S - m \cdot S_{\text{min}}$ parts and $m - 1$ 'sub-batch separators'. This results in $(S - m \cdot S_{\text{min}} + m - 1)!$ arrangements and since both the parts and the 'sub-batch separators' are non-distinct, this number has to be divided by $(S - m \cdot S_{\text{min}})!$ and by $(m - 1)!$. So the number of distributions of $S$ non-distinct parts into $m$ distinct sub-batches with at least $S_{\text{min}}$ parts in each sub-batch is equal to

$$\frac{(S - m \cdot S_{\text{min}} + m - 1)!}{(S - m \cdot S_{\text{min}})!(m - 1)!}$$

A complexity of $O(C(S - m \cdot S_{\text{min}} + m - 1, m - 1))$ for the algorithm means that the algorithm is not suited for the general MLT-problem, since the algorithm is of exponential complexity.

**Algorithm 4-1**

```plaintext
minimum ← \text{S}^{*} (p_{\text{OVL}} \cdot p_{\text{NXT}})
{ investigate each possible number of sub-batches }
for m ← 2 to \left\lfloor \frac{S}{S_{\text{min}}} \right\rfloor do
{ for this number of sub-batches, examine each possible parts distribution }
for i ← 1 to C(S - m \cdot S_{\text{min}} + m - 1, m - 1) do
    determine next parts distribution
    maximum ← 0
    for j ← 1 to m do
        result ← 0
        for k ← 1 to j do
            result ← result + s_{k} \cdot p_{\text{OVL}}
        end for
        for k ← j to m do
            result ← result + s_{k} \cdot p_{\text{NXT}}
        end for
        if result > maximum then
            maximum ← result
        end if
    end for
    if maximum < minimum then
        minimum ← maximum
    end if
end for
```

Table 4-1 shows the time span involved for some $m$, $S$ and $S_{\text{min}}$, only considering the number of combinations for the parts' distribution over the sub-batches and using a processing time for each distribution of one micro
second \((10^{-6} \text{ seconds})\). Clearly, this shows the inadequacy of the given algorithm, which means that we have to search for another way to solve the MLT-problem, for instance, through a linear programming approach.

<table>
<thead>
<tr>
<th>(S_{\text{min}})</th>
<th>(S)</th>
<th>(m)</th>
<th>\text{number of distributions}</th>
<th>\text{time span needed}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>2</td>
<td>49</td>
<td>0.000 049 seconds</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>12</td>
<td>2.913*10^{10}</td>
<td>8.1 hours</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>26</td>
<td>6.321*10^{13}</td>
<td>2 years</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>2</td>
<td>41</td>
<td>0.000 041 seconds</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>7</td>
<td>54264</td>
<td>0.054 264 seconds</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>11</td>
<td>1.558*10^{13}</td>
<td>180.3 days</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>15</td>
<td>3.800*10^{16}</td>
<td>12 centuries</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>7</td>
<td>1.432*10^{8}</td>
<td>2.39 minutes</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>13</td>
<td>5.225*10^{10}</td>
<td>14.5 hours</td>
</tr>
</tbody>
</table>

\textbf{table 4-1} \hspace{1cm} \text{time span needed if simple computing is needed for all possible distributions of parts over the sub-batches}

\subsection{4.2.2 Linear programming (LP) approach}

The MLT-problem has an LP-type structure: the optimisation of a linear objective function subject to linear constraints. The general formula of an LP-problem is

\[
\text{maximise} \quad \sum_{j=1}^{m} c_j x_j \quad \text{(LP-0)}
\]

\[
\text{subject to} \quad \sum_{j=1}^{m} a_{ij} \cdot x_j \leq b_i \quad i = 1, 2, \ldots, n \quad \text{(LP-1)}
\]

\[
x_j \geq 0 \quad j = 1, 2, \ldots, m \quad \text{(LP-2)}
\]

For the MLT-problem we transform the constraints (MLT-1) and (MLT-2) into the LP-problem constraints formula. First, constraint (MLT-2) is split into two constraints \(s_k \leq S\) and \(s_k \geq S_{\text{min}}\). By substituting \(s_k' = s_k - S_{\text{min}}\) we obtain

\[
s_k' \leq S - S_{\text{min}} \quad \text{and} \quad s_k' \geq 0 \quad \text{for} \ k = 1, 2, \ldots, m
\]
Now constraint (MLT-1) can be transformed. Again we substitute \( s'_k = s_k - S_{\text{min}} \)

\[
\sum_{k=1}^{m} s'_k = S - m \cdot S_{\text{min}}
\]

By splitting this constraint we obtain

\[
\sum_{k=1}^{m} s'_k \leq S - m \cdot S_{\text{min}} \quad \text{and} \quad -\sum_{k=1}^{m} s'_k \leq m \cdot S_{\text{min}} - S
\]

Because constraint (MLT-3) determines the number of variables in the MLT-problem, this constraint is left out and \( \left\lfloor \frac{S}{S_{\text{min}}} \right\rfloor - 1 \) MLT-problems have to be converted to LP-problems.

In conclusion, the constraints of the MLT-problem can be transformed to the following set of constraints

\[
s'_k \leq S - S_{\text{min}} \quad \text{for } k = 1, 2, \ldots, m
\]

\[
\sum_{k=1}^{m} s'_k \leq S - m \cdot S_{\text{min}}
\]

\[
-\sum_{k=1}^{m} s'_k \leq m \cdot S_{\text{min}} - S
\]

\[
s'_k \geq 0 \quad \text{for } k = 1, 2, \ldots, m
\]

If \( n = m + 2 \) and \( x_j = s'_k \) (for \( j = k \) and \( j = 1, 2, \ldots, m \)), then for

\[
b_i = \begin{cases} 
S - S_{\text{min}} & \text{for } i = 1, 2, \ldots, n-2 \\
S - m \cdot S_{\text{min}} & \text{for } i = n-1 \\
m \cdot S_{\text{min}} - S & \text{for } i = n 
\end{cases}
\]

\[
a_{ij} = \begin{cases} 
0 & i \neq j \quad \forall j \quad i = 1, 2, \ldots, n-2 \\
1 & i = j \quad \forall j \quad i = 1, 2, \ldots, n-2 \\
1 & i = n-1 \quad \forall j \\
-1 & i = n \quad \forall j
\end{cases}
\]

54
the transformed set of MLT_\(m\)-problem constraints has been written as

\[
\sum_{j=1}^{m} a_{ij} x_j \leq b_i \quad \text{for } i = 1, 2, \ldots, n \text{ and } \quad x_j \geq 0 \quad \text{for } j = 1, 2, \ldots, m.
\]

Transforming the objective function of the MLT_\(m\)-problem we obtain

\[
\text{minimise } \max_{1 \leq j \leq m} \{ z_j = \sum_{k=j}^{m} x_k \cdot p_{\text{ov}l} + \sum_{k=j}^{m} x_k \cdot p_{\text{nxt}} + S_{\text{min}} \cdot (j \cdot p_{\text{ov}l} + (m - j + 1) \cdot p_{\text{nxt}}) \}
\]

This objective function is quite complicated since its result is the maximum of the results of a set of \(m\) objective functions. To solve the MLT_\(m\)-problem is not an easy task as can be seen in example 4-1.

**Example 4-1**

We investigate the instance of the MLT_2-problem with \(p_{\text{ov}l} = 1, \ p_{\text{nxt}} = 2, \ S = 10\) and \(S_{\text{min}} = 1\).

After transforming the MLT_2-problem and substituting these variables, we obtain the problem

\[
\begin{align*}
\text{minimise} & \quad \max \{ z_1 = 3x_1 + 2x_2 + 5, z_2 = x_1 + 3x_2 + 4 \} \\
\text{subject to} & \quad x_1 + x_2 = 8 \\
& \quad x_1 \leq 9 \\
& \quad x_2 \leq 9 \\
& \quad x_1, x_2 \geq 0
\end{align*}
\]

Nine possible combinations exist to divide the batch into two sub-batches. The implications on the lead time are given in table 4-2.

As long as the variable \(x_1\) has a value less than 3, the \(z_2\) objective function presents the maximum of both objective functions, otherwise the \(z_1\) objective function evaluates as the maximum. The complexity increases with the increase in variables.

The transformed MLT_\(m\)-problem, which in fact is an integer programming problem, shows a set of objective functions from which the one generating the maximum value has to be selected. The MLT_\(m\)-problem resembles the integer multiple objective programming problem as described by Hartley (1983), however, we do not pursue this direction.
Instead, some features of the MLT-problem are used which make the MLT-problem solving easier. These features are described in the next subsection and they are used by the knowledge-based system described in chapter 6.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$z_1$</th>
<th>$z_2$</th>
<th>lead time $\max{z_1,z_2}$</th>
<th>$s_1$</th>
<th>$s_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>21</td>
<td>28</td>
<td>28</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>22</td>
<td>26</td>
<td>26</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>23</td>
<td>24</td>
<td>24</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>24</td>
<td>22</td>
<td>24</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>25</td>
<td>20</td>
<td>25</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>26</td>
<td>18</td>
<td>26</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>27</td>
<td>16</td>
<td>27</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>28</td>
<td>14</td>
<td>28</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>29</td>
<td>12</td>
<td>29</td>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 4-2** For a given set of variables, the solution for the MLT$_2$-problem is given. The distributions with the minimum lead time of 24 for the MLT$_2$-problem are $[s_1,s_2] = [3,7]$ and $[s_1,s_2] = [4,6]$

### 4.3. Optimal solution by search space reduction

The MLT$_m$-problem has some features that enable us to reduce the space that has to be searched to derive a solution. The division based on the operation times as used in section 4.1 will be continued in this section.

The lead time $LT$ for the OVL-operation and the NXT-operation together is given by formula (4.1). However, $LT$ can also be expressed as the sum of the time between the start of the operation on the OVL-machine and the start on the NXT-machine, the time to manufacture the batch on the NXT-machine and the not allocatable idle time (NAIT) on the NXT-machine for this job:

$$LT = s_1 \cdot p_{ovl} + s_2 \cdot p_{nxt} + \text{NAIT}$$  \hspace{1cm} (4.2)

#### 4.3.1 Equal operation time per part: $p_{ovl} = p_{nxt}$

Typically, if we view this situation the objective function (MLT-0) of an arbitrary MLT$_m$-problem reduces from
\[
\text{minimise } \max_{1 \leq j \leq m} \left\{ \sum_{k=1}^{j} s_k \cdot p_{ovl} + \sum_{k=j}^{m} s_k \cdot p_{nxt} \right\}
\]

to

\[
\text{minimise } \max_{1 \leq j \leq m} \left\{ \sum_{k=1}^{m} s_k \cdot p_{ovl} + s_j \cdot p_{ovl} \right\}
\]

The latter is equivalent to \( \minimise \max_{1 \leq j \leq m} \{ S \cdot p_{ovl} + s_j \cdot p_{ovl} \} \). This means that we have to find a distribution of parts for which the largest sub-batch size is minimised. Thus, having the set \( D \) of all possible distributions, a particular subset \( D' \subseteq D \) for which \( \max_{1 \leq j \leq m} \{ s_j \} \) is minimised has to be found. Because \( m \) is fixed for the MLT\(_m\)-problem, the characteristics of the subset \( D' \) are known: for each distribution in \( D' \) all parts must be distributed evenly amongst the sub-batches with \( \max_{1 \leq j \leq m} \{ s_j \} = \left\lfloor \frac{S}{m} \right\rfloor \).

For example, if \( S = 18 \), \( S_{\min} = 1 \) and \( m = 4 \), then \( D = \{ [15,1,1,1], [14,2,1,1], [14,1,2,1], ..., [6,4,4,4], [5,5,4,4], [5,4,5,4], ..., [1,1,1,1] \} \). Apparently, the set of maxima is \( \{ 5, 6, ..., 14, 15 \} \) and, because the minimum of these is required, \( D' = \{ [5,5,4,4], [5,4,5,4], [5,4,4,5], [4,5,5,4], [4,5,4,5], [4,4,5,5], [4,5,4,5] \} \) and

\[
\max_{1 \leq j \leq m} \{ s_j \} = \max_{1 \leq j \leq m} \{ 4, 5 \} = 5 = \left\lfloor \frac{18}{4} \right\rfloor = \left\lfloor \frac{S}{m} \right\rfloor.
\]

Now, for the general MLT-problem, \( m \) varies from 2 to \( \left\lfloor \frac{S}{S_{\min}} \right\rfloor \) which means that for the corresponding minima the largest sub-batch sizes vary from \( \left\lfloor \frac{S}{2} \right\rfloor \) to \( \left\lfloor \frac{S}{S_{\min}} \right\rfloor \), where \( \left\lfloor \frac{S}{2} \right\rfloor \geq \left\lfloor \frac{S}{3} \right\rfloor \geq \ldots \geq \left\lfloor \frac{S}{S_{\min}} \right\rfloor \).

As the lead time is determined by the size of the largest sub-batch size and this size has to be minimised, the lead time is the smallest for \( m \) equal to the largest number of sub-batches possible. This sets the largest sub-batch size for the optimal solution equal to
Improving Production Schedules Without Scheduling

\[ \left\lfloor \frac{S}{S_{\text{min}}} \right\rfloor = \begin{cases} S_{\text{min}} & \text{for } S \text{ an integer multiple of } S_{\text{min}} \\ S_{\text{min}} + 1 & \text{otherwise} \end{cases} \]

In the process of establishing the set of distributions \( D' \) for which the lead time is minimised, the NAIT does not play any role because the lead time depends only on the size of the largest sub-batch. However, one of the objectives is to minimise the NAIT, which can be achieved by selecting from \( D' \) the distribution for which the number of parts in a sub-batch is larger or equal to the preceding sub-batch. Hence, the distribution which will be selected is of the format

\[ [S_{\text{min}}, \ldots, S_{\text{min}}, S_{\text{min}} + 1, \ldots, S_{\text{min}} + 1]. \]

Clearly, since another objective is to minimise the number of sub-batches, knowing the forward shift needed to finish the job in time enables us to select a value \( m \) for which the job is in time and the number of sub-batches is minimal.

### 4.3.2 Operation times per part not equal: \( p_{\text{ovl}} < p_{\text{NXT}} \)

For this situation the objective function of an arbitrary MLT\(_{\text{pr}}\)-problem can be reduced to

\[
\text{minimise } \max \left\{ \sum_{k=1}^{m} s_k p_{\text{ovl}} - \sum_{k=j+1}^{m} s_k p_{\text{ovl}} + \sum_{k=j}^{m} s_k p_{\text{NXT}} \right\}
\]

or

\[
\text{minimise } \max \left\{ S_j p_{\text{ovl}} + s_j p_{\text{ovl}} + \sum_{k=j}^{m} s_k (p_{\text{NXT}} - p_{\text{ovl}}) \right\}
\]

This does not really simplify the problem, so we investigate this specific situation further.

Because the NXT-operation starts as soon as the first sub-batch has been finished by the OVL-operation and each subsequent sub-batch needs more time at the OVL-machine than at the NXT-machine, the smaller the first sub-batch the earlier the start time for the first sub-batch on the NXT-machine. The lead time for the job is \( s_1 p_{\text{ovl}} + S_j p_{\text{NXT}} + \text{NAIT} \). This is minimal for \( s_1 = S_{\text{min}} \) and NAIT = 0. For the NXT-machine, NAIT is equal to 0 if at the end of operating on each sub-batch the next sub-batch is available. If \( m \cdot S_{\text{min}} < S \), to obtain the minimum lead time one or more sub-batches will have a size of \( S_{\text{min}} + 1 \).
For this situation, minimising the number of sub-batches may also be an additional objective. Contrary to the \( p_{ovl} = p_{nxt} \) situation, the lead time may stay the same for fewer sub-batches but with \( s_1 = S_{min} \). This means that NAIT should remain 0 or, equivalently, the arrival time for each sub-batch \( i \) (except for the first sub-batch) at the NXT-machine should be at the same time as the end of the previous sub-batch (sub-batch \( i-1 \)) or before the end of the previous sub-batch.

![Diagram showing lead time and number of sub-batches](image)

**Figure 4-4** example of minimising both the lead time and the number of sub-batches

Figure 4-4 shows an example in which the batch size is 20, \( S_{min} = 5 \), \( p_{ovl} = 2 \) and \( p_{nxt} = 3 \). Setting each sub-batch size equal to \( S_{min} \) we obtain the largest forward shift possible, namely 30, hence the first objective, minimising the lead time, has been satisfied. The objective of minimising the number of sub-batches, can be achieved by increasing the sizes of the second and third sub-batch to 7 and 8 respectively, which decreases the number of sub-batches.

The number of sub-batches can be decreased, without increasing the lead time, if one is able to increase the size of some of the sub-batches (except the first sub-batch). For this, the following constraints have to be met

\[
\sum_{k=2}^{i} s_k \cdot p_{ovl} \leq \sum_{k=1}^{i-1} s_k \cdot p_{nxt} \quad i = 2, 3, \ldots, m \tag{4.3}
\]

\[
s_k \begin{cases} = S_{min} & k = 1 \\ \geq S_{min} & k > 1 \end{cases} \tag{4.4}
\]

Rewriting constraint (4.3) and replacing \( s_1 \) by \( S_{min} \) we find

\[
s_i \leq S_{min} + \sum_{k=1}^{i-1} s_k \left( \frac{p_{nxt}}{p_{ovl}} - 1 \right) \quad i = 2, 3, \ldots, m
\]

59
For some sub-batch \( i \) to increase by 1, the term \( \sum_{k=1}^{i-1} s_k \left( \frac{p_{\text{NXT}}}{p_{\text{OVL}}} - 1 \right) \) has to be larger than or equal to 1. Hence, the size of the first \( i - 1 \) sub-batches has to be larger than

\[
\frac{1}{\frac{p_{\text{NXT}}}{p_{\text{OVL}}} - 1}
\]

or

\[
\sum_{k=1}^{i-1} s_k \geq \frac{p_{\text{OVL}}}{p_{\text{NXT}} - p_{\text{OVL}}}
\]

Finding the first sub-batch able to be increased by 1 is easy now. Suppose the index of this sub-batch is \( i \), then all the previous sub-batches are of size \( S_{\text{min}} \). Hence,

\[
(i - 1) \cdot S_{\text{min}} \geq \frac{p_{\text{OVL}}}{p_{\text{NXT}} - p_{\text{OVL}}}
\]

and

\[
i \geq \frac{p_{\text{OVL}}}{S_{\text{min}} \cdot (p_{\text{NXT}} - p_{\text{OVL}}) + 1}
\]

The index of the first sub-batch able to increase is

\[
i = \left\lfloor \frac{p_{\text{OVL}}}{S_{\text{min}} \cdot (p_{\text{NXT}} - p_{\text{OVL}}) + 1} \right\rfloor
\]

Similarly, the index \( j \) of the first sub-batch to increase by some \( q \) whilst the size of the first \( (j - 1) \) sub-batches remain equal to \( S_{\text{min}} \) is

\[
j = \left\lfloor \frac{q \cdot p_{\text{OVL}}}{S_{\text{min}} \cdot (p_{\text{NXT}} - p_{\text{OVL}}) + 1} \right\rfloor
\]

When \( p_{\text{OVL}} < p_{\text{NXT}} \), consecutive sub-batches may increase in size. For each sub-batch the possible increase relative to \( S_{\text{min}} \) can be determined. Let \( d_i \) denote the increment for sub-batch \( i \). Then,

\[
\sum_{k=1}^{i-1} s_k \geq d_i \cdot \frac{p_{\text{OVL}}}{p_{\text{NXT}} - p_{\text{OVL}}}
\]

or

\[
d_i \leq \sum_{k=1}^{i-1} s_k \left( \frac{p_{\text{NXT}} - p_{\text{OVL}}}{p_{\text{OVL}}} \right)
\]

Because \( s_i = S_{\text{min}} + d_i \) and \( s_1 = S_{\text{min}} \),
\[
\begin{align*}
    d_1 &= 0 \\
    d_i &= \left[ (i - 1) \cdot S_{\text{min}} + \sum_{k=1}^{i-1} d_k \right] \cdot \left( \frac{p_{\text{NXT}} - p_{\text{OVL}}}{p_{\text{OVL}}} \right)
\end{align*}
\] (4.5)

The process to determine each \( s_i \) such that both the lead time and the number of sub-batches are minimal, can make use of this recursive relation.

### 4.3.3 Operation times per part not equal: \( p_{\text{OVL}} > p_{\text{NXT}} \)

As with the previous situation, reducing the objective function does not simplify the problem

\[
\text{minimise} \quad \max_{1 \leq j \leq m} \left\{ \sum_i s_i \cdot p_{\text{NXT}} + s_j \cdot p_{\text{NXT}} + \sum_{k=1}^{i} s_k \cdot (p_{\text{OVL}} - p_{\text{NXT}}) \right\},
\]

thus we investigate this specific situation further.

As soon as the first sub-batch has been finished by the OVL-operation it can be operated on by the NXT-machine. Since the NXT-operation time is less than the OVL-operation time, depending on the size of the second sub-batch, the NXT-machine will be idle for some time. The lead time for the job is again \( s_1 \cdot p_{\text{OVL}} + S \cdot p_{\text{NXT}} + \text{NAIT} \). If the last sub-batch can be machined at the NXT-machine as soon as it finishes on the OVL-machine, the lead time is equal to \( S \cdot p_{\text{OVL}} + s_m \cdot p_{\text{NXT}} \).

To start, each sub-batch is filled with the minimum quantity and the remainder is distributed amongst the various sub-batches except the last sub-batch. The condition here is that NAIT does not need to be 0 to have the minimum lead time, however, the NXT-machine should be able to operate on the last sub-batch at the time it is released by the OVL-machine. If this constraint is met, the minimum lead time is determined by the size of the last sub-batch, preferably set to the minimum batch size: \( s_m = S_{\text{min}} \).

Similar to the situation with \( p_{\text{OVL}} < p_{\text{NXT}} \), the number of sub-batches can be decreased without increasing the lead time. This is only possible if one is able to increase the size of some sub-batches except the last sub-batch. The constraints that have to be met, with \( s_m \) substituted by \( S_{\text{min}} \), are

\[
\begin{align*}
    s_i &\leq S_{\text{min}} + \sum_{k=m+1}^{m} s_k \left( \frac{p_{\text{OVL}}}{p_{\text{NXT}}} - 1 \right) \\
    i &= 2, 3, \ldots, m \\
    s_k &\geq S_{\text{min}} \quad k = m \\
    s_k &\geq S_{\text{min}} \quad k < m
\end{align*}
\]
Correspondingly, for each sub-batch $i$ the increase $d_i$ relative to $S_{\min}$ can be determined, so

$$d_i \leq \sum_{k=i+1}^{m} s_k \left( \frac{P_{OVL} - P_{NXT}}{p_{NXT}} \right)$$

Because $s_i = S_{\min} + d_i$ and $s_m = S_{\min}$ we find

$$d_m = 0$$

$$d_i = \lceil \left((m-i)\cdot S_{\min} + \sum_{k=i+1}^{m} d_k\right) \cdot \left( \frac{P_{OVL} - P_{NXT}}{p_{NXT}} \right) \rceil$$

(4.6)

The process to determine each $s_i$, such that both the lead time and the number of sub-batches are minimal, can make use of this recursive relation.

It should be noted that sub-batches $s_1, ..., s_l$ will be equal to zero, because the number of sub-batches may decrease. Hence the index $l$ must be determined for the sub-batches $s_1', s_2', ..., s_p'$ with $s_i' = s_{i+l}, i = 1, 2, ..., m - l$ and $p = m - l$.

The preference of the operators to have an NAIT of zero, to prevent "warming-up" the machine repeatedly, can be established after reaching a minimum lead time by releasing the first sub-batch not immediately after finishing it, but NAIT time units later.
Fine Tuning Schedules Through Batch Splitting

5.1 Vertical Splitting

Two or more identical machines have to be available to even consider applying the Vertical Splitting fine tuning technique. The main objectives of applying the Vertical Splitting fine tuning technique are:

- the end time of the job has to be forward shifted as much as possible
- the number of sub-batches has to be as small as possible

The main ingredient of vertical splitting is the combined available capacity of the SPL*-machines during the period between the end of the previous operation and the end of the original SPL-operation, called the SPL-
period. Further, whenever the SPL-operation is not the last operation of a tardy job within the cell, the Vertical Splitting fine tuning technique is only applicable if all succeeding operations can be shifted forward. Hence, the machines that handle these operations have to be idle before operating on the tardy job.

Noticeably, the maximum possible profit of applying the Vertical Splitting fine tuning technique is equal to the smallest preceding free time for the succeeding operations if the SPL-operation is not the last operation of the tardy job. Otherwise, the profit is equal to the number of minutes the operation finishes earlier as a result of splitting the batch.

5.1.1 Mathematical problem description

Having described the objectives and characteristics of the Vertical Splitting fine tuning technique, we will now discuss the problem of finding the best forward shift using Vertical Splitting fine tuning technique.

For the mathematical description we define the following parameters:

- \( p_{su} \) processing time per unit for the SPL-operation
- \( M \) number of SPL-machines
- \( m \) number of sub-batches
- \( s_i \) number of parts in sub-batch \( i \), \( 1 \leq i \leq m , s_i = 0 \) otherwise
- \( S \) total number of parts in the batch
- \( S_{min} \) minimum number of parts in a sub-batch after splitting
- \( G \) set of gaps
- \( g_{ij} \) the \( j \)-th gap on machine \( i \) and its length
- \( g \) \( m \)-tuple of selected gaps
- \( t_{ij} \) start time of \( g_{ij} \)

Typically, we view only those gaps (including the space occupied by the SPL-operation) present within the SPL-period, that is between the end of the previous operation (time point 0) and the end of the SPL-operation (time point \( t_e \)). These gaps constitute the set \( G \). Although involved in the SPL-operation, without loss of generality the set up and tear down times are set to zero.

When investigating the Vertical Splitting fine tuning technique, several constraints have to be met. At first, at least two identical machines that can perform the SPL-operation have to be available. Secondly, after splitting each sub-batch has to contain at least \( S_{min} \) parts and further the maximum
number of sub-batches after splitting depends on the number of identical machines, the batch size $S$ and on $S_{min}$.

Figure 5-1 shows the various gaps within the period $(0, t_e)$, with $g_{11}$ the 'gap' for the original operation. We define the lead time $LT$ for this operation to be the time from time point 0 to the moment all parts have been finished. Evidently, the lead time will be equal to $t_e$ for the original situation and $t_e = t_{11} + g_{11} = t_{11} + S \cdot p_{SPL}$.

![Diagram showing gaps between operations on identical machines, including idle periods and lead time calculations.]

*Figure 5-1* gaps: idle periods on (identical) machines, including the time to perform the original operation.

Our aim to maximise the forward shift of the finish time of the SPL-operation is equivalent to minimising the LT for the SPL-operation. To calculate the minimum LT for the SPL-operation when splitting into $m$ sub-batches, each combination of $m$ gaps $g_{i_1 j_1}, g_{i_2 j_2}, \ldots, g_{i_m j_m}$ ($i_r \neq i_s \leftrightarrow r \neq s$) has to be investigated and for each of these gap combinations, also all possible combinations to fill the sub-batches have to be investigated. Now $m$, the number of sub-batches, ranges from 2 to the minimum of $M$ and $\left\lfloor \frac{S}{S_{min}} \right\rfloor$.

Having an $m$-gap combination $g = (g_{i_1 j_1}, g_{i_2 j_2}, \ldots, g_{i_m j_m})$, the lead time depends on the distribution of the parts over the $m$ sub-batches such that each sub-batch $s_k$ contains at least $S_{min}$ parts and all sub-batches together contain a total of $S$ parts. For each gap $g_{i_k j_k}$ involved, the end time $t_{e_k}$ can be calculated:

65
\[ t_{ek} = t_{i_{kij}} + s_{k} \cdot \text{p}_{\text{SRL}} \]

LT is equal to the maximum of the end times for all gaps involved, so

\[ \text{LT} = \max \{ t_{ek} \} = \max \left\{ t_{i_{kij}} + s_{k} \cdot \text{p}_{\text{SRL}} \right\} \]

The problem of finding the minimum lead time, or the MLT-problem for vertical splitting, can be formulated as follows:

\[ \forall \ m \in \{ 2, ..., \min \{ M, \left\lfloor \frac{S}{S_{\text{min}}} \right\rfloor \} \} \]
\[ \forall \ \mathbf{g} \in \{ (g_{i_{1j1}}, g_{i_{2j2}}, ..., g_{i_{mj}}) \mid g_{i_{kj}} \in G \land i_{r} \neq i_{k} \} \]

\[ \text{minimise} \quad \max \left\{ t_{i_{kij}} + s_{k} \cdot \text{p}_{\text{SRL}} \right\} \]

subject to

\[ \sum_{k=1}^{m} s_{k} = S \]

\[ 1 \leq S_{\text{min}} \leq s_{k} \leq S \quad \text{for} \quad k = 1, 2, ..., m \]

\[ s_{k} \cdot \text{p}_{\text{SRL}} \leq g_{i_{kj}} \quad \text{for} \quad k = 1, 2, ..., m \]

5.1.2 Optimal solution by search space reduction

The problem of finding the minimum lead time incorporates the sub-division of the original batch into several sub-batches. The number of sub-batches into which the original batch has to be split is not known in advance. Fixing the number of sub-batches \( m \), we can solve the MLT\(_{m}\)-problem using a special algorithm. This algorithm uses the characteristics of the MLT\(_{m}\)-problem to solve it. By comparing the solutions to the various MLT\(_{k}\)-problems \( (k = 2, ..., \min \{ M, \left\lfloor \frac{S}{S_{\text{min}}} \right\rfloor \}) \), the best solution is selected.

For the optimal solution a specific number of sub-batches \( m \) is needed and hence we are considering combinations of \( m \) gaps. The restriction of having at least \( S_{\text{min}} \) parts in a sub-batch, leads to the removal of gaps that are not large enough \( (g_{ij} < S_{\text{min}} \cdot \text{p}_{\text{SRL}}) \) from the set \( G \) of 'selectable' gaps. Because each remaining gap will contain at least \( S_{\text{min}} \) parts when selected, we fill each gap with \( S_{\text{min}} \) parts beforehand by adjusting the length and start time of each gap.
Chapter 5  
Fine Tuning Schedules Through Batch Splitting

This means that the number of parts to distribute over the \( m \) selected gaps in the \( m \)-gap combination decreases to \( S - m \cdot S_{\text{min}} \). For further reference we use \( S_{\text{rem}} \) to denote \( S - m \cdot S_{\text{min}} \), the remaining parts to be distributed. With the filling each gap with \( S_{\text{min}} \) parts as a starting-point, \( s_i \) denotes the number of parts in the sub-batch in addition to the \( S_{\text{min}} \) parts already present.

To clarify some algorithms and explanations, we let \( H \) denote the set of gaps that have been selected to comprise an \( m \)-gap combination \( g \). Consequently, for some \( g = (g_{i_{1j_1}}, g_{i_{2j_2}}, \ldots, g_{i_{nj_n}}) \) we have the set \( H = \{g_{i_{1j_1}}, g_{i_{2j_2}}, \ldots, g_{i_{nj_n}}\} \).

In addition to the parameters as defined in section 5.1.1 “Mathematical problem description”, we define the following parameters:

- \( H \) set of gaps selected to be part of the \( m \)-gap combination
- \( H_g \) total number of parts that can be put into the gaps in \( H \)
- \( r_{ij} \) relative finish time for gap \( g_{ij} \), depends on \( G \) and \( H \)
- \( u_{ij} \) usable start time for gap \( g_{ij} \), depends on \( G \) and \( H \)
- \( mL_t_g \) minimum lead time for \( m \)-gap combination \( g \)
- \( S_{\text{rem}} \) remaining number of parts to be distributed

Algorithm 5-1

<table>
<thead>
<tr>
<th>Step 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialisation step.</td>
</tr>
<tr>
<td>• ( t \leftarrow 0 )</td>
</tr>
<tr>
<td>• ( s_k \leftarrow 0, k = 1, 2, \ldots, m )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Progress in time and calculate the number of parts that can be assigned to each sub-batch ( s_k ) (( k = 1, 2, \ldots, m )).</td>
</tr>
<tr>
<td>• ( t \leftarrow t + 1 )</td>
</tr>
<tr>
<td>• for ( k = 1 ) to ( m ) do</td>
</tr>
<tr>
<td>if ( (\lfloor \frac{t - l_{ik}}{P_{\text{ser}}} \rfloor &gt; s_k) ) and ( (\lfloor \frac{g_{ik}}{P_{\text{ser}}} \rfloor &gt; s_k) ) and ( (\sum_{l=1}^{m} s_l &lt; S_{\text{rem}}) ) then</td>
</tr>
<tr>
<td>( s_k \leftarrow s_k + 1 )</td>
</tr>
<tr>
<td>fi</td>
</tr>
<tr>
<td>od</td>
</tr>
</tbody>
</table>
Algorithm 5-1 (continued)

Step 2
Repeat the previous step if not all parts have been assigned.

\[ \text{if } \left( \sum_{k=1}^{m} s_k < S_{rem} \right) \text{ then } \]
\[ \text{goto step 1} \]
\[ \text{fi} \]

Additionally, since we are discussing the Vertical Splitting fine tuning technique, no two gaps in an \( m \)-gap combination \( g \) are allowed to be on the same machine. Subsequently, for a gap to be a potential gap in an \( m \)-gap combination, this gap has to be larger than all preceding gaps on the same machine. That is, each gap \( g_{i+j+1} \) has to be able to contain more parts than the gap \( g_{ij} \) immediately preceding and selectable, otherwise it is impossible for an \( m \)-gap combination holding gap \( g_{ij+1} \) to have a minimum lead time shorter than an \( m \)-gap combination holding gap \( g_{ij} \).

Definition 5-1
An \( m \)-gap combination \( g = (g_{i_{j_1}}, g_{i_{j_2}}, \ldots, g_{i_{j_m}}) \) is a feasible \( m \)-gap combination if \( \sum_{k=1}^{m} \frac{g_{i_{j_k}}}{p_{\text{spl}}} \geq S_{rem} \).

Theorem 5-1
For a given \( m \), let \( g = (g_{i_{j_1}}, g_{i_{j_2}}, \ldots, g_{i_{j_m}}) \) be a feasible \( m \)-gap combination, with \( t_{i_{j_1}} \leq t_{i_{j_2}} \leq \ldots \leq t_{i_{j_m}} \), and let \( s_1, s_2, \ldots, s_m \) be a parts distribution for which \( \text{mlt}_g = \max \{ t_{i_{j_1}} + s_1 \cdot p_{\text{spl}}, t_{i_{j_2}} + s_2 \cdot p_{\text{spl}}, \ldots, t_{i_{j_m}} + s_m \cdot p_{\text{spl}} \} \) denotes the minimum lead time for \( g \).

Consider \( m \)-gap combination \( g' \), constructed by replacing a gap \( g_{i_{j_k}} \) of \( g \) by \( g_{i_{j_k}'} \in G \) with \( t_{i_{j_k}} < t_{i_{j_k}'} \), and \( \frac{g_{i_{j_k}'}}{p_{\text{spl}}} \geq \frac{g_{i_{j_k}}}{p_{\text{spl}}} \). Then \( \text{mlt}_{g'} \geq \text{mlt}_g \).

Proof
To prove the theorem, we consider two situations.

(1) \( \sum_{k=1}^{m} \frac{g_{i_{j_k}}}{p_{\text{spl}}} = S_{rem} \)

If gap \( g_{i_{j_k}} \), can not hold as many parts as \( g_{i_{j_k}'} \), that is \( \frac{g_{i_{j_k}'}}{p_{\text{spl}}} < \frac{g_{i_{j_k}}}{p_{\text{spl}}} \),

then \( \frac{g_{i_{j_k}'}}{p_{\text{spl}}} + \sum_{k=1}^{x-1} \frac{g_{i_{j_k}}}{p_{\text{spl}}} + \sum_{k=x+1}^{m} \frac{g_{i_{j_k}}}{p_{\text{spl}}} < S_{rem} \) and hence \( g' \) is not feasible.
If gap $g_{i,j}^*$ can hold as many parts as $g_{i,j}$, that is \( \left\lfloor \frac{g_{i,j}^*}{p_{spl}} \right\rfloor \leq \left\lfloor \frac{g_{i,j}}{p_{spl}} \right\rfloor \), then

\[
mlt_{g'} = \max \left\{ t_{i_1j_1} + s_1^* p_{spl}, \ldots, t_{i_kj_k} + s_k^* p_{spl}, \ldots, t_{m_jm} + s_m^* p_{spl} \right\}
= \max \left\{ t_{i_1j_1} + s_1^* p_{spl}, \ldots, t_{i_kj_k} + s_k^* p_{spl}, \text{mlt}_g \right\} \geq \text{mlt}_g
\]
since $t_{i_1j_1} > t_{i_kj_k} + s_k^* p_{spl}$.

\[\sum_{k=1}^{m} \left\lfloor \frac{g_{k}^{i,j}}{p_{spl}} \right\rfloor > S_{rem}\]

Because at least one gap will not be completely filled, several distributions of the parts over the gaps (into sub-batches) are possible. Using algorithm 5-1, the optimal distribution $s_1$, $s_2$, ..., $s_x$, ..., $s_m$ for $g$ has been generated such that

\[\text{mlt}_g = \max \left\{ t_{i_1j_1} + s_1^* p_{spl}, \ldots, t_{i_kj_k} + s_k^* p_{spl}, \ldots, t_{m_jm} + s_m^* p_{spl} \right\}.
\]

Replacing $g_{i,j}$ by $g_{i,j}^*$ causes algorithm 5-1 to generate a new optimal distribution $s'_1$, $s'_2$, ..., $s'_x$, ..., $s'_m$ for $g'$. Obviously, for all $k$ for which $t_{i_kj_k} + s_k^* p_{spl} \leq t_{i_kj_k}$ the filling of the gaps will be the same, hence $s_k = s'_k$ and $t_{i_kj_k} + s_k^* p_{spl} = t_{i_kj_k} + s'_k p_{spl}$. For the other gaps in $g'$, except for $g_{i_1j_1}^*$, algorithm 5-1 will fill the gaps with at least as many parts as the same gaps in $g$. Consecutively, for all these gaps $g_{i_kj_k}$, $s_k' \geq s_k$ and $t_{i_kj_k} + s'_k p_{spl} \geq t_{i_kj_k} + s_k p_{spl}$.

Evidently $s'_x \leq s_x$ and $t_{i_1j_1} + s'_x p_{spl} > t_{i_1j_1} + s_x p_{spl}$. Therefore,

\[
\text{mlt}_{g'} = \max \left\{ t_{i_1j_1} + s_1^* p_{spl}, \ldots, t_{i_kj_k} + s_k^* p_{spl}, \ldots, t_{m_jm} + s_m^* p_{spl} \right\}
\geq \max \left\{ t_{i_1j_1} + s_1^* p_{spl}, \ldots, t_{i_kj_k} + s_k^* p_{spl}, \ldots, t_{m_jm} + s_m^* p_{spl} \right\} = \text{mlt}_g.
\]

These measures reduce the search space by reducing the number of gaps to be investigated. However, we still have to investigate several combinations of the gaps available in the set of selectable gaps.

The first thing to do now is to find a first $m$-gap combination $g$ capable of holding the $S_{rem}$ parts. The gaps comprising the $m$-gap combination can be found in the set $H$, which is initially empty ($|H| = 0$). So we have to move gaps one by one from set $G$ to set $H$ until $m$ gaps are available in the set $H$ ($|H| = m$) and an $m$-gap combination can be formed. The criterion we use to select gaps from $G$, before moving them to $H$, is based on the usable start time of a gap.

Firstly, the usable start time $u_{ij}$ of a gap $g_{ij}$ is equal to the start time $t_{ij}$, if none of the gaps in $H$ is on machine $i$: $u_{ij} = t_{ij}$. Otherwise $g_{ij} \in G$ will be
used to replace \( g_{ij'} \in H \ (j > j') \) and consequently part of the gap \( g_{ij} \) will be used for the number of parts originally ‘assigned’ to \( g_{ij'} \). So only the surplus of parts possible to assign to \( g_{ij} \) is of interest and hence the usable start time \( u_{ij} \) will be equal to \( u_{ij} = t_{ij} + p_{spn} \left\lceil \frac{g_{ij'}}{p_{spn}} \right\rceil \).

Secondly, from the moment \( H \) contains \( m \) gaps, a gap can only be added to \( H \) by replacing another gap in \( H \). Similar to the previous situation, if gap \( g_{ij'} \in H \) then gap \( g_{ij} \) will replace \( g_{ij'} \) and \( u_{ij} = t_{ij} + p_{spn} \left\lceil \frac{g_{ij'}}{p_{spn}} \right\rceil \). Otherwise, gap \( g_{ij} \) will replace the smallest gap in \( H \) and \( u_{ij} = t_{ij} + p_{spn} \left\lceil \frac{\min \{H\}}{p_{spn}} \right\rceil \).

Summarising, the usable start time \( u_{ij} \) of a gap \( g_{ij} \) can be calculated as follows:

\[
u_{ij} = \begin{cases} t_{ij} & |H| < m \land \exists j' : g_{ij'} \in H \\ t_{ij} + p_{spn} \left\lceil \frac{\min \{H\}}{p_{spn}} \right\rceil & |H| = m \land \exists j' : g_{ij'} \in H \\ t_{ij} + p_{spn} \left\lceil \frac{g_{ij'}}{p_{spn}} \right\rceil & \exists j' : g_{ij'} \in H \end{cases}
\]

We can now move gaps from \( G \) to \( H \), each time selecting the gap with the earliest usable start time. Before moving a gap from \( G \) to \( H \), the gap in \( H \) that is possibly on the same machine as the new gap, has to be removed. We continue this process until \( H \) contains \( m \) gaps. These gaps together are either able to contain the \( S_{rem} \) parts or not. The number of parts that can be put into the \( m \) gaps of \( H \) is given by \( H_S = \sum_{k=1}^{m} \left\lfloor \frac{g_{ki}}{p_{spn}} \right\rfloor \).

If the \( m \) gaps do not offer enough capacity, we start replacing gaps in \( H \) by gaps in \( G \). To increase \( H_S \), we calculate the usable start time for the gaps in \( G \) and select the gap with the smallest usable start time. If \( H \) contains a gap that is on the same machine as the selected gap, it will be replaced by the selected gap. Otherwise the smallest gap in \( H \) will be replaced. We continue this process until an \( m \)-gap combination has been formed with \( H_S \) large enough, or no gaps are left in \( G \). For the \( m \)-gap combination we can determine the parts distribution and the finish time using algorithm 5-1.
Definition 5-2

Given an \( m \)-gap combination \( g = (g_{i_1 j_1}, g_{i_2 j_2}, \ldots, g_{i_m j_m}) \) and the set of selectable gaps \( G \), then there exists no gap in \( G \) with a usable start time less than the maximum usable start time, \( \max \{ u_{i_1 j_1}, u_{i_2 j_2}, \ldots, u_{i_m j_m} \} \), of all the gaps comprising \( g \).

Having an \( m \)-gap combination and the finish time we can reduce the number of other \( m \)-gap combinations to investigate by removing gaps that will never improve the best solution to date. Since gaps that have a start time later than this finish time can never improve the solution, these gaps will be removed from \( G \).

Theorem 5-2

For a given \( m \), let \( g = (g_{i_1 j_1}, g_{i_2 j_2}, \ldots, g_{i_m j_m}) \) be a feasible \( m \)-gap combination and let \( s_1, s_2, \ldots, s_m \) be a parts distribution for which \( ml_{tg} = \max \{ t_{i_1 j_1} + s_1 \cdot p_{spl}, t_{i_2 j_2} + s_2 \cdot p_{spl}, \ldots, t_{i_m j_m} + s_m \cdot p_{spl} \} \) denotes the minimum lead time for \( g \).

Consider \( m \)-gap combination \( g' \), constructed by replacing a gap \( g_{i_k j_k} \) of \( g \) by \( g_{i_k j_k} \in G \) with \( t_{i_k j_k} \geq ml_{tg} \). Then \( ml_{tg'} \geq ml_{tg} \).

Proof

Replacing \( g_{i_k j_k} \) by \( g_{i_k j_k} \) causes algorithm 5-1 to generate a new optimal distribution \( s'_1, s'_2, \ldots, s'_{m'} \) for \( g' \).

It is evident that, since \( t_{i_k j_k} \geq ml_{tg} \), \( t_{i_k j_k} + s'_x \cdot p_{spl} \geq ml_{tg} \) for any \( s'_x \geq 0 \), hence \( ml_{tg'} = \max \{ t_{i_k j_k} + s'_x \cdot p_{spl}, ml_{tg} \} \geq ml_{tg} \).

Similarly gaps in \( G \), that can hold less than or as many parts as the smallest gap in \( H \), can be removed for they can not improve the current solution either.

Theorem 5-3

For a given \( m \), let \( g = (g_{i_1 j_1}, g_{i_2 j_2}, \ldots, g_{i_m j_m}) \) be a feasible \( m \)-gap combination, with \( t_{i_1 j_1} \leq t_{i_2 j_2} \leq \ldots \leq t_{i_m j_m} \), and let \( s_1, s_2, \ldots, s_m \) be a parts distribution for which \( ml_{tg} = \max \{ t_{i_1 j_1} + s_1 \cdot p_{spl}, t_{i_2 j_2} + s_2 \cdot p_{spl}, \ldots, t_{i_m j_m} + s_m \cdot p_{spl} \} \) denotes the minimum lead time for \( g \).

Consider \( m \)-gap combination \( g' \), constructed by replacing the smallest gap \( g_{i_k j_k} \) of \( g \) (\( g_{i_k j_k} = \min \{ g_{i_1 j_1}, g_{i_2 j_2}, \ldots, g_{i_m j_m} \} \)) by \( g_{i_k j_k} \in G \) with \( \lfloor \frac{g_{i_k j_k}}{p_{spl}} \rfloor \geq \lfloor \frac{g_{i_k j_k}}{p_{spl}} \rfloor \). Then \( ml_{tg'} \geq ml_{tg} \).
Proof
By definition 5-2, we know that $u_{i_{k}j_{s}} \leq u_{ij}$. Additionally, also $t_{i_{k}j_{s}} \leq t_{ij}$ holds, which can be easily seen since in an earlier stage for some $H'$, $g_{i_{k}j_{s}}$ has been selected instead of $g_{ij}$, and then also

$$u_{i_{k}j_{s}} = t_{i_{k}j_{s}} + p_{s_{k}} = \min\left\{H'\right\}$$

$$t_{ij} + p_{s_{k}} = \min\left\{H'\right\}$$

was true and therefore $t_{i_{k}j_{s}} \leq t_{ij}$.

If $g_{ij}$ is on the same machine as $g_{i_{k}j_{s}}$, that is $i = i_{x}$, then according to theorem 5-1 $mlt_{g} \geq mlt_{g}$.

For $i \neq i_{x}$, we consider two situations.

1. \[ \sum_{k=1}^{m} \left\lfloor \frac{g_{u_{ij}}}{p_{s_{k}}} \right\rfloor = S_{rem} \]

If gap $g_{ij}$ can not hold as many parts as $g_{i_{k}j_{s}}$, that is \( \left\lfloor \frac{g_{ij}}{p_{s_{k}}} \right\rfloor < \left\lfloor \frac{g_{i_{k}j_{s}}}{p_{s_{k}}} \right\rfloor \), then

\[ \left\lfloor \frac{g_{ij}}{p_{s_{k}}} \right\rfloor + \sum_{k=1}^{x-1} \left\lfloor \frac{g_{u_{ij}}}{p_{s_{k}}} \right\rfloor + \sum_{k=x+1}^{m} \left\lfloor \frac{g_{u_{ij}}}{p_{s_{k}}} \right\rfloor < S_{rem} \]

and hence $g'$ is not feasible.

If gap $g_{ij}$ can hold as many parts as $g_{i_{k}j_{s}}$, that is \( \left\lfloor \frac{g_{ij}}{p_{s_{k}}} \right\rfloor = \left\lfloor \frac{g_{i_{k}j_{s}}}{p_{s_{k}}} \right\rfloor \), then

\[ mlt_{g'} = \max\left\{ t_{i_{k}j_{s}} + s_{1}p_{s_{k}}, \ldots, t_{ij} + s_{x}p_{s_{k}}, \ldots, t_{i_{k}j_{s}} + s_{m}p_{s_{k}} \right\} \]

\[ = \max\left\{ t_{ij} + s_{x}p_{s_{k}}, mlt_{g} \right\} \geq mlt_{g} \]

since $t_{ij} \geq t_{i_{k}j_{s}}$.

2. \[ \sum_{k=1}^{m} \left\lfloor \frac{g_{u_{ij}}}{p_{s_{k}}} \right\rfloor > S_{rem} \]

Because at least one gap will not be completely filled, several distributions of the parts over the gaps are possible. Using algorithm 5-1, the optimal distribution $s_{1}, s_{2}, \ldots, s_{x}, \ldots, s_{m}$ for $g$ has been generated such that

\[ mlt_{g} = \max\left\{ t_{i_{k}j_{s}} + s_{1}p_{s_{k}}, \ldots, t_{ij} + s_{x}p_{s_{k}}, \ldots, t_{i_{k}j_{s}} + s_{m}p_{s_{k}} \right\} \]

Replacing $g_{i_{k}j_{s}}$ by $g_{ij}$ causes algorithm 5-1 to generate a new optimal distribution $s'_{1}, s'_{2}, \ldots, s'_{x}, \ldots, s'_{m}$ for $g'$.

Obviously, for all $k$ for which $t_{i_{k}j_{s}} + s_{k}p_{s_{k}} \leq t_{i_{k}j_{s}}$, the filling of the gaps will be the same, hence $s_{k} = s'_{k}$ and $t_{i_{k}j_{s}} + s_{k}p_{s_{k}} = t_{i_{k}j_{s}} + s'_{k}p_{s_{k}}$. For the other gaps in $g'$, except for $g_{ij}$, algorithm 5-1 will fill the gaps with at least as many parts as the same gaps in $g$. Consecutively, for all these gaps $g_{u_{ij}}$, $s'_{k} \geq s_{k}$ and $t_{i_{k}j_{s}} + s'_{k}p_{s_{k}} \geq t_{i_{k}j_{s}} + s_{k}p_{s_{k}}$.

Evidently $s'_{x} \leq s_{x}$ and $t_{ij} + s'_{x}p_{s_{k}} \geq t_{i_{k}j_{s}} + s_{x}p_{s_{k}}$. Therefore,
Chapter 5  Fine Tuning Schedules Through Batch Splitting

\[ \text{mlt}_g' = \max \{ t_{i_j} + s'_1 \cdot p_{sp}, \ldots, t_{i_j} + s'_x \cdot p_{sp}, \ldots, t_{i_m} + s'_m \cdot p_{sp} \} \]

\[ \geq \max \{ t_{i_j} + s_1 \cdot p_{sp}, \ldots, t_{i_j} + s_x \cdot p_{sp}, \ldots, t_{i_m} + s_m \cdot p_{sp} \} = \text{mlt}_g. \]

Using the usable start time we form a new \( m \)-gap combination by replacing a gap in \( H \) by the gap from \( G \) with the earliest usable start time. We continue this process until no gap is left in \( G \). Comparing the various \( m \)-gap combinations and their finish times, we can select the \( m \)-gap combination with the earliest finish time and hence the shortest lead time or largest forward shift.

Algorithm 5-2 reflects the previous paragraphs.

**Algorithm 5-2**

\( H_{MLT} \) denotes the \( m \)-gap combination or set of gaps that leads to the minimum lead time \( MLT \). Initially, \( G \) contains all gaps and both \( H \) and \( H_{MLT} \) are empty.

**Step 0**

Initialisation step.

- Remove all gaps that can not hold at least \( S_{\text{min}} \) parts.
  
  \[ G \leftarrow G \setminus \{ g_{pq} \mid g_{pq} < S_{\text{min}} \cdot p_{sp} \} \]

- Each selected gap will contain at least \( S_{\text{min}} \) parts.
  Adjust the gap sizes and the gap start times.
  
  \[ \forall g_{pq} \in G : SP_q \leftarrow SP_q + S_{\text{min}} \cdot p_{sp} \]
  
  \[ \forall g_{pq} \in G : g_{pq} \leftarrow g_{pq} - S_{\text{min}} \cdot p_{sp} \]

- For each SPL*-machine remove from \( G \) the gaps that can not hold more parts than the preceding gaps.
  
  \[ G \leftarrow G \setminus \{ g_{pq} \in G \mid \frac{g_{pq}}{p_{sp}} \leq \max \{ \frac{u_{av}}{p_{sp}} \mid g_{av} \in G \land \]
  
  \[ u = p \land v < q \} \}

**Step 1**

Select \( m \) gaps (on different machines).

**Sub-step 1-1**

Calculate the usable start time for each gap in \( G \).

- \( \forall g_{pq} \in G : u_{pq} \leftarrow \begin{cases} t_{pq} + p_{sp} \cdot \frac{g_{pr}}{p_{sp}} & \exists r : g_{pr} \in H \\ t_{pq} & \nexists r : g_{pr} \in H \end{cases} \)
Algorithm 5-2 (continued)

Sub-step 1-2
Move the gap with the earliest usable start time from G to H and, if necessary, remove the gap already in H that is on the same machine.

- \( H \leftarrow H \cup \{ g_{pq} \in G \mid u_{pq} = \min \{ u_{rs} \mid g_{rs} \in G \} \wedge p = \min \{ x \mid u_{xy} = \min \{ u_{rs} \mid g_{rs} \in G \} \wedge g_{xy} \in G \} \} \)

- \( G \leftarrow G \setminus \{ g_{pq} \in G \mid u_{pq} = \min \{ u_{rs} \mid g_{rs} \in G \} \wedge p = \min \{ x \mid u_{xy} = \min \{ u_{rs} \mid g_{rs} \in G \} \wedge g_{xy} \in G \} \} \)

- \( H \leftarrow H \setminus \{ g_{pq} \in H \mid \exists g_{pq'} \in H : q' > q \} \)

Sub-step 1-3
Determine whether \( m \) gaps have been selected and if not, whether additional gaps can be selected.

- if (\( |H| < m \)) then
  - if (\( G = \emptyset \)) then
    stop \( \{ \text{no gaps left, no } m \text{-gap combination creatable} \} \)
  - else
    goto sub-step 1-2
  - fi
- fi

Step 2
Determine the first feasible \( m \)-gap combination \( g \).

Sub-step 2-1
Calculate the number of parts that can be put into the \( m \)-gap combination.

- \( H_S \leftarrow \sum_{g \in G} \frac{g}{p_{SPL}} \)

Sub-step 2-2
Determine whether a feasible \( m \)-gap combination has been found or whether one can be found.

- if (\( H_S \geq S_{rem} \)) then
  goto step 3
- elseif (\( G = \emptyset \)) then
  stop \( \{ \text{no gaps left, no feasible } m \text{-gap combination possible} \} \)
- fi

Sub-step 2-3
Calculate the usable start time for each gap in \( G \).

- \( \forall g_{pq} \in G : u_{pq} \leftarrow \begin{cases} \frac{t_{pq} + p_{SPL}}{p_{SPL}} \leftarrow \frac{g_{pr}}{p_{SPL}} \forall r : g_{pr} \in H \end{cases} \)

- \( \forall g_{pq} \in G : u_{pq} \leftarrow \begin{cases} \min \{ H \} \leftarrow \frac{t_{pq} + p_{SPL}}{p_{SPL}} \forall r : g_{pr} \in H \end{cases} \)
Algorithm 5-2 (continued)

Sub-step 2-4
- Move the gap with the earliest usable start time from G to H.
  \[ H \leftarrow H \cup \{ g_{pq} \in G \mid u_{pq} = \min \{ u_{rs} \mid g_{rs} \in G \} \land \\
  p = \min \{ x \mid u_{xy} = \min \{ u_{rs} \mid g_{rs} \in G \} \land g_{xy} \in G \} \} \]
  \[ G \leftarrow G \setminus \{ g_{pq} \in G \mid u_{pq} = \min \{ u_{rs} \mid g_{rs} \in G \} \land \\
  p = \min \{ x \mid u_{xy} = \min \{ u_{rs} \mid g_{rs} \in G \} \land g_{xy} \in G \} \} \]
- If present remove the gap already in H that is on the same machine, otherwise remove the smallest gap.
  \[ H \leftarrow H \setminus \{ g_{pq} \in H \mid \exists g_{pq': q' > q} \} \]
  \[ |H| > m : H \leftarrow H \setminus \{ g_{pq} \in H \mid g_{pq} = \min \{ H \} \land \\
  p = \min \{ x \mid g_{xy} = \min \{ H \} \land g_{xy} \in H \} \} \]

Step 3
Fill the m sub-batches using algorithm 5-1 and determine MLT and H_{MLT}.
- \( LT \leftarrow \text{final } t \text{ from algorithm 5-1} \)
- \( \text{if } (H_{MLT} = \emptyset) \text{ or } (H_{MLT} \neq \emptyset \text{ and } LT < MLT) \text{ then} \)
  - \( \text{MLT} \leftarrow LT \)
  - \( H_{MLT} \leftarrow H \)
  \fi

Step 4
Reduce search space by removing undesirable gaps from G.
- Remove all gaps from G for which the start time is later than LT.
  \[ G \leftarrow G \setminus \{ g_{pq} \in G \mid SP_q \geq LT \} \]
- Remove all gaps from G that are not larger than the smallest gap of the m-gap combination g.
  \[ G \leftarrow G \setminus \{ g_{pq} \in G \mid g_{pq} \leq \min \{ g_{rs} \mid g_{rs} \in H \} \} \]
- Determine whether there are still gaps left to be investigated.
  \( \text{if } (G = \emptyset) \text{ then} \)
  - \( \text{stop} \{ \text{no gaps left}\} \)
  \fi

Step 5
Determine the next m-gap combination.
Sub-step 5-1
Calculate the usable start time for each gap in G.
- \( \forall g_{pq} \in G : u_{pq} \leftarrow \begin{cases} 
  t_{pq} + p_{SPL} \cdot \frac{g_{pr}}{p_{SPL}} & \exists r : g_{pr} \in H \\
  t_{pq} + p_{SPL} \cdot \min \{ H \} & \nexists r : g_{pr} \in H 
\end{cases} \)
Algorithm 5.2 (continued)

<table>
<thead>
<tr>
<th>Sub-step 5.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Move the gap with the earliest usable start time from $G$ to $H$.</td>
</tr>
<tr>
<td>$H \leftarrow H \cup { g_{pq} \in G \mid u_{pq} = \min { u_{rs} \mid g_{rs} \in G } \land p = \min { x \mid u_{xy} = \min { u_{rs} \mid g_{rs} \in G } \land g_{xy} \in G } }$</td>
</tr>
<tr>
<td>$G \leftarrow G \setminus { g_{pq} \in G \mid u_{pq} = \min { u_{rs} \mid g_{rs} \in G } \land p = \min { x \mid u_{xy} = \min { u_{rs} \mid g_{rs} \in G } \land g_{xy} \in G } }$</td>
</tr>
<tr>
<td>- If present remove the gap already in $H$ that is on the same machine, otherwise remove the smallest gap.</td>
</tr>
<tr>
<td>$H \leftarrow H \setminus { g_{pq} \in H \mid \exists g_{pq}': q' &gt; q }$</td>
</tr>
<tr>
<td>$</td>
</tr>
<tr>
<td>- Goto step 3.</td>
</tr>
</tbody>
</table>

5.2 Horizontal Splitting

Due to the Look Ahead heuristic, idle time can be artificially created to suit some critical job (see section 2.2.3). These artificially created gaps in the schedule, may be useful for partly processing some job’s operation. This decreases the finish time for the entire operation and possibly also the lead time of the job decreases.

This splitting of a job’s batch into several sub-batches, all processed in different and not contiguous time periods, is called Horizontal Splitting. It can also be regarded as invoking pre-emption after scheduling, for the result is the same. For actual schedules, additional gaps arise on the cancellation of jobs or due to the updating of the schedule using actual values.

The main objectives of applying the Horizontal Splitting fine tuning technique are:

- the end time of the job has to be forward shifted as far as possible
- the number of sub-batches has to be as small as possible

5.2.1 Mathematical problem description

Before discussing the problem of finding the best forward shift using Horizontal Splitting fine tuning technique and presenting the mathematical description, we define the following parameters:
\( p_{\text{src}} \)  
processing time per unit for the SPL-operation

\( m \)  
number of sub-batches

\( s_i \)  
number of parts in sub-batch \( i \), \( 1 \leq i \leq m, s_i = 0 \) otherwise

\( S \)  
total number of parts in the batch

\( S_{\text{min}} \)  
minimum number of parts in a sub-batch after splitting

\( G \)  
set of gaps

\( g_j \)  
the \( j \)-th gap on the SPL-machine and its length

\( g \)  
\( m \)-tuple of selected gaps

\( t_j \)  
start time of \( g_j \)

Typically, we view only those gaps (including the space occupied by the SPL-operation) present within the SPL-period, that is between the end of the previous operation \( (t = 0) \) and the end of the SPL-operation \( (t = t_c) \). These gaps form the set \( G \). Although involved in the SPL-operation, without loss of generality we set the set up and tear down times to zero since we can view the gaps as the idle time minus the set up.

Looking at the Horizontal Splitting fine tuning technique, we can identify several constraints. First, after splitting each sub-batch has to contain at least \( S_{\text{min}} \) parts and further the maximum number of sub-batches after splitting depends on the number of gaps that can hold at least \( S_{\text{min}} \) parts, the batch size \( S \) and on \( S_{\text{min}} \).

The problem of finding the minimum lead time, or the MLT-problem for horizontal splitting, can be formulated as follows:

\[
\forall m \in \{2, \ldots, \min\left\{ \left\lfloor \frac{S}{S_{\text{min}}} \right\rfloor \right\} \},
\forall g \in \{(g_{j_1}, g_{j_2}, \ldots, g_{j_m}) \ | \ g_j \in G\}
\]

minimise

\[
\max_{1 \leq k \leq m} \left\{ t_{j_k} + s_k^* p_{\text{src}} \right\}
\]

subject to

\[
\sum_{k=1}^{m} s_k = S
\]

\[
1 \leq S_{\text{min}} \leq s_k \leq S \quad k = 1, 2, \ldots, m
\]

\[
s_k^* p_{\text{src}} \leq g_j \quad k = 1, 2, \ldots, m
\]

5.2.2 Optimal solution by search space reduction

The Horizontal Splitting fine tuning technique is similar to the Vertical Splitting fine tuning technique considering it attempts to forward shift the finish of the SPL-operation by sub-dividing the batch and exploiting gaps in the
schedule. The Horizontal Splitting fine tuning technique also investigates one machine type, but instead of using the combined capacity of the SPL-machines, it uses the combined capacity of several gaps on the SPL-machine. Having only one SPL-machine to consider, solving the problem should be easier.

The Horizontal Splitting problem is a special case of the Vertical Splitting problem when we view the $M$ gaps ($M = \left| G \right|$) on the SPL-machine as $M$ gaps distributed over $M$ different SPL-machines. Note that $G$ only contains gaps that are large enough, thus gaps that are not large enough ($g_j < S_{min} \cdot p_{spl}$) are removed from the set $G$ of ‘selectable’ gaps. Further, the gaps are numbered in sequence of their start times. Because each remaining gap will contain at least $S_{min}$ parts when selected, we fill each gap with $S_{min}$ parts on beforehand by adjusting the length and start time of each gap. This means that the number of parts to distribute over the $m$ selected gaps in the $m$-gap combination decreases to $S_{rem} = S - m \cdot S_{min}$. With the filling each gap with $S_{min}$ parts as a starting-point, $s_i$ denotes the number of parts in the sub-batch along with the $S_{min}$ parts already present.

Again, fixing the number of sub-batches $m$ and using algorithm 5-2, enables us to solve the $MLT_m$-problem and we obtain a solution for which the parts distribution is determined by algorithm 5-1, if $m \leq \left| G \right|$. By comparing the solutions to the various $MLT_k$-problems ($k = 2, \ldots, \min \left( \left| G \right|, \left\lfloor \frac{S}{S_{min}} \right\rfloor \right)$), we can select the best solution, that is the solution with enough forward shift and the lowest number of sub-batches used.
Using the Fine Tuning Techniques

In the previous chapters we have discussed various fine tuning techniques. These fine tuning techniques have been implemented in two Expert Management Support Systems that have been developed as part of the ESPRIT Project 2415. For both Imperial College and Delft University of Technology, this project is part of the route towards human-centred advanced manufacturing technology (AMT).

The human production planner for instance faces a rapidly evolving technology. To support the human planner, several support systems have been and are being developed. Often these systems lack the possibilities to adapt to changes. Verbraeck (1991) however, shows that it is possible to design and to build scheduling support systems that incorporate knowledge of the planner in the system, that adapt to changes in the environment and that effectively support the human planner.

Imperial College has focused on human-centred AMT in various projects. Research products include a design framework for human-centred AMT
and the design and implementation of a flexible human-centred turning cell (Slatter, 1990).

In this chapter we will view some implementation aspects of the EMS as developed to support the OMS, which in turn supports the human operator with his or her tasks.

6.1 Implementation concept

At the start of the project, the EMSs were placed at area level and at the cell level. Each EMS, supporting either a PMS or an OMS, is supposed to be able to work as an individual system, that is without any knowledge from other EMSs within the same level or at other levels within the factory. Hence such an EMS merely solves local problems. Nevertheless, communication between EMS and the system it is supporting is very crucial, and so is data storage and retrieval. Further, it is envisaged that, eventually, EMSs will be communicating with each other to solve a global problem. Adding other considerations as described later, evidently effort has to be put into selecting the proper means to establish communication and data storage and retrieval (DSR).

6.1.1 Communication and DSR

The product by the ESPRIT Project 2415 is an Integrated System consisting of one PMS and several OMSs supported by various EMSs. Certainly during development and probably also for the final product, when commercially available, the Integrated System will be running on several linked hardware platforms of different types or manufacturers. Further, it is envisaged that the Integrated System or the separate systems also have to support other hardware platforms than used by the ESPRIT Project consortium.

The nature of these connected systems points at the common data to be used and at the inter-system communication. This puts some constraints on the development of the various systems as the developer has to comply with the functioning of the other systems to assure

- data compatibility,
- data interchangeability and
- data integrity
which also implies that the data definition must be upheld by each system and the communication between the various systems must be established and maintained.

The ability to implement such an Integrated System within different environments is a major must and advantage to be able to market the product. Therefore, the software does not only have to be able to be run on several different hardware platforms, but the communication among these various platforms must also be ensured in case the Integrated System itself uses several hardware platforms.

6.1.2 Implementing communication and DSR

Besides linking systems physically, the communication between the systems has to be established. To promote the ESPRIT Projects, the consortium favoured using the results of another ESPRIT Project regarding the definition and implementation of communication protocols. This implies that effort has to be put into the implementation of these communication protocols, along with the work on the main subject of the project. Advantages of using the results of the other ESPRIT Project and implementing communication protocols ourselves are:

- display of the quintessence and importance of ESPRIT Projects and their results
- total control over the communication implementation

Due to the fast developments of hardware and the corresponding operating systems and other software, changes concerning the hardware platforms used are likely to occur. This, among others, is disadvantageous since it causes additional work to upgrade the communication protocols’ implementation. Some of the disadvantages are listed below:

- extensive effort required for the implementation of communication protocols
- additional effort required for upgrading upon hardware or software changes
- availability of the finished product, other hardware platforms and operating systems also need communication protocol implementation

Viewing the length of the project and the available effort, the implementation of communication protocols uses a disproportionate amount of available effort relative to the development of the main software itself.
Especially because communication largely depends on the hardware and operating systems used, changes in operating systems and upgrading of hardware cause even more effort to be spent on this issue.

Observing the various systems to be built, one recognises the need for storing, retrieving and communicating data. So not only the communication, but also data storage and retrieval facilities have to be developed.

Although the intention of the consortium is amiable, the additional effort required and the possible problems commercialising the Integrated System do not justify the usage of other ESPRIT Projects’ results. After a closer examination of the various systems, the data definition, the data storage and retrieval, and the corresponding communication issues, we did not pursue the ‘domestic’ implementation of the communication protocols and so on. Our effort focused on the various systems that constitute the Integrated System, without bothering about how to implement and how to maintain the communication protocols that form the back-bone of the Integrated System. For this, we presented a more elaborate and applicable solution, which was accepted by the consortium.

6.1.3 Purchasing communication and DSR
Knowing the comments made in the previous section, some requirements regarding the solution to be found can be defined. The solution should

- deal with communication between various hardware platforms,
- involve no implementation on communication protocols by the consortium and
- offer sophisticated data storage and retrieval handling.

Relational Database Management Systems
Data storage and retrieval has been a research topic for a long time. Implementations using database management systems (DBMSs), either hierarchical, network or relational (Date, 1990), are widely spread and the development of such systems is still ongoing. The commercially available relational DBMSs (RDBMSs), such as Sybase, Ingress, Informix and Oracle, are available on a wide variety of hardware platforms and they are also available as distributed versions. This implies that problems of communication between the same or different hardware platforms have been solved by the manufacturers of the RDBMSs.

With this, the continuity of the Integrated System has been reasonably secured. Changes in hardware platforms and operating systems will be
studied by such commercial organisations and, if eligible, the RDBMSs will be updated accordingly.

Additionally, the usage of a commercially available RDBMS will enhance the possibilities of integrating the Integrated System with existing software at the user's site. For instance, the Integrated System may be linked to MRP II systems or tool management systems that use the same RDBMS, or even another manufacturer's RDBMS. In addition, new applications may also use this commercially available RDBMS, which may lengthen the life cycle of the product.

**Portability and maintenance**

Since commercially available RDBMSs allow for the communication between various hardware platforms, it is possible to offer the Integrated System, or the separate systems, on this wide range of hardware platforms.

Porting software from one computer to another depends on the availability of the proper compilers to transform the source code into object code, before linking it to the system specific libraries, creating an executable program on that machine.

For example, the OMS has been coded using Modula-2 and some additional procedures have been written using C. Porting the OMS to another machine only requires a Modula-2 and a C compiler to be available. Obviously, these compilers have to offer the same functionality as the compilers on the original machine. Fortunately, the choice of programming languages is such that no real problems will occur. Modula-2 has been designed to be modular so operating specific procedures can be isolated in one module, making it easy to adjust. C causes even less trouble since C is more or less a standard, especially when using a Unix operating system.

The EMS consists of programs written in Delphi3 and C. Delphi3 code is independent of the hardware and can be transferred even without recompilation. Obviously, this is too good to be true, so here is the catch. The Delphi3 compiler, linker, maker, interpreter and debugger are all written in Modula-2. This presents us with the same conditions as for OMS, a Modula-2 compiler and a C compiler have to be available.

Having the proper compilers available only guarantees the easy transfer of software from one machine onto another. However, we are also using some RDBMS which puts some additional constraints on the implemen-
tation. Working with such an RDBMS, we can manipulate the data base contents by using SQL (Structured Query Language).

To perform data storage and retrieval actions from within the Modula-2 or C programs, a lot of complex coding has to be incorporated in the programs to establish a link with the RDBMS. To make programming easier, manufacturers offer the possibility of using pre-compilers. This enables the programmer to embed the SQL-statements for data storage and retrieval within the normal program. The pre-compiler translates the SQL-statements into the complex coding required for the link with the RDBMS.

Figure 6-1 shows the code for counting the number of available machine using the data stored in the database. Notice that the SQL-statements are approximately the same as when entering these statements interactively. Another property of embedding SQL-statements is the readability of the code and hence the ease to maintain the program. This is stressed by figure 6-2 that shows only a minor part of the same program after the pre-compiler has replaced these SQL-statements by the appropriate code. Clearly, this is less readable due to the numerous complicated instructions and less maintainable.

A further advantage is the independence of changes in the version that is being used. Each version may have its own peculiarities, however using the embedded SQL-statements and consecutively pre-compiling them will almost prevent any changes to be made. If there are any, they can best be made in the coding that is the easiest to read or maintain by the programmer.

Another advantage of this is the possibility to easily change over to another manufacturer's RDBMS since most of the embedded SQL-statements are standardised. Manufacturer specific elements can also easily be changed.

Until now we have discussed the embedded SQL-statements using C as an example. For Modula-2 it is different because Modula-2 pre-compilers are less available, which depends on the manufacturer. However, the needed data storage and retrieval statements in the Modula-2 coding can be accomplished by writing the statements in C procedures and subsequently pre-compiling, compiling and linking them to the Modula-2 object code.
/* Simple C-program with embedded SQL-statements */
/* Retrieves the number of machines available */

#include <stdio.h>

EXEC SQL BEGIN DECLARE SECTION;
  VARCHAR user[50];
  int numberOfMachines;
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;

main()
{
  strcpy(user.arr, "ep2415/tud");
  user.len = strlen(user.arr);

  EXEC SQL
    WHENEVER SQLERROR GOTO sqlerror;

  EXEC SQL
    CONNECT :user;

  EXEC SQL
    SELECT COUNT(*)
    INTO :numberOfMachines
    FROM MACHINE;

  EXEC SQL
    COMMIT RELEASE;

  printf("There are %d machines defined. \n", numberOfMachines);
  exit(0);

  sqlerror:
    printf("\nExample: (main)\n% .70s \n", sqlca.sqlerrm.sqlerrmc);
}

**Figure 6-1** sample C-code with SQL-statements embedded

Concluding, the advantages of using a well-known RDBMS, suited for distributed implementation, within the project include:

- wide variety of hardware platforms supported
- communication between similar and different hardware platforms supported by the RDBMS
- changes in hardware platforms and operating systems will be followed by the RDBMS manufacturers
- communication between different versions of the RDBMS possible
- possibilities to integrate with other available or new software packages
- strong position because of using the commercial RDBMS
- embedded SQL simplifies programming
/* Simple C-program with embedded SQL-statements */
/* Retrieves the number of machines available */

#include <stdio.h>
struct {
    unsigned short len;
    unsigned char arr[50];
    user;
    int numberOfMachines;
} static struct {
    int sq001N;
    unsigned char *sq001V[4];
    unsigned long sq001I[4];
    unsigned short sq001T[4];
    unsigned short *sq001T1[4];
    } sq001 = (4);

main()
{
    strcpy(user.arr, "ep2415/tudm");
    user.len = strlen(user.arr);
    sqlscs(&sqlca);
    sq001.sq001V[0] = (unsigned char *)&user.len;
    sq001.sq001I[0] = (unsigned long)52;
    sq001.sq001T[0] = (unsigned short)9;
    sq001.sq001T1[0] = (unsigned short)10;
    sq001.sq001T2[0] = (unsigned short)10;
    sq001.sq001T3[0] = (unsigned short)10;
    SQLTM[0] = (int)0;
    SQLTM[1] = (int)10;
    sqlca2(
        &sq001.sq001N,sq001.sq001V,sq001.sq001I,sq001.sq001T,
        &sqlami, &SQLTM[0], &SQLTM[1], &sqlvsn);
    if (sqlca.sqlcode < 0) goto sqerror;
    sqlscs(&sqlca);
    if (Isquali[0])
    { /* OPEN SCOPE */
        sq002.sq002T[0] = (unsigned short)10;
        SQLTM[0] = (int)4;
        sqlba2(&sq002.sq002V, sq002.sq002I,
            sq002.sq002T, sq002.sq002T1, sq002.sq002T2, sq002.sq002T3,
            &SQLTM[0], &sqlusi[0]);
    } /* CLOSE SCOPE */
    sqlach(&sqlusi[0]);
    sqlacct(&sqlcum[0]);
    sqltf(&SQLTM[0], &SQLBT0);
    sqlach(&sqlusi[0]);
    SQLTM[0] = (int)11;
    sqlcom(&SQLTM[0]);
    if (sqlca.sqlcode < 0) goto sqerror;
    printf("There are %d machines defined. \n",
        numberOfMachines);
    exit(0);
    sqerror:
    printf("\nExample: (main) .70s \n",
        sqlca.sqlerrm,sqlerrmc);
    }

figure 6-2 sample C-code of figure 6-1 after pre-compilation. Code has been omitted to fit one page instead of more than three
• using embedded SQL-statements causes changeovers to other manufacturer’s RDBMSs easier

6.1.4 Integrated System Architecture

Although we have developed software on SUN-380 workstations with the Unix operating system, the EMS has been transferred to the computer as they are in use by our partners in the consortium. Before transferring the EMS software, the Delfi3 compiler, linker, maker, interpreter and debugger had to be ported to the various computers, a PS/2 with OS/2 for OMS and an HP9000 with HP-UX for PMS. After the successful porting of these systems, we were able to transfer the EMS software.

As the RDBMS to be used by the consortium the Oracle RDBMS has been selected. The choices made on the usage of a commercial RDBMS have proved to be worthwhile, since the changes that had to be made to the EMS software were minor concerning the OMS (PS/2) and most of them where due to the version 5.1 used on the PS/2 instead of version 6 as on the SUN-380. Further developments on the EMS were performed on a 80386DX computer.

![Diagram](image)

*Figure 6-3  linking of a PMS on an HP9000 and two OMSs on PS/2s*
Although after a while we could make use of an HP9000, we did not have an Oracle RDBMS, so also the developments on the EMS for PMS were performed on a 80386DX computer. This did not pose a problem since of each of the C-programs compiled without errors on the HP9000.

The various sessions of porting the EMS were a fortification of the advantages as stated in the previous section and promise future changes of hardware platforms to be without too many problems.

In the ESPRIT Project 2415, an HP9000 and a PS/2 were linked using Ethernet displaying the Integrated System accompanied by a stand-alone PS/2. Figure 6-3 shows two PS/2 computers, each running OMS with EMS, linked to one HP9000 running PMS with EMS.

6.1.5 EMS: an intelligent control mechanism

As discussed in the previous chapters, for solutions concerning fine tuning techniques such as Overlapping Manufacture and Vertical Splitting, the search space can be decreased by using problem specific knowledge. The knowledge-based system EMS that has been developed for the ESPRIT Project 2415 is not 'just' an expert system, but it is an intelligent control mechanism that reacts on requests by invoking the proper programs to calculate specific data. The intelligent control mechanism incorporates the problem specific knowledge.

6.2 Implementing EMS for OMS

For both the PMS and the OMS, an EMS has been developed. In this section we only view the EMS that supports the OMS, hence we only discuss schedules that deal with the work allocation in a cell and we describe the process of improving some schedule using fine tuning techniques.

Assisted by the OMS, the operator or human scheduler that allocates resources for processing the jobs' operations can request assistance if one or more jobs will finish too late according to some schedule. This schedule can be the current schedule or some candidate schedule. Requesting assistance for improving the schedule, the operator invokes the knowledge-based system EMS, specifying which jobs have to be investigated and which of the fine tuning techniques might apply. For example, if a cell contains machines of different types only and no machine exists that can be used as an alternative for another machine, only the options Overlapping Manufacture and Horizontal Splitting may apply. This pre-selection of possible
options can be made by the operator. Figure 6-4 shows the link between the EMS and the OMS through the RDBMS.

EMS consists of various modules coded using the logic programming environment Delfi3 (Goedhart, 1991b) and additional external programs written in C. Some supporting modules excluded, the contents of the modules that comprise the EMS is listed in Appendix A: EMS implementation. The numerous external programs are not listed in the appendix. Also no implementation of the Horizontal Splitting fine tuning technique is shown, since Horizontal Splitting was not possible in the testing environment.

![Diagram showing OMS and EMS connection with advice and replies]

**Figure 6-4** link between OMS and EMS and an indication of the knowledge modules of EMS

### 6.3 The control module EMS-advisor

The typical working of the EMS is to retrieve the request as entered by OMS, identifying the problem jobs, investigating them one by one and consecutively compiling a reply containing the suggestions or the advice.

The control module of EMS retrieves the requests from the database, using initial information as supplied by the calling system. This initial information consists of the name of the OMS which is calling the EMS (further referred to as the caller), the number of the request and the type of operation sequencing that has been performed.

As OMS, EMS operates on data that can be found in the database. During execution, EMS uses the database. The various deductions made by the EMS largely depend on the contents of this database. Some intermediate results derived from data found in the database are stored in that data-
base to prevent the same calculations to appear in a later stage of the consultation.

6.3.1 The request

Considering unique cell names, it is obvious to adopt the cell name as the name of the OMS that handles this cell. Through this unique name the proper data can be retrieved from the database. For some OMS to keep track of the various requests that have been made, each request by this OMS can be identified by a unique number.

Important for the evaluation of the applicability and influence of the various fine tuning techniques, the EMS has to know the type of sequencing operations. For instance, will the next machine be set up either after finishing the previous operation on the part or after tearing down the machine used for the previous operation. When referring to the previous machine or operation we use the prefix PRV, for the machine or operation on hand we use the prefix NXT. The types of sequencing, referring to the previous machine or operation using the prefix PRV and the prefix NXT for the next machine or operation, are:

- the setting up of the NXT-machine is after tearing down the PRV-machine,
- the setting up of the NXT-machine is after finishing the PRV-operation on the last part of the batch,
- the start of the NXT-operation on the first part of the batch is after the tearing down of the PRV-machine and
- the start of the NXT-operation on the first part of the batch is after finishing the PRV-operation on the last part of the batch.

Through the OMS, the operator can specify the problem jobs that have to be investigated by the EMS. The names of the jobs are then passed with the request. The operator can also opt for the possibility to have the EMS check all problem jobs by passing the name ‘ALL’ as the name for the problem job. Clearly, this excludes the name ‘ALL’ as a possible name for a job.

Additionally, the request contains information on the schedule which is the basis of the investigation. Either the name (‘CURRENT’) of the current schedule or the name of some candidate schedule is passed to the EMS. The candidate schedules are possible schedules as generated by the OMS-scheduler according to some priority rules and heuristics. All information on a particular schedule can be found in the SCHEDULE table.
On beforehand it may be known to either OMS or the operator that some fine tuning techniques are not valid for a specific job. To accommodate this, the request also includes an indication of which options should be investigated.

6.3.2 The main body

If the operator indicates that the EMS has to check for problem jobs, the EMS selects all jobs in the specified schedule that violate their due dates. Otherwise, EMS selects all indicated jobs. After the identification of the problem jobs, the EMS investigates one job at a time. To be able to differentiate the impact of the various options, EMS calculates the amount of due date violation.

The amount of due date violation is the number of manufacturing minutes from the end of the day indicated by the due date until the end of the job's last operation in the cell managed by the OMS. For manufacturing minutes we use the time that can actually be used for manufacturing. Holidays and shift breaks are excluded, but overtime, if any, is included. Section 6.3.4 discusses how the difference between two date and time combinations is calculated, taking holidays and shift breaks into account.

Depending on the indication of the fine tuning techniques to investigate, control is transferred to the modules containing the knowledge to investigate the indicated fine tuning technique. When all indicated fine tuning techniques have been investigated, the EMS compiles the advice or reply for the caller. This reply is stored in the database before the EMS stops executing.

6.3.3 The reply

After the EMS has finished investigating the problem jobs, control is returned to the OMS and the operator can retrieve the advice. The data that comprises the advice, has been stored by the EMS in the REPLY table and will be interpreted by the OMS.

The reply in the REPLY table contains

- the number of the reply
- each reply concerning one problem job is numbered
- the number of the request as defined by the caller
- the ID of the caller
- the ID of the job that has been investigated
a sequence number if several options may be advised
• the type of fine tuning technique that is advised
• No Solution, Overlapping Manufacture, Horizontal Splitting, Vertical Splitting
  or Alternative Manufacture
• the ID of the operation involved
• the number of sub-batches after overlapping or splitting
• the distribution of the parts over the sub-batches
• the IDs of the machines involved in the Vertical Splitting fine tuning technique or in the Alternative Manufacture fine tuning technique
• the start date for the NXT-operation when the Overlapping Manufacture fine tuning technique is advised, that is on which day the first sub-batch may be operated on
• the start time for the NXT-operation when the Overlapping Manufacture fine tuning technique is advised
• a number indicating the value of the advised fine tuning technique, this can be the forward shift of the end time for the job

6.3.4 Calculating length of time periods

Typically a period is bounded by two \langle date, time\rangle-tuples: \langle startdate, starttime\rangle and \langle enddate, endtime\rangle. The number of manufacturing minutes between the start-tuple and end-tuple can be calculated using the information stored in the database. Information on work-shift lengths, breaks, overtime and working days can be retrieved from the database using SQL interactively or embedded in a programming language. Since for the start day and the end day only part of those days can be included in the calculation and all working days in between have to be included, the calculation consists of three steps.

The first step concerns the calculation of the remaining manufacturing time from starttime until the end of the day of the period’s first day. Besides the time between starttime and the end of the shift, we include the time spent on overtime and we exclude the duration or part of the duration of shift breaks during the period of starttime to the shift end. Hence, for these calculations we need to retrieve the start and finish times for the shift and the start times and duration of the possible overtime and shift breaks (see figure 6-5 (A)).
A Retrieval of data regarding the start of the period

```sql
SELECT STIME, FTIME, WORK_STIME, WORK_DUR,
       BREAK1_STIME, BREAK1_DUR, BREAK2_STIME, BREAK2_DUR,
       BREAK3_STIME, BREAK3_DUR, BREAK4_STIME, BREAK4_DUR,
FROM CALENDAR, SHIFTYPE, SHIFTALL
WHERE SHIFT_TYPE = SH_TYPE
AND CELL_ID = cell-id
AND MACH_ID = mach-id
AND FAC_DAY = 'Y'
AND TRUNC(C_DATE) = TO_DATE(startdate)
AND TRUNC(SDATE) <= TRUNC(C_DATE)
AND TRUNC(FDATE) >= TRUNC(C_DATE);
```

B Retrieval of data regarding the end of the period

```sql
SELECT STIME, FTIME, WORK_STIME, WORK_DUR,
       BREAK1_STIME, BREAK1_DUR, BREAK2_STIME, BREAK2_DUR,
       BREAK3_STIME, BREAK3_DUR, BREAK4_STIME, BREAK4_DUR,
FROM CALENDAR, SHIFTYPE, SHIFTALL
WHERE SHIFT_TYPE = SH_TYPE
AND CELL_ID = cell-id
AND MACH_ID = mach-id
AND FAC_DAY = 'Y'
AND TRUNC(C_DATE) = TO_DATE(enddate)
AND TRUNC(SDATE) <= TRUNC(C_DATE)
AND TRUNC(FDATE) >= TRUNC(C_DATE);
```

C Retrieval of the total manufacturing time in the period, excluding the start of the period and the end of the period

```sql
SELECT SUM(FTIME-STIME+WORK_DUR-BREAK1_DUR-BREAK2_DUR-
                 BREAK3_DUR-BREAK4_DUR)
FROM CALENDAR, SHIFTYPE, SHIFTALL
WHERE SHIFT_TYPE = SH_TYPE
AND CELL_ID = cell-id
AND MACH_ID = mach-id
AND FAC_DAY = 'Y'
AND TRUNC(C_DATE) > TO_DATE(startdate)
AND TRUNC(C_DATE) < TO_DATE(enddate)
AND TRUNC(SDATE) <= TRUNC(C_DATE)
AND TRUNC(FDATE) >= TRUNC(C_DATE);
```

**Figure 6-5** SQL-queries to retrieve the proper manufacturing time information from the database

Similarly for the period’s last day, we calculate the manufacturing time that is possible from the beginning of the day until `enddate`, the finish of the period. Again, we need to retrieve the start and finish times for the shift and the start times and duration of the possible overtime and shift breaks (see figure 6-5 (B)).
The third step concerns the calculation of the total minutes in between the start day and the finish day. Figure 6-5 (C) shows how to calculate the number of minutes for the period between startdate and enddate.

After calculating the total minutes for each step, we add them to obtain the length of the period. However, the possibility exists that both the start and the end of the period are on the same day. If this is true, the length of the shift for that day has to be subtracted from the total.

6.4 The module Overlapping Manufacture

The result of applying the Overlapping Manufacture fine tuning technique depends on the ability to forward shift the NXT-operation that succeeds the OVL-operation. Evidently, directly preceding the NXT-operation no other operation should be scheduled on the same machine to allow the forward shift to occur.

The investigations by the Overlapping Manufacture fine tuning technique concern

- the calculation of the free time preceding each operation of the problem job,
- the determination of the operation to be overlapped, the number of sub-batches after overlapping and the distribution of parts over the sub-batches that offers the best forward shift, and
- the calculation of the start date and start time for the operating on the job on the next machine, after the first sub-batch has arrived.

6.4.1 Calculating the preceding free time

For each operation of the job, which is currently being investigated, the length of the idle machine period directly preceding the operation on that machine, has to be calculated. The process of forward shifting an operation depends on the idle machine period belonging to this operation.

The calculation of the idle machine period involves either the period the machine is idle between the problem job operation and the previous operation on the same machine or, if the problem job operation is the first one, the period between the problem job operation and the start of the previous operation of the same problem job. By definition, the preceding free time for the first operation is 0.
After calculating the preceding free time for each operation of the problem job, we can make a first selection of operations that may be involved when investigating the Overlapping Manufacture fine tuning technique. Just by viewing the list of operations and their corresponding free times, a first selection of which operations might be involved in the Overlapping Manufacture fine tuning technique can be made. Apparently an operation can only be overlapped, i.e. be an OVL-operation, if all succeeding operations have a non-zero preceding free time. Otherwise, the complete job can not be forward shifted, because one of the succeeding operations can not be forward shifted.

This leads to a simple deduction of which operations are eligible. Starting with the last operation (of the problem job), if the preceding free time is larger than 0, the preceding operation is marked as a possible OVL-operation. As long as one can mark such a preceding operation, this operation is then investigated and treated in the same way. The marking of operations stops as soon as an operation can not be marked, or equivalently, as soon as a zero preceding free time has been found.

Table 6-1 shows an example of operations with corresponding preceding free times, local forward shift and actual forward shift. The local forward shift is the forward shift as a result of overlapping without the constraining preceding free time of the succeeding operations. The actual forward shift incorporates this preceding free time constraint.

<table>
<thead>
<tr>
<th>OVL-operation</th>
<th>NXT-operation</th>
<th>possible OVL-operation</th>
<th>preceding free time</th>
<th>local forward shift</th>
<th>actual forward shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>-</td>
<td>102</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>♦</td>
<td>100</td>
<td>90</td>
<td>75</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>♦</td>
<td>75</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>♦</td>
<td>96</td>
<td>78</td>
<td>78</td>
</tr>
</tbody>
</table>

Table 6-1 operations with preceding free time, marking (♦) as possible OVL-operation, local and actual forward shift

The total amount of preceding free time per marked operation is not an indication to which operation will be selected, since this is not a degree of forward shift! The total amount of possible forward shift offers us a better
indication, as the forward shift for the last operation always places an upper bound.

In the example in table 6-1 the forward shift for operation 6, operation 5 is the OVL-operation, informs us that the forward shift for the complete job can be at most 78 minutes. If one selects operation 3 or 4 to be the OVL-operation, the maximum forward shift for the job decreases to 75 minutes. Although operation 5 has the largest preceding free time, selecting operation 4 as the OVL-operation and operation 5 as the NXT-operation, does not offer the best result concerning the possible forward shift. Because operation 3 has no preceding free time, overlapping operation 2 will not lead to any improvement and therefore operation 2 is not marked as a candidate OVL-operation.

After marking operations as candidate OVL-operations, one of these candidates has to be selected.

6.4.2 Determining the best overlap

To determine the best overlap we first retrieve, among others, the batch size \( S \) and \( S_{\text{min}} \), the minimum batch size after overlapping. Consecutively we determine the maximum number of sub-batches \( m \) that could be formed. Evidently, \( m \) has to be larger than 1 or else we can not apply the Overlapping Manufacture fine tuning technique.

Although the maximum number of sub-batches can become very large, even equal to the batch size, this could be undesirable. Therefore the operator's or OMS' preference for the maximum number of sub-batches should be used to artificially and temporarily reset the minimum batch size and the maximum number of sub-batches if this preference is violated.

The task now is to investigate each candidate OVL-operation to determine the forward shift as a result of a particular parts distribution over a number of sub-batches. The results for the various candidate OVL-operations will be compared to offer the best solution as a suggestion to the caller.

In chapter 4 we discussed the problem of finding the minimum lead time for both the OVL-operation and the corresponding NXT-operation. By identifying three different situations concerning the operation times of the OVL-operation and the NXT-operation, we have reduced the search space. This strategy has been implemented.
**OVI-operation time equals NXT-operation time**

The distribution of parts over the $m$ sub-batches is such that each sub-batch contains either $S_{\text{min}}$ or $S_{\text{min}} + 1$ parts and each subsequent sub-batch contains at least as many parts as the previous sub-batch. This means that after filling each sub-batch with $S_{\text{min}}$ parts, the remaining $n$ parts are distributed over the last $n$ sub-batches one by one.

Distributing the parts in this way, when $m$ sub-batches have to be filled, offers us the best forward shift with the NAIT (not allocatable idle time) equal to 0. Obviously, we regret it if our solution does not enable the job to finish in time, but this is still the best result that can be achieved.

We face another situation if the forward shift obtained causes the job to be in time. Obviously, we are satisfied with this solution and we can not improve the forward shift further, so how can we improve the solution. First, the current forward shift causes the job to be no problem anymore, hence any step we take may decrease the forward shift but should not cause the job to become a problem again. The further improvement of the solution can be found in the number of sub-batches needed to obtain enough forward shift. This number can be decreased, given the constraint that the forward shift has to remain large enough for the job not being a problem job. Clearly, for the solution the number of sub-batches can only decrease if there are more than two sub-batches to begin with.

Having found the best solution after distributing the parts over the sub-batches, we can investigate the solution when allowing fewer sub-batches. This can be accomplished by decreasing $m$ by 1 to $m'$ and by temporarily setting the $S_{\text{min}}$ equal to $\lfloor \frac{S}{m'} \rfloor$. Again the best distribution can be derived and the forward shift can be calculated. As long as the new solution also causes the job to be in time, this process can be continued providing that the number of sub-batches is larger than 2. If the new solution does not lead to enough forward shift, this solution has to be disregarded.

**OVI-operation time less than NXT-operation time**

Contrary to the 'simple' distribution when $p_{\text{ovi}}$ and $p_{\text{nxt}}$ are equal, deriving the distribution is somewhat more complicated. Because $p_{\text{ovi}}$ is less than $p_{\text{nxt}}$, keeping the batch size for each sub-batch the same causes sub-batches to pile up at the NXT-machine. One can avoid this by increasing the size of some sub-batch $i$ by $d_i$ such that $s_i = S_{\text{min}} + d_i$ and $s_1 = S_{\text{min}}$ (see chapter 4 "Fine Tuning Schedules Through Overlapping Manufacture"). Formula (4.5) shows the increment $d_i$ for each sub-batch such that not
more than one sub-batch is waiting to be processed by the NXT-machine. The formula only shows the possible increment $d_i$ for sub-batch $i$, based on the minimum batch size after overlapping $S_{\text{min}}$ the processing times for the OVL-operation and the NXT-operation and the increments for the preceding sub-batches.

Figure 6-6 shows the definition relation (DREL) PossibleDistribution which is the implementation of formula (4.5), using the built-in back-tracking mechanism of Delfi3. The DREL PossibleDistribution determines whether a parts distribution can be found, in which case it is returned, or not.

```
DREL PossibleDistribution
DOMAIN pOVL : <INT>
  | pNXT : <INT>
  | Smin : <INT>
  | Size : <INT>
  | surplus : <INT>
  | subbatch : <INT>
RANGE partsdistribution : <LIST <INT>>
VAR di : <INT>
  | si : <INT>
[ Size <= 0 AND ! OR
di = (((subbatch-1) * Smin+surplus) * (pNXT-pOVL)) DIV pOVL AND
si = Smin+di AND
  | Size-si <= 0 AND ! AND
`si = Size
  OR
Size-si >= Smin AND !
  OR
`si = Size-Smin AND
  | si >= Smin AND !
  OR
subbatch = 2 AND
si = 1 AND ! AND
`si = Size
  OR
FALSE
]
] AND PossibleDistribution(pOVL,pNXT,Smin,Size-si,surplus+di,
  subbatch+1,`partsdistribution) AND
`partsdistribution = LISTER(si,partsdistribution)
]
ERELE
```

*figure 6-6*  Delfi3 code to determine a possible parts distribution
Contrary to the formula, the implementation regards the batch size. If, after assigning a value to \( s_i \), the remaining number of parts is larger or equal to \( S_{\min} \), \( s_i \) remains unchanged, otherwise \( s_i \) will be decreased until the remaining number of parts is equal to \( S_{\min} \). If this causes \( s_i \) to become less than \( S_{\min} \), \text{PossibleDistribution} will evaluate to \text{FALSE}, with one exception: if \( s_2 \) has been decreased to 1, \( s_2 \) will be set to \( S_{\min} + 1 \). This is to assure that a distribution can be found.

The DREL \text{CalculateOverlapLess}, which is using the DREL \text{PossibleDistribution}, invokes \text{PossibleDistribution} to obtain a distribution with a specific value for \( S_{\min} \). If \text{PossibleDistribution} does not succeed in generating such a distribution, \text{CalculateOverlapLess} temporarily increases \( S_{\min} \) by 1 and tries to obtain a distribution of parts by invoking \text{PossibleDistribution} with this new value for \( S_{\min} \).

This process will be repeated until \text{PossibleDistribution} returns a distribution. The increasing of \( S_{\min} \) continues until \( S_{\min} = \lceil \frac{S}{2} \rceil \), and \text{PossibleDistribution} will be able to generate a distribution, using the exception if \( S \) is odd. The DREL \text{PossibleDistribution} will be called with the parameters \( p_{\text{OVL}} \), \( p_{\text{NXT}} \), \( S_{\min} \) and \( S \). Further, the total of \( d_i \) so far (surplus) and the number of the sub-batch are supplied:

\[
\text{PossibleDistribution}(p_{\text{OVL}}, p_{\text{NXT}}, S_{\min}, \text{Size}, 0.1; \text{partsdistribution}).
\]

The parts distribution will be returned and the DREL evaluates to \text{TRUE}.

Once the distribution has been established, the forward shift and the NAIT are computed. If the forward shift is enough to get the job done in time, we can try to improve the solution by decreasing the number of sub-batches as described in the previous section.

**OVL-operation time larger than NXT-operation time**

For \( p_{\text{OVL}} \) being larger than \( p_{\text{NXT}} \) we find the NXT-machine to run out of parts to handle if all sub-batches are of the same size. Without affecting the lead time, one can try to avoid this by having each next sub-batch either the same size or smaller than its predecessor. However, the last sub-batch should be equal to \( S_{\min} \). Formula (4.6) shows the decrement \( d_i \) for each sub-batch such that the NAIT on the NXT-machine is minimised.

The implementation of formula (4.5) can be used to calculate the sub-batch sizes for the occasion where \( p_{\text{OVL}} \) is larger than \( p_{\text{NXT}} \). In fact, formula (4.6) calculates the increments for the sub-batches, starting with the last sub-batch going to the first. The procedure followed is the same, so the DREL
PossibleDistribution can be used by interchanging the parameters $p_{cvl}$ and $p_{nxr}$ in the DREL-call, whilst the resulting parts distribution has to be inverted (e.g. $5,8,13 \rightarrow 13,8,5$). Again, we can try to improve the solution by decreasing the number of sub-batches if enough forward shift has been achieved.

6.4.3 Calculating the start date and start time

If Overlapping Manufacture is possible, the start of the NXT-operation will be earlier than scheduled. Although offering the amount of forward shift should be enough to determine the new start date and start time for the NXT-operation, the new start date and start time are calculated by EMS to enhance the support as offered to the OMS (and hence the operator). The external program for calculating the start time and date, is invoked by the external relation (XREL) ComputeStartDateAndTime.

Since not only the selected NXT-operation but also the succeeding operations have some preceding free time, the calculations have to regard the effective preceding free time. That is the minimum of the preceding free time for the NXT-operation up to the last operation.

6.5 The module Vertical Splitting

The result of the Vertical Splitting fine tuning technique depends on the gaps that are present on the machines of a specific type. The gaps are used for performing a portion of a specific operation of the problem job. By sharing the capacity of various machines of the same type, the finish of this operation can be earlier than scheduled. If the succeeding operations are able to start earlier, the finish of the entire problem job will also be earlier.

The investigations by the Vertical Splitting fine tuning technique concern

- the determination of the number of machines per operation,
- the calculation of the free time preceding each operation of the problem job and
- the determination of the operation to be split vertically, the number of sub-batches after splitting and the distribution of parts over the sub-batches that offers the best forward shift.

6.5.1 Number of machines per operation and preceding free time

Vertical Splitting means dividing a batch for some operation into several sub-batches. All available machines of the type that is originally used for this
operation are called SPL*-machines and each sub-batch will be handled by one of these SPL*-machines. If there are \( m \) sub-batches, \( m \) different SPL*-machines will be used.

Before any attempt can be made to apply the Vertical Splitting fine tuning technique, the number of identically type machines that can perform the given operation has to be determined for each operation. If only the original SPL-machine can be used, splitting the batch vertically is impossible, hence at least two SPL*-machines have to be present in the cell to even contemplate Vertical Splitting.

If this constraint is met, the operations succeeding the SPL-operation have to be forward shifted to effectuate any improvement of the lead time of the job. Clearly, the succeeding operations may start earlier if for each of these operations, the corresponding machine is idle before the original start of the operation. The forward shift of the job depends on this idle time or preceding free time as well. How preceding free time can be calculated, is described in section 6.4.1. However, if the preceding free times have already been calculated when investigating another fine tuning technique, the preceding free times are retrieved from the database.

Table 6-2 shows an example of operations with corresponding preceding free times, local and actual forward shift. Contrary to Overlapping Manufacture, Vertical Splitting can be a possibility even if the last operation has no preceding free time.

<table>
<thead>
<tr>
<th>SPL-operation</th>
<th>splitting possible</th>
<th>preceding free time</th>
<th>local forward shift</th>
<th>actual forward shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>65</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>*</td>
<td>0</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 6-2 operations with preceding free time, marking (*) for possible splitting, local and actual forward shift

Because splitting the batch for some operation causes this operation to finish earlier, splitting the last operation's batch causes this operation, and hence the job, to finish earlier despite the lack of preceding free time.
6.5.2 Determining the best vertical split

Since we only benefit from improving the finish time of some operation if the succeeding operations can be forward shifted, we investigate the job's operations starting with the last operation. If some operation has no preceding free time, the preceding operations will not be regarded.

An operation can be skipped if only one machine in the cell is capable of handling this operation, that is the number of SPL*-machines is one. If more than one SPL*-machine is available we first retrieve the available capacity on the SPL*-machines. Here, available capacity is a general term for the existing gaps or idle periods of the SPL*-machines. The available capacity is captured in the gap set G.

The MLT-problem for Vertical Splitting involves finding the largest forward shift after splitting the batch for some operation, whilst the number of sub-batches is as small as possible. Solving the MLT-problem, we vary the number of sub-batches \( m \) from 2 to \( \min \left\{ M, \left\lfloor \frac{S}{S_{\text{min}}} \right\rfloor \right\} \) (\( M \) is the number of SPL*-machines), and for each setting of \( m \) we solve the MLT\(_m\)-problem using algorithm 5-2. The DREL HandleThisSetting (see appendix A, module Gaps) fills the gap set \( H \) from steps 1 and 2 in algorithm 5-2 (DREL FillGapSetH) and calls the DREL FindBestCombination. The DREL FindBestCombination (see figure 6-7) invokes various other DRELs to effectuate step 3, 4 and 5 of the algorithm 5-2.

6.6 The module Alternative Manufacture

In chapter 3 the fine tuning techniques Overlapping Manufacture, Horizontal Splitting, Vertical Splitting and Alternative Manufacture have been generally introduced, explaining some characteristics of their performance. A further discussion on the problematic nature of the fine tuning techniques Overlapping Manufacture, Horizontal Splitting and Vertical Splitting can be found in chapters 4 and 5.
Figure 6-7 implementation of steps 3, 4 and 5 of algorithm 5-2

Viewing the Alternative Manufacture fine tuning technique, we do not have to find the best distribution of parts over sub-batches, since Alternative Manufacture regards the transfer of the complete batch from the original machine to an alternative machine. The main aspects of the Alternative Manufacture fine tuning technique concern the availability of an alternative machine for some job’s operation. In this section the implementation of the Alternative Manufacture fine tuning technique is described.
The investigations by the Alternative Manufacture fine tuning technique concern

- the determination of the number of alternative machines per operation and
- the calculation of the free time preceding each operation of the problem job and

6.6.1 Number of alternative machines and preceding free time

Alternative Manufacture means moving some operation (of some job) in the manufacturing process from one machine to another. This other machine is not of the same type but of an alternative type. One particular assumption when regarding Alternative Manufacture is that any type of machine can have at most one alternative machine type. An ALT-machine is an alternative to some machine if the type of the ALT-machine is the same as the other machine's alternative machine type.

To start the investigations into the Alternative Manufacture fine tuning technique, the number of ALT-machines has to be determined per operation. Further, the preceding free time has to be calculated for each operation. Clearly, the Alternative Manufacture fine tuning technique can only be considered if at least one alternative machine exists for any operation of the problem job. Also we have to be able to forward shift the operations that succeed the ALT-operation, hence the preceding free time has to be larger than zero for all succeeding operations.

Table 6-2 shows for Vertical Splitting an example of operations with corresponding preceding free times, local and actual forward shift. This table is also applicable for Alternative Manufacture.

6.6.2 Determining the best alternative machine choice

Analogous to the Vertical Splitting fine tuning technique, we do not regard operations that precede an operation with no preceding free time. Also if there exists no ALT-machine for the machine used in some operation, this operation will not be investigated since it can not be moved to an ALT-machine.

If at least one ALT-machine exists for some operation, we retrieve all ALT-machines that are available in the cell. We also determine the periods when these ALT-machines are idle or available, that is we identify the
Chapter 6  Using the Fine Tuning Techniques

gaps on the ALT-machines from the end of the previous operation until the end of the ALT-operation.

After retrieving this information we investigate each available ALT-machine. For each ALT-machine we select the gap on that machine which is the earliest in time and which is large enough for handling the complete batch. The finish time for the operation on this ALT-machine is calculated and the forward shift can be concluded. The ALT-machine that offers the largest forward shift will then be selected for this operation.

6.7 EMS for PMS

PMS supports the production planner with assigning orders to cells. Obviously, the capacity of the cells plays an important role. The capacity of a cell is not just an addition of the available capacity on the machines or machine types in the cell. Because, for instance, the sequence of the operations determines when and to what extend the machine types have to be available. Because the PMS acts at the area level, we do not regard the individual machines. For calculating capacity we do regard the machine types within the cells.

When allocating cell capacity for some order, the production planner may find that not enough capacity is available causing the order to be tardy. The production planner can trigger the PMS to search for alternatives. On request by the PMS, EMS tries to find such an alternative by applying fine tuning techniques.

Regarding the function of PMS, the fine tuning techniques as discussed in chapters 4 and 5 have to applied on the orders using the cells as resources. Applying the fine tuning techniques within the cells is a task of the OMSs. Obeying this constraint, we cannot use the Horizontal Splitting fine tuning technique since, by definition, it focuses on idle periods on the original resource. The Overlapping Manufacture fine tuning technique relies on moving sub-batches in sequence from the current resource to the next resource upon finishing them on the current resource. For two machines this can be accomplished, but there are too many factors involved when dealing with two cells to apply Overlapping Manufacture properly.

6.7.1 Cell capacity

To investigate the fine tuning techniques we need a suitable definition of cell capacity. Since several machine types are present in a cell, one can not
speak of the general capacity of a cell. Meaning, the capacity as offered by the machine types may be enough for some order but not for another. For example, if some machine type has no capacity left, the capacity of the cell is 0 for orders that use that particular machine type. Hence we will use the capacity of a cell in relation to the part that has to be manufactured and we express this capacity in the number of parts that can be manufactured within the given period.

6.7.2 Vertical Splitting

Vertical Splitting at the area level refers to splitting orders between cells. Since cells have part families assigned to them, which is the basic idea of the Group Technology or cellular manufacturing philosophy, parts should be manufactured in the cell its family is assigned to. Hence splitting, or any other kind of cell routing flexibility, may go against the rigid notion of cellular manufacturing.

However, in problem situations it may be desirable to move the order to another cell that is not dedicated to manufacture this particular part family. Wemmerlöv (personal communication), who surveyed cellular manufacturing in the U.S. industry (1989), acknowledges that the ability to process parts in another cell is an advantage that should be taken care of if the situation calls for this solution. Exploiting another cell may be necessary to compensate for the loss of flexibility due to dedication of equipment. Clearly, there is a problem if such splitting between cells becomes the norm rather than the exception. This would lead to a loss of identity of the cell.

Knowing the importance and draw-backs of Vertical Splitting, how can Vertical Splitting be realised?

Similar to Vertical Splitting at the cell level, the ‘identical’ cells have to be identified. For the ESPRIT Project, if cell A manufactures part 26051, then cell B is identical to cell A if it contains the machine types needed to manufacture part 26051. This means that cell B may be identical to cell A regarding the manufacturing of part 26051, but may not be identical to cell A concerning the manufacturing of part 26031.

Knowing the identical cells, we determine the capacity of each cell using the method as described in section 6.7.1. If the total capacity is larger or equal to the order size, the order is split into several sub-orders and the cells are assigned.
6.7.3 Alternative Manufacture

Alternative Manufacture at the area level refers to moving orders to other cells. Again, this is an instance of cell routing flexibility, which goes against the rigid notion of cellular manufacturing. Still, if the benefits are high enough we may opt for such a solution.

Investigating the Alternative Manufacture fine tuning technique, we have to identify the alternative cells. Once more, we define alternative cells with respect to some part. Cell B is considered an alternative cell for cell A when producing part 26051, if it can perform all operations on part 26051 using the machine types, or their alternative machine types, as defined for part 26051.

To advise the usage of an alternative cell, we have to determine the capacity of each alternative cell. If the total capacity of an alternative cell is enough to manufacture the complete order, the order can be assigned to this cell. Since by definition identical cells are alternative cells, such an identical cell can also be suggested. Obviously, if such a situation occurs, the order has been originally assigned to the wrong cell.
Test Results EMS

Using product and resource data from Harmonic Drive Antriebstechnik, one of the partners in the ESPRIT Project 2415, a set of data has been constructed for testing the various systems. A sub-set of this data has been used for testing the EMS that supports OMS.

7.1 Data used for testing

OMS supports the operator that has to manage a cell. To enhance the operator support, the EMS has been developed. To test the EMS, a sub-set of the test data has been used. This sub-set contains data on one of the CNC-cells as used in the Integrated System evaluation. The example CNC-cell consists of 6 CNC-machines:

- two INDEX GFG250 (turning) of type TURN1,
- one INDEX GFG800 (turning) of type TURN2,
- one MAHO500 (milling) of type MILL1,
- one MAHO700 (milling) of type MILL2 and
- one FROMAG RAPIDA (grooving) of type GROOVE.
The machines of type TURN2 are alternatives for machines of type TURN1 and machines of type MILL2 are alternatives for machines of type MILL1.

For the tests six parts, and the operations to be performed to produce these parts, have been defined. Table 7-1 shows the six parts and the consecutive operations that have to be performed in the CNC-cell.

<table>
<thead>
<tr>
<th>parts</th>
<th>26011</th>
<th>26021</th>
<th>26031</th>
<th>26041</th>
<th>26051</th>
<th>26061</th>
</tr>
</thead>
<tbody>
<tr>
<td>operations</td>
<td>TURN1</td>
<td>TURN1</td>
<td>TURN1</td>
<td>MILL2</td>
<td>TURN2</td>
<td>TURN1</td>
</tr>
<tr>
<td>MILL1</td>
<td>MILL1</td>
<td>TURN1</td>
<td>MILL2</td>
<td>GROOVE</td>
<td>MILL1</td>
<td></td>
</tr>
<tr>
<td>GROOVE</td>
<td>TURN1</td>
<td>MILL1</td>
<td>GROOVE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*table 7-1* six parts and the operations to be performed to produce them

### 7.2 EMS supporting OMS

When in operation, the OMS supports the operator in managing a cell. The operator can employ various tools within OMS to improve the performance of the cell. For instance, if enough time available, the operator uses the OMS to regenerate a schedule using different dispatching rules (see section 2.2.2) or by invoking heuristics (see section 2.2.3). Using the evaluation reports of the various schedules that have been generated, the operator selects some schedule for implementation.

The OMS has been extended with the EMS to increase the number of possibilities to improve on the schedules. With this extension, for some schedule the operator can investigate some jobs that will finish too late when using this schedule. To invoke the investigation, the operator uses the expert function of the OMS and enters the names of the problem jobs that he has selected. If he wants to, at this stage he can indicate that some fine tuning technique has not to be investigated by the EMS.

The operator then invokes the investigation and the EMS will investigate each job individually using the information found in the schedule. EMS
does not change any data in the schedule and the influence of implementing the advice for some job on the advice generation for another job is not incorporated.

7.3 Parameter settings for testing

Ordinarily, the OMS considers a period of one week for scheduling. Any operation or job that does not fit into the schedule for that particular week will not be scheduled. Obviously, the scheduling session for the succeeding week will regard such a remaining operation or job. For the testing of the EMS, the OMS is used to schedule some set of jobs for the period of one week. This set of jobs is generated at random.

The scheduler only schedules operations that can be started in the scheduling period. This means that for some jobs there may be operations that will not be started within the defined period. To obtain the due date violation, the EMS retrieves the finish date of the last operation scheduled for some job, and compares this with the job’s due date. For the EMS a job becomes a problem job if this finish date is past the due date. The unscheduled operations are not taken into account.

For example in figure 7-1 only job 1 is considered to be tardy. All operations of job 1 are scheduled, with the third operation too late. The third operation of job 3 is past job 3’s due date, but because this operation lies outside the scheduling period, it is not taken into account.
For the tests, we vary the number of jobs that have to be generated: we generate sets that contain 10, 15, 20, 25 or 30 jobs. The number of jobs per run will also be referred to as the job set size. Each job concerns the production of a part, which is selected from the six parts (see table 7-1) at random. The batch size $S$ is also generated at random for each job and lies within a specified interval. The intervals used are 5 - 25, 5 - 25, 20 - 70 and 50 - 70.

The due dates for the jobs are established by taking a multiple of the total work content (the TWK method). The multiplication factor or multiplier $k$ influences the tightness of the due dates. Lower values of $k$ imply tighter due dates, that is the slack between the required processing time and the time available until the due date becomes smaller for lower values of $k$. Because of the schedule period of just one week and our aim to investigate tardy jobs, the multiplier $k$ is set to 3.

For scheduling the generated set of jobs, each machine uses the same dispatching rule. However, for each set of jobs we vary the dispatching rule that has to be used by each machine. The rules used are:

- **SPT** - Shortest Processing Time
- **S/OPN** - Least Slack per Remaining Operation
- **MQT** - Maximum Queuing Time
- **EDD** - Earliest Due Date

Varying the number of jobs in a set, the batch size and the dispatching rule that has to be used, we find 80 different situations to consider. For each situation, we generate 25 different sets of jobs, leading to 2000 test runs.

### 7.4 Experimental results

In this section we describe the results from the 2000 test runs. Throughout this section we refer to the OMS scheduler by simply saying scheduler.

#### 7.4.1 Problem jobs

For each batch size range as defined in the previous sub-section, a sub-set of the available jobs may finish past their due dates and these jobs are problem jobs. For each job set size, we find a number of problem jobs. The total number of problem jobs found, when performing 25 runs for each job set size and for a specific batch size range, is given in figure 7-2. The
number of problem jobs is less for runs that concern jobs with large batch sizes, as can be expected when using the revised definition of a problem job (section 7.3). For example, a job with a large batch size obtains a later due date (TWK method), which may fall into the next scheduling period.

![Graph showing the number of problem jobs found in 25 runs per batch size range for a variation of jobs per run](image)

**Figure 7-2**  the number of problem jobs found in 25 runs per batch size range for a variation of jobs per run

The various graphs in figure 7-2 show that scheduling using the EDD dispatching rule presents us with the lowest number of problem jobs. The SPT dispatching rule mostly performs second best regarding the number of problem jobs, especially for smaller batch sizes (ranges 5-25 and 5-55). This appears to contradict with the survey of dispatching rules by Blackstone, Phillips and Hogg, (1982). They conclude that when having the due date set to a constant beyond the date of the job's receipt, the SPT dispatching rule appears to have the best performance.
Blackstone, Phillips and Hogg also refer to Elvers (1973), who states that due date rules perform very poorly. They do not only generate more tardy jobs than the SPT rule, but also produce more jobs that are very late. For his experiments, Elvers has used five variations of the TWK due date assignment method (setting the due dates as 3, 4, 5, 6 and 7 times the total processing time). When setting the due date at six times the total processing time or less, the SPT dispatching rule performs best.

However describing the scheduler used in the OMS, Hatzikonstantis (1991, 1992) finds, in contrast to Blackstone, Phillips and Hogg, that although SPT is at least performing better than S/OPN in the tight and medium tight due date region, SPT is outperformed by most rules tested, especially for tight due dates ($k \leq 4$). Together with the MDD (modified due date) and SPT$^2$ (two class truncated SPT), the EDD dispatching rule performs best.

### 7.4.2 Problem jobs solved

For each batch size range, figure 7-3 shows the percentage of problem jobs that may be completely solved. A problem job is considered to be completely solved, if the number of minutes the job is tardy is less than or equal to the forward shift gained by applying some fine tuning technique. During the investigation of problem jobs, possible solutions are not implemented in the schedule. Therefore, the solutions for two or more problem jobs may interfere with each other when they are implemented.

Figure 7-3 clearly shows that no problem job is solved for the test runs that concern jobs with batch sizes in the range 50 - 70, presumably due to the time consuming operations and the lack of enough space to shift operations. Also the number of problem jobs to begin with, is very small (see section 7.4.1). Further, the percentage of problem jobs that can be completely solved, is higher for test runs with fewer jobs. This lives up to our expectations since the number of problem jobs is small for the test runs with only 10 or 15 jobs (see figure 7-2), and for these test runs, the schedule will show more idle time that can be used to accommodate the fine tuning techniques.
Figure 7-3 the percentage of problem jobs that may be completely solved in 25 runs per batch size range for a variation of jobs per run.

Another observation we can make, is that the batch size influences the solvability of the problem jobs. Mostly, for test runs involving job sizes from the range 5 - 25, more problem jobs are solved than for the ranges 5 - 55, 20 - 70 and 50 - 70, in that order. The EDD dispatching rule however, causes the ordering to be significantly different for test runs with 10 or 15 jobs: the percentage of problem jobs solved is higher for the ‘wide’ batch size ranges 20 - 70 and 5 - 55.

7.4.3 Problem jobs not completely solved

Figure 7-3 clearly shows a decrease of the number of jobs that may be completely solved as the number of jobs per run increases. Apparently, the increase in the number of jobs per run, causes less machine idle time and henceforth less possibilities for shifting operations. This is corroborated by figure 7-4, that shows the percentage of problem jobs that may be shifted forward but not sufficient to have these jobs finishing in time.

The peaks in the graphs have shifted to the test runs that involve 20 to 25 jobs, and again, the percentages are the highest for experiments concerning the batch size range 5 - 25.
**Figure 7-4**  the percentage of problem jobs not completely solved in 25 runs per batch size range for a variation of jobs per run

**Figure 7-5**  possible application of various fine tuning techniques when using the dispatching rules SPT and SOP/N
Like figure 7-3, figure 7-4 shows no improvement for problem jobs that are part of test runs concerning the batch size range 50 - 70. Considering the revised definition of a problem job, having a scheduling period of one week and that large jobs leave little room for shifting, this result comes up to expectations.

![Figure 7-6: Possible application of various fine tuning techniques when using the dispatching rules MQT and EDD](image)

**Figure 7-6** Possible application of various fine tuning techniques when using the dispatching rules MQT and EDD

### 7.4.4 Application of Fine Tuning Techniques

In the previous sections we have seen the percentage of problem jobs that can be completely and partly solved. In this section we view how often the fine tuning techniques Overlapping Manufacture, Vertical Splitting and Alternative Manufacture have been suggested to solve problem jobs.

Using the dispatching rule S/OPN, we find that the finish time most of the problem jobs can be improved by applying the Alternative Manufacture fine
tuning technique (see figure 7-5). For the remaining problem jobs Vertical Splitting turns out to be most applicable. Investigating the situation for the EDD dispatching rule (see figure 7-6), we find that the Overlapping Manufacture and the Alternative Manufacture fine tuning techniques have been advised almost equally often. For the MQT dispatching rule however, the Overlapping Manufacture fine tuning technique dominates, with most of the remaining problem jobs tackled by the Alternative Manufacture fine tuning technique. The SPT dispatching rule (figure 7-5) shows a mixture of fine tuning techniques being advised.

\[ \text{Figure 7-7 for SPT and S/OPN: the mean tardiness, the mean forward shift and the difference for each batch size range and job set size} \]

7.4.5 Impact on tardiness

In the previous sections we have shown the percentages of problem jobs that could be or could not be completely solved and the distributions of
the fine tuning techniques that have been suggested by EMS. In this section we view the mean tardiness for the problem jobs and examine the influence of the fine tuning techniques by inspecting the mean forward shift.

**Figure 7-8** for MQT and EDD: the mean tardiness, the mean forward shift and the difference for each batch size range and job set size.

In figures 7-7 and 7-8, we find that the best results can be found for small job set sizes. Also the results are better for runs with small batch sizes. When we compare the results in these figures with the results in the figures 7-5 and 7-6, we notice that most peaks in forward shift fall in with a very high percentage of problem jobs being solved using the Alternative Manufacture fine tuning technique.
Since most of these peaks occur for a job set size of 10, obviously we may conclude that some machine, of some alternative machine type, is not used for many jobs and therefore this machine is often idle. Because it is often idle and for a long period, it can be used to apply the Alternative Manufacture fine tuning technique. As the machines, that can be used as alternatives, become more frequently used, the effect of Alternative Manufacture fine tuning technique decreases.

The overall effect of the fine tuning techniques is clear; they help the best for small job set sizes. Further, fine tuning techniques have no effect at all concerning job sets with large batch sizes (range 50-70).

7.4.6 Conclusions

Using the findings in the previous sections, the investigation into the application of fine tuning techniques can be improved. For instance, the Alternative Manufacture fine tuning technique may be tried during each investigation, whereas the Vertical Splitting fine tuning technique offers not real opportunities when dealing with schedules generated using the EDD dispatching rule. The Overlapping Manufacture fine tuning technique however, does not seem to be a good technique for fine tuning schedules that are generated using the S/OPN dispatching rule.

Nevertheless, these conclusions depend on the type of machines used in the cell and the type of parts that have to be manufactured by the cell. For instance, if in some cell no alternative machine exists, the Alternative Manufacture fine tuning technique will never be applied.
Evaluation and Conclusions

The ESPRIT Project 2415 has been a great stimulus for the research presented in this thesis. Throughout the project various opinions have emerged concerning the application of knowledge-based systems to enhance production planning and scheduling. A major issue during this period was the focus on the human planner and the human operator as the centre of the Integrated System.

8.1 Environment

The ideas and their implementation have frequently been discussed with the project reviewers of the European Commission. Although at first some reviewers were reluctant to adopt the idea of splitting orders between cells, we have shown during the project that such splitting may be necessary. Another problem occurred during the actual integration of the
systems, for example due to lack of communication concerning order status information.

Nevertheless integration sessions have proved to be worthwhile, especially concerning the functionality of the individual systems. The EMS originally only reacted on a specific request concerning some problem order or problem job. The EMS investigated the problem and returned an advice to either PMS or OMS, which in turn informed the production planner or the operator.

The reviewers suggested that the production planner or operator could be assisted more by offering the possibility of investigating all problem orders or problem jobs present. This suggestion has been adopted. Along with specifying the problem order or problem job, the production planner or operator can indicate that he wants the EMS to select all orders or jobs that are tardy and to investigate them one by one.

In spite of the human-centred approach by the ESPRIT Project 2415, some specific constraints have been set for the EMS such as

- direct interaction with the operator is not allowed
- interaction with the operator through PMS or OMS has to be prevented
- data entered by PMS and OMS may not be altered

These constraints have been set to have PMS and OMS keeping control over the interaction with the user. These constraints have had a large impact on the implementation of the fine tuning techniques as presented in this thesis.

8.1.1 Direct interaction not allowed

Since direct interaction with the operator is not allowed and interaction through PMS or OMS has to be avoided, the EMS has to generate advice using data from the database and data supplied by PMS or OMS when the EMS was invoked. This prevents the EMS from tuning the advice by interacting with the operator.

8.1.2 Data alteration not allowed

For the EMS, two alternatives exist when investigating two or more problem jobs.
First, applying fine tuning technique X to improve the lead time of problem job A may influence the solution or advice for problem job B. Hence, the order in which the problem jobs are investigated is important. This influence can only be measured when adjusting the schedule by implementing the advice for problem job A. Since data entered by PMS and OMS may not be altered by the EMS, the schedule should be copied for such calculations.

Time is needed for copying the schedule, for implementing the advice and for investigating the problem jobs in different orders. Moreover, for problem job A fine tuning technique X as well as fine tuning technique Y may be recommended. Considering that the overall improvement of the schedule is dependent of the lead time improvement of either X or Y, the influence of both X and Y has to be investigated. This complicates the investigations even more.

Second, each problem job may be investigated as if it is the only problem job available. Consequently, a solution for problem job X may contradict with the solution for problem job Y when both solutions use the same idle periods for the same machines.

8.1.3 Alternative used

The first alternative for investigating two or more problem jobs involves a large amount of additional calculations and may require direct interaction with the operator.

The second alternative involves offering advice without actually changing any schedule, a process which generally does not take as long as the process involved in the first alternative. For this alternative, the advice for problem job X may contradict with the advice for problem job Y. It is up to the operator to select the advice that has to be implemented. Any dependence in the sequence of advice implementation is then controlled by the operator.

The EMS as implemented for the ESPRIT Project 2415 is based on the second alternative.

8.2 Tools and implementation

Due to the nature of the research, the type of knowledge representation was not known beforehand. Hence to be flexible, the choice was made to
use a tool that incorporates several knowledge representations instead of, for instance, a tool that is only rule-based.

Having Delft University of Technology as a partner, the consortium was able to use the knowledge engineering environment Delfi3 (Jonker, 1990; Goedhart, 1991b) as a tool during the ESPRIT Project 2415. Being able to change and enhance such a tool is one of the advantages of using a tool built by one of the partners. For Delfi3, the ESPRIT Project 2415 offered an interesting environment to show Delfi3's features.

The intention to use knowledge-based systems within the project was presented in the Technical Annex of the ESPRIT Project 2415 (1989). However, the actual area of application crystallised during the initial stage of the project. Already during this period, Delfi3 has been used for developing prototypes and evaluating various possibilities such as distributed knowledge-based support (De Swaan Arons, 1990b).

Additionally, the Technical Annex outlines the tasks for PMS and OMS. Although the system specifications have been described during the project, changes in functionality kept appearing until the end of the project. These changes were caused either by new insights or from reviewers' suggestions.

8.2.1 Rapid prototype development

During the remainder of the project, Delfi3 has proved to be a tool for rapid prototype development concerning the knowledge and logic of the intelligent control mechanism. Certainly, Delfi3 code is easily readable and maintainable, but knowledge of logic programming is required to maintain the code. Because prototype development is made easy using Delfi3, changes in functionality of the various systems were smoothly adopted in the EMSs.

Still, performing large calculations can be a nuisance, especially when various loops are required. Moreover, various calculations for the project involve data from the database.

8.2.2 Database access

To perform the many tedious calculations, to store data in and to retrieve data from the database, the programming language C has been used. One of the advantages of C is its portability. This made it possible to write
various supporting C-programs such that they support the EMSs for both
PMS (Unix on HP9000) and OMS (OS/2 on PS/2).

Delfi3 communicates with the C-programs by exchanging data through
files. Data for the C-program is stored in a file and the C-program returns
data using another file. Further, each supporting C-program that is in-
voked and that has to communicate with the database, has to gain access
to this database by entering a user-name and a password.

8.2.3 Performance
The implementation of the EMS consists of a main program developed
using Delfi3 and various supporting C-programs. Because the individual
C-programs have to connect to the database each time they are invoked
and the ESPRIT-database has to be accessed often when investigating
problem jobs, these connections decrease the overall performance of the
EMS.

The overall performance of the EMS is also influenced by the regular file
exchange between the Delfi3-program and the supporting C-programs.

8.2.4 Conclusions
Although combining the strengths of Delfi3 and C during the project sim-
plifies the implementation of the EMS (for instance concerning adaptation
to changes), this construction has its influence on the overall time per-
formance of the EMS.

Having used this combination for the rapid prototype development and
calibration, the EMS may be enhanced to obtain a final implementation.
For instance, with the intelligent control mechanism defined and imple-
mented using Delfi3, for a commercial release the EMS may be completely
implemented using C. Therewith, the EMS does not need file exchange
between programs and the connection to the ESPRIT-database has to be
established only once.

Another possibility is to link the C-programs directly to the Delfi3-pro-
gram. This requires changes in the Delfi3 knowledge engineering envi-
rnment. Also it may be considered to change Delfi3 such that C-
programs can be linked during the development. For both options Delfi3
has to offer the possibility to connect to a database.
8.3 Results

In the previous chapter we have seen that applying various fine tuning techniques to tardy jobs often causes the mean tardiness to decrease. The best results have been accomplished when dealing with small numbers of scheduled jobs and a variety in products (six parts with different operation sequences have been used during the tests). In this case more usable idle time is available than when less variety among the products exists or when a large number of jobs has been scheduled. Since the fine tuning techniques use the idle periods of machines as resources, these results live up to the expectations.

The operator requests and obtains advice from the OMS on how to shorten the lead time for some job. Using knowledge-based system techniques, the EMS has been built. The EMS receives the operator's request from the OMS and returns some advice after investigating the indicated (problem) job. The advice generated by the EMS, includes one or more of the fine tuning techniques Overlapping Manufacture, Horizontal Splitting, Vertical Splitting or Alternative Manufacture.

Viewing these results, we conclude that we have an affirmative answer to the main question (as presented in chapter 1) for the research discussed in this thesis: Is it possible to devise and to build a knowledge-based support system that

1. is able to counsel on how to improve the lead time of a job,
2. uses the idle periods of the machines,
3. and does not schedule all over again.

8.4 Recommendations

Ideally when generating advice one weighs various factors such as costs and so on. In the ESPRIT Project, costs of transportation, machines, labour and so forth, have not been used. Advice may therefore be judged only on tardiness and lead time improvement. For instance, if enough capacity is available on some machine, this machine may be selected for Alternative Manufacture. Taking the additional costs of this machine into account, a gain of only ten minutes may not be enough to use the alternative machine. On the other hand even one minute earlier may have the job finishing in time and therefore a penalty may be prevented.

Further, the constraints set by the ESPRIT Project 2415 consortium (see section 8.1) prohibit the EMS to enhance human-centred approach. Allowing
the EMS to communicate with the operator not only improves the human-centred approach, but it may also improve the advice generated by the EMS. For instance, the operator’s expertise may be needed during the advice generation to offer even better advice than without including the operator’s expertise.

Although the advice by the EMS is not based on costs for transportation, machines and labour, the system offers the operator a tool for improving production schedules. Within an environment where the scheduling system makes use of these costs, the EMS may use this information for generating advice. Also, allowing the operator to interact with the EMS could improve the advice generated by the EMS.

Concluding, using costs for transportation, machines and labour, and including interaction between EMS and the operator may be part of future research.
Bibliography

(Abou-Zeid, 1985)

(Baker, 1974)

(Baker, 1984)

(Besant, 1986)
Improving Production Schedules Without Scheduling

(Besant, 1990)

(Black, 1983)

(Blackstone, 1982)

(Cavestro, 1990)

(Date, 1990)
C.J. Date, An Introducion to Database Systems, 5th edn., 1, Addison-Wesley, 1990.

(De Swaan Arons, 1989)

(De Swaan Arons, 1990)
(De Swaan Arons, 1991)

(Edwards, 1971)

(Eliyahu, 1987)

(Elvers, 1973)

(Falcão, 1990)

(Feldmann, 1989)

(French, 1982)
Improving Production Schedules Without Scheduling

(Fujiwara, 1986)

(Gallagher, 1986)

(Gere, 1966)

(Goedhart, 1991a)

(Goedhart, 1991b)

(Goldratt, 1989)

(Hartley, 1983)
(Hatzikonstantis, 1991)

(Hatzikonstantis, 1992)

(Jain, 1990)

(Jansen, 1993)
E.P. Jansen, “Fine-Tuning Schedules by Overlapping Manufacture”, International Journal of Advanced Manufacturing Technology, 8(6), December 1993, pp. ..-..

(Jonker, 1990)

(Kanet, 1989)

(Kusiak, 1988)
Improving Production Schedules Without Scheduling

(Liu, 1991)

(Parrello, 1988)

(Pegler, 1990)

(Ranky, 1990)

(Sen, 1984)

(Slatter, 1990)

(Technical Annex ESPRIT Project 2415, 1989)
(Thunissen, 1987)
R. Thunissen, “Tariff information at the Dutch Railways”, 
_Informatie_, 1(2), 1987, pp. 6-8. (in Dutch)

(Van der Drift, 1990)

(Verbraeck, 1990)

(Verbraeck, 1991)

(Waarts, 1992)

(Wemmerlöv, 1989)
Index

80386DX 87, 88
advanced manufacturing technology 11, 79
AGV see Automated Guided Vehicle
ALT-machine 104, 105
ALT-operation 104, 105
Alternate Operation 20, 24
Alternative Manufacture 36, 37, 92, 102-107, 117-120, 126
Atlas Datensysteme 6
Automated Guided Vehicle 4
C 51, 52, 83-94, 124, 125
CAD see Computer Aided Design
CalculateOverlapLess 99
candidate schedule 11, 21-24, 27, 88, 90
CAP see Computer Aided Planning
cell capacity 105
CNC see Computer Numerical Control
computer 80386DX 87, 88
HP9000 87, 88, 125
PS/2 87, 88, 125
SUN-380 87
Computer Aided Design 4, 14, 15
Computer Aided Planning 4
Computer Numerical Control 4, 14, 109, 110
Consortium
Atlas Datensysteme 6
Delft University of Technology 6, 7, 79, 124
Harmonic Drive Antriebstechnik 6
Imperial College of Science, Technology and Medicine 6, 79
Harmonic Drive Antriebstechnik 109

Database Management System 82
DBMS see Database Management System
Delfi Project 6, 7, 8
Delfi1 8
Delfi2 8
Delfi2+ 8
Delfi3 6, 9, 83, 87, 89, 98, 124, 125
Delft University of Technology 6, 7, 79, 123
dispatching rule
EDD 19, 112-120
FCFS 20
FIFO 20
MDD 19, 114
MQTI 20, 112, 117-119
MRO 19
MST 19
SI 19
SIO 19
SPT 19, 112, 114-118
SRPT 19
S/OPN 112, 114, 118, 120
DREL 98-102
CalculateOverlapLess 99
FindBestCombination 102, 103
HandleThisSetting 102
PossibleDistribution 98, 99, 100
due date 11, 17-35, 91, 111-114
EDD 19, 112-120
Enmycin 8
EMS see Expert Management System
ESPRIT 6, 9, 12-16, 79-88, 106, 109, 121-126
ESPRIT database table
REPLY 91
SCHEDULE 90
Improving Production Schedules Without Scheduling

Ethernet 88
European Commission 121
Expert Management System 7, 12, 80-100, 105, 109-111, 119, 123-126
expert system shell
  Delfi 8
  Delfi2 8
  Delfi2+ 8
  Enycin 8

Factory Automation 2
FCFS 20
FIFO 20
FindBestCombination 102, 103
fine tuning technique
  Alternative Manufacture 36, 37, 92, 102-107, 117-120, 126
  Horizontal Splitting 11, 35, 36, 76-78, 88-92, 102, 105, 126
  Overlapping Manufacture 11, 30, 31, 39-51, 88, 92-105, 117, 118, 120, 126
  Vertical Splitting 11, 33, 34, 40, 63-68, 77, 78, 88, 92, 100-106, 117-120, 126
Flexible Manufacturing System 4, 10
FMS see Flexible Manufacturing System
free time see preceding free time
forward shift 43-50, 58, 59, 64, 65, 72, 76-78, 92-105, 114, 118, 119

Group Technology 3, 106
GT see Group Technology

HandleThisSetting 102
Harmonic Drive Antriebstechnik 6, 109
heuristic 11-27, 36, 76, 90, 110
  Alternate Operation 20, 24
Look Ahead 20, 24, 25, 36, 76
Horizontal Splitting 11, 35, 36, 76-78, 88-92, 102, 105, 126
HP9000 87, 88, 125
human-centred 79, 80, 122, 126

idle machine period 94
idle time 1, 20, 25, 29, 30, 34-36, 39-46, 76, 77, 101, 115, 126
NAIT 43, 47, 56-62, 97, 99
Imperial College of Science, Technology
  and Medicine 6, 79
Integrated System 7, 8, 12, 16, 80-88, 109, 121
JIT see just-in-time
job-shop scheduling 11, 13, 17, 25
just-in-time 2, 3, 21
knowledge-based system
  Delfi3 6, 9, 83, 87, 89, 98, 124, 125
lead time 1, 2, 4, 10, 22, 27-41, 48-77, 96, 99, 101, 122, 123, 126
linear programming 52-54
Look Ahead 20, 24, 25, 36, 76
LP see linear programming
LT see lead time

machine
  NXT-machine 32, 44-49, 56-61, 90, 97, 99
  OVL-machine 32, 44-49, 56-61
  SPL*-machine 34, 35, 63, 64, 73, 77, 101, 102
  SPL-machine 34-36, 76-78, 101
  ALT-machine 104, 105
  PRV-machine 90
Manufacturing Resource Planning 3, 5, 6, 7, 83
Material Requirements Planning 3, 6
MDD 19, 114
minimum lead time 50-62, 66-77, 96
MLT see minimum lead time
MLT-problem 51-57, 66, 77, 102
MLTm-problem 51-58, 66, 78, 102
Modula-2 83, 84
MQT 20, 112, 117-119
MRO 19
MRP see Material Requirements Planning
MRP II see Manufacturing Resource Planning
MST 19
NAIT 43, 47, 56-62, 97, 99
NC see Numerical Control
not allocatable idle time see NAIT
Numerical Control 4
NXT-machine 32, 44-49, 56-61, 90, 97, 99
NXT-operation 44-50, 56-61, 90-100
OMS see Operator Management System
  operation
  OVL-operation 32, 44-50, 56-61, 94-99
  SPL-operation 35, 63-65, 76, 77, 101
  NXT-operation 44-50, 56-61, 90-100
Index

ALT-operation 104, 105
PRV-operation 90
Operator Management System 7, 80-91, 96, 100, 105-112, 123-126
Overlapping Manufacture 11, 30, 31, 39-43, 48-51, 88, 92-105, 117-120, 126
OVL-machine 32, 44-49, 56-61
OVL-operation 32, 44-50, 56-61, 94-99

Performance measure 21-23, 27
PMS see Production Management System
Possible distribution 98-100
Preceding free time 64, 94, 95, 100-104
Pre-emption 17, 25, 35, 76
Problem
MLT 51-57, 66, 77, 102
MLTM 51-58, 66, 78, 102
Processing time 18, 19, 26, 44, 48, 52, 64, 76, 98, 112, 114
Production Management System 7, 80, 87, 88, 105, 123, 124
Programming language
C 51, 52, 83-89, 94, 124, 125
Modula-2 83, 84
Project
Delfi 6-8
PRV-machine 90
PRV-operation 90
PS/2 87, 88, 125

RDBMS see Relational Database Management System
Relational Database Management System 82-89
REPLY table 91

SCHEDULE table 90
Scheduling period 17, 111-117
Shop floor 1, 5-7, 13, 21-28, 36
SI 19
SIO 19
S/OPN 112, 114, 118, 120
SPL*-machine 34, 35, 63, 64, 73, 77, 101, 102
SPL-machine 34-36, 76-78, 101
SPL-operation 35, 63-65, 76, 77, 101
SPL-period 64, 77
SPT 19, 112-118
SQL 84-87, 92, 93
SRPT 19
Structured Query Language see SQL
SUN-380 87

System
DBMS 82
EMS 7, 12, 80-100, 105, 109-111, 119, 122-126
FMS 4
OMS 7, 80-91, 96, 100, 105-112, 122-126
PMS 7, 80, 87, 88, 105, 122-124
RDBMS 82-89

Total work content 112-114
TWK see total work content
Vertical Splitting 11, 33, 34, 40, 63-68, 77, 78, 88, 92, 100-106, 117-120, 126

XREL 100

139
Chapter 6 “Using the Fine Tuning Techniques” discusses the EMS, which is described as an intelligent control mechanism. Various modules that are part of the EMS have been collected in this appendix. These modules have been implemented using the knowledge engineering environment Delfi3. For detailed information on the Delfi3 syntax, we refer to the Delfi3 user manual (Goedhart, 1991b).

**Main module: EMS-advisor**

```plaintext
{ Module : EMSoms
  Author : R.P. Jansen
  Project: Esprit 2415
  Date : 92 07 03 (final EC-review date)

  This module checks the applicability of several fine tuning techniques.
}
```

**MODULE EMSoms**

**USE** BasicInf | Utility | PrtOMS | OVOptOMS | VOOptOMS | AMOptOMS

the options to be investigated are indicated through the option_indicator in the request table. OptionIndicated checks whether an option has to be investigated or not. The option_indicator is constructed by adding 2 to the power of (option number - 1) of the options to be investigated.

1 overlapping manufacture
2 horizontal splitting
3 vertical splitting
4 alternative manufacture

Example: if the options overlapping manufacture, vertical splitting and alternative manufacture have to be investigated, the option_indicator is equal to:

```
option_indicator = 1 + 4 + 8 = 13
```
Improving Production Schedules Without Scheduling

```
)
DREL OptionIndicated
DOMAIN option,optionindicator : <INT>
[
  option=0 AND 1 AND FALSE OR
  option = optionindicator MOD 2 AND 1 AND TRUE OR
  OptionIndicated(option,1,optionindicator DIV 2)
]
}
)
{ read the specifics to set up the EMS
  - caller id (= cell id for OMS)
  - request number
  - sequence indicator concerning the start of the
    NXT-operation with respect to the PREV-operation
    1  NXT-setup    after PREV-setdown
    2  NXT-setup    after PREV-processing
    3  NXT-processing after PREV-setdown
    4  NXT-processing after PREV-processing
}
QREL go  
  [ request ]
VAR length,index : <INT>
| statement  : <TEXT>
  textVAL   : <LIST <LIST <TEXT>>>
  intVAL    : <LIST <LIST <INT>>>
  realVAL   : <LIST <LIST <REAL>>>
  val       : <TEXT>
  problemOrders : <LIST <TEXT>>
  sched_id   : <LIST <TEXT>>
  indicators : <LIST <INT>>
  sizes      : <LIST <INT>>
  violations : <LIST <INT>>
  requestnumber : <INT>
  caller_id  : <TEXT>
[
  SelectRequest(currentProblem."requestnumber,
    currentProblem."caller_id,
    currentProblem."sequencetype)   AND
  PrintIntroInfo()   AND
  WRITE("+ deleting old information concerning ",
    currentProblem.caller_id," ...") AND
  WRITELN(1)   AND
  DeleteOldEMSInfo(currentProblem.caller_id)   AND
  WRITE("+ selecting request information ...") AND
  WRITELN(2)   AND
  "violations = []
[
  Check whether all problem jobs have to be selected
]
[ CheckForAll(currentProblem.caller_id,
    currentProblem.requestnumber,
    problemOrders,sched_id,indicators,
    "sizes,"violations) AND

142
```
problemOrders <> [] AND ! AND
`textVAL = [sched_id] AND
`textVAL = LISTER(problemOrders, textVAL) AND
`intVAL = [sizes] AND
`intVAL = LISTER(indicators, intVAL)
OR
{
   Get information on the problem jobs from the database
}

SQL(["select ord_id, option_indicator, sched_id, o_size ",
   "from request, ord where caller_id=:t1 ",
   "and id=ord_id ",
   "and rnumber=:i1 ",
   ["t"],["i"],["t"],["i"],
   [currentProblem.caller_id],
   [currentProblem.requestnumber], [],
   `textVAL, `intVAL, `realVAL]
] AND
[ textVAL = [] AND ! AND
SQL(["select ord_id ",
   "from request where caller_id=:t1 ",
   "and rnumber=:i1 ",
   ["t"],["i"],["t"],
   [currentProblem.caller_id],
   [currentProblem.requestnumber], [],
   `textVAL, `intVAL, `realVAL] AND
InsertReply(0, 1,
   currentProblem.requestnumber,
   currentProblem.caller_id,
   HEAD(HEAD(textVAL)), 1, 1,"", 0,"", ",","", 0,0,0,0)
AND
WRITE("Advice request from: ", currentProblem.caller_id) AND
WRITELN(1) AND
WRITE("No problem jobs present") AND
WRITELN(1) OR
GetItem(1, textVAL, currentProblem.\`orders)
AND
currentProblem.\`sched_id = HEAD(HEAD(TAIL(textVAL)))
AND
GetItem(1, intVAL, currentProblem.\`indicator)
AND
GetItem(2, intVAL, currentProblem.\`sizes)
AND
currentProblem.\`optionnumber = 0
AND
PrintRequestInfo()
AND
GetLength(currentProblem.\`orders, length)
AND
length > 0
AND
index = 1
AND
[ Repeat() AND
          ProcessJob(index, violations)
   `index := index+1
   index > length AND ! AND
   TRUE
] OR TRUE
] AND
WRITE("---

AND

WRITELN(2) AND 1 AND
WRITE('Further investigations: ') AND FALSE
]

EREL

DRELC ProcessJob
DOMAIN index : <INT>
| violations : <LIST <INT>>
VAR freelist : <LIST <INT>>
| maxbatches : <LIST <INT>>
| batchsize : <INT>
| forwardshifts : <LIST <INT>>
| option_indicator : <INT>
| optionFound : <BOOL>
[
GetItem(index, currentProblem.orders, currentOrder.`order_id) AND
GetItem(index, currentProblem.sizes, currentOrder.`Size) AND
GetItem(index, currentProblem.indicator, `option_indicator) AND
PrintProcessingOrderInfo() AND
currentOrder.`solutiontype := 0 AND
currentProblem.`optionnumber := 0 AND
keepOrder.`order_id := currentOrder.order_id AND
keepOrder.`solutiontype := currentOrder.solutiontype AND
{
Calculate the severity of the due date violation in minutes
}
[ violations = [] AND 1 AND
WRITE('Calculating ...') AND
WRITELN(1) AND
GetDuedateViolation(currentProblem.caller_id,
currentOrder.order_id,
currentProblem.sched_id,
currentOrder.`violationMinutes)
OR
WRITE('Retrieving ...') AND
WRITELN(1) AND
GetItem(index, violations, currentOrder.`violationMinutes)
] AND
[ currentOrder.violationMinutes >= 0 AND 1 OR
currentOrder.`violationMinutes = 0
] AND
PrintDuedateViolation() AND
optionFound = FALSE AND
[
{ Overlapping Manufacture }
OptionIndicated(1, option_indicator)
AND
PrintCheckingOption("Overlapping Manufacture")
AND
CheckOverlapping()
AND
`optionFound := TRUE
AND
PutReply(index)
AND
FALSE
{ Vertical Splitting }
  OptionIndicated(3, option_indicator)
  PrintCheckingOption("Vertical Splitting")
  CheckVerticalSplitting()
  'optionFound := TRUE
  PutReply(index)
FALSE
{ Alternative Manufacture }
  OptionIndicated(4, option_indicator)
  PrintCheckingOption("Alternative Manufacture")
  CheckAlternativeManufacture()
  'optionFound := TRUE
  PutReply(index)
FALSE
TRUE
[ optionFound AND ! OR PutReply(index) ] AND ! AND
WRITELN(1)
]
ERELE
DREL PutReply
DOMAIN replynumber : <INT>
VAR intVAL : <LIST <LIST <INT>>>
  realVAL : <LIST <LIST <REAL>>>
  textVAL : <LIST <LIST <TEXT>>>
  string : <TEXT>
  string2 : <TEXT>
  length : <INT>
  OVLOperation : <TEXT>
  AMoperation : <TEXT>
[ currentProblem.`optionnumber := currentProblem.optionnumber+1
  AND
  
  { No Solution }
  keepOrder.solutiontype = 0 AND ! AND
  InsertReply(0, replynumber,
    currentProblem.requestnumber,
    currentProblem.caller_id,
    keepOrder.order_id,
    currentProblem.`optionnumber,
    1,"","","","",0,0,
    currentOrder.violationMinutes,
    currentOrder.Size) OR
  
  { Overlapping Manufacture }
  keepOrder.solutiontype = 1 AND ! AND
  ComposeDistributionString(keepOrder.maxpartsdistribution,
    string) AND
  GetLength(keepOrder.maxpartsdistribution, length) AND

Improving Production Schedules Without Scheduling

GetItem(keepOrder.operation, keepOrder.operations, `OVLOperation) AND
InsertReply(keepOrder.solutiontype, replynumber,
currentProblem.requestnumber, 
currentProblem.caller_id, 
keepOrder.order_id, 
currentProblem.optionnumber, 
1, OVLOperation, 
length,string,"", 
keepOrder.sdate,keepOrder.stime, 
keepOrder.maxforwardshift, 
currentOrder.violationMinutes, 
currentOrder.Size) OR

{ 
Vertical Splitting
} 

keepOrder.solutiontype = 3 AND ! AND
ComposeDistributionString(keepOrder.maxpartsdistribution, 
string) AND
ComposeResourceString(keepOrder.machines,string2) AND
GetLength(keepOrder.maxpartsdistribution,length) AND
GetItem(keepOrder.operation, keepOrder.operations, 
`OVLOperation) AND
InsertReply(keepOrder.solutiontype, replynumber, 
currentProblem.requestnumber, 
currentProblem.caller_id, 
keepOrder.order_id, 
currentProblem.optionnumber, 
1, OVLOperation, 
length,string,string2, 
"",0, 
keepOrder.maxforwardshift, 
currentOrder.violationMinutes, 
currentOrder.Size) OR

{ 
Alternative Manufacture
} 

keepOrder.solutiontype = 4 AND ! AND
GetItem(keepOrder.operation, keepOrder.operations, 
`AMOperation) AND
InsertReply(keepOrder.solutiontype, replynumber, 
currentProblem.requestnumber, 
currentProblem.caller_id, 
keepOrder.order_id, 
currentProblem.optionnumber, 
1, AMOperation, 
1,"",keepOrder.machines, 
keepOrder.sdate,keepOrder.stime, 
keepOrder.maxforwardshift, 
currentOrder.violationMinutes,
currentOrder.Size) OR
]
ERELE

XREL CheckForAll
DOMAIN caller_id : <TEXT>
| rnumber : <INT>
RANGE problemOrders : <LIST <TEXT>>
| sched_ids : <LIST <TEXT>>
| indicators : <LIST <INT>>
| sizes : <LIST <INT>>
| violations : <LIST <INT>>
CALL OChkAll
ERELE

XREL SelectRequest
RANGE rnumber : <INT>
| caller_id : <TEXT>
| sched_type : <INT>
CALL OSelReq
ERELE

XREL InsertReply
DOMAIN option_type : <INT>
| reply_no : <INT>
| request_no : <INT>
| caller_id : <TEXT>
| order_id : <TEXT>
| option_no : <INT>
| option_line_no : <INT>
| operation : <TEXT>
| no_sub_batches : <INT>
| distribution : <TEXT>
| resource_ids : <TEXT>
| sdate : <TEXT>
| stime : <INT>
| value : <INT>
|duedateViolation : <INT>
| size : <INT>
CALL BInsRep
ERELE

XREL DeleteOldEMSinfo
DOMAIN caller_id : <TEXT>
CALL BDelInfo
ERELE

XREL GetDuedateViolation
DOMAIN cell_id : <TEXT>
| order_id : <TEXT>
| sched_id : <TEXT>
RANGE minutes : <INT>
CALL OCalcDDV
EREL

XREL ComposeDistributionString
DOMAIN distribution : <LIST <INT>>
RANGE string : <TEXT>
CALL BCompose
EREL

XREL ComposeResourceString
DOMAIN machines : <LIST <TEXT>>
RANGE string : <TEXT>
CALL BMakeRes
EREL

END

FTT module: Overlapping Manufacture
{ Module : OVOptOMS
  Author : E.P. Jansen
  Project: Esprit 2415
  Date : 92 07 03 (final EC-review date)

  This module checks the Overlapping Manufacture fine tuning technique.
}
MODULE OVOptOMS

USE OptSup | BasicInf | Utility

DREL *CheckOverlapping
VAR freelist : <LIST <INT>>
copyfreelist : <LIST <INT>>
maxbatches : <LIST <INT>>
batchsize : <INT>
numberofoperations : <INT>
freetime : <INT>
OLLOperation : <TEXT>
NXToperation : <TEXT>
forwardshift : <INT>
machines : <LIST <TEXT>>

[ EmptyCurrent()
  AND
  currentOrder.^preference = 5
  AND
  currentOrder.^usableFreetime = 9999
  AND

  Calculate the preceding freetime for each operation of the
  problem order/job
]
GetOperationsAndFreetime(currentProblem.caller_id,
currentOrder.order_id,
currentProblem.sched_id,
'freelist,'machines) AND

{ For forward shift to be a possible alternative it is imperative that the last operation has some preceding freetime attached }
ForwardShiftLastOperationPossible(freelist,TRUE,
    'Overlapping') AND

    copyfreelist = freelist

{ Invert the list of preceding free time such that the investigations can be made from the last preceding free time to the first using the HEAD construction }
InvertList('copyfreelist)
GetLength(currentOrder.operations,numberOfOperations) AND
InvertList('machines)

{ Determine the overlap and forward shift for each operation, starting with the last, until the preceding freetime is equal to 0. Also present the largest forward shift and according overlap }
DetermineBestOverlap(numberOfOperations,copyfreelist,
machines) AND

[ currentOrder.maxforwardshift > 0 AND !
    OR
    FALSE
] AND
GetItem(currentOrder.operation+1,freelist,'fertime) AND
GetItem(currentOrder.operation,currentOrder.operations,
    'OVLoperation) AND
GetItem(currentOrder.operation+1,currentOrder.operations,
    'NXToperation) AND

{ Determine the start date and start time for this forward shift }
ComputeStartDateAndTime(currentProblem.caller_id,
    currentOrder.order_id,
    currentProblem.sched_id,
    currentProblem.sequenceType,
    OVLoperation,
    NXToperation,
    currentOrder.NAIt,
    currentOrder.maxforwardshift,
    'forwardshift,
    currentOrder.'sdate,
    currentOrder.'stime) AND

currentOrder.'maxforwardshift = forwardshift AND

WRITELN(1) AND
WRITE('Due date violation job ',currentOrder.order_id,': ',
    currentOrder.violationMinutes) AND WRITELN(1) AND
WRITE('Best results') AND WRITELN(1) AND

149
WRITE('--------------') AND WRITELN(1) AND
WRITE('  Overlap operation : ', 
  OVLoperation) AND WRITELN(1) AND
WRITE('  Next operation : ', 
  NXToperation) AND WRITELN(1) AND
WRITE('  Start date first sub-batch : ', 
  currentOrder.sdate) AND WRITELN(1) AND
WRITE('  Start time first sub-batch : ', 
  currentOrder.stime) AND WRITELN(1) AND
WRITE('  Total forward shift : ', 
  currentOrder.maxforwardshift) AND WRITELN(1) AND
WRITE('  Parts distribution : ', 
  currentOrder.maxpartsdistribution) AND WRITELN(1) AND
WRITE('  ---- used') AND WRITELN(1) AND
WRITE('  Order size : ', 
  currentOrder.Size) AND WRITELN(1) AND
WRITE('  Preceding free time operat. ', 
  currentOrder.operation+1, ':' ,freetime) AND WRITELN(1)
AND
WRITE('  Freetime list : ', 
  freelist) AND WRITELN(1) AND
WRITE('  NAIT before setting date/time: ', 
  currentOrder.NAIT) AND WRITELN(2) AND
CurrentToKeep() AND
keepOrder.'solutiontype := 1
]

EREL

DREL DetermineBestOverlap
DOMA IN numberofoperations : <INT>
  | freelist : <LIST <INT>>
  | machines : <LIST <TEXT>>
VAR freetime : <INT>
  | intlist : <LIST <LIST <INT>>>>
  | tdummy : <LIST <LIST <TEXT>>>>
  | rdummy : <LIST <LIST <REAL>>>>
  | pOVL : <INT>
  | pNXT : <INT>
  | Smin : <INT>
  | maxNoSubbatches : <INT>
  | forwardshift,NAIT : <INT>
  | partsdistribution : <LIST <INT>>
  | OVLoperation : <TEXT>
  | NXToperation : <TEXT>
  | machine : <TRXT>

[ numberofoperations <= 1 AND !
  OR
  "freetime = HEAD(freelist) AND
  "machine = HEAD(machines) AND
  [ [TAIL(machines) <> [] AND

150
machine = HEAD(TAIL(machines))
AND ! AND
DetermineBestOverlap(numberofoperations-1, TAIL(freelist), TAIL(machines))
OR
[ currentOrder.usableFreetime <= freetime AND ! OR currentOrder.\`usableFreetime = freetime ] AND
[ freetime = 0 AND ! OR GetItem(numberofoperations-1, currentOrder.operations, \`OVLoperation) AND GetItem(numberofoperations, currentOrder.operations, \`NXTOperation) AND SQL('
select ptpu, sp_size_min, sp_size_max, o_size ,
  from part_op, ord ,
  where part_op.part_id=ord.part_id ,
  and ord.id=a ,
  and (part_op.no=t1 or part_op.no=t2) ,
  `order by part_op_no`
),
[`t`,`t`,`t`],`i`,`i`,`i`,`i`],
currentOrder.order_id,OVLoperation,NXTOperation],
[],[],
`tdummy,`intlist,`rdummy) AND
{
  intlist = [ [pOVL,pNX], [SminOVL,SminNX], [SmaxOVL,SmaxNX], [SizeOVL,SizeNX] ]
}
pOVL = HEAD(HEAD(intlist)) AND
pNX = HEAD(TAIL(HEAD(intlist))) AND
currentOrder.\`Smin = HEAD(HEAD(TAIL(intlist))) AND
currentOrder.\`Size =
  HEAD(TAIL(TAIL(TAIL(intlist)))) AND
\`forwardshift = 0 AND
\`NAIT = 0 AND
[ currentOrder.Size DIV currentOrder.Smin < 2 AND ! OR Smin = currentOrder.Smin AND
{
  calculate the maximum number of sub-batches possible: | _ Size/Smin _ |
}
maxNoSubbatches = currentOrder.Size DIV Smin AND
{
  the maximum number should be less or equal to a preferred maximum
}
[ maxNoSubbatches > currentOrder.preference AND ! AND \`Smin = currentOrder.Size DIV currentOrder.preference AND \`maxNoSubbatches = currentOrder.Size DIV Smin OR

151
TRUE
] AND
{
Based on the relation between pOVL and pNXT the proper strategy is used
}
[ pOVL=pNXT AND ] AND
CalculateOverlapEqual(numberofoperations,pOVL,pNXT,
Smin,maxNoSubbatches,TRUE,
``forwardshift,`NAIT,
`partsdistribution)
OR
[ pOVL<pNXT AND ] AND
CalculateOverlapLess(numberofoperations,pOVL,pNXT,
Smin,maxNoSubbatches,TRUE,
``forwardshift,`NAIT,
`partsdistribution)
OR
CalculateOverlapMore(numberofoperations,pOVL,pNXT,
Smin,maxNoSubbatches,TRUE,
``forwardshift,`NAIT,
`partsdistribution)
]
]
AND
currentOrder.`forwardshifts =
LISTER(forwardshift,currentOrder.forwardshifts) AND
currentOrder.`NAITs = LISTER(NAIT,currentOrder.NAITs) AND
{
if the forward shift is equal to the maximum forward shift so far, this solution is selected if the NAIT is smaller than the previous NAIT if the forward shift is larger than the maximum forward shift so far, this solution is selected
}
[ forwardshift <= currentOrder.maxforwardshift AND ] AND
[ [ forwardshift = currentOrder.maxforwardshift AND
NAIT < currentOrder.NAIT
] AND ] AND
currentOrder.`maxpartsdistribution =
partsdistribution AND
currentOrder.`NAIT = NAIT AND
currentOrder.`operation = numberofoperations-1
OR
TRUE
]
OR
currentOrder.`maxforwardshift = forwardshift AND
currentOrder.`NAIT = NAIT AND
currentOrder.`maxpartsdistribution =
partsdistribution AND
currentOrder.`operation = numberofoperations-1
] AND
DetermineBestOverlap(numberofoperations-1,
TAIL(freelist),
TAIL(machines))
[ AND
  [ forwardshift > currentOrder.violationMinutes AND
    maxNoSubbatches > 2
  ] AND ! AND
  [ `maxNoSubbatches = maxNoSubbatches-1 AND
    `Smin = currentOrder.Size DIV maxNoSubbatches AND
    CalculateOverlapEqual(op_no,pOVL,pNXT,Smin,maxNoSubbatches,
      FALSE,`forwardshift,`NAIT,
      `partsdistribution)
    OR
    TRUE
  ]
  OR
  forwardshift > currentOrder.violationMinutes AND !
  OR
  first
] } EREL

{ pOVL < pNXT:
  When all sub-batches have the same size, NAIT is zero.
  The surplus has to be distributed among the sub-batches
  without increasing NAIT and without increasing the lead time.
  The latter implies that the first sub-batch size can not be
  increased!
}

DREL CalculateOverlapLess

DOMAIN op_no : <INT>
  pOVL : <INT>
  pNXT : <INT>
  smin : <INT>
  maxno : <INT>
  first : <BOOL>
RANGE forwardshift : <INT>
  NAIT : <INT>
  partsdistribution : <LIST <INT>>
VAR Smin : <INT>
  maxNoSubbatches : <INT>

{ Smin = smin AND
  maxNoSubbatches = maxno AND
  `partsdistribution = [ ] AND
  [ PossibleDistribution(pOVL,pNXT,Smin,currentOrder.Size,
    0,1,`partsdistribution) AND !
    OR
    maxNoSubbatches = 2 AND ! AND
    `Smin = Smin + 1 AND
    CalculateOverlapEqual(op_no,pOVL,pNXT,Smin,maxNoSubbatches,
      first,`forwardshift,`NAIT,
      `partsdistribution)

154
OR
`maxNoSubbatches = maxNoSubbatches-1` AND
`Smin = currentOrder.Size DIV maxNoSubbatches AND
CalculateOverlapLess(op_no,pOVL,pNXT,Smin,maxNoSubbatches,
first,`forwardshift`,`NAIT,
`partsdistribution)
OR
TRUE
] AND
ComputeForwardshiftAndNAIT(currentOrder.Size,
pOVL,pNXT,partsdistribution,
`forwardshift`,`NAIT) AND
[ forwardshift <= currentOrder.usablePretteime AND !
OR
`forwardshift = currentOrder.usablePretteime
] AND
[
[ forwardshift > currentOrder.violationMinutes AND
maxNoSubbatches > 2
] AND ! AND
[ `maxNoSubbatches = maxNoSubbatches-1` AND
`Smin = currentOrder.Size DIV maxNoSubbatches AND
CalculateOverlapLess(op_no,pOVL,pNXT,Smin,maxNoSubbatches,
FALSE,`forwardshift`,`NAIT,
`partsdistribution)
OR
TRUE
]
OR
forwardshift > currentOrder.violationMinutes AND !
OR
first
]
ERE

{ pOVL > pNXT }
} DREL CalculateOverlapMore
DOMAIN op_no : <INT>
| pOVL : <INT>
| pNXT : <INT>
| smin : <INT>
| maxno : <INT>
| first : <BOOL>
RANGE forwardshift : <INT>
| NAIT : <INT>
| partsdistribution : <LIST <INT>>
VAR Smin : <INT>
| maxNoSubbatches : <INT>
[
Smin = smin AND
\[
\text{maxNoSubbatches} = \text{maxno} \ \text{AND}
\]
\[
\text{partsdistribution} = [] \ \text{AND}
\]
\[
\text{PossibleDistribution}(\text{pNXT}, \text{pOVL}, \text{Smin}, \text{currentOrder.Size}, 0, 1, \text{partsdistribution}) \ \text{AND} \ \text{!}
\]
\[
\text{OR}
\]
\[
\text{maxNoSubbatches} = 2 \ \text{AND} \ \text{!} \ \text{AND}
\]
\[
\text{Smin} = \text{Smin} + 1 \ \text{AND}
\]
\[
\text{CalculateOverlapMore}(\text{op_no}, \text{pOVL}, \text{pNXT}, \text{Smin}, \text{maxNoSubbatches}, \text{first}, \text{forwardshift}, \text{NAIT}, \text{partsdistribution})
\]
\[
\text{OR}
\]
\[
\text{maxNoSubbatches} = \text{maxNoSubbatches} - 1 \ \text{AND}
\]
\[
\text{Smin} = \text{currentOrder.Size} \ \text{DIV} \ \text{maxNoSubbatches} \ \text{AND}
\]
\[
\text{CalculateOverlapMore}(\text{op_no}, \text{pOVL}, \text{pNXT}, \text{Smin}, \text{maxNoSubbatches}, \text{first}, \text{forwardshift}, \text{NAIT}, \text{partsdistribution})
\]
\[
\text{OR}
\]
\[
\text{TRUE}
\]
\[
\text{AND}
\]
\[
\text{InvertList}(\text{partsdistribution}) \ \text{AND}
\]
\[
\text{ComputeForwardshiftAndNAIT}(\text{currentOrder.Size}, \text{pOVL}, \text{pNXT}, \text{partsdistribution}, \text{forwardshift}, \text{NAIT})
\]
\[
\text{AND}
\]
\[
\text{forwardshift} \leq \text{currentOrder.usableFreetime} \ \text{AND} \ \text{!}
\]
\[
\text{OR}
\]
\[
\text{forwardshift} = \text{currentOrder.usableFreetime} \ \text{AND}
\]
\[
\text{[}
\text{forwardshift} > \text{currentOrder.violationMinutes} \ \text{AND}
\]
\[
\text{maxNoSubbatches} > 2
\]
\[
\text{] \ \text{AND} \ \text{!}
\]
\[
\text{[}
\text{maxNoSubbatches} = \text{maxNoSubbatches} - 1 \ \text{AND}
\]
\[
\text{Smin} = \text{currentOrder.Size} \ \text{DIV} \ \text{maxNoSubbatches} \ \text{AND}
\]
\[
\text{CalculateOverlapMore}(\text{op_no}, \text{pOVL}, \text{pNXT}, \text{Smin}, \text{maxNoSubbatches}, \text{FALSE}, \text{forwardshift}, \text{NAIT}, \text{partsdistribution})
\]
\[
\text{OR}
\]
\[
\text{TRUE}
\]
\[
\text{]
\]
\[
\text{OR}
\]
\[
\text{forwardshift} > \text{currentOrder.violationMinutes} \ \text{AND} \ \text{!}
\]
\[
\text{OR}
\]
\[
\text{first}
\]
\[
\]
\text{EREL}

\text{DREL PossibleDistribution}

\text{DOMAIN}
\begin{align*}
\text{pOVL} &: \text{<INT> } \\
\text{pNXT} &: \text{<INT> } \\
\text{Smin} &: \text{<INT> } \\
\text{Size} &: \text{<INT> } \\
\text{surplus} &: \text{<INT> } \\
\text{subbatch} &: \text{<INT> }
\end{align*}
```
RANGE partsdistribution : <LIST <INT>>
VAR di : <INT>
  | si : <INT>

[ Size <= 0 AND !
  OR di = ((subbatch-1)*Smin+surplus)*(pNXT-pOVL)) DIV pOVL AND
  si = Smin+di AND
  [ Size-si <= 0 AND ! AND
    `si = Size
  OR
    Size-si >= Smin AND !
  OR
    `si = Size-Smin AND
  [ si >= Smin AND !
    OR
    si = 1 AND ! AND
    `si = Size
  OR
    FALSE
]
] AND
PossibleDistribution(pOVL,pNXT,Smin,Size-si,surplus+di,
  subbatch+1,`partsdistribution) AND
`partsdistribution = LISTER(si,partsdistribution)

EREL

XREL ComputeForwardshiftAndNAIT
DOMAIN size : <INT>
  | pOVL : <INT>
  | pNXT : <INT>
  | partsdistribution : <LIST <INT>>
RANGE forwardshift : <INT>
  | NAIT : <INT>
CALL OCalcOVL
EREL

XREL ComputeStartDateAndTime
DOMAIN caller_id : <TEXT>
  | order_id : <TEXT>
  | sched_id : <TEXT>
  | sequencetype : <INT>
  | OVLOperation : <TEXT>
  | NXToperation : <TEXT>
  | NAIT : <INT>
  | forwardshift : <INT>
RANGE newforwardshift : <INT>
  | sdate : <TEXT>
  | stime : <INT>
CALL OCalcDat
EREL
```
END

FTT module: Vertical Splitting
{
  Module : VSoptOMS
  Author : E.P. Jansen
  Project: Esprit 2415
  Date : 92 07 03 (final EC-review date)

  This module checks the Vertical Splitting fine tuning technique.
}

MODULE VSoptOMS

USE OptSup | BasicInf | Utility | GapsOMS

DREL *CheckVerticalSplitting
VAR machinesPerOp : <LIST <INT>>
  freelist : <LIST <INT>>
  optionPossible : <BOOL>
  continueChecking : <BOOL>
  copyMachinesPerOp : <LIST <INT>>
  copyFreelist : <LIST <INT>>
  numberOfOperations : <INT>
  batchsize : <INT>
  freetime : <INT>
  forwardshift : <INT>
  solutionFound : <BOOL>
  SPOperation : <TEXT>
  machines : <TEXT>

EmptyCurrent() AND
  "solutionFound " = FALSE AND
{
  Check for which operation several machines of the same type exist.
}

GetNumberOfMachinesPerOperation(currentProblem.caller_id,
  currentOrder.order_id,
  currentProblem.sched_id,
  machinesPerOp,
  "optionPossible") AND

optionPossible AND
{
  Calculate the preceding freetime for each operation of the problem order/job
}

GetOperationsAndFreetime(currentProblem.caller_id,
  currentOrder.order_id,
  currentProblem.sched_id,
  currentOrder."operations,
  "freelist,"machines") AND

GetLength(currentOrder.operations,numberOfOperations) AND
copymachinesPerOp = machinesPerOp
copyfreelist = freelist
InvertList('copymachinesPerOp)
InvertList('copyfreelist)
[ DetermineBestVerticalSplit(TRUE, numberofoperations,
copymachinesPerOp,
copyfreelist) AND ! AND

`solutionFound = TRUE
OR
TRUE
] AND
`continueChecking = FALSE AND
[ solutionFound AND ! AND
  [ ForwardshiftLastOperationPossible(freelist, FALSE, '') AND
    ! AND
    `continueChecking = TRUE
    OR
    TRUE
  ]
OR
ForwardshiftLastOperationPossible(freelist, TRUE,
  'Vertical Splitting') AND
`continueChecking = TRUE
] AND
[ continueChecking AND
  `copymachinesPerOp = TAIL(copymachinesPerOp) AND
  `copyfreelist = TAIL(copyfreelist) AND
  DetermineBestVerticalSplit(FALSE, numberofoperations-1,
copymachinesPerOp,
copyfreelist) AND ! AND

currentOrder.operation <> 0
OR
solutionFound
] AND
GetItem(currentOrder.operation, currentOrder.operations,
  'SPLoperation) AND WRITELN(1) AND
WRITE('Due date violation job ', currentOrder.order_id, ':: ',
currentOrder.violationMinutes) AND WRITELN(1) AND
WRITE('Best results') AND WRITELN(1) AND
WRITE('--------------') AND WRITELN(1) AND WRITELN(1)
WRITE('Splitting operation : ',
SPLoperation) AND WRITELN(1) AND
WRITE('Total forward shift : ',
currentOrder.maxforwardshift) AND WRITELN(1) AND
WRITE('Parts distribution : ',
currentOrder.maxpartsdistribution) AND WRITELN(1) AND
WRITE('Machines :
currentOrder.machines) AND WRITELN(2) AND

CurrentToKeep() AND
keepOrder.` solutiontype := 3
]
REPL
DREL DetermineBestVerticalSplit
DOMAIN lastoperation : <BOOL>
| numberofOperations : <INT>
| machinesPerOperation : <LIST <INT>>
| freelist : <LIST <INT>>
VAR numberOfMachines : <INT>
| freetime : <INT>
| operation : <TEXT>
| machines : <LIST <TEXT>>
| partsdistribution : <LIST <INT>>

[ numberofOperations = 0 AND !
OR
numberOfMachines = HEAD(machinesPerOperation) AND
freetime = HEAD(freelist) AND
[ numberOfMachines = 1 AND !
OR
getItem(numberofOperations, currentOrder.operations, operation) AND
DetermineVerticalSplit(lastoperation, operation, numberOfMachines)
] AND
[ currentOrder.usableFreetime <= freetime AND !
OR
currentOrder.`usableFreetime = freetime
] AND
[ numberOfMachines = 1 AND !
OR
[ BestGapSet.forwardshift <= currentOrder.maxforwardshift AND !
OR
`partsdistribution = [] AND
`machines = [] AND
GetMachinesAndDistribution(BestGapSet.gaps,
`machines,
`partsdistribution) AND
currentOrder.`maxforwardshift = BestGapSet.forwardshift AND
currentOrder.`maxpartsdistribution = partsdistribution AND
currentOrder.`machines = machines AND
currentOrder.`operation = numberofoperations
] ] AND
[ lastoperation AND ! AND numberOfMachines <> 1
OR
DetermineBestVerticalSplit(FALSE, numberofOperations-1,
TAIL(machinesPerOperation),
TAIL(freelist))
]
]}
ERE
DREL GetMachinesAndDistribution

DOMAIN GapSet : <LIST < Gap >>

RANGE machines : <LIST < TEXT >>
| partsdistribution : <LIST < INT >>

VAR aGap : < Gap >

[ GapSet = [] AND ! OR
  `aGap = HEAD(GapSet) AND
  `machines = LISTER(aGap.mach_id,machines) AND
  `partsdistribution = LISTER(aGap.s+aGap.Smin,
  partsdistribution) AND
  GetMachinesAndDistribution(TAIL(GapSet),`machines,
  `partsdistribution)
]

EREL

DREL DetermineVerticalSplit

DOMAIN lastoperation : < BOOL >
| operation : < TEXT >
| numberOfMachines : < INT >

VAR set_up : < INT >
| set_down : < INT >
| ptpu : < INT >
| Smin : < INT >
| orig_finish : < INT >
| machines : < LIST < TEXT >>
| sdates : < LIST < LIST < TEXT >> >
| stimes : < LIST < LIST < INT >> >
| sizes : < LIST < LIST < INT >> >
| minutes : < LIST < LIST < INT >> >
| GapSetG : < LIST < Gap >>
| length : < INT >

[ GetMachinesAndCapacity(currentProblem.caller_id,
currentOrder.order_id,
currentProblem.sched_id,operation,TRUE,
`set_up,`set_down,`ptpu,`Smin,
`orig_finish,`machines,`sdates,`stimes,
`sizes,`minutes) AND

currentOrder. `Smin := Smin AND

`GapSetG = [] AND

CreateGapSetG(set_up,set_down,ptpu,machines,sizes,
minutes,Smin,TRUE,`GapSetG) AND

GapSetG <> [] AND

GetLength(GapSetG,`length) AND

length > 1 AND

FindVSplit(GapSetG,numberOfMachines,currentOrder.Size,
orig_finish,Smin)
]

EREL

XREL GetNumberOfMachinesPerOperation
Improving Production Schedules Without Scheduling

XREL GetMachinesAndCapacity
DOMAIN caller_id : <TEXT>
| ord_id : <TEXT>
| sched_id : <TEXT>
| no : <TEXT>
| verticalSearch : <BOOL>
RANGE set_up : <INT>
| set_down : <INT>
| ptpu : <INT>
| Smo : <INT>
| orig_finish : <INT>
machines : <LIST <TEXT>>
sdates : <LIST <LIST <TEXT>>>
stimes : <LIST <LIST <INT>>>
sizes : <LIST <LIST <INT>>
| minutes : <LIST <LIST <INT>>
CALL OCapMc
EREL

END

FTT module: Alternative Manufacture

{ Module : AMOptOMS
   Author : E.P. Jansen
   Project : Esprit 2415
   Date : 92 07 03 (final EC-review date)

This module checks the Alternative Manufacture fine tuning technique.
}

MODULE AMOptOMS

USE OptSup | BasicInf | Utility | PrtOMS

DREL *CheckAlternativeManufacture
VAR freelist : <LIST <INT>>
| copyfreelist : <LIST <INT>>
| altMachinesPerOp : <LIST <INT>>
| copyMachinesPerOp : <LIST <INT>>
| batchsize : <INT>
| numberOfOperations : <INT>
| freetime : <INT>
| forwardshift : <INT>
| solutionPound : <BOOL>

162
| optionPossible        : <BOOL>
| AMOperation         : <TEXT>
| PREVOperation       : <TEXT>
| machines            : <LIST <TEXT>>

EmptyCurrent() AND
currentTimeOrder.'Smin = 0 AND
'solutionFound = FALSE AND

Check for which operation an alternative machine exists

GetNumberOfAlternativeMachinesPerOperation(
currentProblem.caller_id,
currentOrder.order_id,
currentProblem.sched_id,
'altMachinesPerOp,
'optionPossible) AND

optionPossible AND

Calculate the preceding freetime for each operation of the problem order/job

GetOperationsAndFreetime(currentProblem.caller_id,
currentOrder.order_id,
currentProblem.sched_id,currentOrder.
'operations,'freetime,'machines) AND

GetLength(currentOrder.operations,numberofoperations) AND
'copyMachinesPerOp = altMachinesPerOp AND
'copyfreetime = freetime AND
InvertList('copyMachinesPerOp) AND
InvertList('copyfreetime) AND
InvertList('machines)

DetermineBestAlternativeManufacture(numberofoperations,
copyMachinesPerOp,
copyfreetime,
machines) AND

[currentOrder.maxforwardshift > 0 AND !
OR
FALSE
] AND

GetItem(currentOrder.operation,currentOrder.operations,
'AMoperation) AND

WRITE(') AND
WRITE('Due date violation job ',currentOrder.order_id,': ',
currentOrder.violationMinutes) AND WRITE(') AND
WRITE('Best results') AND WRITE(') AND
WRITE('----------') AND WRITE(') AND
WRITE('Operation for ') AND WRITE(') AND
WRITE('alternative manufacture : ',
AMoperation) AND WRITE(') AND
WRITE('Start date : ',
currentOrder.sdate) AND WRITE(') AND
WRITE('Machine : ',

163
currentOrder.machine) AND WRITELN(1) AND
WRITE('   Start time : ',
currentOrder.stime) AND WRITELN(1) AND
WRITE('   Total forward shift : ',
currentOrder.maxforwardshift)) AND WRITELN(2) AND
CurrentToKeep() AND
keepOrder.\$solutiontype := 4 AND
\$solutionFound = TRUE
]
EREL

DREL DetermineBestAlternativeManufacture
DOMAIN
  numberOfoperations : <INT>
  | altMachinesPerOperation : <LIST <INT>>
  | freelist : <LIST <INT>>
  | machines : <LIST <TEXT>>
VAR
  numberOfMachines : <INT>
  | freetime : <INT>
  | operation : <TEXT>
[
  numberOfoperations = 0 AND !
OR
  numberOfMachines = HEAD(altMachinesPerOperation) AND
  freetime = HEAD(freelist)
  [ numberOfMachines = 0 AND !
OR
  GetItem(numberOfoperations, currentOrder.operations, \$operation) AND
  DetermineAlternativeManufacture(operation, numberOfoperations)
  ]
] EREL

DREL DetermineAlternativeManufacture
DOMAIN
  operation : <TEXT>
  | opnoindex : <INT>
VAR
  set_up : <INT>
  | set_down : <INT>
  | ptpu : <INT>
  | machines : <LIST <TEXT>>
  | sdateslist : <LIST <LIST <TEXT>>>
  | stimeslist : <LIST <LIST <INT>>>
  | sizeslist : <LIST <LIST <INT>>>
  | minuteslist : <LIST <LIST <INT>>>
  | sdates : <LIST <TEXT>>
  | stimes : <LIST <INT>>
  | sizes : <LIST <INT>>
  | minutes : <LIST <INT>>
  | machine : <TEXT>
  | sdate : <TEXT>
  | stime : <INT>
  | size : <INT>
| minute    : <INT> |
| nummachines : <INT> |
| forwardshift : <INT> |
| thisforwardshift : <INT> |

GetAlternativeMachinesAndCapacity(currentProblem.caller_id, 
currentOrder.order_id, 
currentProblem.sched_id, 
operation, 
`machines`, `sdateslist`, 
`stimeslist`, `sizeslist`, 
`minuteslist`) AND

GetLength(`machines`, `nummachines`) AND
nummachines <> 0 AND
`forwardshift` = 0 AND
[ Repeat() AND
  `machine` = HEAD(`machines`) AND
  `sdates` = HEAD(`sdateslist`) AND
  `stimes` = HEAD(`stimeslist`) AND
  `sizes` = HEAD(`sizeslist`) AND
  `minutes` = HEAD(`minuteslist`) AND
  `machines` := TAIL(`machines`) AND
  `sdateslist` := TAIL(`sdateslist`) AND
  `stimeslist` := TAIL(`stimeslist`) AND
  `sizeslist` := TAIL(`sizeslist`) AND
  `minuteslist` := TAIL(`minuteslist`) AND
  `nummachines` := nummachines - 1 AND
  PrintInvestigatingAlternativeMachine(`machine`) AND
  InvestigateMachine(`machine`, operation, opnoindex, `sdates`, 
  `stimes`, `sizes`, `minutes`, 
  `sdate`, `stime`, `size`, `minute`) AND
  nummachines = 0 AND ! AND
  TRUE
]
]

EREL

DRELD InvestigateMachine

DOMAIN
machine : <TEXT>
operation : <TEXT>
opnoindex : <INT>
sdates : <LIST <TEXT>>
stimes : <LIST <INT>>
sizes : <LIST <INT>>
minutes : <LIST <INT>>

RANGE
sdate : <TEXT>
stime : <INT>
size : <INT>
minute : <INT>

VAR
forwardshift : <INT>
[

Get the gap which is large enough and is the earliest on
Improving Production Schedules Without Scheduling

the machine

GetFirstOnMachine(sdates, stimes, sizes, minutes,
    `sdate, `stime, `size, `minute) AND 1 AND

Determine the amount of forward shift when using the gap
just found

DetermineForwardshift(currentProblem.caller_id,
    currentOrder.order_id,
    currentProblem.sched_id,
    operation, machine,
    sdate, stime, `forwardshift) AND

[ currentOrder.machine = " "
  OR
  [ forwardshift >= currentOrder.maxforwardshift AND
    minute < currentOrder.minutes
  ]
] AND 1 AND

currentOrder.`machine := machine AND

currentOrder.`operation := opnoindex AND

currentOrder.`maxforwardshift := forwardshift AND

currentOrder.`sdate := sdate AND

currentOrder.`stime := stime AND

currentOrder.`minutes := minute

OR

TRUE

]

OR

TRUE

]

ERSEL

DREL GetFirstOnMachine

DOMAIN sdates : <LIST <TEXT>>
| stimes : <LIST <INT>>
| sizes : <LIST <INT>>
| minutes : <LIST <INT>>

RANGE sdate : <TEXT>
| stime : <INT>
| size : <INT>
| minute : <INT>

[ sdates = [] AND 1 AND FALSE

OR

`size = HEAD(sizes) AND

[ size >= currentOrder.Size AND
  `sdate = HEAD(sdates) AND
  `stime = HEAD(stimes) AND
  `minute = HEAD(minutes)

OR

GetFirstOnMachine(TAIL(sdates), TAIL(stimes), TAIL(sizes),

166
Additional module: Gap Handling

{ Module: GapsOMS
  Author: E.P. Jansen
  Project: Esprit 2415
  Date : 92 07 03 (final EC-review date)

  This module contains relations for handling gaps in the schedules and assists research into the Vertical Splitting FTT.
}

MODULE GapsOMS
USE Utility

DOBJ *GapSet
PRIVATE
  gaps : <LIST <Gap>>
  | forwardshift : <INT>
  | orig_finish : <INT>
EOBJ

IOBJ
  *BestGapSet : GapSet([],0,0)
EOBJ

DOBJ *Gap
PRIVATE
  mach_id : <TEXT>
  | Smin : <INT>
  | pSPL : <INT>
  | g : <INT>
  | t : <INT>
  | u : <INT>
  | s : <INT>
EOBJ

DREL *CreateGapSetG
DOMAIN set_up : <INT>
  | set_down : <INT>
  | ptpu : <INT>
  | machines : <LIST <TEXT>>
  | sizesList : <LIST <LIST <INT>>>
  | minutesList : <LIST <LIST <INT>>>
  | Smin : <INT>
  | Vertical : <BOOL>
RANGE GapSetG : <LIST <Gap>>
VAR machine : <TEXT>
  | sizes : <LIST <INT>>
  | minutes : <LIST <INT>>
[ machines = [] AND !
  OR
  machine = HEAD(machines) AND
  sizes = HEAD(sizesList) AND
  minutes = HEAD(minutesList) AND
  HandleThisMachine(machine,sizes,minutes,ptpu,
  -ptpu,Smin,`GapSetG) AND
  CreateGapSetG(set_up,set_down,ptpu,TAIL(machines),
  TAIL(sizesList),TAIL(minutesList),Smin,Vertical,
  `GapSetG)
]
EREL

DREL HandleThisMachine

168
Additional module: Gap Handling

DOMAIN machine : <TEXT>
    | sizes  : <LIST <INT>>
    | minutes : <LIST <INT>>
    | ptpu   : <INT>
    | lastSize : <INT>
    | Smin   : <INT>
RANGE GapSetG   : <LIST <Gap>>
VAR
    | usable : <INT>
    | size   : <INT>

[ sizes = [] AND ! OR
size = HEAD(sizes) AND
[ size DIV ptpu <= lastSize DIV ptpu AND ! AND
   HandleThisMachine(machine,TAIL(sizes),TAIL(minutes),
   ptpu,lastSize,Smin,\GapSetG)
   OR
   start = HEAD(minutes) AND
   [ lastSize = -ptpu AND ! AND
     `usable = start
            OR
     `usable = start + ptpu*(lastSize DIV ptpu)
   ] AND
   `GapSetG = LISTER(Gap(machine,Smin,ptpu,size,start,usable,0),
   GapSetG) AND
   HandleThisMachine(machine,TAIL(sizes),TAIL(minutes),
   ptpu,lastSize,Smin,\GapSetG)
]
]
ERE

DREL *FindVSplit

DOMAIN TheGapSet  : <LIST <Gap>>
    | numMachines : <INT>
    | totalSize   : <INT>
    | orig_finish : <INT>
    | Smin        : <INT>
VAR keepGapSetG   : <LIST <Gap>>
    | GapSetG     : <LIST <Gap>>
    | GapSetH     : <LIST <Gap>>
aGap  : <Gap>
totalPossible : <INT>
    | m           : <INT>

[ `GapSetG = TheGapSet AND
  BestGapSet."gaps"  = [] AND
  BestGapSet."forwardshift" = 0 AND
  BestGapSet."orig_finish" = orig_finish AND

  { Sort the gaps in G according to their usable start times
  }
  SortGapSet(`GapSetG) AND
  `keepGapSetG = GapSetG AND
Improving Production Schedules Without Scheduling

m = 2 AND
[ Repeat() AND
  'GapSetG = keepGapSetG AND
  'GapSetH = [] AND
  HandleThisSetting(m,totalSize,Smin,'GapSetG,'GapSetH) AND
  m := m+1 AND
  m > numMachines AND ] AND
TRUE
] AND
BestGapSet.gaps <> []
]

DREL HandleThisSetting

DOMAIN m : <INT>
| totalSize : <INT>
| Smin : <INT>
RANGE GapSetG : <LIST <Gap>>
| GapSetH : <LIST <Gap>>
[
  totalSize-m*Smin < 0 AND ] OR
[
  Fill gap set H with m gaps on different machines
]
FillGapSetH(m,'GapSetG,'GapSetH) AND
FindBestCombination(m,totalSize-m*Smin,'GapSetG,'GapSetH)
]

DREL FindBestCombination

DOMAIN m : <INT>
| totalSize : <INT>
RANGE GapSetG : <LIST <Gap>>
| GapSetH : <LIST <Gap>>
VAR aGap : <Gap>
| totalPossible : <INT>
| finish : <INT>
[
  Calculate the maximum number of parts that an be assigned to the gaps in H: Hs
]
'totalPossible = 0 AND
GetTotalPossible(GapSetH,'totalPossible) AND
[ totalPossible < totalSize AND ] OR
FillCombination(totalSize,GapSetH,'GapSetH,'finish) AND
[ BestGapSet.orig_finish-finish <= BestGapSet.forwardshift AND ] OR
BestGapSet.'gaps : = GapSetH AND
BestGapSet.\`forwardshift :=
    BestGapSet.orig_finish-finish AND
    RemoveLaterFromGapSet(finish,`GapSetG)
]
] AND
[ GapSetG = [] AND
  OR
{
  Determine the gap that can hold the least
}
\`aGap = HEAD(GapSetH)
AND
GetMinimumGap(TAIL(GapSetH),`aGap)
AND
RemoveNotLargerFromGapSet(aGap.g DIV aGap.pSPL,
    `GapSetG)
AND
AdjustUsableStartTimes(aGap,GapSetH,`GapSetG)
AND
SortGapSet(`GapSetG)
AND
RemoveMinOrExistingFromGapSetH(m,aGap,HEAD(GapSetG).mach_id,
    `GapSetH)
AND
\`aGap = HEAD(GapSetG)
AND
\`GapSetG = TAIL(GapSetG)
AND
RemoveFromGapSet(aGap.mach_id,`GapSetH)
AND
\`GapSetH = LISTER(aGap,GapSetH)
AND
FindBestCombination(m,totalSize,`GapSetG,`GapSetH)
]
]
EREL

DREL *PrintBestGapSet
[
  WRITELN(1) AND
  WRITE('Best gap set') AND WRITELN(1) AND
  WRITE('------------') AND WRITELN(1) AND
  WRITE(' gap set :',
    BestGapSet.gaps) AND WRITELN(1) AND
  WRITE(' original finish: ',
    BestGapSet.orig_finish) AND WRITELN(1) AND
  WRITE(' new finish : ', BestGapSet.orig_finish-
    BestGapSet.forwardshift) AND WRITELN(1) AND
  WRITE(' forwardshift :',
    BestGapSet.forwardshift) AND WRITELN(1)
]
EREL

DREL SortGapSet
RANGE GapSet : <LIST <Gap>>
VAR GapSetSorted : <LIST <Gap>>
[
  GapSetSorted = [] AND
  SortGap(GapSet,`GapSetSorted) AND
  `GapSet = GapSetSorted
]
EREL
DREL SortGap
DOMAIN inlist : <LIST <Gap>>
RANGE sortlist : <LIST <Gap>>
VAR h : <Gap>
| t : <LIST <Gap>>
[
  inlist = [] AND ! OR
  h = HEAD(inlist) AND
  t = TAIL(inlist) AND
  InsertGap(h, `sortlist) AND
  SortGap(t, `sortlist)
]
EREL

DREL InsertGap
DOMAIN item : <Gap>
RANGE list : <LIST <Gap>>
VAR h : <Gap>
| t : <LIST <Gap>>
[
  list = [] AND ! AND
  `list = LISTER(item, list)
  OR
  h = HEAD(list) AND
  t = TAIL(list) AND
  [ h.u > item.u AND ! AND
    `list = LISTER(item, list)
    OR
    InsertGap(item, `t) AND
    `list = LISTER(h, t)
  ]
]
EREL

DREL RemoveGap
DOMAIN index : <INT>
RANGE fromlist : <LIST <Gap>>
| returnitem : <Gap>
VAR tail : <LIST <Gap>>
| keepitem : <Gap>
[
  `returnitem = HEAD(fromlist) AND
  `tail = TAIL(fromlist) AND
  [ index = 1 AND ! AND
    `fromlist=tail
    OR
    keepitem=returnitem
  ]
  RemoveGap(index-1, `tail, `returnitem) AND
  `fromlist = LISTER(keepitem, tail)
]
EREL
```
DREL RemoveFromGapSet
DOMAIN mach_id : <TEXT>
RANGE GapSet : <LIST <Gap>>
VAR aGap : <Gap>
[ GapSet = [] AND !
  OR ~aGap = HEAD(GapSet) AND
  ~GapSet = TAIL(GapSet) AND
  [ aGap.mach_id = mach_id AND !
    OR
    RemoveFromGapSet(mach_id,~GapSet) AND
    ~GapSet = LISTER(aGap,GapSet)
  ]
]
EREEL

DREL RemoveLaterFromGapSet
DOMAIN finish : <INT>
RANGE GapSetG : <LIST <Gap>>
VAR aGap : <Gap>
[ GapSetG = [] AND !
  OR ~aGap = HEAD(GapSetG) AND
  ~GapSetG = TAIL(GapSetG) AND
  [ aGap.t >= finish AND !
    OR
    RemoveFromGapSet(aGap.mach_id,~GapSetG) AND
    ~GapSetG = LISTER(aGap,GapSetG)
  ]
]
EREEL

DREL RemoveNotLargerFromGapSet
DOMAIN size : <INT>
RANGE GapSetG : <LIST <Gap>>
VAR aGap : <Gap>
[ GapSetG = [] AND !
  OR ~aGap = HEAD(GapSetG) AND
  ~GapSetG = TAIL(GapSetG) AND
  [ aGap.g DIV aGap.pSPL <= size AND !
    OR
    RemoveFromGapSet(aGap.mach_id,~GapSetG) AND
    ~GapSetG = LISTER(aGap,GapSetG)
  ]
]
EREEL

DREL RemoveMinOrExistingFromGapSetH
DOMAIN m : <INT>
```
Improving Production Schedules Without Scheduling

\[ \text{minGap} : \langle \text{Gap} \rangle \]
\[ \text{mach_id} : \langle \text{TEXT} \rangle \]
\text{RANGE} \begin{align*} \text{GapSetH} & : \langle \text{LIST} \langle \text{Gap} \rangle \rangle \\
\text{VAR} & \text{ length} : \langle \text{INT} \rangle \\
\end{align*} \]
\[
\begin{align*}
\text{RemoveFromGapSet(mach_id, GapSetH)} & \text{ AND} \\
\text{GetLength(GapSetH, length)} & \text{ AND} \\
\begin{align*}
\text{length} & < m \text{ AND I} \\
\text{OR} & \text{ RemoveFromGapSet(minGap, mach_id, GapSetH)} \\
\end{align*}
\]
\}
\]
\text{ERELE}

\text{DREL} \text{ FillGapSetH}
\begin{align*}
\text{DOMAIN m} & : \langle \text{INT} \rangle \\
\text{RANGE} \begin{align*} \text{GapSetG} & : \langle \text{LIST} \langle \text{Gap} \rangle \rangle \\
\text{GapSetH} & : \langle \text{LIST} \langle \text{Gap} \rangle \rangle \\
\end{align*} \\
\text{VAR} \begin{align*} \text{aGap} & : \langle \text{Gap} \rangle \\
\text{length} & : \langle \text{INT} \rangle \\
\end{align*} \\
\end{align*} \]
\[
\begin{align*}
\text{GetLength(GapSetH, length)} & \text{ AND} \\
\begin{align*}
\text{length} & = m \text{ AND I} \\
\text{OR} & \begin{align*}
\text{aGap} & = \text{HEAD}(\text{GapSetG}) \\
\text{GapSetG} & = \text{TAIL}(\text{GapSetG}) \\
\text{RemoveFromGapSet(aGap, mach_id, GapSetH)} & \text{ AND} \\
\text{GapSetH} & = \text{LISTER}(aGap, GapSetH) \\
\text{FillGapSetH}(m, \text{GapSetG, GapSetH}) & \text{ AND} \\
\end{align*}
\end{align*}
\]
\}
\]
\text{ERELE}

\text{DREL} \text{ GetTotalPossible}
\begin{align*}
\text{DOMAIN GapSetH} & : \langle \text{LIST} \langle \text{Gap} \rangle \rangle \\
\text{RANGE total} & : \langle \text{INT} \rangle \\
\end{align*} \]
\[
\begin{align*}
\text{GapSetH} & = [] \text{ AND I} \\
\text{OR} & \begin{align*}
\text{total} & = \text{total} + (\text{HEAD}(\text{GapSetH}).g \text{ DIV } \text{HEAD}(\text{GapSetH}).pSPL) \text{ AND} \\
\text{GetTotalPossible(TAIL(\text{GapSetH}), total)} & \text{ AND} \\
\end{align*}
\]
\}
\]
\text{ERELE}

\text{DREL} \text{ GetMinimumGap}
\begin{align*}
\text{DOMAIN GapSet} & : \langle \text{LIST} \langle \text{Gap} \rangle \rangle \\
\text{RANGE minGap} & : \langle \text{Gap} \rangle \\
\end{align*} \]
\[
\begin{align*}
\text{GapSet} & = [] \text{ AND I} \\
\text{OR} & \begin{align*}
(\text{HEAD}(\text{GapSet}).g \text{ DIV } \text{HEAD}(\text{GapSet}).pSPL) & >= \\
\text{AND} & \begin{align*}
\text{minGap.g DIV minGap.pSPL} & \text{ AND I} \\
\end{align*}
\end{align*}
\]
\]
\}

174
Additional module: Gap Handling

```
`minGap = HEAD(GapSet)
] AND
GetMinimumGap(TAIL(GapSet), `minGap)
]
EREL

DREL AdjustUsableStartTimes
DOMAIN minGap : <Gap>
   | GapSetH : <LIST <Gap>>
RANGE GapSetG : <LIST <Gap>>
VAR aGap : <Gap>
   | size : <INT>
[
   GapSetG = [] AND ! OR
   `aGap = HEAD(GapSetG) AND
   `GapSetG = TAIL(GapSetG) AND
   GetGapSize(aGap, GapSetH, `size) AND ! AND
   aGap. u = aGap.t + aGap.pSpl* (size DIV aGap.pSpl)
   OR
   aGap. u = aGap.t + aGap.pSpl* (minGap.g DIV aGap.pSpl)
   ] AND
AdjustUsableStartTimes(minGap, GapSetH, `GapSetG) AND
`GapSetG = LISTER(aGap, GapSetG)
]
EREL

DREL GetGapSize
DOMAIN aGap : <Gap>
   | GapSetH : <LIST <Gap>>
RANGE size : <INT>
[
   GapSetH = [] AND ! AND FALSE OR
   [ HEAD(GapSetH).mach_id = aGap.mach_id AND ! AND
     size = HEAD(GapSetH).g
     OR
     GetGapSize(aGap, TAIL(GapSetH), `size)
   ]
]
EREL

XREL FillCombination
DOMAIN totalSize : <INT>
   | GapSetH : <LIST <Gap>>
RANGE GapSetHFilled : <LIST <Gap>>
   | finish : <INT>
CALL OF11Gaps
EREL

END
```
Appendix B: ESPRIT Project 2415 database

In Chapter 6 “Using the Fine Tuning Techniques” we discuss the EMS as part of the Integrated System. An Oracle RDBMS serves as the communication and data handling back-bone of the Integrated System (PMS, OMS, EMS).

All systems use tables of the common database, but not all tables are used by all systems. For example, the tables CELL, MACHINE and MACHTYPE are used by PMS, OMS and EMS, whilst EMSINFO and GAPINFO are solely used by EMS.

The main ESPRIT Project 2415 database tables are the following (in SQL-notation):

```sql
table cell
{
  id             char(4),
  name           char(25)
};

table machtype
{
  id             char(6),
  name           char(25),
  alter_type     char(6),
  no_control     char(16),
  inf_cap        char(1)
};

table machine
{
  id             char(6),
  name           char(25),
  type           char(6),
  cell_id        char(4),
  vb_modul       char(2),
  status         number(2),
  break_off_date date,
  break_off_time number(4),
  break_off_dur  number(4)
};
```
Improving Production Schedules Without Scheduling

break_period number(4), sdate date, stime number(4), fdate date, ftime number(4)

table machrule
{ id char(6), efficiency number(3), rule number(3), load_ratio number(5,2) }

table part
{ id char(16), name char(25), mat_id char(12), family number(3) }

table part_op
{ part_id char(16), part_op_no char(3), type_id char(6), cell_id char(4), mach_type char(6), mach_id char(6), set_up number(5), set_down number(5), ptpu number(5), nc_prog char(8), sp_marker char(1), sp_size_min number(5), sp_size_max number(5) }

table ord
{ id char(10), name char(25), part_id char(16), mat_id char(12), marker char(1), o_size number(5), status number(2), ext_prio number(2), sdate date, act_sdate date, pps_sdate date, fdate date, act_fdate date, pps_fdate date }

table op
{ ord_id char(10), no char(3), sub_no char(1) }
type_id char(6),
con_ord_id char(5),
job_id char(5),
cell_id char(4),
mach_type char(6),
mach_id char(6),
status number(2),
prio number(2),
op_size number(5),
sp_marker char(1),
sp_size_min number(5),
sp_size_max number(5),
set_up number(5),
set_down number(5),
ptpu number(5),
nc_prog char(8),
nc_name char(8),
sdate date,
stime number(4),
act_sdate date,
act_stime number(4),
ftime number(4),
act_fdate date,
act_ftime number(4),
part_fin number(5) ;

table calendar
{   c_date date,
    fac_day char(1),
    week number(2),
    day number(1) } ;

table shiftype
{   sh_type char(3),
    stime number(4),
    ftime number(4),
    break1_stime number(4),
    break2_stime number(4),
    break3_stime number(4),
    break4_stime number(4),
    break1_dur number(4),
    break2_dur number(4),
    break3_dur number(4),
    break4_dur number(4),
    work_stime number(4),
    work_dur number(4) } ;

table shiftall
{   cell_id char(4),
    mach_id char(6),
    sdate date,
    fdate date,
shift_type  char(3),
overtime   char(1)  )

<table>
<thead>
<tr>
<th>Table</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>kapmasch</td>
<td>mach_id  char(6), cap_date date, mach_cap number(5), ord_id char(10), no char(3), sub_no char(1)</td>
</tr>
<tr>
<td>con_ord</td>
<td>id  char(5), op_id char(6), mach_id char(6), mat_id char(12), prio number(2), vb_modul char(2), status number(2), sdate date, act_sdate date, fdate date, act_fdate date</td>
</tr>
<tr>
<td>stcell</td>
<td>strategy char(1), cell_id char(4), rule1 char(2), rule2 char(2), rule3 char(2), rule4 char(2)</td>
</tr>
<tr>
<td>prioreg</td>
<td>rule char(2), text char(60)</td>
</tr>
<tr>
<td>konf</td>
<td>asco_modul1 char(2), con_number number(4), st_trans number(5), wait_time number(5), job_number number(4), idle_time number(5), asco_modul2 char(2), asco_modul3 char(2), side_pos number(5), opt_factor number(5), dmc_factor number(2)</td>
</tr>
<tr>
<td>constraint</td>
<td>mach_id char(6), no char(2), type number(1),</td>
</tr>
</tbody>
</table>
Appendix B: ESPRIT Project 2415 database

```c
job1 char(10),
job2 char(10),
op1 char(3),
op2 char(3)
);

table schedule
{
sched_id char(16),
creator_id char(4),
cell_id char(4),
mach_id char(6),
ord_id char(10),
no char(3),
sub_no char(1),
sdate date,
stime number(4),
setup_fdate date,
setup_ftime number(4),
op_fdate date,
op_ftime number(4),
ftime date,
ftime number(4)
};

table schedule_info
{
creator_id char(4),
sched_id char(16),
cell_id char(4),
sdate date,
stime number(4),
ftime date,
ftime number(4),
sched_type number(1)
};

table request
{
rt_number number(2),
caller_id char(4),
ord_id char(10),
sched_id char(16),
sched_type number(1),
option_indicator number(4),
max_split number(2),
reply_id number(2)
};

table reply
{
id number(2),
request_no number(2),
caller_id char(4),
ord_id char(10),
option_no number(2),
option_line_no number(2),
option_type number(2),
operation_id char(3),
no_sub_batches number(2),
distribution char(60)
};
Improving Production Schedules Without Scheduling

```
  resource_ids  char(90),
  sdate        date,
  stime        number(4),
  value        number(4) );

table emsinfo
  (    caller_id   char(4),
       ord_id      char(10),
       no          char(3),
       free_prec_cap number(8) );

table gapinfo
  (    caller_id   char(4),
       sched_id    char(16),
       mach_id     char(6),
       no          char(3),
       sdate       date,
       stime       number(4),
       o_size      number(5),
       minutes     number(5),
       set_up      number(5),
       set_down    number(5),
       ptpu        number(5) );
```
Appendix C: ESPRIT Project 2415 Brochures

To present the Integrated System, the consortium compiled a two page leaflet. This leaflet describes the Distributed Manufacturing Planning and Control philosophy which constitutes the foundation of the ESPRIT Project 2415. The PMS, the OMS and the EMS have been developed according to this philosophy and they are described together with the computer infrastructure.

Imperial College of Science, Technology and Medicine, and Delft University of Technology have designed a four page brochure on the OMS and the EMS. This brochure is meant to support commercialisation of the systems. The next pages show the leaflet and the brochure.
DISTRIBUTED MANUFACTURING PLANNING AND CONTROL

The Distributed Manufacturing Planning and Control philosophy is based on decentralizing responsibility to management levels where the most relevant expertise and data exist to facilitate decision making. This approach leads to improved productivity, greater job satisfaction, and improved data flow between management levels. The system enjoys complete modularity and is based on three distinctive components:

- The Production Management System (PMS): A machine shop production system
- The Operational Management System (OMS): A cell level manufacturing and control system
- The System (CIM): An expert system for scheduling in the machine shop or cell level.

The system is based on a combination of dedicated hardware and software components, distributed technology with all communication achieved via the ORACLE database system. The development of the module has been under the direction of the ESPRIT (European Strategic Programme for Research in Information Technology) Project 2415 Distributed Manufacturing Planning and Control completed in summer 1992. The partners on this project are Imperial College of Science, Technology and Medicine, Delft University of Technology, Atlas Datensysteme, and Harmonic Drive Aktiengesellschaft.

figure C-1 leaflet Distributed Manufacturing Planning and Control (page 1)
DISTRIBUTED MANUFACTURING PLANNING AND CONTROL

ESPRIT Project 2415

PMS
Production Management System
- Provides production-oriented interactive planning functionality based on a modern windows user interface
- Supplements MRP II systems for the control of shop floor and cells
- Simulates the manufacturing process on cell and machine type level, taking into account user-defined priorities and scheduling strategies for meeting competing goals
- Allows comparison of various planning alternatives using Gantt charts and statistical evaluations.

OMS
Operator Management System
- Dynamic scheduling for manufacturing cells
- User programmable scheduling rules
- Gantt charts show the sequence and flow of work orders through a cell
- Bar charts display available and required capacity
- Statistical stock evaluation and report generation
- Feedback of shop floor information into the current schedule
- Storage, upload, and downloading of NC programs
- Ease of use through windows user interface.

EMS
Expert Management System
- Acts on requests by PMS and OMS
- Investigates problem orders (PMS) and problem jobs (OMS)
- Uses idle times to forecast delays using various techniques such as:
  - overlapping manufacture (OMS)
  - horizontal splitting, i.e. on one machine (OMS)
  - vertical splitting, i.e. between cells (PMS) and machines (OMS)
  - alternative manufacture (PMS, OMS)
- Acts without disturbance to other orders and jobs.

For more information please contact:

**figure C-2** leaflet Distributed Manufacturing Planning and Control
(page 2)
figure C-3  brochure Operator Management System with expert management support (page 1)
The OMS is a planning and scheduling tool that assists the day-to-day management of a manufacturing cell or a small workshop. The OMS can alleviate the task of scheduling by generating workplans much faster than it could be done by hand. However, selecting the "best" alternative is left to the operator's judgement and expertise. Numerical information is presented in an easy to understand graphical format.

1. Capacity analysis is performed before detailed scheduling. Overloads are quickly identified and the relevant action is taken to provide excess capacity or accept delayed completion.

2. A schedule is generated for each machine, using strategies defined by the user. A prediction can then be made on the degree to which production targets, such as due dates can be met.

3. Candidate and current schedules are evaluated according to measures of performance such as number of tardy jobs, machine utilisation, and so on.

4. Once a schedule has been implemented on the shop-floor, finish times of operations can be fed back to the OMS. If the operator judges the current schedule to be unsatisfactory, he can re-schedule or re-assess available capacity.

5. Problem jobs may be improved on by overlapping operations, splitting batches (on one machine or between machines) and using alternative machines. Advice is offered by the expert management support system.

6. The advice can be automatically implemented during re-scheduling.

The OMS is capable of linking to higher and lower levels of a computer integrated manufacturing (CIM) system. On a higher level a facility-wide Production Management System holds the basic database, and assigns orders to a manufacturing cell managed by OMS. The OMS feeds back actual operation times and the status of resources. The OMS uses the common ORACLE database-management system for data storage. It is used in conjunction with a Production Management System, networking allows OMS to access the database of the higher level system. The OMS also provides storage for NC programs that can be uploaded or downloaded to machine tool controllers via a serial RS232 link.

**Figure C-4** brochure Operator Management System with expert management support (page 2)
Basic data is entered, viewed, or modified through the OMS’s data entry tables. They contain information about:
- orders
- operations
- parts
- routes
- machine types
- machines
- production calendar
- shift definition
- shift allocation

A number of scheduling related functions are available, in order to facilitate decision making. These functions include:
- Capacity analysis
- Schedule generation
- Schedule evaluation
- Schedule updating with shop floor feedback
- Schedule improvement through fine tuning techniques
- Report generation and printing facilities

Scheduling strategies are defined by the user using a format supported by OMS. Each machine can be associated to a different rule. These rules are then interpreted when a schedule is generated.

Three types of scheduling strategies can be defined:
- Simple priority rules
- Filtered scheduling strategies
- Conditional strategies

The Operator Management System (OMS) was developed within the CEC-funded ESPRIT project 2415, entitled “Distributed Manufacturing Planning and Control”. This was a 42-month Research and Development Programme aimed at providing a computer integrated manufacturing (CIM) decision support structure, that will enable evolution of decision making to the lowest possible levels of management hierarchy.

In addition to the OMS, complementary systems have been developed by Atlas Datensysteme of Germany (Production Management System for multi-cell control) and Delft University of Technology of The Netherlands (Expert Management Support System for the fine tuning of schedules).

The OMS can work in conjunction with the Production Management System using its database and receiving from it the work assigned to the cell.

The OMS operator can request advice from the Expert Management Support System with respect to options such as batch overlapping, splitting and use of alternative machines. This option is intended as a fine tuning operation for interactive production control.

The OMS runs on an IBM PC or compatible computer. The minimum hardware requirements are:
- Intel 80386 processor based computer
- 8 Mbyte RAM
- 60 Mbyte hard disc

Basic software requirements are:
- OS/2 operating system (version 1.1 or later)
- ORACLE relational database for OS/2

Figure C-5  brochure Operator Management System with expert management support (page 3)
OMS

Shop floor management for production cells and small enterprises

The OMS is a human-centred planning and scheduling tool that assists the day to day management of a manufacturing cell or a small workshop.

Application area

- Metal working batch production
- Small to medium quantity production, medium to high product variety
- Small, workshop type factories
- Manufacturing cells within a larger production facility comprising of typically 3 to 15 machines
- The OMS can be used as a stand-alone system, or in conjunction with a higher level factory Production Management System.

Why OMS?

- The concept of manufacturing cells within larger production facilities is becoming more and more common
- Computerised production control should not be restricted to large enterprises; it is also necessary in the small workshop
- Quick response is crucial in the highly variable environment of small batch manufacturing
- Decision making can be improved by devolution to the lower levels of production control
- Easy to use on the shop floor.

Features

- Gantt chart shows the sequencing and flow of work orders
- Bar charts display available and required capacity
- User defined scheduling strategies
- Statistical evaluation of work schedules
- Multiple candidate workplans stored and compared
- Forward loading of work on finite or infinite capacity basis
- Feedback of shopfloor information into current schedule
- Job, part, operation hierarchies
- Cell, machine type, machine hierarchy of resources
- Knowledge-based system for advice on improving schedules with respect to problem jobs.

Benefits

- Helps devolve decision making to the lowest levels of a manufacturing organisation
- Increased operator involvement and job satisfaction
- Improved accuracy of current status information
- Ability to quickly respond to and solve problems
- More efficient scheduling for cells or small workshops
- Offers alternatives for dealing with problem jobs avoiding major disturbances to schedules.

For more information please contact:

Figure C-6: brochure Operator Management System with expert management support (page 4)
Summary

Developments in manufacturing and technology lead to the development of manufacturing systems that reduce the amount of decision making by those responsible for operating and supervising the manufacturing process. The reduction of decision making can especially be witnessed at the cell and workstation levels. Moreover, such manufacturing systems have often been developed to centralise authority higher up the management: the plant and factory levels. Still, the urge exists to improve the efficiency and accuracy of the manufacturing process and of the usage of resources during the manufacturing process. Regularly, the allocation of resources for the various steps of the production process is done either manually or, for instance, by using an MRP II system.

The ESPRIT Project 2415 focused on decision making at the appropriate levels and on developing systems to support production planners and operators with their decision making processes. For instance, such systems support production planners and operators when trying to solve conflicts due to events such as machine break-downs.

The Production Management System (PMS) and the Operator Management System (OMS) are used to perform scheduling runs using various dispatching rules and heuristics. Nevertheless, improvements to the schedules generated by PMS or OMS may be desired.

In this thesis, various techniques to fine tune schedules are presented and discussed. To support the PMS and OMS, and hence the users of these systems, these techniques have been implemented in an Expert Management System (EMS). On requests by the PMS or OMS, and initiated by the user, the EMS investigates the indicated orders or jobs that are tardy. If one or more fine tuning technique can be applied to improve the order or job's lead time, advice is generated and returned to the caller, that is either PMS or OMS.
Fine tuning techniques that have been implemented are Overlapping Manufacture, Horizontal Splitting, Vertical Splitting and Alternative Manufacture. The investigation of the applicability of these fine tuning techniques for an order or a job does not involve a scheduling run. Further, these fine tuning techniques exploit resources’ idle periods to improve the lead time of the orders or jobs that are being investigated. As a spin off, the lead time of other orders or jobs may improve as well if the advice is implemented.

The EMS as developed within the ESPRIT Project 2415, has been built using the knowledge engineering environment Delfi3. The knowledge on applying fine tuning techniques has been put into various modules of the EMS. To access the ESPRIT database and to perform calculations on the retrieved data, various programs have been developed using the C programming language.

In this thesis we have shown that it is possible to devise and to build a knowledge-based support system that is able to counsel on how to improve the lead time of a job, that uses idle periods of the resources and that does not need to re-schedule to achieve a lead time improvement.
Samenvatting

De vooruitgang in fabricage en technologie heeft geleid tot de ontwikkeling van fabricagesystemen, die op hun beurt ervoor hebben gezorgd dat het aantal beslissingen op cel en werkplek niveau afgenomen is. Over het algemeen zijn de beslissingsbevoegdheden binnen dergelijke systemen op een hoger niveau binnen het management gecentraliseerd. Toch bestaat nog steeds de behoefte fabricageprocessen en het gebruik van middelen (mankracht, machines, gereedschappen) tijdens fabricageprocessen te verbeteren qua effectiviteit en efficiëntie. Regelmatic worden de middelen die voor bepaalde stappen in een productieproces nodig zijn, handmatig of met behulp van een MRP II systeem geselecteerd.

De aandacht van het ESPRIT Project 2415 was gericht op het brengen van beslissingsbevoegdheden op het juiste niveau en op het ontwikkelen van beslissingenondersteunende systemen voor produktieplanners en operators. Bijvoorbeeld ondersteunend bij het oplossen van problemen door kapotte machines.

Het produktie beheersysteem (PMS) en het operator beheersysteem (OMS) worden gebruikt om werkplannen te maken aan de hand van verschillende selectiecriteria en heuristiek. Hoewel deze systemen goede werkplannen kunnen maken, kan het wenselijk zijn deze werkplannen verder te verbeteren.

In dit proefschrift worden verschillende technieken beschreven en besproken om werkplannen verder te verfijnen. Deze technieken zijn in een ondersteunend kennis systeem (EMS) geïmplementeerd om PMS of OMS te ondersteunen. Hiermee wordt direct de gebruiker van deze systemen ondersteund. Vanuit een aanvraag door PMS of OMS en gestart door de gebruiker, onderzoekt het EMS de bewerkingen van produktieorders die door de gebruiker zijn geselecteerd. Wanneer een of meer verfijningstechnieken kan worden toegepast om de doorlooptijd van een produktie-
order te verbeteren, dan wordt een advies opgesteld en gestuurd aan PMS of OMS.

De verfijningstechnieken Overlapping Manufacture, Horizontal Splitting, Vertical Splitting en Alternative Manufacture zijn geïmplementeerd. Om de toepasbaarheid van deze verfijningstechnieken voor een bepaalde produktieorder te onderzoeken, hoeft niet een nieuw werkplan gemaakt te worden. Deze verfijningstechnieken maken gebruik van de periodes dat machines niet gebruikt worden. Een prettige bijkomstigheid is dat mogelijk andere produktieorders eveneens eerder klaar kunnen zijn wanneer een advies overgenomen wordt.

Het binnen het ESPRIT Project 2415 ontwikkelde EMS, is ontwikkeld met behulp van de kennisstelsel ontwikkelomgeving Delfi3. In verschillende modules van het EMS is kennis opgenomen over het toepassen van verfijningstechnieken. Voor toegang tot gegevens in de ESPRIT database en voor het uitvoeren van berekeningen op deze gegevens, zijn verschillende ondersteunende programma's ontwikkeld met behulp van C.

In dit proefschrift hebben we aangegeven dat het mogelijk is een kennisstelsel te ontwikkelen dat advies levert over het verbeteren van de doorlooptijd van een produktieorder, dat hiervoor de stilstand periodes van de productiemiddelen gebruikt en dat doorlooptijd verbetering verkregen kan worden zonder opnieuw een werkplan te maken.
Curriculum Vitae

Eric-Paul Jansen was born on 7 January 1960 in Vlissingen. After passing his Atheneum-B exam in 1980, he started studying at Delft University of Technology. In 1987, he received his M. Sc. in mathematics with a specialisation in computer science.

During his study, Eric-Paul Jansen worked on the Delfi Project and he was one of the developers of the Delfi2 expert system shell. After working as a student-assistant, he joined the Unilever Research Laboratory in Vlaardingen.

In 1987 at the end of his study, Eric-Paul Jansen started working for Philips Telecommunication and Information Systems. He was project leader for the development of an Account Management Information System and later on for an internal Office Automation project. From 1989 until August 1992, he worked at Delft University of Technology on the ESPRIT Project 2415.

Since August 1992, Eric-Paul Jansen works for CVI. CVI is a full service system house and is a full daughter company of the Dutch Railways. Currently, he works in the business-unit Manufacturing and Logistics as a consultant on innovation and logistics. Eric-Paul Jansen participates in a group that is involved in defining and implementing a Competence Centre for TRITON (logistics software).
Stellingen

behorende bij het proefschrift

Improving Production Schedules Without Scheduling

Eric-Paul Jansen
I. Bij cellular manufacturing zijn cellen aangewezen voor het maken van produkten die deel uitmaken van bepaalde families van produkten. Deze toewijzing vermindert de flexibiliteit om capaciteitsproblemen binnen de cellen op te lossen. Het verdient aanbeveling om in voorkomende gevallen de mogelijkheden te onderzoeken een produktie-order in een andere, niet aangewezen cel te verwerken of een produktie-order te splitsen over gelijkwaardige cellen.
[dit proefschrift]

II. De doorlooptijd voor het maken van een serie van een bepaald artikel, kan soms verkort worden door deze serie voor een bepaalde bewerking te splitsen in subseries op verschillende gelykssoortige machines. Hierbij wordt gebruik gemaakt van de beschikbare capaciteit van deze machines tijdens stilstandperioden.
Wanneer de capaciteit $C_X$ van machine M gedurende stilstandperiode X gebruikt wordt en tot een kortere doorlooptijd leidt, dan kan een latere stilstandperiode $Y$ geen verbetering van deze doorlooptijd opleveren wanneer de capaciteit $C_Y$ gedurende deze periode kleiner of gelijk is aan $C_X$.
[dit proefschrift]

III. Voor de leesbaarheid en voor de overdraagbaarheid naar andere apparatuur, andere programmatuur en andere RDBMSen, moet gebruik worden gemaakt van ESQL (Embedded Structured Query Language) voor het benaderen van deze RDBMSen vanuit derde generatie programmatuur.
[dit proefschrift]

IV. De FNV komt niet op voor de belangen van haar leden, maar opereert slechts uit eigenbelang.
Stakingen hebben bijvoorbeeld in de havens wel geleid tot hogere lonen, maar ook tot hogere kosten en verlies van werkgelegenheid omdat meer en meer uitgeweken wordt naar andere havens.
V. De beleving van een ramp is omgekeerd evenredig met de afstand tot de ramp en de nationaliteit van de slachtoffers.

VI. Discriminatie is niet uit te bannen zolang positieve discriminatie gepredikt wordt.

VII. Word voor Windows is gemaakt voor de gebruiker die op eenvoudige wijze zoveel mogelijk wil doen. WordPerfect is voor toetsvirtuozen en zelfkastigers die op moeilijke wijze zo min mogelijk willen doen.

VIII. De juiste spelling van woorden in de Nederlandse taal wordt beschreven in de *Herziene Woordenlijst van de Nederlandse taal* (het zogenaamde Groene Boekje).
Het is niet alleen raadzaam regelmatig deze woordenlijst te raadplegen om woorden juist te spellen, het is ook raadzaam regelmatig na te gaan wat de laatst uitgegeven druk van deze woordenlijst is.
In de eerste druk (maart 1990) worden woorden als elektronica en elektronisch nog met een c geschreven. In de vijfde druk (april 1992) worden deze woorden met een k geschreven. Woorden als functie en locatie worden nog steeds met een c geschreven.


X. Ook voor de laatste stelling is geen hulp te vinden bij bedrijven die vermeld zijn onder *Stellingen* in de Gouden Gids.