Assignment and scheduling algorithms in automated manufacturing
Assignment and scheduling algorithms in automated manufacturing

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Rijksuniversiteit Limburg te Maastricht, op gezag van de Rector Magnificus, Prof.Mr. M.J. Cohen, volgens het besluit van het College van Dekanen, in het openbaar te verdedigen op vrijdag, 21 februari 1992 om 16.00 uur

door

Frederik Cornelis Rafaël Spieksma

geboren te Leiden in 1964
Promotor : Prof.dr.ir.drs. O.J. Vrieze
Co-promotor : Dr. Y. Crama

Leden van de beoordelingscommissie:
  Prof.dr.ir. A.W.J. Kolen (voorzitter)
  Prof.dr. H.-J. Bandelt
  Prof.dr. J.K. Lenstra
  Prof.dr. S.H. Tijs
  Prof.dr. D. de Werra

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Spieksma, Frits C.R.

ISBN 90-9004828-6
NUGI 811
Subject headings: assignment / scheduling / manufacturing.
Preface

The research described in this thesis has been carried out at the Department of Mathematics of the University of Limburg in Maastricht. I wish to thank here a number of people who contributed to this thesis. First of all, without the support of Yves Crama and Koos Vrieze, this thesis would not have existed at all. I feel privileged that they have guided my efforts to accomplish this thesis.

To illustrate this, a glance through the names of the co-authors of the chapters of this thesis, immediately reveals that the influence of Yves Crama can hardly be overestimated. I never made in vain a request for his time, and I always left his room wiser than I came in; things for which I owe him many thanks.

Also, the support of Koos Vrieze has meant a great deal to me. His optimistic character, his enthusiasm and his interest, not only for scientific matters but for personal matters as well, created an ideal research environment. Moreover, his insightful remarks and suggestions have been an important incentive for this research. Therefore, I feel very much indebted to him.

Thanks are also due to other co-authors: Antoon Kolen, for introducing me to the topic of this thesis and for our work on Chapters 2 and 6, Hans-Jürgen Bandelt, for showing me the benefits of precision and Alwin Oerlemans, for our cooperation.

Furthermore, I wish to thank my colleagues: Maarten Oosten, my roommate, for enduring my moods and for tolerating the mess I make of our room, Jeroen Kuipers and Hans van der Stel for the pleasant discussions during lunch, and Piet Boekhoudt, Jean Derks, Reinoud Joosten, Hans Peters, Frank Thuijsman and Marijke Versteegen, for their support and interest.

I am also very grateful to Marlies Haenen for typesetting this manuscript and for her flexibility to incorporate last-minute changes on numerous occasions.
Then I would like to thank Berry Schuur, who never fails to put into perspective the merits of science in general and mathematics in particular.

Special thanks are for Annemiek Derksen, not only for her moral support, but also for her attentive listening when I practised talks.

Finally, I want to express my deepest gratitude to my parents for the education and the opportunities they gave me.
Contents

Preface

1 Introduction and overview
   1.1 Introduction ................................................. 2
   1.2 Overview of the thesis ..................................... 4
       1.2.1 Throughput rate optimization in the automated assembly of printed circuit boards ............................................. 4
       1.2.2 Approximation algorithms for three-dimensional assignment problems with triangle inequalities ......................... 6
       1.2.3 Approximation algorithms for multidimensional assignment problems with decomposable costs ..................... 7
       1.2.4 Scheduling jobs of equal length: complexity and facets 8
       1.2.5 Minimizing the number of tool switches on a flexible machine ................................................................. 9

2 Throughput rate optimization in the automated assembly of printed circuit boards 11
   2.1 Introduction ............................................... 12
   2.2 Technological environment .................................. 13
   2.3 The throughput rate optimization problem .................. 16
   2.4 Workload balancing ......................................... 18
       2.4.1 Subproblem (A) ......................................... 18
       2.4.2 Subproblem (B) ......................................... 22
   2.5 Scheduling of individual machines ......................... 24
       2.5.1 Subproblem (C) ......................................... 25
       2.5.2 Subproblem (D) ......................................... 27
       2.5.3 Subproblem (E) ......................................... 30
       2.5.4 Subproblem (F) ......................................... 32
   2.6 An example .................................................. 35
3 Approximation algorithms for three-dimensional assignment problems with triangle inequalities 41
  3.1 Introduction ................................................. 42
  3.2 Complexity of $TΔ$ and $SΔ$ .................................. 43
  3.3 Approximation algorithms .................................... 45
  3.4 Computational results ...................................... 52

4 Approximation algorithms for multidimensional assignment problems with decomposable costs 57
  4.1 Introduction .................................................. 58
  4.2 Some decomposable cost functions .......................... 60
  4.3 Approximation algorithms and their performance .......... 61
  4.4 Sum costs ..................................................... 66
  4.5 Star costs ..................................................... 71
  4.6 Tour costs ..................................................... 73
  4.7 Tree costs ..................................................... 77
  4.8 Recursive heuristics ....................................... 78

5 Scheduling jobs of equal length: complexity and facets 87
  5.1 Introduction .................................................. 88
  5.2 Complexity of SEL .......................................... 89
  5.3 A partial polyhedral description of SEL ................. 93
  5.4 A cutting-plane algorithm for SEL ....................... 103

6 Minimizing the number of tool switches on a flexible machine 107
  6.1 Introduction ................................................ 108
  6.2 Basic results ............................................... 111
    6.2.1 $\mathcal{NP}$-hardness results .......................... 111
    6.2.2 Finding the minimum number of setups for a fixed job sequence .......... 113
  6.3 Heuristics .................................................. 117
    6.3.1 Traveling salesman heuristics .......................... 117
    6.3.2 Block minimization heuristics .......................... 118
    6.3.3 Greedy heuristics ..................................... 120
    6.3.4 Interval heuristic .................................... 121
    6.3.5 2-Opt strategies .................................... 122
    6.3.6 Load-and-Optimize strategy ............................ 123
  6.4 Computational experiments .................................. 124
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4.1</td>
<td>Generation of problem instances</td>
<td>124</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Computational results</td>
<td>125</td>
</tr>
<tr>
<td>6.5</td>
<td>Lower bounds</td>
<td>131</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Traveling salesman paths</td>
<td>131</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Structures implying extra setups</td>
<td>132</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Valid inequalities</td>
<td>134</td>
</tr>
<tr>
<td>6.5.4</td>
<td>Lagrangian relaxation</td>
<td>135</td>
</tr>
<tr>
<td>Appendix</td>
<td>Graph-theoretic definitions</td>
<td>137</td>
</tr>
</tbody>
</table>

References 139

Samenvatting 145

Curriculum vitae 149
Chapter 1

Introduction and overview
1.1 Introduction

In the past decade, automated manufacturing has been gradually adopted by most industries. This technology has an impact on all aspects of manufacturing (for an elaborate discussion on this subject, the reader is referred to Groover and Zimmers (1984) and Hartley (1984)). By means of illustration, the availability of computer numerically controlled machines makes it possible to produce a variety of parts with virtually no setup costs, as already described in Cook (1975). Other potential benefits include shorter lead times and higher machine utilization (see, among others, Van Looveren, Gelders and Van Wassenhove (1986) and Van Vliet and Van Wassenhove (1989)). Also, automation increases the flexibility to start manufacturing new products. This is important in view of today's fast changing demand and short product-cycles (Jaikumar (1986)).

Evidently, automated manufacturing offers opportunities. However, it is in general a difficult task to make an efficient use of the newly offered possibilities. Indeed, the planning of a system which consists of a number of connected, complicated machines using limited resources can constitute quite a challenge.

To be more specific, consider an automated system manufacturing discrete parts. The planner has to decide:

- at which moments parts are allowed to enter the system;
- which tools are attached to which machines during which period so that required operations can be performed;
- which operations are performed by which machines;
- how to allocate scarce resources such as pallets and fixtures.

These questions belong to the so-called system setup problem, i.e. those decisions that have to be made before the system can start producing (see Kiran and Tansel (1988) and Spieksma, Vrieze and Oerlemans (1990)). Of course, depending on the specific situation, some of the issues in this system-setup problem might be irrelevant or trivial. However, in general these decisions determine to a great extent whether the potential advantages of automated manufacturing will be achieved or not.
Section 1.1

A typical process where automated manufacturing gives rise to a complex planning problem, can be found in the electronics industry. Printed circuit boards (PCBs) are manufactured by numerically controlled machines. In general terms, the assembly process for PCBs can be described as follows (see e.g. Chapter 2 of this thesis, Ball and Magazine (1988) and Van Laarhoven and Zijm (1990)).

A line of machines is devoted to the assembly of several types of PCBs. An automated transport band is used to carry each PCB from one machine to the next. The assembly of an individual PCB consists of inserting electronic components of prespecified types into prescribed locations on the board. In order to handle the components, each machine is equipped with a device called its arm. This arm picks components from so-called feeders, moves to the appropriate locations, inserts the components into the board and moves back to the feeders to pick new components. These feeders deliver components of a certain type (one type per feeder) and they are attached to the machine by means of so-called slots. Each machine has a row of slots available; a feeder occupies 1, 2 or even more adjacent slots (see Figure 1.1). A sequence of operations consisting of picking components from feeders moving to the appropriate locations, and inserting them into the board is called a pick-and-place round.

Figure 1.1
It is assumed that the main objective of a PCB-manufacturer is to maximize the throughput rate, i.e. the number of PCBs produced per time-unit. These rates appear to be quite sensitive to the quality of the process planning decisions, and it is therefore important to have good optimization programs available. On the other hand, the assembly of a typical PCB implies the insertion of hundreds of components by complex machines. Thus, optimizing the throughput rates for several types of PCBs, on a line of several placement machines constitutes a very difficult planning problem. In fact, the sheer size of the problem seems to prohibit the development of an exact algorithm. These observations have led several researchers to tackle the PCB throughput rate optimization problem by means of a hierarchical approach. In such decomposition schemes a number of hierarchically coupled subproblems are identified, each of which is easier to solve than the global problem. By solving these subproblems consecutively, a solution to the global problem can be found. Of course, this methodology does not guarantee global optimality of the final solution, even when all subproblems are solved to optimality. Nevertheless, such hierarchical approaches have previously proven to deliver good quality solutions to similar hard problems (e.g. in VLSI design, see Korte (1989)).

The purpose of this thesis is to develop and analyze algorithms for certain classes of combinatorial optimization subproblems arising in decomposition approaches to the PCB throughput rate optimization problem.

In the next section, a brief description of the contents of the thesis is given chapter by chapter. Notice that each chapter (except this one) is written in the form of an article, and can be read independently of the other ones.

1.2 Overview of the thesis

1.2.1 Throughput rate optimization in the automated assembly of printed circuit boards

In Chapter 2, a specific PCB assembly system is considered. It is assumed there that only one type of PCB is to be assembled. Also, in addition to the general description of PCB assembly systems given in Section 1.1, two more features of the machines are worth mentioning. First, in the system under study the arm of each machine has three heads. Each head can carry one component at the time. Consequently, in one pick–and–place round three components are inserted in the board. Secondly, in order to be able to
collect a component from a feeder, a head of the arm of the machine must be provided with some tools or equipments. Every component type can only be handled by a restricted set of alternative equipments. For more details concerning this PCB assembly system, the reader is referred to Chapter 2.

We propose to decompose the resulting planning problem into the following, hierarchically coupled, subproblems:

(A) determine how many components each machine should insert, and with what equipment;

(B) assign feedertypes to machines;

(C) determine which components each head should insert into the board;

(D) cluster the locations into subsets of size at most three, to be processed in one pick–and–place round;

(E) determine the sequence of pick–and–place operations to be performed by each machine;

(F) assign the feeders to the slots.

Subproblems (A) and (B) determine the workload of each machine. Obviously, the objective here is to minimize the maximum workload over all machines (since this is equivalent to maximizing the throughput rate). The remaining subproblems (C)–(F) deal with the scheduling of individual machines.

In Chapter 2, we model more precisely each of the subproblems (A)–(F), and we develop heuristic approaches for their solution. Let us now briefly summarize these contributions.

For subproblem (A), an integer programming model is constructed. Thanks to the aggregation of certain types of components, the number of integer variables in this model is limited (< 80). A two-phase approach, based on relaxing the integrality requirements on some variables and fixing other variables, enables us to find good solutions to this model, that is solutions for which the estimated bottleneck workload is low. Subproblem (B) is modelled as a pure fixed-charge transportation problem (see Fisk and Mckown (1979)). A simple heuristic is described to solve it.

As mentioned earlier, the next subproblems are solved separately for each machine. First, in subproblem (C), the components are assigned to the heads so as to minimize the number of pick–and–place rounds. A model and a heuristic similar to those used for subproblem (A) are developed to tackle
this problem. Next, the locations have to be clustered into subsets which can be processed in each pick-and-place round (subproblem (D)). This is done in such a way that the travel time within clusters is minimized. This problem is modeled as a specially structured three-dimensional assignment problem. It is solved using a heuristic based on the solution of some related "two-dimensional" assignment problems (that is bipartite matching problems). Much more will be said about this particular model in Chapter 3, and in Chapter 4 about some of its higher dimensional generalizations.

Subproblem (E) asks for an optimal sequence of the pick-and-place operations, consistent with the rounds determined in subproblem (D). It turns out that this decision is highly dependent on the assignment of feeders to feederslots (i.e. on the solution of subproblem (F)). Therefore, as suggested by Leipälä and Nevalainen (1989), the following approach is adopted. First, subproblem (E) is heuristically solved; using this solution as input, a solution of subproblem (F) is computed, which in turn is used to revise the previous solution of subproblem (E), and so on. Subproblem (E) can be reduced to the shortest Hamiltonian path problem. Some well-known heuristics (nearest neighbor, farthest insertion) are used to solve this problem. Subproblem (F) is modeled as a quadratic assignment problem. Local improvement techniques are implemented for its solution; shifting a feeder or exchanging two feeders are the main ingredients of this method. The study of subproblem (F) originally motivated the research presented in Chapter 5.

The performance of our approach is tested on a problem instance described in CQM (1988): 258 components of 39 types have to be inserted in each PCB by a line of three machines. The bottleneck workload reported in CQM (1988) for the actual operation sequence implemented by the plant equals 81 seconds. Using the approach described here, the bottleneck workload could be reduced to 66 seconds. More details can be found in Chapter 2.

1.2.2 Approximation algorithms for three-dimensional assignment problems with triangle inequalities

Chapter 3 in this thesis is concerned with subproblem (D) from the decomposition described in Subsection 1.2.1. This problem is a special case of the three-dimensional assignment problem (3DA), which can be described as follows (see also Balas and Saltzman (1991)). Given are three disjoint n-sets of points, and nonnegative costs associated with every triple consisting of exactly one point from each set. The problem is to find a minimum-cost
collection of $n$ triples covering each point exactly once.

In subproblem (D), the three disjoint point-sets correspond to the locations where components have to be inserted by the first, second or third head respectively. The cost of a triple reflects the travel time of the arm between the corresponding locations.

Instances of subproblem (D) are specially structured instances of 3DA in the sense that the cost of each triple is determined by a distance defined on the set of all points and satisfying the triangle inequality. We call $T\Delta$ the special case of 3DA where the cost of a triple is equal to the sum of the distances between its points, and $S\Delta$ the case where the cost of a triple is equal to the sum of the two smallest distances between its points.

(Notice that instances of $T\Delta$ and $S\Delta$ also arise in the scheduling of teaching practices at colleges of education, see Frieze and Yadegar (1981)).

We prove in Chapter 3 that $T\Delta$ as well as $S\Delta$ are $\mathcal{NP}$-hard problems. For both $T\Delta$ and $S\Delta$, we present two polynomial-time heuristics based on the solution of a small number (either two or six) of related two-dimensional assignment problems. We prove that these heuristics always deliver a feasible solution whose cost is at most $\frac{3}{2}$, resp. $\frac{4}{3}$ of the optimal cost. Computational experiments indicate that the performance of these heuristics is excellent on randomly generated instances of $T\Delta$ and $S\Delta$.

1.2.3 Approximation algorithms for multidimensional assignment problems with decomposable costs

The problems considered in Chapter 4 are a straightforward generalization of the problems $T\Delta$ and $S\Delta$ described in Subsection 1.2.2. More specifically, given are now $k$ disjoint sets of points, $k \geq 3$, and a distance between every two points in different sets. Call a clique any subset of $k$ points, consisting of one point from each set. The cost of a clique is a function of the distances induced by the points in the clique. The problem is to find a minimum cost partition of the set of points into cliques. (When the cost of a clique is given by an arbitrary number, the problem is known as the $k$-dimensional assignment problem; applications of this problem are described in Pierskalla (1968)). In Chapter 4, four specific cost functions are considered; namely, the cost of a clique is either the sum of the distances induced by the clique (sum costs), or the minimum length of a spanning star (tour costs) or of a traveling salesman tour (tour costs), or of a spanning tree (tree costs) of the induced subgraph.

Several heuristics for these problems are proposed, and worst-case bounds
are derived on the ratio between the cost of the solutions produced by the heuristics and the cost of an optimal solution. All worst-case bounds are stated in terms of \( k \) and \( r \), where the parameter \( r \) indicates how close the distance function comes to satisfying the triangle inequality. In particular, when the distance function satisfies the triangle inequality, we prove the existence of a 1-approximate polynomial-time algorithm, in the case of sum costs and star costs (that is, of a heuristic which always returns a solution whose cost is at most twice the cost of an optimal assignment).

1.2.4 Scheduling jobs of equal length: complexity and facets

Chapter 5 is devoted to the following problem. Given are \( n \) jobs which have to be processed on a single machine within a fixed timespan \( \{1, 2, \ldots, T\} \). The processing time, or length, of each job equals \( p, p \in \mathbb{N} \). The processing cost of each job is an arbitrary function of its start-time. The problem is to schedule all jobs so as to minimize the sum of the processing costs. We refer to this problem as problem SEL (Scheduling jobs of Equal Length).

We prove that SEL is strongly \( \mathcal{NP} \)-hard already for \( p = 2 \) and \( c_{jt} \in \{0, 1\} \). On the other hand, if the number of time-units equals \( n \cdot p + c \), where \( c \) denotes a constant, then the problem is shown to be polynomially solvable. We also study a 0-1 programming formulation of SEL from a polyhedral point of view. In particular, we show that all facets defined by set-packing inequalities have been previously listed by Sousa and Wolsey (1989). Two more classes of facet-defining inequalities (one of them exponentially large) are derived. The separation problem for these inequalities is solvable in polynomial time.

It should be noted that SEL is a special case of a very general scheduling problem, say problem \( S \), considered by Sousa and Wolsey (1989), where the jobs may have arbitrary, usually distinct, processing times. It is an easy observation that, if \( \{1, \ldots, n\} \) is any subset of the jobs occurring in \( S \), and all jobs in \( \{1, \ldots, n\} \) have the same length \( p \), then any valid inequality for SEL is also valid for \( S \). This suggests that the polyhedral description presented in Chapter 5 may prove useful, not only when all jobs have strictly equal length, but also in cases where the number of distinct lengths is small.

SEL is strongly related to subproblem (F) in the decomposition described in Subsection 1.2.1. This can be seen as follows: each feeder \( j \) requires a certain number of slots, say \( p_j \), depending on the feeder type; usually, \( p_j \) only takes a small number of values, say \( p_j \in \{1, 2, 3\} \). In order to maximize the throughput rate, it is desirable to position the feeders close to
the locations where the corresponding components must be inserted. More precisely, for each combination of feeder $j$ and slot $t$ a cost coefficient can be computed which captures the cost of assigning feeder $j$ to slots $t$, $t + 1$, \ldots $t + p_j - 1$. It follows that finding a minimum-cost assignment of feeders to slots is equivalent to solving a scheduling problem where the number of distinct processing times is small.

### 1.2.5 Minimizing the number of tool switches on a flexible machine

The problem studied in Chapter 6 can be described as follows. A number of jobs have to be successively processed on a single flexible machine. Each job requires a subset of tools which have to be placed in the tool magazine of the machine before the job can be processed. The tool magazine has a limited capacity, and, in general the number of tools needed to produce all the jobs exceeds this capacity. Hence, it is sometimes necessary to change tools between two jobs in a sequence. We call a switch the combination of a tool-removal and a tool-insertion. The problem is then to determine a job sequence and an associated sequence of loadings for the tool magazine, such that the total number of tool switches is minimized.

The previous formulation, in terms of flexible machines and tools, is due to Bard (1988) and Tang and Denardo (1988). Bard (1988) also mentions another interesting application of this model to PCB assembly. He considers a situation where several types of PCBs are to be produced by the same assembly system (see Section 1.1) and where the time needed to load a feeder into a slot is significantly high. Then it becomes important to find a sequence of the different board types which minimizes the number of “feedersetups”. This problem is equivalent to the toolswitching problem described above.

We prove that the problem of determining whether there exists a sequence of jobs such that each tool is loaded only once, is \textit{NP}-hard. Also, minimizing the number of tool switches is already \textit{NP}-hard for a fixed capacity equal to 2. On the other hand, given a sequence of jobs, there exists a linear time algorithm to find the optimal sequence of tool loadings. A new proof of this result is presented. Finally, we propose several heuristics whose performance is computationally assessed.
Chapter 2

Throughput rate optimization in the automated assembly of printed circuit boards

This chapter is joint work with Yves Crama, Antoon W.J. Kolen and Alwin G. Oerlemans; it has been published as an article in Annals of Operations Research 26, 455-480, 1990.
2.1 Introduction

The electronics industry relies heavily on numerically controlled machines for the placement of electronic components on the surface of printed circuit boards (PCB). These placement (or mounting, or pick-and-place) machines automatically insert components into PCB’s, in a sequence determined by the input program. The most recent among them are characterized by high levels of accuracy and speed, but their throughput rates still appear to be extremely sensitive to the quality of the instructions. On the other hand, the effective programming of the machines becomes steadily more difficult in view of the increasing sophistication of the available technology. The development of optimization procedures allowing the efficient operation of such placement machines therefore provides an exciting challenge for the operations research community, as witnessed by, e.g., the recent papers by Ahmadi, Grotzinger and Johnson (1988), Ball and Magazine (1988), and Leipäla and Nevalainen (1989).

In this chapter we propose a hierarchical approach to the problem of optimizing the throughput rate of a line of several placement machines devoted to the assembly of a single product. As usual in the study of flexible systems, the high complexity of the problem suggests its decomposition into more manageable subproblems, and accepting the solution of each subproblem as the starting point for the next one. Of course, this methodology cannot guarantee the global optimality of the final solution, even assuming that all subproblems are solved to optimality. This is even more true in the present case, where most subproblems themselves turn out to be $\mathcal{NP}$-hard, and hence can only be approximately solved by heuristic procedures. Nevertheless, such hierarchical approaches have previously proved to deliver good quality solutions to similarly hard problems (e.g. in VLSI-design; see Korte (1989)). They also offer the advantage of providing precise analytical models for the various facets of the global problem (see, for example, Buzacott and Yao (1986) for a discussion of analytical models in FMS).

Our approach has been tested on some industrial problems, but more experimentation would be required in order to precisely assess the quality of its performance and its range of applicability. In particular, as pointed out by one of the referees, the validity of some of our models is conditioned by the validity of some exogenous assumptions about the nature of instances “coming up in practice” (see, for instance, Subsection 2.4.1). Even though these assumptions were fulfilled in the industrial settings that motivated our study, they may well fail to be satisfied in other practical situations. This
Section 2.2

would then invalidate the use of the corresponding models. However, we believe that the hierarchical scheme and most of the techniques presented in this chapter would nevertheless remain applicable for a wide range of problem instances.

We now give a brief outline of the chapter. The next section contains a more detailed description of the technological environment, and Section 2.3 provides a precise statement of the problem and a brief account of previous related work. Sections 2.4 and 2.5 present our approach to the solution of the throughput rate optimization problem. Section 2.4 addresses the workload balancing problem for the line of machines, and Section 2.5 deals with the optimal sequencing of operations for individual machines. Both sections present mathematical models and heuristic solution methods for the various subproblems arising in our decomposition of the global problem. Finally, in Section 2.6 we describe the results supplied by our approach on a practical problem instance.

2.2 Technological environment

In this chapter, we are concerned with the automated assembly of a number of identical PCB's. For our purpose, the assembly of a PCB consists of the insertion of electronic components of prespecified types (indexed by $1, \ldots, T$) into prespecified locations (indexed by $1, \ldots, N$) on a board. Prior to operations, the components of different types are collected on different feeders (one type per feeder). Feeders are used by the placement machines as described below. We denote by $N_t$ the number of components of type $t$ ($t = 1, \ldots, T$). So, $N = \sum_{t=1}^{T} N_t$.

We assume that a line of $M$ placement machines is devoted to the assembly of the PCB's. The machines we have in mind are of the CSM (Component Surface Mounting) family. They feature a worktable, a number $S$ of feeder slots, and three pick-and-place heads (see Figure 2.1).
Figure 2.1 Schematic representation of a placement machine.

The PCB is carried from one machine to the next by an automatic transportband until it comes to rest on the worktable. It stays motionless during the mounting operations.

The feeder slots are fixed to two opposite sides of the worktable, $S/2$ of them on each side. The feeders containing the components to be placed by the machine must be loaded in the slots before the mounting begins. Depending on its type, each feeder may require 1, 2, or even more adjacent slots.

The pick-and-place heads are numbered from 1 to $3M$, with heads $3m - 2$, $3m - 1$ and $3m$ on machine $m$ (but, for short, we shall also refer to heads 1, 2 and 3 of each machine). They are fixed along a same arm which always remains parallel to the side of the worktable supporting the feeder slots. The arm can move in a horizontal plane above the worktable. It can perform vertical moves to allow the heads to pick components from the feeders or to insert components into the board.

Each head can carry at most one component at a time. It must be equipped with certain tools (chucks and nozzles) before it can handle any components. The collection of tools necessary to process a given component we call equipment. With every component type is associated a restricted set
of alternative equipments by which it can be handled. In most situations, four or five equipments suffice to mount all component types. Changing the equipment of a head can be done either manually or automatically, depending on the technology (notice that, on certain types of machines, an equipment change can be performed automatically for heads 1 and 2, but only manually for head 3). In either case, an equipment change is a time-consuming operation.

Consider now a typical pick-and-place operation, during which the machine must place components of types \(i\), \(j\) and \(k\) using heads 1, 2 and 3, respectively. Suppose, for instance, that these components are to be placed in the order \(j\), \(i\), \(k\). Such an operation can be decomposed as follows. First, the arm moves until head 1 is positioned above a feeder of type \(i\). Head 1 picks then a component \(i\). Two more moves of the arm between the feeder slots allow heads 2 and 3 to pick components \(j\) and \(k\). Next, the arm brings head 2 above the location where component \(j\) is to be placed, and the insertion is performed. The same operation is subsequently repeated for \(i\) and finally for \(k\).

Some remarks are in order concerning such a pick-and-place round. Notice that the picking of the components must always be performed by head 1 first, then by head 2, then by head 3 (of course, we may decide in some rounds to use only one or two heads), whereas an arbitrary sequence may be selected for their placement. Once a head has been positioned by the arm above the required feeder slot or location, the time needed to pick or to place the corresponding component depends on the type of the component, but is otherwise constant. Thus, on one machine, the only opportunities for a reduction of the total pick-and-place time reside in a clever sequencing of the operations and assignment of the feeders to feeder slots.

We have intentionally omitted many details in this brief description of the placement machines and of their functioning. For example, the insertion heads have to rotate to a predetermined angle before picking or placing components; some feeder slots or board locations are unreachable for certain heads; heads may be unavailable (e.g. broken) or may be assigned fixed equipments; the arm can only move in a limited number of directions; etc.

Some of these features (unreachable locations, unavailable heads, etc.) can be easily introduced in our models by setting variables to fixed values, thus resulting in a simplification of these models. Others will be implicitly incorporated in the models. For instance, parameters of the models such as the pick-and-place time or the travel time between board locations will be assumed to take into account the rotation of the heads and the restricted
moves of the arm. Of course, there remains a possibility that these characteristics could be exploited explicitly to improve the performance of the machines, but we did not attempt to do so.

2.3 The throughput rate optimization problem

With this description of the technological constraints, we can now state a global throughput rate optimization problem as follows. Given the specifications of a PCB and of $M$ placement machines, determine:

(1) an assignment of the components to the $M$ machines;

(2) for each machine, an assignment of feeders to feeder slots;

(3) for each machine, a sequence of pick-and-place rounds, each round consisting itself of a sequence of at most three component locations among those assigned to the machine in step (1);

(4) for each machine and for each pick-and-place round, an assignment of equipment to heads.

These decisions are to be made so that the PCB can be mounted using all $M$ machines, and so as to minimize the processing time on the bottleneck machine (i.e., the machine with the longest processing time).

In our solution of this problem, we shall also take into account a secondary criterion, dictated by cost considerations. Because feeders are rather expensive, it appears desirable (at least, in the practical situations that we encountered) to minimize the total number of feeders used. Ideally, thus, all components of a same type should be processed by one machine. We shall show in Subsection 2.4.2 how this criterion can be accommodated.

This formulation of the throughput rate optimization problem is patterned after a (confidential) report of the Philips Center for Quantitative Methods (CQM (1988); see also Van Laarhoven and Zijm (1990)). This report proposes a hierarchical decomposition of the problem, and heuristics for the resulting subproblems. Our decomposition, as well as all heuristics presented in the next two sections, are different from CQM's. Our heuristics, in particular, rely more explicitly on the precise mathematical modeling of the subproblems.

The throughput rate optimization problem is also mentioned by Ball and Magazine (1988), under somewhat simpler technological conditions. In
Section 2.3

particular, each machine has but one pick-and-place head. The authors
investigate in detail only the sequencing of pick-and-place operations over
one machine (i.e., our step (3) above).

Leipälä and Nevalainen (1989) discuss our steps (2) and (3), for a different
type of one-head machines.

Ahmadi et al. (1988) consider the case of one machine featuring two
heads. They address subproblems (2), (3) and (4), but their technological
constraints are very different from ours, and their models do not seem to be
directly applicable in our framework.

In the next two sections we describe our approach to the throughput
rate optimization problem. This approach is based on a decomposition of
the global problem into the following list of subproblems (which thus refines
the original formulation (1)–(4) given before):

(A) determine how many components each machine must mount, and with
what equipments;

(B) assign feeder types to machines;

(C) determine what components each head must mount;

(D) cluster the locations into subsets of size at most three, to be processed
in one pick-and-place round;

(E) determine the sequence of pick-and-place operations to be performed
by each machine;

(F) assign the feeders to feeder slots.

Subproblems (A) and (B) in this list answer together question (1) and part
of question (4) above. Our main concern in solving these two subproblems
will be to achieve an approximate balance of the workload over the line of
machines. This will be done in Section 2.4.

Subproblems (C), (D), (E), (F) address the scheduling of individual ma-
chines, and are dealt with in Section 2.5.

In our computer experiments, the sequence of subproblems (A)–(F) is
solved hierarchically in a single pass (except for (E) and (F); see Section 2.5).
It may be possible to use an iterative solution procedure, and to exploit the
solution of certain subproblems in order to revise previous ones. We have
not further explored these possibilities.
2.4 Workload balancing

2.4.1 Subproblem (A)

The model

We proceed in this phase to a preliminary distribution of the workload over the machine line, based on the number of equipment changes for each head and on a rough estimate of the time needed to mount each component. The latter estimate is computed as follows.

In Section 2.1, we have seen that the time needed to mount a component of type \( t = 1, \ldots, T \) consists of two terms: a variable term measuring the travel time of the head, and a constant term \( p_t \) representing the total time spent to pick the component when the head is directly above feeder \( t \), plus the time to place the component when the head is above the desired location.

Let now \( v_t \) be an estimate of the first variable term; then, \( v_t + p_t \) is an estimate of the mounting time required by each component of type \( t \). Notice that, in practice, a reasonable value for \( v_t \) does not appear too difficult to come by, e.g. by evaluating the average time required for the arm to travel from feeder slots to mounting locations. The solution of the model given below does not appear to be very sensitive to the exact value of \( v_t \).

(In our computer experimentations, we used a constant value \( v \) for all \( v_t \), \( t = 1, \ldots, T \).) Otherwise, solving the model for a few alternative values of \( v_t \) \( (t = 1, \ldots, T) \) provides different initial solutions for the subsequent phases of the procedure. If necessary, after all subproblems \((A)-(F)\) have been solved, a solution to the global problem can be used to adjust the values \( v_t \) and reiterate the whole solution procedure.

Define now two component types to be equivalent if the quantity \( v_t + p_t \) is the same for both types, and if both types can be handled by precisely the same equipment. This relation induces a partition of the set of components into \( C \) classes, with each class containing components of equivalent types.

We are now almost ready to describe our model. We first introduce a few more parameters:

\[
\begin{align*}
Q & = \text{number of available equipments; } \\
& \quad \text{for } c = 1, \ldots, C, \\
B_c & = \text{number of components in class } c; \\
\omega_c & = \text{common value of } v_t + p_t \text{ for the types represented in class } c; \\
Q(c) & = \text{set of equipment types which can handle the components in class } c;
\end{align*}
\]
Section 2.4

for \( h = 1, \ldots, 3M \),
\[
E_h = \text{time required by an equipment change for head } h.
\]

The decision variables are: for \( c = 1, \ldots, C \), for \( m = 1, \ldots, M \), for \( h = 1, \ldots, 3M \), for \( q = 1, \ldots, Q \):

\[
\begin{align*}
\alpha_{cm} & = \text{number of components of class } c \text{ to be mounted by machine } m; \\
\alpha_{mq} & = 1 \text{ if machine } m \text{ uses equipment } q; \\
& = 0 \text{ otherwise}; \\
\alpha_h & = \text{number of equipment changes required for head } h; \\
W & = \text{estimated workload of the bottleneck machine.}
\end{align*}
\]

The optimization model for subproblem (A) is:

\[
(M_A) \quad \text{minimize } W
\]

\[
\text{subject to } \sum_{m=1}^{M} \alpha_{cm} = B_c \quad (c = 1, \ldots, C), \quad (2.1)
\]

\[
\alpha_{cm} \leq B_c \sum_{q \in Q(c)} \alpha_{mq} \quad (c = 1, \ldots, C; \\
\alpha_{mq} \leq 3m \quad m = 1, \ldots, M), \quad (2.2)
\]

\[
\sum_{q=1}^{Q} \alpha_{mq} \leq \sum_{h=3m-2}^{3m} \alpha_h + 3 \quad (m = 1, \ldots, M), \quad (2.3)
\]

\[
W \geq \sum_{c=1}^{C} w_c \alpha_{cm} + \sum_{h=3m-2}^{3m} E_h \alpha_h \quad (m = 1, \ldots, M), \quad (2.4)
\]

\[
\alpha_{cm} \geq 0 \text{ integer } \quad (c = 1, \ldots, C; \\
\alpha_{mq} \geq 0 \text{ integer } \quad (m = 1, \ldots, M), \quad (2.5)
\]

\[
\alpha_{mq} \in \{0, 1\} \quad (m = 1, \ldots, M; \\
\alpha_h \geq 0 \text{ integer } \quad (h = 1, \ldots, 3M). \quad (2.6)
\]

Constraints (2.1) express that all components must be mounted. Constraints (2.2) ensure that machine \( m \) is assigned at least one of the equipments in \( Q(c) \) when \( \alpha_{cm} \) is nonzero. Constraints (2.3) together with (2.4), (2.7) and the minimization objective, impose that the number of equipment changes on each machine be equal to the number of equipments used minus three, or to zero if the latter quantity becomes negative. The right-handside of (2.4) evaluates the processing time on machine \( m \) (we assume here that
the time needed to bring a new PCB on the worktable, after completion of
the previous one, is always larger than the time required for an equipment
change. Thus, at the optimum of \((M_A)\), \(W\) is equal to the maximum of
these processing times.

Two comments are in order concerning this model. First, we could have
formulated a similar model using variables \(z_{km}\) instead of \(z_{im}\), with the
index \(k\) running over all component locations, from 1 to \(N\). The advantage
of aggregating the components into classes is that the number of variables is
greatly reduced, and that some flexibility remains for the exact assignment
of operations to heads. This flexibility will be exploited in the solution of
further subproblems. Second, observe that we do not impose any constraint
on the number of feeder slots required by a solution of \((M_A)\). This could,
in principle, be done easily, e.g. as in the partitioning model of Ahnadi et
al. (1988), but requires the introduction of a large number of new variables,
resulting again from the disaggregation of classes into types. From a practical
point of view, since we always allocate at most one feeder of each type per
machine (remember the secondary criterion expressed in Section 2.3), the
number of feeder slots never appears to be a restrictive factor; hence the
solutions of \((M_A)\) are implementable.

In practice, the number of equipments needed to mount all components is
often smaller than the number of heads available. When this is the case, we
can in general safely assume that no change of equipments will be performed
in the optimal solution of \((M_A)\) (since \(E_h\) is very large). We may then
replace \((M_A)\) by a more restrictive model, obtained by fixing \(r_h = 0\) for
\(h = 1,\ldots, 3M\).

Complexity and solution of model \((M_A)\)

Every instance of the well-known set-covering problem can be polynomially
transformed to an instance of \((M_A)\) with \(M = 1\), which implies that model
\((M_A)\) is already \(\mathcal{NP}\)-hard when only one machine is available (we assume the
familiarity of the reader with the basic concepts of complexity theory; see, for
example, Garey and Johnson (1979) or Nemhauser and Wolsey (1988); the
proofs of all the complexity results can be found in Crama, Kolen, Oerlemans
and Spieksma (1989)).

In spite of this negative result, obtaining solutions of good quality for
\((M_A)\) turns out to be easy in practical applications. To understand this
better, notice that the number of variables in these applications is usually
small. The real-world machine line which motivated our study features three
machines. A typical PCB may require the insertion of a few hundred components, but these fall into five to ten classes. The number of equipments needed to mount the board (after deletion of a few clearly redundant ones) seems rarely to exceed five. So, we have to deal in \( (M_A) \) with about 10 to 30 zero-one variables and 15 to 50 integer variables.

In view of these favorable conditions, we take a two-phase approach to the solution of \( (M_A) \). In a first phase, we consider the relaxation of \( (M_A) \) obtained by omitting the integrality requirement on the \( x \)-variables (in constraints (2.5)). The resulting mixed-integer program is easily solved by any commercial branch-and-bound code (one may also envision the development of a special code for this relaxed model, but this never appeared necessary in this context).

In the second phase, we fix all \( r \)- and \( z \)-variables of \( (M_A) \) to the values obtained in the optimal solution of the first phase. In this way we obtain a model of the form:

\[
(M'_A) \quad \text{minimize} \quad W \\
\text{subject to} \quad \sum_{m=1}^{M} x_{cm} = B_c \quad (c = 1, \ldots, C), \\
W \geq \sum_{c=1}^{C} u_c x_{cm} + W_m \quad (m = 1, \ldots, M), \\
x_{cm} \geq 0 \text{ integer} \quad (c = 1, \ldots, C; m = 1, \ldots, M),
\]

where some variables \( x_{cm} \) are possibly fixed to zero (by constraints (2.2) of \( (M_A) \)), and \( W_m \) is the total time required for equipment changes on machine \( m \) \( (m = 1, \ldots, M) \).

In practice, model \( (M'_A) \) is again relatively easy to solve (even though one can show by an easy argument that \( (M'_A) \) is \( \mathcal{NP} \)-hard). If we cannot solve it optimally, then we simply-round up or down the values assumed by the \( x \)-variables in the optimal solution of the first phase, while preserving equality in the constraints (2.1).

In our implementation of this solution approach, we actually added a third phase to the procedure. The goal of this third phase is twofold: 1) to improve the heuristic solutions found in the first two phases; 2) to generate alternative "good" solutions of \( (M_A) \), which can be used as initial solutions for the subsequent subproblems of our hierarchical approach.

Two type of ideas are applied in the third phase. On the one hand, we
modify "locally" the solutions delivered by phase 1 or 2, e.g. by exchanging the equipments of two machines, or by decreasing the workload of one machine at the expense of some other machine. On the other hand, we slightly modify model \((M_A)\) by imposing an upperbound on the number of components assigned to each machine, and we solve this new model.

Running the third phase results in the generation of a few alternative solutions associated with reasonable low estimates of the bottleneck workload.

2.4.2 Subproblem \((B)\)

The model

At the beginning of this phase, we know how many components of each class are to be mounted on each machine, i.e. the values of the variables \(x_{cm}\) in model \((M_A)\). Our goal is now to disaggregate these figures and to determine how many components of each type must be handled by each machine. The criterion to make this decision will be the minimization of the number of feeders required (this is the secondary criterion discussed in Section 2.3).

So, consider now an arbitrary (but fixed) class \(c\). Reorder the types of the components so that the types of the components contained in class \(c\) are indexed by \(t = 1, \ldots, R\). Recall that \(N_t\) is the total number of components of type \(t\) to be placed on the board for all \(t\). To simplify our notations, we also let \(X_m = x_{cm}\) denote the number of components of class \(c\) to be mounted by machine \(m\). So, \(\sum_{t=1}^{R} N_t = \sum_{m=1}^{M} X_m = B_c\). We define the following decision variables: for \(t = 1, \ldots, R\), for \(m = 1, \ldots, M\):

\[
\begin{align*}
u_{tm} & = \text{number of components of type } t \text{ to be mounted by machine } m; \\
v_{tm} & = 1 \text{ if a feeder of type } t \text{ is required on machine } m; \\
& = 0 \text{ otherwise.}
\end{align*}
\]

Our model for subproblem \((B)\) is:

\[
(M_B) \quad \text{minimize} \quad \sum_{t=1}^{R} \sum_{m=1}^{M} v_{tm}
\]

subject to

\[
\begin{align*}
\sum_{t=1}^{R} u_{tm} & = X_m \quad (m = 1, \ldots, M), \\
\sum_{m=1}^{M} u_{tm} & = N_t \quad (t = 1, \ldots, R),
\end{align*}
\]
Section 2.4

\[ u_{tm} \leq \min(X_m, N_t)v_{tm} \quad (t = 1, \ldots, R; \quad m = 1, \ldots, M), \]
\[ u_{tm} \geq 0 \text{ integer} \quad (t = 1, \ldots, R; \quad m = 1, \ldots, M), \]
\[ v_{tm} \in \{0, 1\} \quad (t = 1, \ldots, R; \quad m = 1, \ldots, M). \]

Model \((M_B)\) is a so-called pure fixed-charge transportation problem (see Fisk and McKeown (1979), Nemhauser and Wolsey (1988)).

Another way of thinking about model \((M_B)\) is in terms of machine scheduling. Consider \(R\) jobs and \(M\) machines, where each job can be processed on any machine. Job \(t\) needs a processing time \(N_t\) \((t = 1, \ldots, R)\) and machine \(m\) is only available in the interval \([0, X_m]\) \((m = 1, \ldots, M)\). Recall that \(\sum_{t=1}^{R} N_t = \sum_{m=1}^{M} X_m\). So, if preemption is allowed, there exists a feasible schedule requiring exactly the available time of each machine. Model \((M_B)\) asks for such a schedule minimizing the number of preempted jobs (in this interpretation, \(v_{tm} = 1\) if and only if job \(t\) is processed on machine \(m\)).

Complexity and solution of model \((M_B)\)

The well-known partition problem can be polynomially transformed to model \((M_B)\), which implies that \((M_B)\) is \(\mathcal{NP}\)-hard.

Model \((M_B)\) can be tackled by a specialized cutting-plane algorithm for fixed-charge transportation problems (Nemhauser and Wolsey (1988)), but we choose to use instead a simple heuristic. This heuristic consists in repeatedly applying the following rule, until all component types are assigned:

Rule:
Assign the type \((\text{say } t)\) with largest number \(N_t\) of components to the machine \((\text{say } m)\) with largest availability \(X_m\); if \(N_t \leq X_m\), delete type \(t\) from the list, and reduce \(X_m\) to \(X_m - N_t\); otherwise, reduce \(N_t\) to \(N_t - X_m\), and \(X_m\) to 0.

Clearly, this heuristic always delivers a feasible solution of \((M_B)\), with value exceeding the optimum of \((M_B)\) by at most \(M - 1\) (since, of all the component types assigned to a machine, at most one is also assigned to another machine). In other words, for a class \(c\) containing \(R\) component types, the heuristic finds an assignment of types to machines requiring at most \(R + M - 1\) feeders. This performance is likely to be quite satisfactory, since \(R\) is usually large with respect to \(M\).
In situations where duplication of feeders is strictly ruled out, i.e. where all components of one type must be mounted by the same machine, we replace the heuristic rule given above by:

**Modified rule:**
Assign the type (say i) with largest number $N_t$ of components to the machine with largest availability $X_m$; delete type $t$ from the list; reduce $X_m$ to $\max(0, X_m - N_t)$.

Of course, this modified rule does not, in general, produce a feasible solution of $(MB)$. In particular, some machine $m$ may have to mount more components of class $c$ than the amount $X_m$ determined by subproblem (A), and the estimated workload $W$ of the bottleneck machine may increase. In such a case, we continue with the solution supplied by the modified rule. A possible increase in estimated workload is the price to be paid for imposing more stringent requirements on the solution.

Before proceeding to the next phase, i.e. the scheduling of individual machines, we still have to settle one last point concerning the distribution of the workload over the machines. Namely, the solution of model $(MB)$ tells us how many components of each type must be processed by each machine (namely, $u_{tm}$), but not which ones. Since the latter decision does not seem to affect very much the quality of our final solution, we neglect to give here many details about its implementation. Let us simply mention that we rely on a model aiming at an even dispersion of the components over the PCB for each machine. The dispersion is measured as follows: we subdivide the PCB into cells, and we sum up the discrepancies between the expected number of components in each cell and their actual number. It is then easy to set up an integer linear programming problem, where the assignment of components to machines is modelled by 0-1 variables, and the objective corresponds to dispersion minimization. The optimal solution of this problem determines completely the final workload distribution.

### 2.5 Scheduling of individual machines

In this section we concentrate on one individual machine (for simplicity, we henceforth omit the machine index). Given by subproblem (B) are the locations (say $1, \ldots, N$) of the components to be mounted by this machine and their types $(1, \ldots, T')$. Given by subproblem (A) are the equipments $(1, \ldots, Q)$ to be used by the machine, and the number $r_h$ of equipment changes per head.
2.5.1 Subproblem (C)

The model

Our current goal is to determine the distribution of the workload over the three heads of the machine (a similar "partitioning" problem is treated by Almadi et al. (1988), under quite different technological conditions). This will be done so as to minimize the number of trips made by the heads between the feeder slots and the PCB. In other words, we want to minimize the maximum number of components mounted by a head. In general, this criterion will only determine how many components each head must pick and place, but not which ones. The latter indeterminacy will be lifted by the introduction of a secondary criterion, to be explained at the end of this subsection.

Here, we are going to use a model very similar to \((M_A)\). Since we are only interested in the number of components mounted by each head, let us redefine two components as equivalent if they can be handled by the same equipments (compare with the definition used in Subsection 2.4.1). This relation determines \(C\) classes of equivalent components. As for subproblem \((M_A)\), we let, for \(c = 1, \ldots, C\):

\[
\begin{align*}
B_c &= \text{number of components in class } c; \\
Q(c) &= \text{set of equipments which can handle the components in class } c.
\end{align*}
\]

We use the following decision variables: for \(c = 1, \ldots, C\), for \(h = 1, 2, 3\), for \(q = 1, \ldots, Q\):

\[
\begin{align*}
x_{ch} &= \text{number of components of class } c \text{ to be mounted by head } h; \\
z_{hq} &= 1 \text{ if head } h \text{ uses equipment } q; \\
&= 0 \text{ otherwise}; \\
V &= \text{number of components mounted by the bottleneck head}.
\end{align*}
\]

The model for subproblem (C) is:

\[
(M_C) \quad \text{minimize } V
\]

subject to

\[
\begin{align*}
\sum_{h=1}^{3} x_{ch} &= B_c \quad (c = 1, \ldots, C), \\
x_{ch} &\leq B_c \sum_{q \in Q(c)} z_{hq} \quad (c = 1, \ldots, C; h = 1, 2, 3), \\
\sum_{q=1}^{Q} z_{hq} &= r_h + 1 \quad (h = 1, 2, 3),
\end{align*}
\]
\begin{align*}
V & \geq \sum_{c=1}^{C} x_{ch} \quad (h = 1, 2, 3), \\
x_{ch} & \geq 0 \text{ integer} \quad (c = 1, \ldots, C; \ h = 1, 2, 3), \\
x_{hq} & \in \{0, 1\} \quad (h = 1, 2, 3; \ q = 1, \ldots, Q).
\end{align*}

(Recall that \( r_h + 1 \) is the number of equipments allocated to head \( h \) by model \((MA)\)).

\textbf{Complexity and solution of model \((MC)\)}

Again, the partition problem is easily transformed to model \((MC)\), implying that the problem is \(\mathcal{NP}\)-hard.

Moreover, as was the case for \((MA)\), model \((MC)\) is actually easy to solve in practice, due to the small number of variables. Here, we can use the same type of two-phase approach outlined for \((MA)\).

As mentioned earlier, the solution of \((MC)\) does not identify which components have to be mounted by each head. To answer the latter question, we considered different models taking into account the dispersion of the components over the board. However, it turned out empirically that a simple assignment procedure performed at least as well as the more sophisticated heuristics derived from these models. We describe now this procedure.

Consider a coordinate axis parallel to the arm along which the three heads are mounted. We orient this axis so that the coordinates of heads 1, 2 and 3 are of the form \(X\), \(X + k\) and \(X + 2k\) respectively, where \(k\) is the distance between two heads \((k > 0)\). Notice that \(X\) is variable, whereas \(k\) is fixed, since the arm cannot rotate.

The idea of our procedure is to assign the component locations with smallest coordinates to head 1, those with largest coordinates to head 3, and the remaining ones to head 2. Since this must be done within the restrictions imposed by \((MC)\), let us consider the values \(x_{ch}\) obtained by solving \((MC)\). Then, for each \(c\), the components of class \(c\) to be mounted by head 1 are chosen to be the \(x_{c1}\) components with smallest coordinates among all components of class \(c\). Similarly, head 3 is assigned the \(x_{c3}\) components with largest coordinates among the components of class \(c\), and head 2 is assigned the remaining ones.

As mentioned before, this heuristic provided good empirical results. The reason for this good performance may be sought in the fact that the inter-head distance \(k\) is of the same order of magnitude as the length of a typical
PCB. Thus, our simple-minded procedure tends to minimize the distance travelled by the heads.

2.5.2 Subproblem (D)

The model

For simplicity, we first consider the case where every head has been assigned exactly one piece of equipment (i.e., \( r_1 = r_2 = r_3 = 0 \) in model \((M_C)\)). Thus, at this point, the components have been partitioned into three groups, with group \( h \) containing the \( G_h \) components to be mounted by head \( h \) \((h = 1, 2, 3)\). Let us further assume that \( G_1 = G_2 = G_3 = G \) (if this is not the case, then we add a number of "dummy" components to the smaller groups). We know that \( G \) is also the minimum number of pick-and-place rounds necessary to mount all these components. We are now going to determine the composition of these rounds, with a view to minimizing the total travel time of the arm supporting the heads.

Suppose that the components in each group have been (arbitrarily) numbered \( 1, \ldots, G \). Consider two components \( i \) and \( j \) belonging to different groups, and assume that these components are to be mounted successively, in a same round. We denote by \( d_{ij} \) the time necessary to reposition the arm between the insertions of \( i \) and \( j \). For instance, if \( i \) is in group 1, \( j \) is in group 2, and \( i \) must be placed before \( j \), then \( d_{ij} \) is the time required to bring head 2 above the location of \( j \), starting with head 1 above \( i \).

For a pick-and-place round involving three components \( i, j, k \), we can arbitrarily choose the order in which these components are mounted (see Section 2.2). Therefore, an underestimate for the travel time of the arm between the first and the third placements of this round is given by:

\[
\begin{align*}
(\text{i}) \quad d_{ijk} &= \min\{d_{ij} + d_{jk}, d_{ik} + d_{kj}, d_{ji} + d_{ik}\} \text{ if none of } i, j, k \text{ is a dummy;} \\
(\text{ii}) \quad d_{ijk} &= d_{ij} \text{ if } k \text{ is a dummy;} \\
(\text{iii}) \quad d_{ijk} &= 0 \text{ if at least two of } i, j, k \text{ are dummies.}
\end{align*}
\]

Let us introduce the decision variables \( u_{ijk} \), for \( i, j, k \in \{1, \ldots, G\} \), with the interpretation:

\[
\begin{align*}
u_{ijk} &= 1 \text{ if components } i, j \text{ and } k \text{, from groups 1, 2 and 3, respectively,} \\
&\quad \text{are mounted in the same round;} \\
&= 0 \text{ otherwise.}
\end{align*}
\]
Then, our optimization model for subproblem (D) is:

\[(M_D) \text{ minimize } \sum_{i=1}^{G} \sum_{j=1}^{G} \sum_{k=1}^{G} d_{ijk} u_{ijk}\]

subject to \[
\sum_{i=1}^{G} \sum_{j=1}^{G} u_{ijk} = 1 \quad (k = 1, \ldots, G),
\]
\[
\sum_{i=1}^{G} \sum_{k=1}^{G} u_{ijk} = 1 \quad (j = 1, \ldots, G),
\]
\[
\sum_{j=1}^{G} \sum_{k=1}^{G} u_{ijk} = 1 \quad (i = 1, \ldots, G),
\]
\[u_{ijk} \in \{0, 1\} \quad (i, j, k = 1, \ldots, G).\]

An optimal solution of \((M_D)\) determines \(G\) clusters, of three components each, such that the sum of the (underestimates of the) travel times "within clusters" is minimized.

In cases where some or all of the heads have been assigned more than one piece of equipment in model \((M_A)\), we adapt our approach as follows. Let \(q_h\) be the first piece of equipment to be used by head \(h\) and \(G_h\) be the number of components which can be handled by \(q_h\) among those to be mounted by head \(h\) \((h = 1, 2, 3)\). Say for instance that \(G_1 \leq G_2 \leq G_3\). We can now set up a model similar to \((M_D)\) for the clustering of these \(G_1 + G_2 + G_3\) components. Any feasible solution of this model determines exactly \(G_1\) clusters containing no dummy components. These clusters correspond to our first \(G_1\), pick-and-place rounds, to be performed by equipments \(q_1, q_2\) and \(q_3\). Next, \(q_1\) is replaced by a new equipment \(q_4\), and the process can be repeated with \(q_4, q_2\) and \(q_3\).

**Complexity and solution of \((M_D)\)**

Model \((M_D)\), with arbitrary coefficients \(d_{ijk}\), has been studied in the literature under the name of three-dimensional assignment problem. The problem is known to be \(\mathcal{NP}\)-hard (see Garey and Johnson (1979)). However, observe that, in our case, the coefficients \(d_{ijk}\) are of the very special type defined by (i)–(iii). Moreover, the travel times \(d_{ij} (i, j = 1, \ldots, G)\) are themselves far from arbitrary; in particular, they satisfy the triangle inequality: \(d_{ij} \leq d_{ik} + d_{kj}\) for \(i, j, k = 1, \ldots, G\). However, even under these added restrictions, model \((M_D)\) remains \(\mathcal{NP}\)-hard (Chapter 3 of this thesis).
Section 2.5

A number of heuristic and exact algorithms have been proposed to solve the three-dimensional assignment problem (see, for example, Frieze and Yadegar (1981) and the references therein). In view of the role of (MP) as a subproblem in the hierarchy (A)-(F), and of the special structure of its cost coefficients, we opt here for a specialized heuristic procedure.

Our heuristic works in two phases. We start by solving an (ordinary) assignment problem, obtained by disregarding the components of the third group. Thus, we solve:

\[
(AP1) \quad \text{minimize} \quad \sum_{i=1}^{G} \sum_{j=1}^{G} d_{ij} u_{ij} \\
\text{subject to} \quad \sum_{j=1}^{G} u_{ij} = 1 \quad (i = 1, \ldots, G), \\
\sum_{i=1}^{G} u_{ij} = 1 \quad (j = 1, \ldots, G), \\
u_{ij} \in \{0, 1\} \quad (i, j, k = 1, \ldots, G),
\]

where $d_{ij} = 0$ if either $i$ or $j$ is dummy. An optimal solution $u^*$ of (AP1) can be computed in time $O(G^3)$ (Papadimitriou and Steiglitz (1982)).

Let now $A = \{(i, j) : u^*_{ij} = 1\}$. Thus, $A$ is the set of pairs $(i, j)$ matched by the solution of (AP1). The second phase of our heuristic consists in assigning the (previously disregarded) components of the third group to the pairs in $A$. Formally, we solve:

\[
(AP2) \quad \text{minimize} \quad \sum_{(i,j) \in A} \sum_{k=1}^{G} d_{ijk} u_{ijk} \\
\text{subject to} \quad \sum_{k=1}^{G} u_{ijk} = 1 \quad ((i, j) \in A), \\
\sum_{(i,j) \in A} u_{ijk} = 1 \quad (k = 1, \ldots, G), \\
u_{ijk} \in \{0, 1\} \quad ((i, j) \in A; \ k = 1, \ldots, G).
\]

The optimal solution of (AP2) can be obtained in time $O(G^3)$ and provides a heuristic solution of (MP). Frieze and Yadegar (1981) proposed a closely related heuristic for general 3-dimensional assignment problems, and observed its good empirical performance.

Let $\beta_3$ denote the optimal value of (AP2). The notation $\beta_3$ is a reminder
that, in the first phase of our heuristic, we arbitrarily decided to disregard the components from the third group. Of course, similar procedures could be defined, and corresponding bounds $\beta_1$ and $\beta_2$ would be derived, by initially disregarding the components from either group 1 or group 2.

In our computer implementations, we compute the three bounds, $\beta_1$, $\beta_2$, $\beta_3$, and we retain the clustering of the components corresponding to the smallest bound. In Chapter 3 of this thesis it is proven that this bound is never worse than $\frac{4}{3}$ times the optimal value for any instance of $(M_D)$. The computer experiments reported in there indicate that the practical performance of this heuristic is excellent.

### 2.5.3 Subproblem (E)

The solution of subproblem (D) has supplied a list $C_1, \ldots, C_G$ of clusters, with each cluster containing (at most) three components to be placed in the same round (if some heads must use more than one piece of equipment, then we successively consider several such lists, where each list consists of clusters which can be processed without equipment changes). Subproblem (E) asks for the sequence of pick-and-place operations to be performed by the machine, given this list of clusters.

This problem has been studied by Ball and Magazine (1988) and Leipälä and Nevalainen (1989), for machines featuring only one insertion head. In both papers, the authors observed that the decisions to be made in subproblem (E) are highly dependent on the assignment of feeders to feeder slots (i.e. on the solution of our subproblem (F)), and conversely. On the other hand, a model simultaneously taking into account both subproblems is far too complicated to be of any practical value.

We therefore choose an approach already suggested by Leipälä and Nevalainen (1989). Namely, we first solve subproblem (E); using this solution as input, we compute a solution of subproblem (F), which in turn is used to revise the solution of subproblem (E), and so on. This process is iterated until some stopping condition is verified.

### The models

According to the previous discussion, we need two models for subproblem (E): the first one to be used when no feeder assignment is yet known, and the second one taking into account a given feeder assignment. In either case, we reduce (E) to the solution of a shortest Hamiltonian path problem (see Lawler, Lenstra, Rinnooy Kan and Shmoys (1985)) over the set of clusters
Section 2.5

\{C_1, \ldots, C_G\}: for \(i, j = 1, \ldots, G\), we define a cost (travel time) \(c(i, j)\) for processing \(C_i\) immediately before \(C_j\); the problem is then to find a permutation \(\sigma = (\sigma_1, \ldots, \sigma_G)\) of \(\{1, \ldots, G\}\) which minimizes

\[
c(\sigma) = \sum_{i=1}^{G-1} c(\sigma_i, \sigma_{i+1})
\]  

(2.8)

The definition of \(c(i, j)\) depends on the given feeder assignment (if any), as explained hereunder.

Consider first the situation where feeders are already assigned to feeder slots, and let \(C_i, C_j\) be two arbitrary clusters. In this case, the appropriate definition of \(c(i, j)\) is given by CQM (1988) as follows. Denote by \(l_1, l_2, l_3\) the component locations in \(C_i\), where \(l_h\) is to be processed by head \(h\) \((h = 1, 2, 3)\). We assume that the feeder needed for \(l_h\) is in slot \(s_h\) \((h = 1, 2, 3)\). Similarly, \(l_4\) is the location to be processed by head 1 in cluster \(C_j\), and slot \(s_4\) contains the corresponding feeder (for simplicity, we assume that \(C_i\) and \(C_j\) consist of exactly three locations; obvious modifications of our description are required when this is not the case).

Suppose now for a moment that \(l_1, l_2\) and \(l_3\) are to be mounted in the order \(\pi = (\pi_1, \pi_2, \pi_3)\), where \((\pi_1, \pi_2, \pi_3)\) is a permutation of \(\{1, 2, 3\}\). For this fixed order, we can easily compute the time (say, \(c_{ij}(\pi)\)) required to carry out the following operations: starting with head 1 above slot \(s_1\), sequentially pick one component from each of \(s_1, s_2, s_3\) using heads 1, 2, 3 respectively; mount \(l_{\pi_1}, l_{\pi_2}, l_{\pi_3}\), in that order; bring head 1 above slot \(s_4\).

Obviously, in an optimal pick-and-place sequence, we would select the permutation \(\pi^*\) of \(\{1, 2, 3\}\) which minimizes \(c_{ij}(\pi)\). We accordingly define:

\(c(i, j) = c_{ij}(\pi^*)\).

Now, if \(\sigma\) is any permutation of \(\{1, \ldots, G\}\), then \(c(\sigma)\) (given by (2.8)) is the time required by a complete pick-and-place sequence processing the clusters in the order \((\sigma_1, \ldots, \sigma_G)\). The shortest Hamiltonian path problem with costs \(c_{ij}\) thus provides a natural model for subproblem (E). As a last remark on this model, notice that the computation of \(c_{ij}(\pi)\) can be simplified by omitting from its definition those elements which are independent of \(\pi\) or \(\sigma\). Namely, we can use a "modified \(c_{ij}(\pi^*)\)" defined as the time needed, starting with head 3 above \(s_3\), to bring successively head \(\pi_1\) above \(l_{\pi_1}\), head \(\pi_2\) above \(l_{\pi_2}\), head \(\pi_3\) above \(l_{\pi_3}\) and finally head 1 above \(s_4\).

Let us return now to the initial solution of (E), when the feeder positions are still unknown. Since this initial sequence will be modified by the subsequent iterations of our procedure, it does not seem necessary at this stage to look for a solution of very high quality (actually, one may even argue
that an initial sequencing of lower quality is desirable since it provides more flexibility in the next phases of the procedure; see, for example, Leipälä and Nevalainen (1989) for more comments along this line). Accordingly, we define the coefficients $c(i, j)$ for our initial traveling salesman problem as rough estimates of the actual travel times. We experimented with some possible definitions, which seem to lead to comparable results (in terms of the final solution obtained). One such definition is as follows. Let $g_i$ and $g_j$ be the centers of gravity of the clusters $C_i$ and $C_j$, respectively. Let $s$ be the feeder slot minimizing the total distance from $t_i$ to $s$ to $g_j$. Then, $c(i, j)$ is the time needed for the arm to travel this total distance.

**Complexity and solution of the models**

The shortest Hamiltonian path problem is closely related to the traveling salesman problem, and is well-known to be $NP$-hard, even when the costs $c(i, j)$ satisfy the triangle inequality (Lawler et al. (1985)). Many heuristics have been devised for this problem, and we have chosen to experiment with two of the simplest: nearest neighbor (with all possible starting points) and farthest insertion, which respectively run in $O(G^3)$ and $O(G^2)$ steps (we refer to Lawler et al. (1985) for details on these procedures). Both heuristics produced results of comparable quality.

**2.5.4 Subproblem (F)**

**The model**

As input to this subproblem, we are given the types $(1, \ldots, T)$ and the locations $(1, \ldots, N)$ of the components to be mounted, where $(1, \ldots, N)$ is the mounting sequence determined by the previous solution of subproblem (E). Our problem is now to allocate each feeder $1, \ldots, T$ to one of the feeder slots $1, \ldots, S$, so as to minimize the total mounting time (for the sake of clarity, we first assume that every feeder can be loaded in exactly one slot; we indicate later how our model can be modified when some feeders require two or more slots).

We use the decision variables $u_{ts}$ ($t = 1, \ldots, T; s = 1, \ldots, S$) with the interpretation:

- $u_{ts} = 1$ if feeder $t$ is loaded in slot $s$;
- $u_{ts} = 0$ otherwise.
These variables must obey the following restrictions, expressing that every feeder occupies exactly one slot, and no slot contains two feeders:

\[
\sum_{s=1}^{S} v_{ts} = 1 \quad (t = 1, \ldots, T), \quad (2.9)
\]

\[
\sum_{t=1}^{T} v_{ts} \leq 1 \quad (s = 1, \ldots, S), \quad (2.10)
\]

\[
v_{ts} \in \{0, 1\} \quad (t = 1, \ldots, T; \ s = 1, \ldots, S). \quad (2.11)
\]

Before describing the other elements of our model, we first introduce some terminological conventions. We say that a movement of the arm is a feeder-board movement if it occurs between the last picking and the first placing of the same round, or between the last placing of a round and the first picking of the next one. By contrast, a feeder-feeder movement takes place between two pickings of a same round.

Consider now a fixed solution \( v_{ts} \) \((t = 1, \ldots, T; \ s = 1, \ldots, S)\) of (2.9)–(2.11). For the corresponding assignment of feeders to slots, the total mounting time of the PCB can be broken up into three terms:

1) a term \( \sum_{t=1}^{T} \sum_{s=1}^{S} a_{ts} v_{ts} \), where \( a_{ts} \) is the total time spent in feeder-board movements from or to feeder \( t \), when feeder \( t \) is loaded in slot \( s \); this term represents the total feeder-board travel time; notice that the value of each coefficient \( a_{ts} \) is completely determined by the technological features of the machine, and by the sequence of pick-and-place operations to be performed by the machine (i.e., by the solution of subproblem (E));

2) a term \( \sum_{p,t=1}^{T} \sum_{r,s=1}^{S} b_{pts} v_{pt} v_{ts} \), where \( b_{pts} \) is the total time spent in feeder-feeder movements between feeders \( p \) and \( t \), when feeder \( p \) is in slot \( r \) and feeder \( t \) is in slot \( s \); this term gives the total feeder-feeder travel time; here again, the coefficients \( b_{pts} \) are easily computed;

3) a term accounting for all other operations (picking and placing of all components, and travel time between placements of the same round); for a fixed pick-and-place sequence, this term is independent of \( v_{ts} \). According to this discussion, our model for subproblem (F) can be formulated as:

\[\text{(MP)} \quad \text{minimize} \quad \sum_{t=1}^{T} \sum_{s=1}^{S} a_{ts} v_{ts} + \sum_{p,t=1}^{T} \sum_{r,s=1}^{S} b_{pts} v_{pt} v_{ts} \]

subject to \((2.9),(2.10),(2.11)\).
Problem \((M_F)\) is a quadratic assignment problem (see Burkard (1984)). As mentioned earlier, this formulation can easily be modified to accommodate additional restrictions. For instance, if feeder \(t\) must occupy two slots, we reinterpret:

\[
v_{ts} = 1 \text{ if feeder } t \text{ is loaded in slots } s \text{ and } s + 1; \]
\[
= 0 \text{ otherwise.}
\]

Straightforward restrictions must then be added to (2.9)–(2.11) to preclude the assignment of any feeder to slot \(s + 1\) when \(v_{ts} = 1\). This can also be achieved while preserving the quadratic assignment structure of \((M_F)\), by raising all coefficients \(b_{p,s+1,t,s}\) to very high values.

As a last remark on \((M_F)\), let us observe that this model boils down to a linear assignment problem for machines featuring only one insertion head. On the other hand, Leipälä and Nevalainen (1989) proposed a quadratic assignment formulation of the feeder assignment subproblem \((F)\) for another type of one-head machines. This discrepancy is obviously due to the different technologies.

**Complexity and solution of \((M_F)\)**

The quadratic assignment problem is well-known to be \(\mathcal{NP}\)-hard, and to be particularly difficult to solve exactly for values of \(T\) and \(S\) larger than twenty (Burkard (1984)). A typical instance of \((M_F)\) may involve as many as twenty feeder types and sixty slots, and hence must be tackled by heuristic methods.

For \((M_F)\), we have used a local improvement method, based on pairwise exchanges of feeders (see Burkard (1984)). This procedure starts with an initial solution of (2.9)–(2.11), and applies either of the following steps, as long as they improve the objective function value in \((M_F)\):

**Step 1**: move a feeder from its current slot to some empty slot;

**Step 2**: interchange the slot assignments of two feeders.

To determine an initial assignment of feeders to slots, we proceed in two phases. First, we solve the assignment problem \((M_F')\) obtained by setting all coefficients \(b_{p,t,s}\) to zero in \((M_F)\) (this amounts to disregarding the feeder-feeder movements of the arm). Let \(v^*\) be an optimal solution of \((M_F')\).

Next, we consider those feeders (say \(1, \ldots, P\)) whose components are only picked by head 2. Observe that the associated variables \(v_{ts} (t = 1, \ldots, P; s = 1, \ldots, S)\) do not appear in the objective function of \((M_F')\), since there
are no feeder-board movements to or from these feeders (i.e., $a_{ts} = 0$ for $t = 1, \ldots, P$; $s = 1, \ldots, S$). Consequently, the value of these variables in $v^*$ is conditioned only by the constraints (2.9)-(2.11), and may as well be random. In order to determine more meaningful values for these variables, we solve the restriction of $(M_P)$ obtained by setting $v_{ts} = v_{ts}^*$ for $t = P + 1, \ldots, T$ and $s = 1, \ldots, S$. It is easy to see that this again is a linear assignment problem, aiming at the minimization of the total feeder-feeder travel time under the partial assignment $v_{ts}^*$ ($t = P + 1, \ldots, T$; $s = 1, \ldots, S$). The optimal solution of this problem together with the values $v_{ts}^*$ ($t = P + 1, \ldots, T$; $s = 1, \ldots, S$), provides the initial solution for the improvement procedure described above.

2.6 An example

In this section, we discuss the performance of our heuristics on a problem instance described in CQM (1988). The placement line under consideration consists of three machines. The third head is broken and unavailable on machine 3. The 258 components to be mounted on the PCB are grouped in 39 types (actually, the PCB is partitioned into three identical blocks, of 86 components each; we shall make use of this peculiarity in the solution of subproblem (A)). Three distinct pieces of equipments suffice to handle all the component types; moreover, each type can be handled by exactly one of these three pieces of equipments.

For the sake of comparison, let us mention that CQM (1988) evaluates to 74, 65 and 81 seconds, respectively, the mounting times required by the three machines for the actual operations sequence implemented by the plant (notice that this sequence is not known in full detail, and that these “plant times” appear to be underestimates). The hierarchical decomposition and the heuristics developed in CQM (1988) produce a solution with mounting times 68.41, 66.52 and 68.88 seconds for the three machines. A still better solution is obtained in CQM (1988) after imposing that the equipments used remain fixed as in the plant situation. Under these conditions, production times of 66.12, 65.25 and 65.47 are achieved on the three machines, i.e. an improvement of at least 18 percent of the bottleneck time with respect to the plant solution. To fully appreciate these figures, one should also know that a constant time of 106 seconds is needed for the pick-and-place operations alone, independently of the production sequence (see Section 2.2). These unavoidable 106 seconds represent more than half of the total mounting time required by the CQM solutions.
Subproblem (A)

We now take up our subproblem (A). With a constant estimate of \( v = 0.3 \) (secs) for the travel time of the heads between two insertions, the components fall into five classes, characterized by the parameters in Table 2.1.

<table>
<thead>
<tr>
<th>Class</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_c )</td>
<td>201</td>
<td>27</td>
<td>24</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>( w_c )</td>
<td>0.6</td>
<td>1.35</td>
<td>0.75</td>
<td>1.65</td>
<td>1.15</td>
</tr>
<tr>
<td>( Q(c) )</td>
<td>{1}</td>
<td>{2}</td>
<td>{3}</td>
<td>{3}</td>
<td>{3}</td>
</tr>
</tbody>
</table>

Table 2.1 Parameters for subproblem (A).

We set up model \((M_A)\) with these parameters and \( E_h = 2 \) \((h = 1, \ldots, 8)\) (and the obvious modifications implied by the unavailability of head 9). This model is easily solved by the approach described in Subsection 2.4.1. Notice that the relaxation of \((M_A)\) obtained by omitting the integrality requirement for the \( x \)-variables has several alternative optima. As expected, \( r_h = 0 \) \((h = 1, \ldots, 8)\) in all these optimal solutions, i.e. equipment changes are ruled out.

As explained in Subsection 2.4.1, the solutions found for subproblem (A) can be considered as alternative inputs for the subsequent subproblems in the decomposition. In the present case, most of these solutions led us to production plans with processing times of 66 to 68 seconds. To illustrate the next steps of our approach, we shall concentrate now on a specific solution of \((M_A)\), derived as follows.

We mentioned before that our PCB consists of three identical blocks. So, rather than solving \((M_A)\) for the complete board, we can solve first the model corresponding to one of the blocks, and eventually multiply all figures by 3. A workload distribution obtained in that way is displayed in Table 2.2.

<table>
<thead>
<tr>
<th>Machine</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equipments</td>
<td>1</td>
<td>1,3</td>
<td>1,2</td>
</tr>
<tr>
<td>( x_{cm} = \text{number of components} ) of class ( c ) on machine ( m )</td>
<td>( x_{11} = 102 )</td>
<td>( x_{12} = 57 )</td>
<td>( x_{13} = 42 )</td>
</tr>
<tr>
<td></td>
<td>( x_{32} = 24 )</td>
<td>( x_{42} = 3 )</td>
<td>( x_{23} = 27 )</td>
</tr>
<tr>
<td></td>
<td>( x_{52} = 3 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2 Workload distribution.
Section 2.6

Subproblem (B)

Since all components of class 2 are to be handled by machine 3, and all components of classes 3, 4, 5 by machine 2, we see that the distribution shown in Table 2.3 need only be further refined for class 1. Specifically, 28 components types are represented in class 1. The number of components of each type \((1, \ldots, 28)\) is given in Table 2.3.

<table>
<thead>
<tr>
<th>Type</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N_t)</td>
<td>24</td>
<td>18</td>
<td>18</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N_t)</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

**Table 2.3** Number of components of each type for subproblem (B).

The heuristic rule described in Subsection 2.4.2 produces the assignment shown in Table 2.4. Observe that each type is assigned to exactly one machine, and hence exactly one feeder of each type will be needed in the final solution (in particular, the heuristic delivers here an optimal solution of \((M_B)\)).

<table>
<thead>
<tr>
<th>Machine</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,2,3,5,10,11,14,17,20,23,26</td>
</tr>
<tr>
<td>2</td>
<td>4,6,8,12,15,18,21,24,27</td>
</tr>
<tr>
<td>3</td>
<td>7,9,13,16,19,22,25,28</td>
</tr>
</tbody>
</table>

**Table 2.4** Assignment of component types to machines.

Subproblem (C)

Since model \((M_C)\) attempts to minimize the maximum workload of the heads (per machine), in this case we obviously find an assignment of the type given in Table 2.5.

<table>
<thead>
<tr>
<th>Head</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equipment</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Number of components</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>29</td>
<td>28</td>
<td>30</td>
<td>42</td>
<td>27</td>
</tr>
</tbody>
</table>

**Table 2.5** Assignment for subproblem (C).
The components to be mounted by heads 1, 2, 3, 4, 5 are further identified as explained at the end of Subsection 2.5.2. In the present case, this amounts to assigning to head 1 all components of block 1, to head 2 all components of block 2, and to head 3 all components of block 3, among those previously assigned to machine 1.

**Subproblem (D)**

We now solve the three-dimensional assignment model \( M_D \) for each of the three machines. Since machine 3 only has two heads, \( M_D \) actually reduces to the assignment problem (AP1) for this machine, and hence can be solved exactly (optimal value: 3.26 secs).

For machines 1 and 2, we solve \( M_D \) using the heuristics described in Subsection 2.5.2. For machine 1, these heuristics supply a very good clustering of the components (value: 4.95 secs), where each cluster simply contains corresponding components from each block of the PCB. For machine 2 we obtain a clustering with value 8.95 secs.

**Subproblems (E) and (F)**

These two subproblems are solved alternately and iteratively for each machine.

On machine 2, for instance, the first Hamiltonian path (corresponding to travel times between centers of gravity of the clusters) has value 13.16 secs. An initial feeder assignment is obtained as in Subsection 2.5.4. The pick-and-place sequence determined by this assignment and the first Hamiltonian path corresponds to a total feeder-board time of 14.10 secs and a total feeder-feeder time of 11.63 secs, for a total travel time of 25.73 secs.

The local improvement procedure is next applied to this initial solution. In each iteration of this procedure, we sequentially consider all feeders, and we attempt to perform one of the exchange steps 1 and 2 on each of them. After four iterations of the procedure, no more improving steps are found. The corresponding feeder-board and feeder-feeder times are respectively 14.68 secs and 8.62 secs, and hence the previous total travel time is improved to 23.30 secs.

Taking this feeder assignment into account, a revised Hamiltonian path with value 14.07 secs is computed. The feeder assignment is in turn modified, resulting in (after three iterations of the local improvement procedure) a total travel time of 22.94 secs. No better Hamiltonian path or assignment
are found in the next solutions of subproblems (E) and (F). Therefore, we adopt this solution for machine 2.

Similar computations are carried out for the other machines. The pick-and-place sequences obtained in this way correspond to processing times of 63.83, 66.27 and 65.82 secs on machines 1, 2 and 3 respectively. These times are comparable to the best ones obtained by CQM.

Acknowledgements
We are grateful to P. van Laarhoven and H. Zijm for introducing us to the problem, and for providing us with the data (CQM, 1988).
Chapter 3

Approximation algorithms for three-dimensional assignment problems with triangle inequalities

This chapter is joint work with Yves Crama; it will be published as an article in the European Journal of Operational Research.
3.1 Introduction

Consider the following classical formulation of the (axial) three-dimensional assignment problem (3DA) (see e.g. Balas and Saltzman (1989)). Given is a complete tripartite graph \( K_{n,n,n} = (I \cup J \cup K, (I \times J) \cup (I \times K) \cup (J \times K)) \), where \( I, J, K \) are disjoint sets of size \( n \), and a cost \( c_{ijk} \) for each triangle \( (i,j,k) \in I \times J \times K \). The problem 3DA is to find a subset \( A \) of \( n \) triangles, \( A \subseteq I \times J \times K \), such that every element of \( I \cup J \cup K \) occurs in exactly one triangle of \( A \), and the total cost \( c(A) = \sum_{(i,j,k) \in A} c_{ijk} \) is minimized. Some recent references to this problem are Balas and Saltzman (1989), Frieze (1974), Frieze and Yadegar (1981), Hansen and Kaufman (1973).

When one formulates 3DA in graph-theoretic terms, as we just did it, it is natural to assume that the costs \( c_{ijk} \) are not completely arbitrary, but are rather defined in terms of costs attached to the edges of the graph. More precisely, we shall restrict our attention in this chapter to the special cases of 3DA where each edge \((u,v) \in (I \times J) \cup (I \times K) \cup (J \times K)\) is assigned a nonnegative length \( d_{uv} \), and where the cost of a triangle \((i,j,k) \in I \times J \times K\) is defined either by its total length \( t_{ijk} \):

\[
t_{ijk} = d_{ij} + d_{ik} + d_{jk},
\]

or by \( s_{ijk} \), the sum of the lengths of its two shortest edges:

\[
s_{ijk} = \min\{d_{ij} + d_{ik}, \ d_{ij} + d_{jk}, \ d_{ik} + d_{jk}\}
\]

(notice that the lengths \( d_{uv} \) are symmetric: \( d_{uv} = d_{vu} \) for all \((u,v)\)).

We refer to the problem 3DA with cost coefficients \( c_{ijk} = t_{ijk} \), or \( c_{ijk} = s_{ijk} \), as problem \( T \) or \( S \), respectively.

Instances of problem \( T \) arise in the scheduling of teaching practices at colleges of education (Frieze and Yadegar (1981)).

Either \( T \) or \( S \) can also be used to model a situation encountered in the production of printed circuit boards by numerically controlled machines featuring three placement heads (see Chapter 2 of this thesis).

In the latter application (which motivated the present study), the lengths \( d_{uv} \) represent travel times of the arm of the machine between locations \( u \) and \( v \), where electronic components are to be inserted. In particular, and even though the exact definition of these travel times may be quite intricate, the lengths \( d_{uv} \) define a distance, i.e. they satisfy the triangle inequalities:

\[
d_{uv} \leq d_{uw} + d_{vw} \text{ for all } u, v, w \in I \cup J \cup K.
\]
Section 3.2

In the remainder of this chapter, we concentrate on problems $T\Delta$ and $S\Delta$, i.e., on the special cases of $T$ and $S$ for which the triangle inequalities (3.3) hold. We show in Section 3.2 that $T\Delta$ and $S\Delta$ are $NP$-hard. In Section 3.3, we describe some heuristics for $T\Delta$ and $S\Delta$, and establish tight bounds on their worst-case performance. The results of computational experiments with these heuristics are presented in Section 3.4.

3.2 Complexity of $T\Delta$ and $S\Delta$

The problem 3DA is well-known to be $NP$-hard, even when the costs $c_{ijk}$ can only take two distinct values (see e.g. Garey and Johnson (1979) for a proof). We show now that its special cases $T\Delta$ and $S\Delta$ remain $NP$-hard too.

**Theorem 3.1** Problem $T\Delta$ is $NP$-hard.

**Proof:**

We use the argument presented by Garey and Johnson (1979) to establish the $NP$-hardness of the problem Partition into Triangles. Consider an instance $I$ of 3DA, defined by three sets $I_0, J_0, K_0$ of size $n$, and $c_{ijk} \in \{0, 1\}$ for all $(i,j,k) \in I_0 \times J_0 \times K_0$.

With $I$, we associate an instance of $T\Delta$, as follows. Let $M = \{(i,j,k) : c_{ijk} = 0\}, |M| = m$, and

$$I = I_0 \cup \{i_l(e) : e \in M, l = 1,2,3\},$$

$$J = J_0 \cup \{j_l(e) : e \in M, l = 1,2,3\},$$

$$K = K_0 \cup \{k_l(e) : e \in M, l = 1,2,3\},$$

where $i_l(e), j_l(e), k_l(e) (e \in M, l = 1,2,3)$ are $9m$ new elements.

In order to conveniently define the lengths of the edges of the complete tripartite graph $G$ on $I \cup J \cup K$, we first introduce $m$ subgraphs of $G$. For each $e = (i,j,k) \in M$, $G(e)$ is the graph represented in Figure 3.1.
Chapter 3

Figure 3.1

Now for each \((u, v) \in (I \times J) \cup (I \times K) \cup (J \times K)\), we let: \(d_{uv} = 1\) if \((u, v)\) is an edge in some graph \(G(e) (e \in M)\), and \(d_{uv} = 2\) otherwise. Clearly, the triangle inequalities (3.3) are satisfied by this assignment, so that \(I, J, K\) and the lengths \(d_{uv}\) together define an instance \(T\) of \(T\Delta\).

Observe that every feasible solution of \(T\) contains exactly \(n + 3m\) triangles, each with cost at least 3. We claim that \(T\) has an optimal solution with value \(3n + 9m\) if and only if \(I\) has a solution with value 0. We leave details to the reader (see Garey and Johnson (1979), pp. 68-69). □

**Theorem 3.2** Problem \(S\Delta\) is \(\mathcal{NP}\)-hard.

**Proof:**
The proof is similar to the previous one: simply delete from each subgraph \(G(e)\) the edges \((i_1(e), j_2(e)), (i, j_2(e)), (j_1(e), k_3(e)), (j, k_3(e)), (i_3(e), k_1(e)), (i_3(e), k)\), for all \(e \in M\). The resulting instance of \(S\Delta\) has an optimal solution with value \(2n + 6m\) if and only if \(I\) has a solution with value 0. □
3.3 Approximation algorithms

In this section, we present approximation algorithms for $T\Delta$ and $S\Delta$. First, we recall a definition from Papadimitriou and Steiglitz (1982) (see also Garey and Johnson (1979)). Consider a minimization problem $P$, and an algorithm $H$ which, given any instance $I$ of $P$, returns a feasible solution $H(I)$ of $I$. Denote by $c(H(I))$ the value of this heuristic solution, and by $\text{OPT}(I)$ the value of an optimal solution of $I$. Then, $H$ is called an $\varepsilon$-approximate algorithm for $P$, where $\varepsilon$ is a nonnegative constant, if:

$$c(H(I)) \leq (1 + \varepsilon) \text{OPT}(I)$$

for all instances $I$ of $P$.

We will show that $\frac{1}{3}$-approximate polynomial-time algorithms exist for problems $T\Delta$ and $S\Delta$. As indicated by our next theorem, the triangle inequalities (3.3) play an instrumental role in the proof of such results:

**Theorem 3.3** Unless $P = \mathcal{NP}$, there is no $\varepsilon$-approximate polynomial algorithm for problems $T$ and $S$, for any $\varepsilon \geq 0$.

**Proof:**

We establish the statement for problem $T$ (the other case being similar). Assume that there is an $\varepsilon$-approximate algorithm for $T$, say $H$.

As in the proof of Theorem 3.1, consider an instance $I$ of 3DA with $c_{ijk} \in \{0, 1\}$ for all $(i,j,k)$, the corresponding sets $I,J,K$, and the subgraphs $G(e) (e \in M)$.

For $(u,v) \in (I \times J) \cup (I \times K) \cup (J \times K)$, let: $d_{uv} = 1$ if $(u,v)$ is an edge of $G(e) (e \in M)$, and $d_{uv} = (3n + 9m)\varepsilon + 2$ otherwise. This defines an instance $T$ of problem $T$, with the property that $T$ has an optimal solution with value $3n + 9m$ if and only if $I$ has a solution with value 0.

Now, it is easy to see that the $\varepsilon$-approximate algorithm $H$ always returns a solution of $T$ with value $3n + 9m$, if there is one (because the second best solution has value at least $(1 + \varepsilon)(3n + 9m) + 1$). Hence, unless $P = \mathcal{NP}$, $H$ cannot be a polynomial-time algorithm.

We describe now informally a polynomial-time heuristic $H_{I,J}$ for problems $T$ and $S$. This heuristic was proposed in Chapter 2 of this thesis. The input to $H_{I,J}$ is the set of edge-lengths $d_{uv}$, where $(u,v) \in (I \times J) \cup (I \times K) \cup (J \times K)$, and $|I| = |J| = |K| = n$.

The heuristic proceeds in two phases, first matching the elements of $I$ and $J$,
and next assigning the elements of $K$ to the pairs thus formed (Frieze and Yadegar (1981) propose a similar heuristic for the general 3DA problem). More precisely:

**Phase 1.** Find an optimal solution $\mathbf{x}^*$ of (P1):

(P1) \[ \begin{align*} & \text{minimize} \quad \sum_{i \in I} \sum_{j \in J} d_{ij} x_{ij} \\ & \text{subject to} \quad \sum_{i \in I} x_{ij} = 1 \quad (j \in J) \\ & \quad \sum_{j \in J} x_{ij} = 1 \quad (i \in I) \\ & \quad x_{ij} \in \{0, 1\} \quad (i \in I, j \in J). \end{align*} \]

Let $M = \{(i, j) : x_{ij}^* = 1\}$.

**Phase 2.** Find an optimal solution $\mathbf{y}^*$ of (P2):

(P2) \[ \begin{align*} & \text{minimize} \quad \sum_{(i, j) \in M} \sum_{k \in K} c_{ijk} y_{ijk} \\ & \text{subject to} \quad \sum_{(i, j) \in M} y_{ijk} = 1 \quad (k \in K) \\ & \quad \sum_{k \in K} y_{ijk} = 1 \quad ((i, j) \in M) \\ & \quad y_{ijk} \in \{0, 1\} \quad ((i, j) \in M, k \in K), \end{align*} \]

where $c_{ijk} = t_{ijk}$ (respectively $c_{ijk} = s_{ijk}$) if the problem to be solved is an instance of $T$ (respectively $S$).

The feasible solution of $T$ (or $S$) returned by the heuristic $H_{IJ}$ is $A = \{(i, j, k) : y_{ijk}^* = 1\}$, and its cost is denoted by $c_{IJ}$.

Notice that both (P1) and (P2) are instances of the classical (two-dimensional) assignment problem, or weighted bipartite matching problem, and hence can be solved in $O(n^3)$ operations (Papadimitriou and Steiglitz (1982)). It follows that $H_{IJ}$ also runs in time $O(n^3)$.

We leave it as an easy exercise to verify that, as suggested by Theorem 3.3, $H_{IJ}$ is not an $\varepsilon$-approximate algorithm for either $T$ or $S$, for any $\varepsilon \geq 0$.

On the other hand, when the lengths $d_{uv}$ satisfy the triangle inequalities, we get:

**Theorem 3.4** $H_{IJ}$ is a $\frac{1}{2}$-approximate algorithm for problem $T\Delta$. Moreover, there exist arbitrary large instances $\mathcal{T}$ of $T\Delta$ such that $c_{IJ} = \frac{3}{2} \text{OPT}(\mathcal{T})$. 
Section 3.3

Proof:
Let $T$ be an instance of $T \Delta$. Let $M$ be the matching of $I \cup J$ found by the first phase of $H_{ij}$, and $A$ be the assignment returned by $H_{ij}$.

Consider now an optimal solution of $T$, say $F$. With $F$, we associate another feasible solution $B = \{(i, j, k) : (i, j) \in M$, and $(u, j, k) \in F$ for some $u \in I\}.

We obtain successively:

\[
c_{ij} = \sum_{(i, j, k) \in A} t_{ijk} \leq \sum_{(i, j, k) \in B} t_{ijk} = \sum_{(i, j, k) \in B} (d_{ij} + d_{ik} + d_{jk}) \leq 2 \sum_{(i, j, k) \in B} (d_{ij} + d_{jk}) = 2 \sum_{(i, j, k) \in A} d_{ij} + 2 \sum_{(i, j, k) \in F} d_{jk} \leq 2 \sum_{(i, j, k) \in F} (d_{ij} + d_{jk}) \]

(3.4) holds because $A$ is optimal for $(P2)$, (3.5) is by definition of $t_{ijk}$, (3.6) uses the triangle inequality, (3.7) is by definition of $B$, and (3.8) follows from optimality of $M$ for $(P1)$.

By symmetry with (3.8), we can also derive:

\[
c_{ij} \leq 2 \sum_{(i, j, k) \in F} (d_{ij} + d_{ik}) \]

(3.9)

Now, (3.8) and (3.9) together entail:

\[
c_{ij} \leq \sum_{(i, j, k) \in F} (2d_{ij} + d_{ik} + d_{jk}) = \sum_{(i, j, k) \in F} \left(\frac{3}{2}d_{ij} + \frac{1}{2}d_{ij} + d_{ik} + d_{jk}\right) \]

and, using the triangle inequalities to bound $\frac{1}{2}d_{ij}$:
\[ c_{ij} \leq \frac{3}{2} \sum_{(i,j,k) \in F} t_{ijk} = \frac{3}{2} \text{OPT}(\mathcal{T}). \]  \hspace{1cm} (3.11)

To see that equality may hold in (3.11), consider first the graph \( G \) represented in Figure 3.2.

![Figure 3.2](image)

Also indicated in Figure 3.2 are the costs \( c_{uv} \in \{1, 2\} \) of the edges of \( G \).

Now, we define an instance \( \mathcal{T} \) of \( T\Delta \) as follows. We let \( I = \{i_1, i_2\}, J = \{j_1, j_2\}, K = \{k_1, k_2\} \). For \((u, v) \in (I \times J) \cup (I \times K) \cup (J \times K)\), \( d_{uv} \) is the length of a shortest path from \( u \) to \( v \) in \( G \), with respect to the costs \( c_{uv} \).

It is easy to see that an optimal solution for this instance is \( F = \{(i_1, j_2, k_1), (i_2, j_1, k_2)\} \), with \( \text{OPT}(\mathcal{T}) = 8 \).

But \( H_{1, J} \) can pick (in Phase 1) \( M = \{(i_1, j_1), (i_2, j_2)\} \), and next (in Phase 2) \( A = \{(i_1, j_1, k_1), (i_2, j_2, k_2)\} \), with cost \( c_{IJ} = 12 = \frac{3}{2} \text{OPT}(\mathcal{T}) \). Arbitrary large instances of \( T\Delta \) can be obtained by taking several copies of \( G \), with very large distances between points in different copies.

\[ \square \]

The previous result also holds mutatis mutandis for problem \( S\Delta \):

**Theorem 3.5** \( H_{1, J} \) is a \( \frac{1}{2} \)-approximate algorithm for problem \( S\Delta \). Moreover, there exist arbitrary large instances \( S \) of \( S\Delta \) such that \( c_{IJ} = \frac{3}{2} \text{OPT}(S) \).
Section 3.3

Proof:
Let $S$ be an instance of $S\Delta$. Define $M, A, F$ and $B$ in the same way as for the proof of Theorem 3.4. We derive the following inequalities:

$$c_{ij} = \sum_{(i,j,k) \in A} s_{ijk}$$

$$\leq \sum_{(i,j,k) \in B} s_{ijk}$$

$$\leq \sum_{(i,j,k) \in A} d_{ij} + \sum_{(i,j,k) \in F} d_{jk}$$

$$\leq \sum_{(i,j,k) \in F} (d_{ij} + d_{jk})$$

(3.12) holds because $A$ is optimal for $(P2)$, (3.13) is by definition of $B$ and of $s_{ijk}$, (3.14) follows from the optimality of $M$ for $(P1)$.

By symmetry with (3.14), the following inequality is also valid:

$$c_{ij} \leq \sum_{(i,j,k) \in F} (d_{ij} + d_{ik}).$$

(3.15)

Using the triangle inequalities, one easily checks:

$$2d_{ij} + d_{ik} + d_{jk} \leq 3s_{ijk} \quad \text{for all } i, j, k.$$  

Hence, (3.14), (3.15) and (3.16) together imply:

$$c_{ij} \leq \frac{3}{2} \sum_{(i,j,k) \in F} s_{ijk} = \frac{3}{2} \text{OPT}(S).$$

(3.17)

The example presented in the proof of Theorem 3.4 also achieves equality in (3.17), and can be used to build arbitrary large instances.

Of course, one can define in a natural way two more $\frac{1}{2}$-approximate algorithms for problems $T\Delta$ and $S\Delta$, namely the heuristics $H_{IK}$ and $H_{JK}$ obtained by permuting the roles of $I$, $J$ and $K$ in the description of $H_{IJ}$. We denote by $c_{iK}$ and $c_{JK}$ the values of the solutions delivered by $H_{IK}$ and $H_{JK}$, respectively.

Consider now the heuristic $H$, which consists in applying all three heuristics $H_{IJ}$, $H_{IK}$ and $H_{JK}$ to the given instance of $T\Delta$ or $S\Delta$, and in retaining the best feasible solution thus produced. We denote by $\gamma$ the value of the solution returned by $H : \gamma = \min\{c_{ij}, c_{iK}, c_{JK}\}$.

Clearly, $H$ can again be implemented to run in time $O(n^3)$, and $H$ is a $\frac{1}{2}$-approximate algorithm for $T\Delta$ and $S\Delta$. But even more is true:
Theorem 3.6 H is a $\frac{1}{3}$-approximate algorithm for problem $T\Delta$. Moreover, there exist arbitrary large instances $T$ of $T\Delta$ such that $\gamma = \frac{4}{3} \text{OPT}(T)$.

Proof:
Let $T$ be an instance of $T\Delta$, and $F$ an optimal solution of $T$. As in the proof of Theorem 3.4, we obtain inequalities (3.8), (3.9), as well as the symmetric inequality:

$$c_{IK} \leq 2 \sum_{(i,j,k) \in F} (d_{ik} + \delta_{jk}). \quad (3.18)$$

Summing up (3.8), (3.9) and (3.18) yields:

$$3\gamma \leq 2c_{ij} + c_{IK} \leq 4 \sum_{(i,j,k) \in F} t_{ijk} = 4 \text{OPT}(T), \quad (3.19)$$

which proves that $H$ is a $\frac{1}{3}$-approximate algorithm.

Equality in (3.19) is achieved by the instance $T$ depicted in Figure 3.3.

![Figure 3.3](image)

Here, $I = \{i_1, i_2, i_3\}$, $J = \{j_1, j_2, j_3\}$, $K = \{k_1, k_2, k_3\}$. The lengths $d_{uv}$ are indicated next to the edges of the "pyramid", with $d_{uv} = 0$ if $u = v$. It is easy to see that $T$ is an instance of $T\Delta$. Moreover, because $T$ is symmetric on $I, J$ and $K$, we can assume that $\gamma = c_{11} = c_{IK} = c_{JK}$. 
Section 3.3

An optimal solution of $\mathcal{T}$ is given by $F = \{(i_1,j_2,k_3), (i_2,j_3,k_1), (i_3,j_1,k_2)\}$ with $\text{OPT}(\mathcal{T}) = 6$. But $H_{IJ}$ can return a solution with cost $c_{IJ} = 8$, by picking $M = \{(i_1,j_1), (i_2,j_3), (i_3,j_2)\}$ in the first phase, and $A = \{(i_1,j_1,k_1), (i_2,j_3,k_2), (i_3,j_2,k_3)\}$ in the second phase.

Notice that we actually proved a little bit more than announced by the statement of Theorem 3.6. Indeed, inequality (3.19) shows that the minimum of any two of the bounds $c_{IJ}, c_{IK}$ and $c_{JK}$ is already bounded by $\frac{4}{3} \text{OPT}(\mathcal{T})$. On the other hand, one can exhibit examples for which $c_{IJ} = c_{IK} = \frac{4}{3} \text{OPT}(\mathcal{T})$, and $c_{JK} = \text{OPT}(\mathcal{T})$. Thus, heuristic $H$ is in general better than the strategy which consists in computing only two of the bounds $c_{IJ}, c_{IK}, c_{JK}$, and retaining the best one.

The same remarks apply to our next result:

**Theorem 3.7** $H$ is a $\frac{1}{3}$-approximate algorithm for problem $S\Delta$. Moreover, there exist arbitrary large instances $S$ of $S\Delta$ such that $\gamma = \frac{4}{3} \text{OPT}(S)$.

**Proof:**
Let $S$ be an instance of $S\Delta$, and $F$ be an optimal solution of $S$. Summing up inequalities (3.14), (3.15) and

$$c_{IK} \leq \sum_{(i,j,k) \in F} (d_{ik} + d_{jk}),$$

we get:

$$3\gamma \leq 2c_{IJ} + c_{IK} \leq 2 \sum_{(i,j,k) \in F} (d_{ij} + d_{ik} + d_{jk}). \tag{3.20}$$

Using the triangle inequalities to bound the right-hand side of (3.20) yields:

$$3\gamma \leq 4 \sum_{(i,j,k) \in F} s_{ijk} = 4 \text{OPT}(S). \tag{3.21}$$

A worst-case instance $S$ is represented in Figure 3.4.
Figure 3.4

All edges of this prism have length 1, and the distances are Euclidean.

The optimal solution \( \{(i_1, j_1), (i_2, j_2, k_3), (i_3, j_3, k_2)\} \) has cost \( \text{OPT} \) = 3. The heuristic \( H_{11} \) may return \( M = \{(i_1, j_1), (i_2, j_2), (i_3, j_3)\} \) in phase 1, and \( A = \{(i_1, j_1, k_1), (i_2, j_2, k_2), (i_3, j_3, k_3)\} \) in phase 2, for a total cost \( c_{11} = 4 \). Hence, by symmetry, \( \gamma = 4 \) is possible.

\[\square\]

3.4 Computational results

Of course, the quality of a heuristic cannot only be judged by its worst-case performance. Very often, it is the case that this worst-case performance is determined by pathological instances of the problem. With this in mind, we conducted some numerical experiments to better assess the quality of the various approximation algorithms discussed in Section 3.3.

For fixed \( n = |I| = |J| = |K| \), we considered random problems of three different types.

Type I The elements of \( I \cup J \cup K \) are generated at random, uniformly in the square \([0, 1] \times [0, 1]\). For each pair of points \((u, v)\), \( d_{uv} \) is the Euclidean distance from \( u \) to \( v \) (we also used for \( d_{uv} \) the Manhattan distance from \( u \) to \( v \), with results similar to those displayed below).

Instances of type I form a "natural" class of random instances for problems \( T \Delta \) or \( S \Delta \). But, due to their high degree of uniformity, one may expect
these instances to be easy to solve for most heuristics when $n$ grows large. The next two types of instances are meant to be more "irregular", and hence more difficult to solve.

**Type II** The elements of $I$ are generated uniformly in $[0, \frac{1}{3}] \times [0, 1]$, those of $J$ in $[\frac{1}{3}, 1] \times [0, \frac{1}{3}]$, those of $K$ in $[\frac{1}{3}, 1] \times [\frac{1}{3}, 1]$. The distances $d_{uv}$ are Euclidean.

**Type III** We fix a parameter $p \in [0, 1]$. Then, for each pair $(u, v)$, we let $d(u, v) = 1$ with probability $p$, and $d(u, v) = 2$ with probability $1 - p$. The value of $p$ was empirically adjusted so as to produce rather difficult problem instances for our heuristics.

For each problem type, we report in Tables 3.1, 3.2 on the solution of three instances with $n = 33$ and three instances with $n = 66$ (more instances were actually tested, but the results displayed here are representative). The problems of type III were generated with $p = \frac{1}{17}$ for $n = 33$ and $p = \frac{1}{50}$ for $n = 66$. Table 3.1 deals with problem $TA$ and Table 3.2 with problem $S\Delta$.

For the sake of comparison, we also give in Table 3.1 and 3.2 a lower-bound $lb$ on the optimal value of each instance, as well as the value of the ratio $\frac{\gamma}{lb}$. The bound $lb$ was computed using a Lagrangean relaxation scheme and subgradient optimization, as proposed by Frieze and Yadegar (1981).
<table>
<thead>
<tr>
<th>Type</th>
<th>n</th>
<th>$c_{ij}$</th>
<th>$c_{ik}$</th>
<th>$c_{jk}$</th>
<th>$\gamma$</th>
<th>lb</th>
<th>$\gamma/lb$</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>33</td>
<td>16.18</td>
<td>16.36</td>
<td>16.55</td>
<td>16.18</td>
<td>16.07</td>
<td>1.007</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>14.16</td>
<td>14.16</td>
<td>14.11</td>
<td>14.11</td>
<td>13.95</td>
<td>1.011</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>16.09</td>
<td>16.33</td>
<td>16.32</td>
<td>15.09</td>
<td>16.04</td>
<td>1.003</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>26.92</td>
<td>26.87</td>
<td>26.68</td>
<td>26.68</td>
<td>26.54</td>
<td>1.005</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>25.00</td>
<td>24.81</td>
<td>24.69</td>
<td>24.69</td>
<td>24.33</td>
<td>1.015</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>28.13</td>
<td>27.75</td>
<td>27.91</td>
<td>27.75</td>
<td>27.48</td>
<td>1.010</td>
</tr>
<tr>
<td>II</td>
<td>33</td>
<td>48.83</td>
<td>48.61</td>
<td>48.75</td>
<td>48.61</td>
<td>47.72</td>
<td>1.019</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>51.72</td>
<td>51.42</td>
<td>51.49</td>
<td>51.42</td>
<td>50.35</td>
<td>1.021</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>43.52</td>
<td>43.83</td>
<td>44.01</td>
<td>43.52</td>
<td>42.60</td>
<td>1.022</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>98.09</td>
<td>97.80</td>
<td>99.15</td>
<td>97.80</td>
<td>96.33</td>
<td>1.015</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>91.47</td>
<td>91.60</td>
<td>91.42</td>
<td>91.42</td>
<td>88.31</td>
<td>1.035</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>99.39</td>
<td>98.88</td>
<td>99.57</td>
<td>98.88</td>
<td>96.70</td>
<td>1.023</td>
</tr>
<tr>
<td>III</td>
<td>33</td>
<td>140</td>
<td>135</td>
<td>136</td>
<td>135</td>
<td>133</td>
<td>1.015</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>141</td>
<td>137</td>
<td>139</td>
<td>137</td>
<td>130</td>
<td>1.054</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>135</td>
<td>136</td>
<td>137</td>
<td>135</td>
<td>130</td>
<td>1.038</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>295</td>
<td>293</td>
<td>296</td>
<td>293</td>
<td>283</td>
<td>1.035</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>294</td>
<td>298</td>
<td>294</td>
<td>294</td>
<td>281</td>
<td>1.046</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>295</td>
<td>296</td>
<td>293</td>
<td>293</td>
<td>280</td>
<td>1.046</td>
</tr>
</tbody>
</table>

Table 3.1 Problem $T\Delta$. 
<table>
<thead>
<tr>
<th>Type</th>
<th>n</th>
<th>$c_{ij}$</th>
<th>$c_{ik}$</th>
<th>$c_{jk}$</th>
<th>$\gamma$</th>
<th>$lb$</th>
<th>$\gamma/lb$</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>33</td>
<td>8.57</td>
<td>8.69</td>
<td>8.64</td>
<td>8.57</td>
<td>8.45</td>
<td>1.014</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>7.61</td>
<td>7.54</td>
<td>7.53</td>
<td>7.53</td>
<td>7.39</td>
<td>1.019</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>8.43</td>
<td>8.55</td>
<td>8.58</td>
<td>8.43</td>
<td>8.37</td>
<td>1.007</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>14.23</td>
<td>14.31</td>
<td>14.09</td>
<td>14.09</td>
<td>13.90</td>
<td>1.014</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>13.38</td>
<td>13.13</td>
<td>13.13</td>
<td>13.13</td>
<td>12.84</td>
<td>1.023</td>
</tr>
<tr>
<td>II</td>
<td>33</td>
<td>26.54</td>
<td>26.65</td>
<td>27.32</td>
<td>26.54</td>
<td>25.96</td>
<td>1.022</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>23.78</td>
<td>23.79</td>
<td>24.21</td>
<td>23.78</td>
<td>23.11</td>
<td>1.029</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>53.86</td>
<td>54.05</td>
<td>55.90</td>
<td>53.86</td>
<td>53.12</td>
<td>1.014</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>49.29</td>
<td>49.47</td>
<td>50.43</td>
<td>49.29</td>
<td>47.88</td>
<td>1.029</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>54.70</td>
<td>54.84</td>
<td>56.19</td>
<td>54.70</td>
<td>53.52</td>
<td>1.022</td>
</tr>
<tr>
<td>III</td>
<td>33</td>
<td>75</td>
<td>71</td>
<td>71</td>
<td>71</td>
<td>69</td>
<td>1.029</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>75</td>
<td>72</td>
<td>72</td>
<td>72</td>
<td>67</td>
<td>1.075</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>71</td>
<td>72</td>
<td>72</td>
<td>71</td>
<td>67</td>
<td>1.060</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>163</td>
<td>161</td>
<td>165</td>
<td>161</td>
<td>151</td>
<td>1.066</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>163</td>
<td>167</td>
<td>164</td>
<td>163</td>
<td>152</td>
<td>1.072</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>163</td>
<td>165</td>
<td>161</td>
<td>161</td>
<td>147</td>
<td>1.095</td>
</tr>
</tbody>
</table>

**Table 3.2** Problem $S\Delta$.

The results exhibited in these tables indicate that, from a practical viewpoint, the heuristics presented in Section 3.3 perform quite satisfactorily. In particular, heuristic $H$ solved all randomly generated instances within 10\% of optimality, and often came within 3\% of the optimal value (or, more precisely, of the lower-bound $lb$).

**Acknowledgements**

The authors wish to thank Koos Vrieze for his insightful suggestions, which led to the discovery of the worst-case examples presented in the proofs of Theorems 3.6 and 3.7, and Hans-Jürgen Bandelt for his comments on this chapter.
Chapter 4

Approximation algorithms for multidimensional assignment problems with decomposable costs

This chapter is joint work with Hans-Jürgen Bandelt and Yves Crama; it will be published as an article in Discrete Applied Mathematics.
4.1 Introduction

For \( k \geq 2 \), the \( k \)-dimensional assignment problem is formulated as follows. Let \( X_0, \ldots, X_{k-1} \) be pairwise disjoint sets of equal cardinality, say \( p \). Regarding \( V = \cup_i X_i \) as the vertex-set of a complete \( k \)-partite graph with edge-set \( E = \{ \{ u, v \} \mid u \in X_i, v \in X_j \} \), we say that a subset \( X \) of \( V \) is a clique of the graph \( (V, E) \) if \( X \) meets every \( X_i \) in exactly one vertex \( (i = 0, \ldots, k-1) \). A \( (k\text{-dimensional}) \) assignment of \((V, E)\) is a partition of \( V \) into cliques, that is, a collection of \( p \) pairwise disjoint cliques of \((V, E)\); we will also call this an assignment between \( X_0, X_1, \ldots, X_{k-1} \). Let now \( c \) be any real-valued cost function defined on the set of cliques of \((V, E)\). The \( k \)-dimensional assignment problem on \((V, E)\) with respect to \( c \) consists in finding an assignment \( M \) of minimum cost, where the cost of \( M \) is defined as \( c(M) = \sum_{X \in M} c(X) \).

When \( k = 2 \), the \( k \)-dimensional assignment problem is nothing but the well-known bipartite weighted matching problem and can be solved in \( O(p^3) \) arithmetic operations by the Hungarian method (see e.g. Papadimitriou and Steiglitz (1982)). Throughout this chapter, we refer to 2-dimensional assignments as to matchings.

The 3-dimensional assignment problem has also been actively investigated in the literature; see e.g. the references contained in Balas and Saltzman (1989), as well as Chapter 3 of this thesis, Frieze (1974), Hansen and Kaufman (1973); it is well-known to be \( \mathcal{NP} \)-hard (Karp (1972)). The \( k \)-dimensional assignment problem has been less thoroughly studied for values of \( k \geq 4 \), and this, in spite of the fact that it constitutes a most natural generalization of the 2- and 3-dimensional cases. Early mention of the problem can be found in Pierskalla (1968), where applications are described.

In this chapter, we concentrate on the restricted version of the \( k \)-dimensional assignment problem which arises when the cost of a clique is not completely arbitrary, but is rather a function of elementary costs attached to the edges of the complete \( k \)-partite graph \((V, E)\). Specifically, let us assume from now on that \( d \) is a nonnegative length function defined on \( E \), and not identically zero; for the sake of simplicity, we use the shorthand \( d(u, v) \) instead of \( d(\{u, v\}) \), and we let \( d(v, v) = 0 \), by convention. We say that the cost function \( c \) is decomposable if there exists a function \( f : R^{[E]} \rightarrow R \) such that, for every clique \( X = \{x_0, \ldots, x_{k-1}\} \),

\[
c(X) = f(d(x_0, x_1), d(x_0, x_2), \ldots, d(x_{k-2}, x_{k-1})).
\]

Different variants of the three-dimensional assignment problem with decom-
Section 4.1

Section 4.1

posable costs arise in applications considered by Frieze and Yadegar (1981) or Chapter 2 of this thesis, and are further investigated in Chapter 3 of this thesis.

The main goal of this chapter is to present some simple heuristics for the k-dimensional assignment problem with decomposable costs and, for various cost functions, to derive worst-case bounds on the ratio between the cost of the heuristic assignments and the cost of an optimal solution.

In the next section, we describe the specific decomposable cost functions which will be considered in the remainder of the chapter. In Section 4.3, we propose some heuristics for the k-dimensional assignment problem with decomposable costs, and we state our main results about the worst-case performance of these heuristics. Proofs of these results are to be found in Sections 4.4 to 4.8 (Sections 4.4 to 4.7 deal with a first type of heuristics and four different cost functions, while Section 4.8 focusses on a second type of heuristics).

The worst-case bounds are stated in terms of two parameters, viz. k and r, where r is the smallest real number for which the following condition holds: for all \( \{u, v\}, \{u, w\}, \{v, w\} \in E, \)

\[
d(u, v) \leq r(d(u, w) + d(v, w)).
\] (4.1)

Observe that r is well-defined, except when there exist three edges \( \{u, v\}, \{u, w\}, \{v, w\} \) such that \( d(u, v) > 0 \) and \( d(u, w) = d(v, w) = 0 \). In the latter case, we let \( r = \infty \).

Clearly, \( r \geq \frac{1}{2} \), and \( r = \frac{1}{2} \) exactly when d is constant on E. If \( r \leq 1 \), then the edge lengths satisfy the usual triangle inequality. More generally, we call (4.1) the \( r \)-relaxed triangle inequality, or \( r \)-inequality for short.

Successive application of the \( r \)-inequality to a sequence of edges \( \{u_i, u_{i+1}\} \in E \) (\( i = 0, \ldots, m - 1 \)) leads to the following \textit{iterated} \( r \)-inequality in case \( r \geq 1 \):

\[
d(u_0, u_m) \leq r \left[ \log m \right], \sum_{i=0,\ldots,m-1} d(u_i, u_{i+1}),
\] (4.2)

where \( \log \) denotes the logarithm to the base 2, and \( \left[ \lambda \right] \), the “ceiling” of a real number \( \lambda \), is the smallest integer greater than or equal to \( \lambda \). In order to verify this inequality by induction, one may assume that \( m = 2^s \) (as \( r \geq 1 \)); then indeed (for \( s \geq 2 \))

\[
d(u_0, u_{2^s}) \leq r(d(u_0, u_{2^s-1}) + d(u_{2^s-1}, u_{2^s}))
\leq r \cdot r^{s-1} \sum_{i=0,\ldots,2^s-1} d(u_i, u_{i+1}).
\]
4.2 Some decomposable cost functions

We now introduce some of the specific decomposable cost functions which will be treated in the remainder of the chapter. Our initial motivation for considering these cost functions stems from the application described in Chapter 2 of this thesis. Let us simply say here that, in that application, the cost of a clique should somehow reflect the total distance travelled by the robot-arm of a machine in order to visit all vertices of the clique. Since the order in which the vertices will eventually be visited is not known in advance, the distance travelled can only be roughly evaluated. This can be done in several ways.

**Sum costs.** The sum cost function assigns to every clique $X = \{x_0, \ldots, x_{k-1}\}$ a cost equal to the sum of its edge lengths, i.e.:

$$c(X) = \sum_{0 \leq i < j \leq k-1} d(x_i, x_j). \quad (4.3)$$

Observe that, since every clique contains the same number of vertices, the cost of a clique with respect to the cost function (4.3) is proportional to the average length of the edges induced by the clique.

Sum cost functions are commonly used in the context of graph partitioning problems, such as those mentioned e.g. in Lengauer (1990). Actually, the $k$-dimensional assignment problem with sum costs can be seen as a special case of the $k$-dimensional matching problem studied by Feo and Khellaf (1990): given an (arbitrary) graph $G$ with $kp$ vertices, and a nonnegative weight for each edge $\{u, v\}$ of $G$, find a partition of the vertex-set of $G$ into $p$ sets $V_1, \ldots, V_p$ such that the sum of the weights of the edges contained in $\cup_i V_i$ is maximized. Feo and Khellaf (1990) present heuristics with guaranteed worst-case performance for this $k$-dimensional problem. But their bounds have little meaning for our problem, due to the fact that we stated it as a minimization rather than a maximization problem.

**Star costs.** The cost of a clique $X = \{x_0, \ldots, x_{k-1}\}$ with respect to the star cost function is equal to the sum of the edge lengths of a minimum length spanning star of $X$, i.e.:

$$c(X) = \min\{ \sum_{0 \leq i \leq k-1} d(x_h, x_i) | 0 \leq h \leq k - 1 \}. \quad (4.4)$$

In the framework of location theory, the vertex $x_h$ realizing the minimum in
Section 4.3

(4.4) is called the median vertex of the clique.

Tour costs. The tour cost of the clique \( X = \{x_0, \ldots, x_{k-1}\} \) is defined as the cost of a traveling salesman tour on the graph induced by \( X \), in other words:

\[
c(X) = \min \{ \sum_{0 \leq i \leq k-1} d(x_{\pi(i)}, x_{\pi(i+1)}) \mid \pi \text{ is a permutation of } \{0,1,\ldots,k-1\} \},
\]

where integers are read modulo \( k \).

Tree costs. The cost of the clique \( X = \{x_0, \ldots, x_{k-1}\} \) with respect to the tree cost function is the cost of a minimum length spanning tree of the complete graph on \( X \), i.e.:

\[
c(X) = \min \{ \sum_{(i,j) \in A} d(x_i, x_j) \mid (X, A) \text{ is a tree } \}.
\]

Observe that, for \( k = 3 \), the sum and the tour cost functions are identical, as are the star and the tree cost functions. In Chapter 3 it is proved that the 3-dimensional assignment problem with sum or star costs is \( \mathcal{NP} \)-hard, even when \( d \) satisfies the triangle inequality.

4.3 Approximation algorithms and their performance

Suppose from now on that we have to solve an instance of the \( k \)-dimensional assignment problem on \((V, E)\) with respect to a decomposable cost function \( c \), depending on a length function \( d \). We denote by \( M_{\text{opt}} \) an optimal solution of this instance.

We propose in this section various heuristics for this problem, and we state our main results about the quality of the solutions which they produce. Namely, we state theorems of the form: "for all instances of the \( k \)-dimensional assignment problem with decomposable costs satisfying such and such assumptions, the heuristic under consideration produces an assignment \( M \) such that \( c(M) \leq \alpha(k, \tau) \cdot c(M_{\text{opt}}) \)", where \( \alpha(k, \tau) \) is an (explicitly given) function of \( k \) and \( \tau \). We sometime add to this that "the statement is sharp", meaning that there exist instances of the problem satisfying the required assumptions and yielding \( c(M) = \alpha(k, \tau) \cdot c(M_{\text{opt}}) \). Notice that, for
the sake of clarity, the theorems are not always stated in the full generality with which they will be proved in subsequent sections.

Our heuristics fall into two classes, namely "hub heuristics", and "recursive heuristics". As we will see, the main ingredient of these procedures consists in solving a sequence of bipartite matching problems on graphs with 2p vertices. As an estimate of the complexity of the heuristics, we will therefore use the number of matching problems which they require to solve. Observe that, in particular, we do not explicitly take into account the time required to compute the cost of a clique in some of the heuristics. This is reasonable for all but the tour cost function; and even the cost of a tour can be quickly computed when k is small.

**Hub heuristics.** We begin with a description of the single-hub heuristic. There is actually one such heuristic for each $h \in \{0, \ldots, k-1\}$; $h$ is then the "hub" of the heuristic. The single-hub heuristic with hub $h$ produces an assignment which is composed of minimum length matchings between $X_h$ and all other parts $X_i$. It proceeds as follows:

**Step 1.** For each $i \neq h$, let $M_{hi}$ be a minimum length matching between $X_h$ and $X_i$ with respect to the length function $d$.

**Step 2.** Return the $k$-dimensional assignment $M_h := \{x_0, \ldots, x_{k-1}\}$ with

$x_h \in X_h$ and $(x_h, x_i) \in M_{hi}, i = 0, \ldots, k-1, i \neq h$.

The single-hub heuristic is clearly polynomial: it only requires the solution of $k-1$ bipartite matching problems. Notice also that the heuristic is quite simple-minded, as it only depends on $d$, and not on the specific cost function $c$ built up from $d$. But in spite of this simplicity, the quality of the solution delivered by the single-hub heuristic cannot be arbitrarily bad, as is attested by the next statement:

**Theorem 4.1** If $c$ is either the sum or the star or the tour or the tree cost function, then $c(M_h) \leq (k-1)\cdot c(M_{\text{opt}})$ for every problem instance satisfying the triangle inequality. This statement is sharp. (See (4.8), (4.19), (4.22), and (4.28) below.)

More precisely, we will establish in Sections 4.4 to 4.7 that, for each of the four cost functions mentioned in Theorem 4.1, there exists a function $\alpha(k, r)$ such that $c(M_h) \leq \alpha(k, r)\cdot c(M_{\text{opt}})$ and $\alpha(k, r) \leq k-1$ when $r \leq 1$. The function $\alpha(k, r)$ grows (at most) linearly with $r$, for the sum and the star cost functions.
Section 4.3

An easy way of improving the single-hub heuristic is to compute a solution $M_h$ for each possible choice of the hub $h$, and to retain the best solution thus found. We then obtain the following multiple-hub heuristic:

**Step 1.** For each pair $i, h \in \{0, \ldots, k-1\}, i \neq h$, let $M_{hi}$ be a minimum length matching between $X_h$ and $X_i$ with respect to the length function $d$.

**Step 2.** For each $h \in \{0, \ldots, k-1\}$, let $M_h$ be the assignment delivered by the single-hub heuristic with hub $h$, that is, $M_h := \{ (x_0, \ldots, x_{k-1}) \mid x_h \in X_h \text{ and } (x_k, x_i) \in M_{hi}, i = 0, \ldots, k-1, i \neq h \}$.

**Step 3.** Return the assignment $M_H := M_0$, where $c(M_i) = \min \{ c(M_h) \mid 0 \leq h \leq k-1 \}$.

This heuristic requires the solution of $k(k-1)/2$ bipartite matching problems. The upper-bound provided by Theorem 4.1 is trivially valid for $c(M_H)$, but even stronger results can be proved. For instance, we will show:

**Theorem 4.2** If $c$ is either the sum or the star cost function, then $c(M_H) \leq (2/k)[(k - 2)\tau + 1] \cdot c(M_{opt})$ for every problem instance. This statement is sharp when $\tau = 1$. (See (4.10) and (4.16) below.)

Observe that, for $\tau \leq 1$, the ratio $c(M_H)/c(M_{opt})$ is bounded by 2. In the terminology of Papadimitriou and Steiglitz (1982), this means that the multiple-hub heuristic is a 1-approximation algorithm for the $k$-dimensional assignment problem with either sum or star costs, when the edge lengths satisfy the triangle inequality (independently of the value of $k$). On the other hand, if $\tau$ is not bounded from above (or $\tau = \infty$), then there exists no polynomial-time $\varepsilon$-approximation algorithm for the $k$-dimensional assignment problem restricted to sum or star costs, for any fixed $\varepsilon \geq 0$ and any $k \geq 3$ (unless, of course, $\mathcal{P} = \mathcal{NP}$); see Chapter 3 of this thesis. As a matter of fact, our theorem only implies that $c(M_H)/c(M_{opt}) \leq 2\tau$.

**Theorem 4.3** If $c$ is either the tour or the tree cost function, then $c(M_H) \leq \frac{1}{2} k \cdot c(M_{opt})$ if $k$ is even and $c(M_H) \leq \frac{1}{2} (k - 1) \cdot c(M_{opt})$ if $k$ is odd, for every problem instance satisfying the triangle inequality. This statement is sharp. (See (4.25) and (4.30) below.)

A more precise, but much more intricate statement can again be obtained by dropping the assumption that the length function satisfies the triangle inequality, and by explicitly taking into account the parameter $\tau$. What
may be more important here is to observe that the ratio \( c(M_H) / c(M_{opt}) \) grows linearly with \( k \), just as for the single-hub heuristic. We do not know whether there exists a polynomial-time \( \varepsilon \)-approximation algorithm, with \( \varepsilon \) an absolute constant, for the \( k \)-dimensional assignment problem restricted to either tour or tree costs, and to edge lengths satisfying the triangle inequality.

**Recursive heuristics.** We propose a single-pass recursive heuristic with respect to a chosen permutation of \( (X_0, \ldots, X_{k-1}) \), say e.g. \( (X_{\sigma(0)}, \ldots, X_{\sigma(k-1)}) \). The idea behind this heuristic is again to produce an assignment of \( (V, E) \) by solving a sequence of \( k - 1 \) bipartite matching problems. The solution of the \( (i - 1) \)st subproblem produces an \( i \)-dimensional assignment between \( X_{\sigma(0)}, X_{\sigma(1)}, \ldots, X_{\sigma(i-1)} \), say \( N_{i-1} \). The \( i \)-th subproblem consists in extending this partial assignment, by computing an optimal bipartite matching between its cliques (regarded as indivisible) and the vertices of \( X_{\sigma(i)} \), with respect to suitably defined edge lengths. The length function for the \( (i - 1) \)st subproblem \((i \geq 3)\) is defined as follows: for all \( X \in N_{i-1} \) and \( u \in X_{\sigma(i)} \), the length of \( \{X, u\} \) is the cost \( c(X \cup \{u\}) \) of the "partial" clique \( X \cup \{u\} \); notice that this definition only makes sense if the cost function \( c \) can be meaningfully extended to subsets of cliques; this is certainly the case for the four cost functions introduced in Section 4.2 since these are well-defined for all \( k \geq 3 \).

We are now ready for a formal description of the \textit{single-pass recursive heuristic} associated to the permutation \( (X_{\sigma(0)}, \ldots, X_{\sigma(k-1)}) \).

**Step 1.** Let \( N_0 := X_{\sigma(0)} \) and \( i := 1 \).

**Step 2.** For all \( X \in N_{i-1} \) and \( u \in X_{\sigma(i)} \), let \( \delta(X, u) := c(X \cup \{u\}) \).

**Step 3.** Let \( N \) be a minimum length matching between \( N_{i-1} \) and \( X_{\sigma(i)} \) with respect to the length function \( \delta \). Let \( N_i := \{X \cup \{u\} | X \in N_{i-1}, u \in X_{\sigma(i)} \} \).

**Step 4.** If \( i < k - 1 \), let \( i := i + 1 \) and go to Step 2. Else, continue.

**Step 5.** Return the assignment \( M_c := N_{k-1} \).

The complexity of the single-pass recursive heuristic is comparable to that of the single-hub heuristic since in both cases \( k - 1 \) bipartite matching problems need to be solved. On the other hand, the recursive heuristic turns out to be much more difficult to analyze than the hub heuristic. In fact, the results to be presented in Section 4.8 will only have bearing on one of our "special" cost functions, namely the sum cost function. We will establish:
Section 4.3

Theorem 4.4 If $c$ is the sum cost function, then $c(M_\sigma) \leq \frac{1}{2}k \cdot c(M_{\text{opt}})$ for every problem instance satisfying the triangle inequality. (See (4.35) and (4.37) below.)

Comparing this result with Theorem 4.1, we see that, in the case of sum costs, the worst-case performance of the single-pass recursive heuristic is better than that of the single-hub heuristic. For $k = 3$, the statement of Theorem 4.4 is known to be valid and sharp, both in the case of sum and of star costs (Chapter 3 of this thesis); but it is no longer sharp for $k \geq 4$, by the next theorem. In particular, we do not even know whether the ratio $c(M_\sigma)/c(M_{\text{opt}})$ really grows linearly in $k$ (although we suspect that it does).

Theorem 4.5 If $c$ is the sum cost function, then $c(M_\sigma) \leq \frac{1}{k^2}c(M_{\text{opt}})$ for all instances of the $4$-dimensional assignment problem satisfying the triangle inequality. This statement is sharp.

Just as we did for the single-hub heuristic, we can improve the performance of the single-pass recursive heuristic by applying it for each possible permutation of $(X_0, \ldots, X_{k-1})$, and retaining the best solution thus found. This results in the multiple-pass recursive heuristic:

Step 1. For each permutation $\sigma$ of $\{0, \ldots, k-1\}$, let $M_\sigma$ be the assignment produced by the single-pass recursive heuristic associated to the permutation $(X_{\sigma(0)}, \ldots, X_{\sigma(k-1)})$.

Step 2. Return the assignment $M_R := M_\pi$, where $c(M_\pi) = \min\{c(M_\sigma)|\sigma$ is a permutation of $\{0, \ldots, k-1\}\}$.

The complexity of the multiple-pass recursive heuristic is quite high, since $k!(k-1)/2$ bipartite matching problems have to be solved in Step 1. As for its performance, we will prove:

Theorem 4.6 If $c$ is the sum cost function, then $c(M_R) \leq 2[(k+1)/(k-1)] \cdot \ln(k+1) \cdot c(M_{\text{opt}})$ for every problem instance satisfying the triangle inequality. (See (4.44) and (4.47) below.)

For $k = 3$, it is known that $c(M_R)/c(M_{\text{opt}}) \leq 4/3$ and that the result is sharp (for both sum and star costs; see Chapter 3 of this thesis). This is the same worst-case performance as for the multiple-hub heuristic (see Theorem 4.2). More generally, in view of the sophistication of the multiple-pass recursive heuristic, one may expect that its worst-case behavior should be at least as good as that of the multiple-hub heuristic. At the very least, one would
hope that $c(M_R)/c(M_{opt})$ be bounded by a constant. The statement of Theorem 4.6 is disappointing in this regard (but maybe is not sharp!). We will nevertheless be able to prove that $c(M_R)/c(M_{opt}) \leq \tau/(1 - \tau)$ (i.e. a bound independent of $k$) for all problem instances such that $\tau < 1$ (see (4.45) below).

In the following sections, we proceed with the proofs of our results.

## 4.4 Sum costs

We assume here that the cost of a clique $X = \{x_0, \ldots, x_{k-1}\}$ equals the sum of all edge lengths $d(x_i, x_j)$, that is,

$$c(X) = \sum_{0 \leq i < j \leq k-1} d(x_i, x_j).$$

We first establish an upper bound on the cost of the assignment $M_h$ produced by the single-hub heuristic ($h = 0, \ldots, k-1$), which is composed of minimum length matchings $M_{hi}$ between $X_h$ and all other parts $X_i$. Applying the $r$-inequality

$$d(x_i, x_j) \leq \tau(d(x_h, x_i) + d(x_h, x_j)),$$

for all $X \in M_h$ and $i, j \neq h$, we obtain

$$c(M_h) = \sum_{X \in M_h} \sum_{i < j} d(x_i, x_j)$$

$$\leq \sum_{X \in M_h} \sum_{i=0, \ldots, k-1} [(k - 2)r + 1] \cdot d(x_h, x_i)$$

$$= [(k - 2)r + 1] \cdot \sum_{i=0, \ldots, k-1} \sum_{X \in M_h} d(x_h, x_i)$$

$$\leq [(k - 2)r + 1] \cdot \sum_{i=0, \ldots, k-1} \sum_{Z \in M_{opt}} d(x_h, z_i)$$

(4.7)

as each $M_{hi}$ is a minimum length matching. The sum of $d(x_h, x_i)$ for $i = 0, \ldots, k - 1$ is a trivial lower bound of $c(Z)$, whence

$$c(M_h) \leq [(k - 2)\tau + 1] \cdot c(M_{opt}).$$

(4.8)

**Example 4.1** To show that this bound is sharp for all $\tau \geq 1$, consider Figure 4.1:
Let $x_{ij}, j = 0, \ldots, k - 1$ denote the $j$-th vertex of $X_i$, or in other words:

$$X_i = \{x_{ij}| j = 0, \ldots, k - 1\} \text{ for } i = 0, \ldots, k - 1.$$  

Observe that in this example each $X_i$ has cardinality $k$, that is $p = k$. Let us now define the length function $d$. The notation $x_{ij} = x_{ir}$ in Figure 4.1 (and in all subsequent figures) indicates that $d(x_{ij}, x_{ir}) = 0$. The length between any two vertices linked by an edge in Figure 4.1 is equal to 1. All other lengths are equal to $2r$, for some $r \geq 1$. Note that the $r$-inequality is fulfilled by this distance function. The single-hub heuristic with hub 0 may find the following assignment (where indices are read modulo $k$):

$$M_0 = \{\{x_{0j}, x_{1,j+1}, x_{2,j+2}, \ldots, x_{k-1,j+k-1}\}| j = 0, \ldots, k - 1\} =$$

$$= \{\{x_{00}, x_{11}, \ldots, x_{k-1,k-1}\}, \{x_{01}, x_{12}, \ldots, x_{k-1,0}\},$$

$$\ldots, \{x_{0,k-1}, x_{10}, \ldots, x_{k-1,k-2}\}\}$$

resulting in

$$c(M_0) = k[(k - 1) + r(k - 1)(k - 2)]$$

$$= k(k - 1)((k - 2)r + 1).$$

It is not difficult to see that

$$M_{opt} = \{\{x_{0j}, x_{1,j}, \ldots, x_{k-1,j}\}| j = 0, \ldots, k - 1\}$$

$$= \{\{x_{00}, x_{10}, \ldots, x_{k-1,0}\}, \{x_{01}, x_{11}, \ldots, x_{k-1,1}\},$$

$$\ldots, \{x_{0,k-1}, x_{1,k-1}, \ldots, x_{k-1,k-1}\}\}$$
with \( c(M_{\text{opt}}) = k(k-1) \) proving that the bound in (4.8) is sharp for \( r \geq 1 \).

The bound (4.8) is certainly not sharp when \( r \) is smaller than 1 because \( d(z_i, z_j) \) cannot be zero for \( i \neq j \). Indeed, using the \( r \)-inequality, we have (given \( Z \in M_{\text{opt}} \))

\[
d(z_i, z_j) \geq \frac{1}{r} \cdot d(z_h, z_i) - d(z_h, z_j)
\]

for \( i \neq j \) and \( i, j \neq h \). Consequently,

\[
\sum_{j=0, \ldots, k-1 \atop j \neq h} d(z_i, z_j) \geq \frac{k-2}{r} \cdot d(z_h, z_i) - \sum_{j=0, \ldots, k-1 \atop j \neq i} d(z_h, z_j) \quad \text{for } i \neq h,
\]

and further

\[
\sum_{i,j \neq h} d(z_i, z_j) \geq \frac{k-2}{r} \cdot \sum_{i=0, \ldots, k-1} d(z_h, z_i) - (k-1) \cdot \sum_{j=0, \ldots, k-1} d(z_h, z_j)
\]

\[
+ \sum_{i=0, \ldots, k-1} d(z_h, z_i)
\]

\[
= (k-2)(\frac{1}{r} - 1) \cdot \sum_{i=0, \ldots, k-1} d(z_i, z_i).
\]

Hence,

\[
[(k-2)(\frac{1}{r} - 1) + 2] \cdot \sum_{i=0, \ldots, k-1} d(z_i, z_i) \leq 2 \cdot \sum_{i<j} d(z_i, z_j).
\]

From this inequality and (4.7) we infer

\[
c(M_h) \leq \frac{2((k-2)r + 1)}{(k-2)(\frac{1}{r} - 1) + 2} \cdot c(M_{\text{opt}}) \quad \text{if } r \leq 1.
\]

(4.9)

Observe that this bound is sharp in the cases \( r = \frac{1}{2} \) (trivially) and \( r = 1 \) (by Example 4.1). It is also sharp for all \( r \leq 1 \) when \( k = 3 \).

**Example 4.2** To see this consider Figure 4.2.
Section 4.4

Here, $X_j = \{x_{j0}, x_{j1}, x_{j2}\}$, for $j = 0, 1, 2$. The lengths between pairs of vertices are indicated along the corresponding edges of Figure 4.2. Note that the $\tau$-inequality is satisfied if $\tau \leq 1$. Clearly, the optimal solution for this problem equals: $M_{\text{opt}} = \{\{x_{00}, x_{10}, x_{20}\}, \{x_{01}, x_{11}, x_{21}\}, \{x_{02}, x_{12}, x_{22}\}\}$, with cost $c(M_{\text{opt}}) = 3(1 + \tau)$. However, the single-hub heuristic with hub 0 may find the following assignment:

$$M_0 = \{\{x_{00}, x_{11}, x_{22}\}, \{x_{01}, x_{12}, x_{20}\}, \{x_{02}, x_{10}, x_{21}\}\}$$

with

$$c(M_0) = 6(\tau + \tau^2).$$

This results in $c(M_0) = 2\tau \cdot c(M_{\text{opt}})$, proving that (4.9) is sharp for all $\tau \leq 1$ when $k = 3$. \hfill \Box
Next we derive from (4.7) an upper bound on the cost of the solution returned by the multiple-hub heuristic, i.e. the cost of a best assignment among $M_0, \ldots, M_{k-1}$:

$$c(M_H) = \min_{h=0, \ldots, k-1} c(M_h)$$

$$\leq \frac{1}{k} \sum_{h=0, \ldots, k-1} c(M_h)$$

$$\leq \frac{1}{k} \sum_{h=0, \ldots, k-1} [(k-2)r + 1] \cdot \sum_{i=0, \ldots, k-1} \sum_{z \in M_{opt}} d(z_i, z_j)$$

$$= \frac{1}{k} [(k-2)r + 1] \cdot \sum_{z \in M_{opt}} \sum_{i,j} d(z_i, z_j)$$

$$= \frac{2}{k} [(k-2)r + 1] \cdot c(M_{opt}). \quad (4.10)$$

This bound is best possible again when $r = \frac{1}{2}$. Observe that

$$\frac{2}{k} [(k-2)r + 1] = 2r \left[ 1 - \frac{1}{k} (2 - \frac{1}{r}) \right] < 2r \quad \text{if } \frac{1}{2} < r.$$

**Example 4.3** In order to prove that the bound (4.10) is also sharp for $r = 1$, consider Figure 4.3.

![Diagram](image)

**Figure 4.3**
The upper vertex in Figure 4.3 is labeled by the points $x_{00}, \ldots, x_{k-1,0}$, while the other $k$ vertices are labeled by the points $x_{0j}, x_{1,j+1}, \ldots, x_{k-1,k-1-j}$, except $x_{k-1,j}(j = 0, \ldots, k-1)$.

The sets $X_0, \ldots, X_{k-1}$ and the length function are defined as for Example 4.1, with $r = 1$. Obviously, an optimal assignment is given by

$$M_{\text{opt}} = \{(x_{i,i+j} : i = 0, \ldots, k-1) | j = 0, \ldots, k-1\}$$

with

$$c(M_{\text{opt}}) = k(k - 1).$$

Since the example is symmetric on $X_0, \ldots, X_{k-1}$, we may assume that the solution returned by the multiple-hub heuristic corresponds to the choice of the hub $h = 0$. Now, for each $i = 1, \ldots, k-1$, an optimal bipartite matching between $X_0$ and $X_i$ is given by:

$$M_{i0} = \{(x_{00}, x_{i0}), \{x_{0k-i}, x_{i0}\} \cup \{x_{0j}, x_{i,i+j} \} | j \neq 0, j \neq k - i\}$$

The cost of $M_{i0}$ is 2 for each $i$. The associated $k$-dimensional assignment $M_H = M_0$ has cost $c(M_H) = 2(k - 1)^2$. Thus, $c(M_H) = 2(1 - \frac{1}{k}) \cdot c(M_{\text{opt}})$, proving that the bound (4.10) is sharp for $r = 1$.

### 4.5 Star costs

Now we assume that the cost of a clique $X = \{x_0, \ldots, x_{k-1}\}$ equals the sum of the edge lengths in a minimum length spanning star of $X$, that is,

$$c(X) = \min \left\{ \sum_{i=0, \ldots, k-1} d(x_h, x_i) : h = 0, \ldots, k-1 \right\}.$$

For every $h$ we obtain

$$c(M_h) \leq \sum_{X \in M_h} \sum_{i=0, \ldots, k-1} d(x_h, x_i)$$

$$= \sum_{i=0, \ldots, k-1} \sum_{X \in M_h} d(x_h, x_i)$$

$$\leq \sum_{i=0, \ldots, k-1} \sum_{z \in M_{\text{opt}}} d(z_h, z_i) \quad (4.11)$$

because each $M_{hi}$ is a minimum length matching. Assume that for a given $Z \in M_{\text{opt}}$. 

\[ c(Z) = \sum_{i=0}^{k-1} d(z_0, z_i). \]

If \( h \neq 0 \), then by the \( \tau \)-inequality we derive
\[
\sum_{i=0}^{k-1} d(z_h, z_i) \leq ((k - 2)\tau + 1) \cdot d(z_0, z_h) + \tau \cdot \sum_{i=0, i \neq h}^{k-1} d(z_0, z_i). \tag{4.12}
\]

So, (4.11) and (4.12) yield the estimation
\[ c(M_h) \leq [(k - 2)\tau + 1] \cdot c(M_{opt}). \tag{4.13} \]

**Example 4.4** To prove that this bound is sharp for \( \tau = 1 \), consider the problem instance described in Example 4.1, with \( \tau = 1 \).

It is easy to see that the optimal solution \( M_{opt} \) as well as the solution \( M_0 \) found by the single-hub heuristic are identical to the ones described there (remember that the single-hub heuristic does not explicitly take into account the cost function \( c \)). With respect to the star costs, we thus have
\[ c(M_0) = k(k - 1) \text{ and } c(M_{opt}) = k. \]

This proves that the bound derived in (4.13) is sharp for \( \tau = 1 \). \( \Box \)

For \( \tau < 1 \) we can again do better than (4.13). First observe that
\[ \frac{1}{\tau} \cdot d(z_0, z_h) \leq d(z_h, z_i) + d(z_0, z_i) \text{ for } i \neq 0, h, \]
and therefore
\[ (1 + (k - 2)\frac{1}{\tau}) \cdot d(z_0, z_h) \leq \sum_{i=0}^{k-1} d(z_h, z_i) + \sum_{i=1, i \neq h}^{k-1} d(z_0, z_i). \]

Adding this inequality to (4.12) results in
\[ (k - 2)(1 - \tau) \cdot d(z_0, z_h) \leq \tau \cdot \sum_{i=1, i \neq h}^{k-1} d(z_0, z_i). \tag{4.14} \]

Now, for \( \tau \leq 1 \), multiply the latter inequality by
\[ \lambda = \frac{(k - 3)\tau + 1}{(k - 3)(1 - \tau) + 1} \]
Section 4.6

and add this to (4.12). Since

\[
(k - 2)r + 1 - \lambda(k - 2)(1 - r) = r + \lambda r = \frac{(k - 1)r}{(k - 3)(1 - r) + 1},
\]

we conclude that

\[
e(M_h) \leq \frac{(k - 1)r}{(k - 3)(1 - r) + 1} \cdot c(M_{opt}) \quad \text{if } r \leq 1.
\]

(4.15)

As to an upper bound on the cost of the assignment produced by the multiple-hub heuristic, apply (4.11) and finally the \(r\)-inequality:

\[
c(M_H) \leq \frac{1}{k} \cdot \sum_{h=0,\ldots,k-1} c(M_h)
\leq \frac{1}{k} \cdot \sum_{h=0,\ldots,k-1} \sum_{i=0,\ldots,k-1} \sum_{Z \in M_{opt}} d(z_h, z_i)
\leq \frac{2}{k} \cdot \sum_{Z \in M_{opt}} \sum_{i<j} d(z_i, z_j)
\leq \frac{2}{k} \cdot \sum_{Z \in M_{opt}} [(k - 2)r + 1] \cdot c(Z)
= \frac{2}{k} [(k - 2)r + 1] \cdot c(M_{opt}).
\]

(4.16)

Example 4.5 To show that (4.16) is sharp for \(r = 1\), consider again the instance described in Example 4.3. We may assume that \(M_H\) and \(M_{opt}\) remain the same as in Example 4.3 (because \(M_H = M_0\) by symmetry of the instance, and \(M_0\) is independent of the cost function \(c\)). Here, \(c(M_H) = 2(k - 1)\) and \(c(M_{opt}) = k\), thus establishing the sharpness of (4.16) when \(r = 1\).

\[\square\]

4.6 Tour costs

The cost of a clique \(X = \{x_0, \ldots, x_{k-1}\}\) is now defined as the sum of the edge lengths in a minimum length Hamiltonian tour (i.e., spanning cycle) of \(X\):

\[
c(X) = \min \left\{ \sum_{i=0}^{k-1} d(x_{\pi(i)}, x_{\pi(i+1)}) \mid \pi \text{ is a permutation of } \{0, \ldots, k-1\} \right\},
\]
where integers $i$ are read modulo $k$. Assume without loss of generality that for a given $X$ (or later on, given $Z \in M_{opt}$) the minimum cost is attained for the identity permutation. Then from the $k - 2$ inequalities

$$d(x_i, x_{i+1}) \leq r(d(x_0, x_i) + d(x_0, x_{i+1})) \quad \text{for} \ i \neq 0, k - 1$$

we obtain

$$c(X) \leq (1 + r)(d(x_0, x_1) + d(x_0, x_{k-1})) + 2r \cdot \sum_{i=2, \ldots, k-2} d(x_0, x_i).$$

To simplify our computations, we assume from now on that $r \geq 1$. Hence

$$c(X) \leq 2r \cdot \sum_{i=1, \ldots, k-1} d(x_0, x_i). \quad (4.17)$$

The analogous inequality holds when 0 is replaced by $h$. Therefore

$$c(M_h) \leq 2r \cdot \sum_{i=0, \ldots, k-1} \sum_{X \in M_h} d(x_h, x_i)$$

$$\leq 2r \cdot \sum_{i=0, \ldots, k-1} \sum_{Z \in M_{opt}} d(z_h, z_i) \quad (4.18)$$

since each $M_{hi}$ is a minimum length matching. Applying the iterated $r$-inequality (4.2), we get for $2 \leq j \leq \frac{k}{2}$

$$d(z_0, z_j) \leq r^{[\log j]} \cdot \sum_{i=0, \ldots, j-1} d(z_i, z_{i+1}). \quad (4.19)$$

So, when $1 \leq j < \frac{k}{2}$, we have

$$\sum_{i=1, \ldots, j} d(z_0, z_i) \leq \left( \sum_{i=1, \ldots, j} r^{[\log i]} \right) \cdot \sum_{i=0, \ldots, j-1} d(z_i, z_{i+1})$$

and similarly,

$$\sum_{i=1, \ldots, j} d(z_0, z_{k-i-1}) \leq \left( \sum_{i=1, \ldots, j} r^{[\log i]} \right) \cdot \sum_{i=0, \ldots, j-1} d(z_{k-i-1}, z_{k-i}).$$

For $k$ odd, adding up these two inequalities immediately yields

$$\sum_{i=1, \ldots, k-1} d(z_0, z_i) \leq \left( \sum_{i=1, \ldots, \frac{k-1}{2}} r^{[\log i]} \right) \cdot \sum_{i=0, \ldots, k-1} d(z_i, z_{i+1}). \quad (4.20)$$
Section 4.6

For \( k \) even, one adds one half of the inequality (4.19) with \( j = \frac{k}{2} \) to each of the above inequalities and then obtains

\[
\sum_{i=1, \ldots, k-1} d(z_0, z_i) \leq \left( \frac{1}{2} \tau^{\frac{1}{\log \frac{k}{2}}} + \sum_{i=1, \ldots, k-1} \tau^{\frac{1}{\log \frac{k}{2}}} \right) \cdot \sum_{i=0, \ldots, k-1} d(z_i, z_{i+1}).
\]

(4.21)

To give a very rough estimate, observe that the sums of powers of \( \tau \) in (4.20) and (4.21) are bounded above by

\[
\frac{k-1}{2} \cdot \frac{1}{\tau^{\frac{1}{\log(k-1)}}}.
\]

Hence from (4.18) we infer

\[
c(M_h) \leq \left( k - 1 \right) \tau^{\frac{1}{\log(k-1)}} \cdot c(M_{\text{opt}}) \quad \text{for } \tau \geq 1.
\]

(4.22)

This inequality is certainly not sharp for \( \tau \neq 1 \).

Example 4.6 The instance described in Example 4.1 achieves the worst-case bound (4.22) when \( \tau = 1 \). Indeed, \( M_0 \) and \( M_{\text{opt}} \) are identical to the ones described there, now at cost \( c(M_0) = k \cdot 2(k-1) \) and \( c(M_{\text{opt}}) = 2 \cdot k \).

From (4.18) one can readily derive a first upper bound of \( c(M_H) \), viz.,

\[
c(M_H) \leq \frac{1}{k} \cdot \sum_{h=0, \ldots, k-1} c(M_h)
\]

\[\leq \frac{1}{k} \cdot \sum_{h=0, \ldots, k-1} 2\tau \cdot \sum_{i=0, \ldots, k-1} \sum_{Z \in M_{\text{opt}}} d(z_h, z_i)
\]

\[= \frac{4}{k} \tau \cdot \sum_{Z \in M_{\text{opt}}} \sum_{i<j} d(z_i, z_j).
\]

(4.23)

In order to estimate the sum of all \( d(z_i, z_j) \) in terms of \( c(Z) \) we make use of (4.19) (with suitably shifted indices):

\[
\sum_{i<j} d(z_i, z_j) \leq \begin{cases} 
\left( \sum_{j=1, \ldots, k+1} \tau^{\frac{1}{\log j}} \right) \cdot c(Z) & \text{if } k \text{ is odd}, \\
\left( \frac{k}{4} \tau^{\frac{1}{\log \frac{k}{2}}} + \sum_{j=1, \ldots, k-1} \tau^{\frac{1}{\log \frac{k}{2}}} \right) \cdot c(Z) & \text{if } k \text{ is even}.
\end{cases}
\]

for \( \tau \geq 1 \).

(4.24)
Chapter 4

The powers of \( r \) occurring in (4.24) are all bounded above by \( \frac{1}{r} \cdot r^{[\log(k-1)]} \). So, applying (4.23), we arrive at the following inequality:

\[
c(M_H) \leq \left\{ \begin{array}{ll} \frac{1}{2} (k - 1)^{r^{[\log(k-1)]}} \cdot c(M_{\text{opt}}) & \text{if } k \text{ is odd}, \\ \frac{1}{2} k r^{[\log(k-1)]} \cdot c(M_{\text{opt}}) & \text{if } k \text{ is even} \end{array} \right.
\]

for \( r \geq 1 \). (4.25)

This bound is certainly not sharp when \( r > 1 \); but for \( r = 1 \) it is best possible, as is confirmed by the next example.

**Example 4.7** Let us assume that \( r = 1 \) and, for ease of notation, that \( k \) is even (the case \( k \) odd will be briefly discussed later on).

The instance we consider involves \( k \left( \frac{k}{2} \right) \) vertices, with \( X_j = \{(j, S, \delta) | j \in S \subseteq \{0, \ldots, k - 1\}, |S| = \frac{k}{2}, \delta \in \{0, 1\}\} \) \( (j = 0, \ldots, k - 1) \). We think of these vertices as being spread over two levels, corresponding respectively to \( \delta = 0 \) or \( \delta = 1 \). Within each level, the vertices are clustered into \( \left( \frac{k}{2} \right) \) groups of cardinality \( \frac{k}{2} \); each group contains all vertices \( (j, S, \delta) \), for a fixed \( S \), and for \( j \in S \). The distance between any two vertices in distinct levels is 1, between two vertices in the same group is 0, and between two vertices in the same level, but in different groups, is 2. More formally,

\[
d((j, S, 0), (i, T, 1)) = 1 \quad \text{for all } j \in S, i \in T,
\]

\[
d((j, S, \delta), (i, S, \delta)) = 0 \quad \text{for all } i, j \in S,
\]

\[
d((j, S, \delta), (i, T, \delta)) = 2 \quad \text{for all } S \neq T, j \in S, i \in T,
\]

for all \( S, T \subseteq \{0, \ldots, k - 1\}, |S| = |T| = \frac{k}{2} \) and for all \( \delta \in \{0, 1\} \).

Notice that, for this problem instance, each sum has cost at least 2. The optimal assignment is:

\[
M_{\text{opt}} = \left\{ \{(j, S, 0) | j \in S\} \cup \{(i, S, 1) | i \in \overline{S}\} | |S| = \frac{k}{2} \right\}
\]

where \( \overline{S} = \{0, \ldots, k - 1\} \setminus S \). To verify this, observe that each set in \( M_{\text{opt}} \) is a sum with tour cost equal to 2.

By symmetry, we can assume that \( M_H = M_0 \). For each \( j = 1, \ldots, k - 1 \), an optimal bipartite matching between \( X_0 \) and \( X_j \) is given by

\[
M_{0j} = \{(0, S, \delta), (j, S, \delta) | 0 \in S, |S| = \frac{k}{2}\}
\]

\[
\cup \{(0, S, \delta), (j, S \cup \{j\} \setminus \{0\}, 1 - \delta) | 0 \in S, j \notin S, |S| = \frac{k}{2}\}.
\]
Section 4.7

The total length of this matching is \( \frac{k}{2} \); this is certainly optimal, since for each \( \delta \) such that \( j \in S, (0, S, \delta) \) must be matched with a vertex at distance at least 1.

Accordingly, the multiple-hub heuristic returns the assignment \( M_H \) consisting of the following sum:

\[
\{(j, S, \delta)| j \in S \} \cup \{(j, S \cup \{j\}\{0\}, 1-\delta)| j \not\in S\}
\]

(4.26)

for each \( S \) such that \( 0 \in S, |S| = \frac{k}{2} \), and for \( \delta = 0, 1 \). In words, each clique of \( M_H \) contains \( \frac{k}{2} \) vertices forming a group in one level, and \( \frac{k}{2} \) vertices from \( \frac{k}{2} \) distinct groups in the other level. The tour cost of a clique is thus equal to \( k \), i.e., \( \frac{k}{2} \) times the cost of a clique in \( M_{opt} \). This implies that \( c(M_H) = \frac{k}{2}c(M_{opt}) \); hence, (4.25) is sharp for \( r = 1 \) and \( k \) even.

When \( k \) is odd, a similar example can be constructed, involving as vertices all triples of the form \( (j, S, \delta) \) for \( j \in S, |S| \in \{[\frac{k}{2}], [\frac{k}{2}] \} \) and \( \delta \in \{0, 1\} \). The definitions of the length function, of \( M_{opt} \) and of \( M_H \) carry over, if we replace everywhere the condition \( |S| = \frac{k}{2} \) by \( |S| \in \{[\frac{k}{2}], [\frac{k}{2}] \} \). The tour cost of a clique (4.26) is then

\[
\begin{align*}
2 \left\lfloor \frac{k}{2} \right\rfloor & \text{ if } |S| = \left\lfloor \frac{k}{2} \right\rfloor, \\
2 \left\lceil \frac{k}{2} \right\rceil & \text{ if } |S| = \left\lceil \frac{k}{2} \right\rceil.
\end{align*}
\]

This entails, after some computations, that (4.25) is sharp when \( r = 1 \) and \( k \) is odd. Details are left to the reader. \( \square \)

4.7 Tree costs

Let now the cost of a clique \( X = \{z_0, \ldots, z_{k-1}\} \) be the sum of edge lengths in a minimum length spanning tree of \( X \). Then, as in the case of star costs, we obtain

\[
c(M_h) \leq \sum_{Z \in M_{opt}} \sum_{t=0, \ldots, k-1} d(z_h, z_t).
\]

(4.27)

When \( r \geq 1 \), the extended \( r \)-inequality immediately implies:

\[
d(z_h, z_t) \leq r^{\lfloor \log(k-1) \rfloor} \cdot c(Z),
\]

and hence
\[ c(M_h) \leq (k - 1) \tau^{\log(k - 1)} \cdot c(M_{opt}) \quad \text{for } \tau \geq 1. \quad (4.28) \]

Also
\[ c(M_H) \leq \frac{2}{k} \sum_{Z \in M_{opt}} \sum_{i < j} d(z_i, z_j) \quad (4.29) \]

follows from (4.27). The maximum number of pairs \( i < j \) for which an edge of the minimum length spanning tree \( T \) of a sum \( Z \) lies on a path from \( z_i \) to \( z_j \) in \( T \) equals \( \left( \frac{k}{2} \right)^2 \) if \( k \) is even and else equals \( \frac{k+1}{2} \cdot \frac{k}{2} \). So, by (4.2) and (4.29), the following rough estimate can be established:

\[ c(M_H) \leq \begin{cases} \frac{1}{2} (k - \frac{1}{2}) \tau^{\log(k - 1)} \cdot c(M_{opt}) & \text{if } k \text{ is odd}, \\ \frac{1}{2} k \tau^{\log(k - 1)} \cdot c(M_{opt}) & \text{if } k \text{ is even}. \end{cases} \quad (4.30) \]

As we show now, the bounds provided by (4.28) and (4.30) are actually sharp for \( \tau = 1 \):

**Example 4.8** For (4.28) consider again the instance described in Example 4.1. The tree costs of the optimal and of the heuristic solutions are the same as their star costs, i.e. \( c(M_{opt}) = k, c(M_0) = k(k - 1) \), thus establishing the sharpness of (4.28) for \( \tau = 1 \).

As for (4.30), we refer to the instance given in Example 4.7 and we assume that \( k \) is even. The optimal and heuristic solutions remain unchanged. The tree cost of a clique in \( M_{opt} \) is 1, while the cost of a clique in \( M_H \) is \( \frac{k}{2} \). This shows that (4.30) is sharp for \( \tau = 1 \) and \( k \) even. The analysis is similar for \( k \) odd. \( \Box \)

### 4.8 Recursive heuristics

In this section we establish the worst-case results for the recursive heuristics described in Section 4.3. As announced there, we only handle here the case of sum costs, i.e. the cost of a clique \( X = \{z_0, z_1, \ldots, z_{k-1}\} \) equals the sum of all edge lengths \( d(z_i, z_j) \).

First, we verify the upper bound stated in Theorem 4.4 for the worst-case performance of \( M_\sigma \). Then, we further refine our analysis for different values of \( \tau \). Finally, we prove similar results for \( M_R \). Note that these results are in fact generalizations of those results given in Chapter 3 of this thesis, where the case \( k = 3, \tau = 1 \) is dealt with.

Assume without loss of generality that \( \sigma \) is the identity permutation. We will also make the convention that, whenever we write a clique \( X \) as
\{x_0, \ldots, x_{k-1}\}$, then $x_j \in X_j$ ($j = 0, \ldots, k - 1$). Let us recursively define a function $\alpha(k, \tau)$ as follows:

$$
\begin{align*}
\alpha(2, \tau) &= 1, \\
\alpha(k, \tau) &= \frac{(k - 1 + 2\tau)(k - 2)\tau}{((k - 2)\tau + 1)(k - 1)} \cdot \alpha(k - 1, \tau) + \frac{1}{k - 1} & \text{for } k \geq 3.
\end{align*}
$$

(4.31)

We will prove by induction on $k$ that $\alpha(k, \tau)$ is an upper bound for the ratio $c(M_\sigma)/c(M_{\text{opt}})$. For $h = 0, \ldots, k - 2$ let

$$
M_{\sigma}^h = \{\{x_0, \ldots, x_{k-2}, x_{k-1}\} | \{x_0, \ldots, x_{k-2}\} \subseteq Y \text{ for some } Y \in M_\sigma \\
\quad \text{and } \{x_h, x_{k-1}\} \subseteq Z \text{ for some } Z \in M_{\text{opt}}\}.
$$

Thus the cliques in $M_{\sigma}^h$ coincide on $X_0 \cup \ldots \cup X_{k-2}$ with the cliques of $M_\sigma$, and on $X_h \cup X_{k-1}$ with the cliques of $M_{\text{opt}}$. Then an upper bound for the cost of $M_\sigma$ is given by the cost of each $M_{\sigma}^h$, viz.

$$
c(M_\sigma) \leq \sum_{X \in M_\sigma} \sum_{i<j \leq k-2} d(x_i, x_j) + \sum_{X \in M_{\sigma}^i} \sum_{i=0, \ldots, k-2} d(x_i, x_{k-1}). \quad (4.32)
$$

Using the $\tau$-inequality $d(x_i, x_{k-1}) \leq \tau (d(x_i, x_h) + d(x_h, x_{k-1}))$ for each $i \neq h$, we obtain

$$
\begin{align*}
\sum_{X \in M_{\sigma}^h} \sum_{i=0, \ldots, k-2} d(x_i, x_{k-1}) \\
\leq \sum_{X \in M_{\sigma}^h} \left[ d(x_h, x_{k-1}) + \tau (k - 2) \cdot d(x_h, x_{k-1}) + \tau \cdot \sum_{i=0, \ldots, k-2} d(x_i, x_h) \right] \\
= [(k - 2)\tau + 1] \cdot \sum_{z \in M_{\text{opt}}} d(z_h, z_{k-1}) + \tau \cdot \sum_{X \in M_{\sigma}} \sum_{i=0, \ldots, k-2} d(x_i, x_h).
\end{align*}
$$

(4.33)

Combining this inequality with (4.32) and summing over $h = 0, \ldots, k - 2$ yields

$$
\begin{align*}
(k - 1) \cdot c(M_\sigma) \leq (k - 1) \cdot \sum_{X \in M_\sigma} \sum_{i<j \leq k-2} d(x_i, x_j) \\
+ 2\tau \cdot \sum_{X \in M_\sigma} \sum_{i<h \leq k-2} d(x_i, x_h) \\
+ [(k - 2)\tau + 1] \cdot \sum_{z \in M_{\text{opt}}} \sum_{h} d(z_h, z_{k-1})
\end{align*}
$$
\[
= (k - 1 + 2r) \cdot \sum_{X \in \mathcal{M}_r} \sum_{i < j \leq k - 2} d(x_i, x_j) \\
+ [(k - 2)\tau + 1] \cdot \sum_{Z \in \mathcal{M}_{\text{opt}}} \sum_{j} d(z_j, z_{k-1}).
\]

By induction, we derive
\[
(k - 1) \cdot c(M_r) \leq \\
(k - 1 + 2r) \cdot \alpha(k - 1, \tau) \cdot \sum_{Z \in \mathcal{M}_{\text{opt}}} \sum_{i < j \leq k - 2} d(z_i, z_j) \\
+ [(k - 2)\tau + 1] \cdot \sum_{Z \in \mathcal{M}_{\text{opt}}} \sum_{j=0, \ldots, k-2} d(z_j, z_{k-1}). \tag{4.34}
\]

In view of this, and using the \(r\)-inequality
\[d(z_i, z_j) \leq \tau (d(z_i, z_{k-1}) + d(z_j, z_{k-1})) \text{ for } i < j \leq k - 2,
\]
we see that (4.34) entails
\[
(k - 1) \cdot c(M_r) \leq \\
[(k - 1 + 2r) \cdot \alpha(k - 1, \tau) - \lambda] \cdot \sum_{Z \in \mathcal{M}_{\text{opt}}} \sum_{i < j \leq k - 2} d(z_i, z_j) \\
+ [(k - 2)\tau + 1 + (k - 2)r\lambda] \cdot \sum_{Z \in \mathcal{M}_{\text{opt}}} \sum_{j} d(z_j, z_{k-1})
\]

for all \(\lambda \geq 0\). Choose \(\lambda = \frac{(k - 1 + 2r) \cdot \alpha(k - 1, \tau)}{(k - 2)\tau + 1} - 1\). From (4.31), one can prove by induction on \(k\) that the following inequality holds for all \(k \geq 3\) and \(r \geq \frac{1}{2}^+\):
\[(k - 1 + 2r) \cdot \alpha(k - 1, \tau) \geq (k - 2)r + 1.\]

Hence \(\lambda \geq 0\), and we arrive at:
\[
(k - 1) \cdot c(M_r) \leq \left[ \frac{(k - 1 + 2r)(k - 2)r}{(k - 2)r + 1} \cdot \alpha(k - 1, \tau) + 1 \right] \cdot c(M_{\text{opt}}),
\]
or equivalently
\[
c(M_r) \leq \alpha(k, \tau) \cdot c(M_{\text{opt}}). \tag{4.35}
\]

In the sequel, we discuss the behavior of the function \(\alpha(k, \tau)\). First, observe that \(\alpha(k, \frac{1}{2}) = 1\), as one would expect. More generally, if \(\frac{1}{2} \leq \tau < \frac{1}{2} \sqrt{2}\), we can prove that \(\alpha(k, \tau)\) is bounded above by a constant only depending on \(\tau\).

To see this, define:
Section 4.8

\[ \gamma(k, r) = \frac{(k - 2)r + 1}{k(1 - 2r^2) + 4r^2 - 1}. \]

It is not hard to prove that, for \( \frac{1}{2} \leq r < \frac{1}{2}\sqrt{2} \) and \( k \geq 3 \)

\[ \gamma(k - 1, r) \leq \gamma(k, r) \leq \frac{r}{1 - 2r^2}, \]

and

\[ \gamma(k, r) = \frac{(k - 1 + 2r)(k - 2)r}{((k - 2)r + 1)(k - 1)} \cdot \gamma(k, r) + \frac{1}{k - 1}. \]

From these observations and from (4.31), it then follows by induction that

\[ \alpha(k, r) \leq \gamma(k, r) \leq \frac{r}{1 - 2r^2} \quad (4.36) \]

for all \( k \geq 3 \) and for \( \frac{1}{2} \leq r < \frac{1}{2}\sqrt{2} \). For \( r = \frac{1}{2}\sqrt{2} \), we can show that \( \alpha(k, \frac{1}{2}\sqrt{2}) \) grows logarithmically with \( k \). Indeed, define \( \xi(k) = \frac{k - 1}{k - 1 + \frac{1}{\sqrt{2}}} \cdot \alpha(k, \frac{1}{2}\sqrt{2}) \) and rewrite (4.31), as

\[ \xi(2) = \frac{1}{1 + \sqrt{2}}, \]

\[ \xi(k) = \xi(k - 1) + \frac{1}{k - 1 + \sqrt{2}} \quad \text{for all } k \geq 3. \]

Solving this difference equation yields the following expression for \( \alpha(k, \frac{1}{2}\sqrt{2}) \):

\[ \alpha(k, \frac{1}{2}\sqrt{2}) = (1 + \frac{\sqrt{2}}{k - 1}) \cdot \sum_{j=2}^{k} \frac{1}{j + \sqrt{2} - 1}. \]

The inequalities

\[ \sum_{j=2}^{k} \frac{1}{j} < \ln k < \sum_{j=1}^{k-1} \frac{1}{j} \]

for all \( k > 1 \)

immediately imply the logarithmic behavior of \( \alpha(k, \frac{1}{2}\sqrt{2}) \). Hence, also \( \alpha(k, r) \) grows at least logarithmically for \( r \geq \frac{1}{2}\sqrt{2} \).

For \( r = 1 \), the reader will easily verify by substitution into (4.31) that

\[ \alpha(k, 1) = \frac{1}{2}k. \quad (4.37) \]
Hence, when the ‘usual’ triangle inequality holds, the upper bound for the worst-case ratio of \( M_o \) is linear in \( k \).

Let us now show that \( \alpha(k, \tau) \) is a sharp upper bound on \( c(M_o)/c(M_{opt}) \) when \( k = 3 \), for all \( \tau \geq \frac{1}{2} \).

**Example 4.9** Let \( X_j = \{x_{j0}, x_{j1}\} \) for \( j = 0, 1, 2 \), and let the edge lengths be defined as indicated on Figure 4.4.

![Figure 4.4](image)

Note that the \( r \)-inequality is satisfied. With \( \sigma = (0, 1, 2) \), we may obtain: \( M_o = \{\{x_{00}, x_{11}, x_{20}\}, \{x_{01}, x_{10}, x_{21}\}\} \) with \( c(M_o) = 2(1 + \frac{1}{2} + \frac{1}{2\tau}) \). Here \( M_{opt} = \{\{x_{00}, x_{10}, x_{20}\}, \{x_{01}, x_{11}, x_{21}\}\} \) with \( c(M_{opt}) = 2(1 + \frac{1}{2}) \). This results in

\[
c(M_o)/c(M_{opt}) = \tau + \frac{1}{2} = \alpha(3, \tau).
\]

Unfortunately, the upper bound \( \alpha(k, \tau) \) is not sharp when \( k > 3 \), as is asserted by Theorem 4.5 for \( k = 4 \). To prove Theorem 4.5, assume w.l.o.g. \( \sigma = (0, 1, 2, 3) \). Combining (4.32) and (4.33) for \( k = 4 \) and \( \tau = 1 \), we obtain for \( h = 0, 1, 2 \) respectively:

\[
c(M_o) \leq \sum_{x \in M_o} (d(x_1, x_2) + 2d(x_0, x_1) + 2d(x_0, x_2))
\]
Section 4.8

\[ + 3 \sum_{z \in M_{\text{opt}}} d(z_0, z_3), \] (4.38)

\[
c(M_0) \leq \sum_{x \in M_0} (d(x_0, x_2) + 2d(x_0, x_1) + 2d(x_1, x_2)) \\
+ 3 \sum_{z \in M_{\text{opt}}} d(z_1, z_3), \] (4.39)

\[
c(M_2) \leq \sum_{x \in M_0} (d(x_0, x_1) + 2d(x_0, x_2) + 2d(x_1, x_2)) \\
+ 3 \sum_{z \in M_{\text{opt}}} d(z_2, z_3). \] (4.40)

Multiplying (4.38) and (4.39) by 4 and (4.40) by 13 and adding up results in

\[ 21c(M_0) \leq \sum_{x \in M_0} (29d(x_0, x_1) + 38d(x_0, x_2) + 38d(x_1, x_2)) \\
+ \sum_{z \in M_{\text{opt}}} (12d(z_0, z_3) + 12d(z_1, z_3) + 39d(z_2, z_3)). \] (4.41)

On the other hand, we can also use (4.32) and (4.33) with \( k = 3 \) and \( h = 0, 1 \) in order to derive

\[
\sum_{x \in M_0} (d(x_0, x_2) + d(x_1, x_2)) \leq 2 \cdot \sum_{z \in M_{\text{opt}}} d(z_0, z_2) \\
+ \sum_{x \in M_0} d(x_0, x_1),
\]

\[
\sum_{x \in M_0} (d(x_0, x_2) + d(x_1, x_2)) \leq 2 \cdot \sum_{z \in M_{\text{opt}}} d(z_1, z_2) \\
+ \sum_{x \in M_0} d(x_0, x_1).
\]

Multiplying both inequalities by 19, further the inequality

\[
\sum_{x \in M_0} (d(x_0, x_1) \leq \sum_{z \in M_{\text{opt}}} d(z_0, z_1)
\]

by 67, and then adding all these to (4.41), we obtain

\[
21c(M_0) \leq \sum_{z \in M_{\text{opt}}} [67d(z_0, z_1) + 38d(z_0, z_2) + 12d(z_0, z_3) \\
+ 38d(z_1, z_2) + 12d(z_1, z_3) + 39d(z_2, z_3)].
\]
Combining this with the triangle inequalities
\[ d(x_0, x_1) \leq d(x_0, x_2) + d(x_1, x_2), \]
\[ 27d(x_0, x_1) \leq 27d(x_0, x_2) + 27d(x_1, x_2) \]
leads to \( c(M_\sigma) \leq \frac{13}{7} c(M_{\text{opt}}) \), as required for Theorem 4.5.

The following example shows that this bound is best possible.

**Example 4.10** Let \( X_j = \{x_{j0}, x_{j1}, x_{j2}\} \) for \( j = 0, 1, 2, 3 \).

![Graph Diagram]

**Figure 4.5**

In order to define the length function, consider the graph \( G \) of Figure 4.5.

The numbers along the edges of \( G \) indicate the length of these edges. For those pairs of vertices \( u, v \) for which no edge is drawn, \( d(u, v) \) is equal to the length of the shortest path between \( u \) and \( v \) in \( G \). It is not difficult to see that the optimal solution is \( M_{\text{opt}} = \{\{x_{0j}, x_{1j}, x_{2j}, x_{3j}\} | j = 0, 1, 2\} \), with \( c(M_{\text{opt}}) = 21 \). For \( \sigma = (0, 1, 2, 3) \), we get \( M_{01} = \{\{x_{00}, x_{12}\}, \{x_{01}, x_{10}\}, \{x_{02}, x_{11}\}\} \). Next, we obtain the 3-dimensional assignment \( M = \{\{x_{00}, x_{12}, x_{21}\}, \{x_{01}, x_{10}, x_{22}\}, \{x_{02}, x_{11}, x_{20}\}\} \). All matchings between the cliques of \( M \) and \( X_3 \) have the same cost, namely 39. Hence, \( c(M_\sigma)/c(M_{\text{opt}}) = \frac{39}{21} = \frac{13}{7} \). \( \square \)
Let us finally handle the case of the multiple-pass recursive heuristic and the proof of Theorem 4.6. Define

\[
\begin{align*}
\beta(2, r) &= 1, \\
\beta(k, r) &= \frac{(k - 1 + 2r)(k - 2)}{k(k - 1)} \cdot \beta(k - 1, r) + \frac{2[(k - 2)r + 1]}{k(k - 1)}.
\end{align*}
\] (4.42)

We want to prove by induction on \( k \) that \( \beta(k, r) \) is an upper bound on \( c(M_R)/c(M_{opt}) \).

Consider an instance of the \( k \)-dimensional assignment problem, and let \( M_{opt} \) be its optimal solution. Assume without loss of generality that, when applying the multiple-pass recursive heuristic to the \((k - 1)\)-dimensional subproblem obtained by disregarding \( X_{k-1} \), the best solution is produced for the permutation \((0,1,\ldots, k-2)\) of the indices. Let now \( \sigma = (0,1,\ldots, k-1) \), and observe that, by the induction hypothesis,

\[
\sum_{X \in M_\sigma} \sum_{i<j \leq k-2} d(x_i, x_j) \leq \beta(k - 1, r) \cdot \sum_{Z \in M_{opt}} \sum_{i<j \leq k-2} d(z_i, z_j).
\]

So, in the same way as we earlier derived (4.34), we now obtain:

\[
(k - 1) \cdot c(M_R) \leq (k - 1 + 2r) \cdot \beta(k - 1, r) \cdot \sum_{Z \in M_{opt}} \sum_{i<j \leq k-2} d(z_i, z_j) + [(k - 2)r + 1] \cdot \sum_{Z \in M_{opt}} \sum_{j \neq k-1} d(z_j, z_{k-1}).
\] (4.42)

Moreover, the reasoning leading to (4.45) can be repeated \( k - 1 \) times, by disregarding \( X_0, \ldots, X_{k-2} \) instead of \( X_{k-1} \). This yields the following inequality, for \( h = 0,1,\ldots, k-1 \):

\[
(k - 1) \cdot c(M_R) \leq (k - 1 + 2r) \cdot \beta(k - 1, r) \cdot \sum_{Z \in M_{opt}} \sum_{0 \leq i<j \leq k-1} d(z_i, z_j) + [(k - 2)r + 1] \cdot \sum_{Z \in M_{opt}} \sum_{j \neq k-1} d(z_j, z_h).
\]

Adding these up, we get:

\[
k(k - 1) \cdot c(M_R) \leq [(k - 1 + 2r)(k - 2) \cdot \beta(k - 1, r) + 2[(k - 2)r + 1] \cdot c(M_{opt}),
\]

or, in view of (4.42)
\[ c(M_R) \leq \beta(k, r) \cdot c(M_{opt}). \quad (4.44) \]

Let us now investigate the behavior of \( \beta(k, r) \) for various values of \( r \). First, when \( \frac{1}{2} \leq r < 1 \), one has

\[ \beta(k, r) \leq \eta(k, r) \overset{def}{=} \frac{(k - 2)r + 1}{k - (k - 2)r - 1} \leq \frac{r}{1 - r}. \quad (4.45) \]

To see this, notice that \( \eta(k, r) \geq \eta(k - 1, r) \) and

\[ \eta(k, r) = \frac{(k - 1 + 2r)(k - 2)}{k(k - 1)} \cdot \eta(k, r) + \frac{2[(k - 2)r + 1]}{k(k - 1)} \]

when \( \frac{1}{2} \leq r < 1 \), and use the definition (4.42).

When \( r = 1 \), \( \beta(k, 1) \) grows logarithmically with \( k \). Indeed, substituting \( r = 1 \) in (4.42) and defining

\[ \varsigma(k) = \frac{k - 1}{k + 1} \cdot \beta(k, 1), \]

we arrive at

\[ \varsigma(k) = \varsigma(k - 1) + \frac{2(k - 1)}{k(k + 1)}. \]

This equation is easily solved, and leads to

\[ \beta(k, 1) = \frac{2(k + 1)}{k - 1} \cdot \sum_{j=1}^{k} \frac{1}{j + 1} - \frac{2k}{k - 1}. \quad (4.46) \]

For \( k \geq 2 \), define

\[ C_k = \sum_{j=1}^{k} \frac{1}{j} - \ln k. \]

It is well known that \( 1 > C_2 > C_3 > \ldots > C > \frac{1}{2} \) and \( \lim_{k \to \infty} C_k = C \), where \( C = 0.57721 \ldots \) is the Euler-Mascheroni constant. Substituting in (4.46), we thus conclude that

\[ \frac{2(k + 1)}{k - 1} \left[ \ln(k + 1) - \frac{1}{2} \right] - \frac{2k}{k - 1} < \beta(k, 1) < \frac{2(k + 1)}{k - 1} \ln(k + 1) - \frac{2k}{k - 1}. \quad (4.47) \]

As mentioned in Section 4.3, we do not know whether \( \beta(k, r) \) is a sharp upper bound on \( c(M_R)/c(M_{opt}) \), except when \( r = \frac{1}{2} \), or \( k = 3 \) and \( r = 1 \) (see Chapter 3 of this thesis).

**Acknowledgements**

We thank Ir. Hans van der Stel for his assistance in solving some of the difference equations arising in this work.
Chapter 5

Scheduling jobs of equal length: complexity and facets

This chapter is joint work with Yves Crama; an article based on parts of this chapter is submitted for publication.
5.1 Introduction

The following problem is studied in this chapter. Given are $n$ jobs, which have to be processed on a single machine within a fixed timespan $\{1, 2, \ldots, T\}$. The processing time, or length, of each job equals $p, p \in \mathbb{N}$. The processing cost of each job is an arbitrary function of its start-time: we denote by $c_{jt}$ the cost of starting job $j$ at time $t$. The problem is to schedule all jobs so as minimize the sum of the processing costs. We refer to this problem as problem SEL (Scheduling jobs of Equal Length).

Mathematically, SEL can be formulated as follows:

\[
\begin{align*}
\text{minimize} & \quad \sum_{j=1}^{n} \sum_{t=1}^{T-p+1} c_{jt} x_{jt} \\
\text{subject to} & \quad \sum_{t=1}^{T-p+1} x_{jt} = 1 \quad \text{for } j = 1, \ldots, n, \quad (5.1) \\
& \quad \sum_{j=1}^{n} \sum_{t=s}^{s+p-1} x_{jt} \leq 1 \quad \text{for } s = 1, \ldots, T-2p+2, \quad (5.2) \\
& \quad x_{jt} \in \{0, 1\} \quad \text{for } j = 1, \ldots, n; \quad t = 1, \ldots, T-p+1, \quad (5.3)
\end{align*}
\]

where $x_{jt} = 1$ if job $j$ starts at time $t$, and $x_{jt} = 0$ otherwise.

In Section 5.2, this problem is shown to be strongly $\mathcal{NP}$-hard, even when all jobs have length $p = 2$. Section 5.3 introduces some facets and valid inequalities for the solution set of (5.1)-(5.3). Section 5.4 describes some computational experiments.

Notice that SEL is a special case of a very general scheduling problem (say, problem S) considered by Sousa and Wolsey (1989). In problem S, the jobs may have arbitrary, usually distinct, processing times. Sousa and Wolsey propose several classes of facets and valid inequalities for S. It is an easy observation that, if $\{1, \ldots, n\}$ is any subset of the jobs occurring in S, and all the jobs in $\{1, \ldots, n\}$ have the same length $p$, then any valid inequality for (5.1)-(5.3) is also valid for S. This suggests that the polyhedral description presented in Section 5.3 may prove useful, not only when all jobs strictly have equal length, but also in situations where the number of distinct lengths is small, or where most of the jobs have the same length. We now proceed to describe an interesting application in which the latter assumptions are fulfilled, and which originally motivated our study.
Section 5.2

The electronics industry relies on numerically controlled machines for the automated assembly of printed circuit boards (PCBs). Prior to the start of operations, a number \( n \) of feeders, containing the electronic components to be mounted on the PCBs, are positioned alongside the machine, in some available slots \( 1, 2, \ldots, T \). Each feeder \( j \) requires a certain number of slots, say \( p_j \), depending on the feeder type; usually, \( p_j \) only takes a small number of values, say \( p_j \in \{1, 2, 3\} \). In order to minimize the production makespan, it is desirable to position the feeders "close" to the locations where the corresponding components must be inserted. More precisely, for each combination of feeder \( j \) and slot \( t \), a coefficient \( c_{jt} \) can be computed which captures the cost of assigning feeder \( j \) to slots \( t, t+1, \ldots, t+p_j - 1 \). It should now be clear that finding a minimum-cost assignment of feeders to slots is equivalent to solving a scheduling problem with "small number of distinct processing times" (see e.g. Ball and Magazine (1988) for a description of this model with \( p_j = 1 \) for all \( j \), and Ahmadi, Ahmadi, Matsuo and Tirupati (1990), Chapter 2 of this thesis, and Van Laarhoven and Zijm (1990) for a more general discussion).

Let us finally mention that SEL may be regarded as a discrete analogue of scheduling problems with unit-length tasks and arbitrary rational start-times (see e.g. Garey, Johnson, Simmons and Tarjan (1981)). SEL is also superficially related to an assignment problem with side constraints investigated by Aboudi and Nemhauser (1990, 1991).

5.2 Complexity of SEL

It is obvious that, when each job has length 1 (the case \( p = 1 \)), SEL reduces to an assignment problem, and hence is solvable in polynomial time. The following theorem shows that SEL is already strongly \( \mathcal{NP} \)-hard for \( p = 2 \):

**Theorem 5.1** SEL is \( \mathcal{NP} \)-hard, even for \( p = 2 \) and \( c_{jt} \in \{0, 1\} \) for all \( j, t \).

**Proof:**

An instance of SEL, with \( p = 2 \) and processing costs equal to 0 or 1, can be described by a bipartite graph \( G = (V_1 \cup V_2, E) \). Each job is represented by a vertex in \( V_1 \), each time-unit is represented by a vertex in \( V_2 \), and there is an edge \( (j, t) \in E \), with \( j \in V_1 \) and \( t \in V_2 \), if and only if starting job \( j \) at time-unit \( t \) has processing cost \( c_{jt} = 0 \). The instance of SEL admits a schedule with zero cost if and only if there exists a set of edges \( A \subseteq E \) such that:
i) each vertex in $V_1$ is incident to precisely one edge in $A$;

ii) each vertex in $V_2$ is incident to at most one edge in $A$;

iii) if vertex $i \in V_2$ is incident to an edge in $A$, then vertex $t + 1$ is not incident to any edge in $A$, for all $t = 1, \ldots, |V_2| - 1$.

We use a reduction from the \textit{NP}-hard 3-dimensional matching problem (see Garey and Johnson(1979)). An instance $I$ of 3-dimensional matching is specified by three mutually disjoint sets $K_1$, $K_2$ and $K_3$ with $|K_i| = n$, for $i = 1, 2, 3$, and a set $Q \subseteq K_1 \times K_2 \times K_3$, with $|Q| = m$. The instance is feasible if there exists a set $Q' \subseteq Q$ such that every element of $K_1 \cup K_2 \cup K_3$ occurs in exactly one element of $Q'$.

With $I$, we associate an instance of SEL as follows. Let

$$
V_1 = K_1 \cup K_2 \cup K_3 \cup \{a_1, \ldots, a_m\} \cup \{b_1, \ldots, b_{m-1}\},
$$

$$
V_2 = \{d_1, \ldots, d_m\}.
$$

In order to define the edge-set $E$, denote by $Q_r = \{k^1_r, k^2_r, k^3_r\}$ the $r$-th triple in $Q$, where

$$
k^1_r \in K_1, \quad k^2_r \in K_2 \quad \text{and} \quad k^3_r \in K_3 \quad (r = 1, \ldots, m).
$$

Now, let $E$ consist of the following edges:

$$(k^1_r, d_{6(r-1)+1}), \quad (k^2_r, d_{6(r-1)+3}) \quad \text{and} \quad (k^3_r, d_{6(r-1)+5})$$

for $r = 1, \ldots, m$ and

$$(a_s, d_{6(r-1)+2}) \quad \text{and} \quad (b_s, d_{6(r-1)+4})$$

for $s = 1, \ldots, m - n$, and $r = 1, \ldots, m$.

A typical piece of the graph is shown in Figure 5.1.
When the instance $I$ of 3-dimensional matching has a feasible solution, it is straightforward to find a set of edges $A \subseteq E$ which defines a zero-cost schedule. Conversely, assume that SEL has a feasible solution specified by an edge set $A$. Define $D_r = \{d_0(r-1)+1, \ldots, d_0r\}$ for $r = 1, \ldots, m$ and let

$R_0 = \{r : \text{neither } d_0(r-1)+2 \text{ nor } d_0(r-1)+4 \text{ is incident to any edge in } A\},$

$R_1 = \{r : \text{exactly one of } d_0(r-1)+2, d_0(r-1)+4 \text{ is incident to an edge in } A\},$

$R_2 = \{r : \text{both } d_0(r-1)+2 \text{ and } d_0(r-1)+4 \text{ are incident to edges in } A\}.$

Since each of $a_s, b_s, (s = 1, \ldots, m-n)$ is incident to an edge in $A$,

$$2(m-n) = |R_1| + 2|R_2|$$

(5.4)

and, since $\{R_0, R_1, R_2\}$ is a partition of $\{1, \ldots, m\}$,

$$m = |R_0| + |R_1| + |R_2|.$$  

(5.5)

Now, we claim that $|R_1| = 0$. Indeed, if $|R_1| > 0$, then (5.4) implies

$$m - n < |R_1| + |R_2|.$$  

(5.6)

However, for each $r \in R_0$, at most three vertices of $D_r$ are incident to some edge of $A$, and for each $r \in R_1 \cup R_2$, at most two vertices of $D_r$ are incident
to an edge of $A$. This implies:

$$
|A| = |V_1| = 2m + n \\
\leq 3|R_0| + 2|R_1| + 2|R_2| \\
= 3m - |R_1| - |R_2| \quad \text{(by (5.5))} \\
< 2m + n. \quad \text{(by (5.6))}
$$

This contradiction proves $|R_1| = 0$. Then, it follows easily by (5.4) and (5.5) that $|R_0| = n$, and $\{Q_r : r \in R_0\}$ defines a feasible solution of the 3-dimensional matching problem.

Notice that the proof of Theorem 5.1 is easily adapted to show that a related problem, in which $n_1$ jobs have length 1, $n_2$ jobs have length 2, and $T = n_1 + 2 \cdot n_2$, is $NP$-hard too. This is to be contrasted with the following statement:

**Theorem 5.2** If $T = n \cdot p + c$, where $c$ denotes a constant, $c \in \mathbb{N}$, then $SEL$ is polynomially solvable.

**Proof:**

We claim that, in this case, it is sufficient to solve $\binom{n + c}{n} = O(n^c)$ assignment problems, where each assignment problem corresponds to a set of starting times allowing a feasible solution to $SEL$. The proof is by induction on $n$.

If $n = 1$, then it is easy to see that $c + 1$ trivial assignment problems need to be solved, since each of the first $T - p + 1 = c + 1$ time-units is a feasible starting time for the job. Assuming that the induction hypothesis is true for $n = k$, consider now the case $T = (k + 1) \cdot p + c$. Let the smallest starting time in a feasible set of starting times be denoted by $i$, with $1 \leq i \leq c + 1$. The remaining $k$ starting times have to be chosen from the set $\{i + p, \ldots, T\}$. By the induction hypothesis, and since $T - (i + p) + 1 = k \cdot p + c - i + 1$, there are $\binom{k + c - i + 1}{k}$ feasible sets of starting times, for $i = 1, \ldots, c + 1$.

As $
\sum_{i=1}^{c+1} \binom{k + c - i + 1}{k} = \sum_{i=0}^{c} \binom{k + i}{k} = \binom{k + c + 1}{k + 1},$
the theorem is proven.
5.3 A partial polyhedral description of SEL

Let us first recall some fundamental definitions from polyhedral theory (for a thorough introduction, the reader is referred to Nemhauser and Wolsey (1988)). Consider a polyhedron \( P = \{ x \in \mathbb{R}^k : A x \leq b \} \). The equality set of \( (A, b) \) is the set of rows of \( (A, b) \), say \( (A^-, b^-) \), such that: \( A^- x = b^- \) for all \( x \) in \( P \). The dimension of \( P \) is given by: \( \dim(P) = k - \text{rank}(A^-, b^-) \). The inequality \( a x \leq a_0 \) is valid for \( P \) if it is satisfied by all points in \( P \). For a valid inequality \( a x \leq a_0 \), the set \( F = \{ x \in P : a x = a_0 \} \) is called a facet of \( P \) if \( \dim(F) = \dim(P) - 1 \). Equivalently, when \( \emptyset \neq F \neq P \), \( F \) is a facet if and only if the following condition holds: if all points in \( F \) satisfy \( \pi x = \pi_0 \), for some \( \pi, \pi_0 \in \mathbb{R}^{k+1} \), then \( (x, \pi_0) \) is a linear combination of \( (A^-, b^-) \) and \( (x, \pi_0) \) (see Nemhauser and Wolsey (1988), p. 91).

Consider now the formulation in Section 5.1, and let \( P \) denote the convex hull of the feasible solutions to constraints (5.1), (5.2) and (5.3). Furthermore, assume from now on that \( T \geq p \cdot (n + 1) \). (Notice that \( \dim(P) \leq n \cdot (T - p + 1) - n = n \cdot (T - p) \). If \( T < p \cdot (n + 1) \), then it is easy to see that \( \dim(P) < n \cdot (T - p) \); for instance \( \sum_{j=1}^{n} \sum_{t=1}^{p} x_{jt} = 1 \) is implied by (5.1) and (5.2)). To avoid trivialities, assume also \( n > 2, p \geq 2 \).

Sousa and Wolsey (1989) established the dimension of \( P \). For the sake of completeness, we also include a proof of this result:

**Theorem 5.3** \( \dim(P) = n \cdot (T - p) \).

**Proof:**

We just noticed that \( \dim(P) \leq n \cdot (T - p) \). Suppose \( \sum_{j=1}^{n} \sum_{t=1}^{T-p+1} \pi_{jt} x_{jt} = \pi_0 \) for all \( x \in P \); we want to show that this equality is implied by constraints (5.1).

To see this, fix \( j \) and \( t, t \leq T - p, \) and consider a solution with job \( j \) starting at time-unit \( t \), while the other jobs start arbitrarily at time-units in \( [1, T - p] \cup [t + p + 1, T - p + 1] \). Note that this is always possible; e.g. let \( t = k \cdot p + q \), with \( 1 \leq q \leq p \); then, a feasible schedule can be found using only start-times in

\[
S_t = \{ j : j = 0, \ldots, k \} \cup \{ j : j = k + 1, \ldots, m \},
\]

where \( m \) is the largest index such that \( m \cdot p + q + 1 \leq T - p + 1 \). Indeed, since \( T \geq p \cdot (n + 1) \), \( S_t \) contains at least \( n \) time periods.

Consider now a second schedule, obtained by starting job \( j \) at time-unit \( t + 1 \), while all other jobs remain untouched. Comparing the two schedules,
it follows easily that $\pi_{jt} = \pi_{j,t+1}$ for all $j = 1, \ldots, n$, $t = 1, \ldots, T - p$. (This construction will be used in subsequent proofs.) Thus, with $\pi_{jt} = \pi_j$ for all $j = 1, \ldots, n$, $t = 1, \ldots, T - p + 1$, we get

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p+1} \pi_{jt} x_{jt} = \sum_{j=1}^{n} \pi_j \sum_{t=1}^{T-p+1} x_{jt} = \pi_0,$$

which is a linear combination of the equalities (5.1).

With the dimension of $P$ established, we now can proceed to show that some inequalities define facets of $P$. First, we prove that the inequalities in the LP-relaxation of (5.1)-(5.3) are facet-defining.

**Theorem 5.4** The inequalities $x_{jt} \geq 0$ define facets of $P$, for all $j = 1, \ldots, n$, $t = 1, \ldots, T - p + 1$.

**Proof:**

Let $F = \{ x \in P : x_{is} = 0 \}$ for any $i, s$ with $1 \leq i \leq n$, $1 \leq s \leq T - p + 1$ and suppose

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p+1} \pi_{jt} x_{jt} = \pi_0 \text{ for all } x \in F.$$

To prove $\pi_j = \pi_{jt}$ for all $j = 1, \ldots, n$, $j \neq i$, $t = 1, \ldots, T - p + 1$, we refer to the construction used in the proof of Theorem 5.3 (it is obvious that it is always possible to ensure that job $i$ is not placed at $s$, for any $s$). Moreover, we can use this construction for job $i$ and starting-time $t$ for all $t \leq s - 2$ and $t \geq s + 1$, proving that $\pi_{i1} = \pi_{i2} = \ldots = \pi_{i,s-1}$ and $\pi_{i,s+1} = \pi_{i,s+2} = \ldots = \pi_{i,T-p+1}$.

If $s \neq 1$ and $s \neq T - p + 1$, consider a solution with job $i$ at time-unit $1$ and the other jobs at time-units $1 + p, 1 + 2p, \ldots, 1 + (n - 1) \cdot p$, and a solution with job $i$ at $T - p + 1$, and all other jobs at the same time-units as before (again, note that this is always possible, since we assumed $T \geq p \cdot (n + 1)$). Comparing these solutions, it follows that $\pi_{i1} = \pi_{i,T-p+1}$ and thus $\pi_i = \pi_{it}$ for all $t \neq s$. So:

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p+1} \pi_{jt} x_{jt} = \sum_{j=1}^{n} \pi_j \sum_{t=1}^{T-p+1} x_{jt} + \rho x_{is},$$

which shows that the equality $\sum_{j=1}^{n} \sum_{t=1}^{T-p+1} \pi_{jt} x_{jt} = \pi_0$ is a linear combination of (5.1) and of $x_{is} = 0$.

**Theorem 5.5** The inequalities (5.2) define facets of $P$. 

Section 5.3

Proof:
Let \( F = \{ x \in \mathbb{P} : \sum_{j=1}^{n} \sum_{t=s}^{T-p-1} x_{jt} = 1 \} \), for any \( 1 \leq s \leq T - 2p + 2 \), and suppose
\[
\sum_{j=1}^{n} \sum_{t=1}^{T-p+1} \pi_{jt} x_{jt} = \pi_0 \quad \text{for all} \quad x \in F.
\]

For any \( j \) and any \( t \), consider a schedule using only start-times in \( S_t \) (as in Theorem 5.3) and with \( x_{jt} = 1 \). There is always such a schedule corresponding to a point in \( F \), unless \( t = s - 1 \). Also, the schedule obtained by delaying the start-time of job \( j \) until \( t + 1 \) is in \( F \), unless \( t = s + p - 1 \). From this, one easily concludes that, for all \( j = 1, \ldots, n \),
\[
\pi_{j1} = \pi_{j2} = \ldots = \pi_{js-1} = \alpha_j,
\]
\[
\pi_{js} = \pi_{j,s+1} = \ldots = \pi_{j,s+p-1} = \beta_j,
\]
\[
\pi_{j,s+p} = \pi_{j,s+p+1} = \ldots = \pi_{j,T-p+1} = \gamma_j. \tag{5.7}
\]

If \( 2 \leq s \leq T - 2p + 1 \), then one can also show as in Theorem 5.4 that \( \pi_{j1} = \pi_{j,T-p+1} \) for all \( j = 1, \ldots, n \), or, more generally:
\[
\gamma_j = \alpha_j \quad \text{for all} \quad j = 1, \ldots, n. \tag{5.8}
\]

Furthermore, simple interchange arguments yield:
\[
\beta_j + \alpha_i = \alpha_j + \beta_i \quad \text{for all} \quad i, j \in \{1, 2, \ldots, n\}, \tag{5.9}
\]
or equivalently \( \delta = \beta_j - \alpha_j \) for all \( j = 1, \ldots, n \).

So, (5.7), (5.8) and (5.9) together imply:
\[
\sum_{j=1}^{n} \sum_{t=1}^{T-p+1} \pi_{jt} x_{jt}
\]
\[
= \sum_{j=1}^{n} \alpha_j \left( \sum_{t=1}^{s-1} x_{jt} + \sum_{t=s+p}^{T-p+1} x_{jt} \right) + \sum_{j=1}^{n} \beta_j \sum_{t=s}^{s+p-1} x_{jt}
\]
\[
= \sum_{j=1}^{n} \alpha_j \sum_{t=1}^{T-p+1} x_{jt} + \delta \sum_{j=1}^{n} \sum_{t=s}^{s+p-1} x_{jt},
\]
which proves the theorem. \( \square \)
Now, let us consider the following inequalities:

$$\sum_{i=s}^{s+p+i-1} x_{it} + \sum_{j=1}^{n} \sum_{i=s+i}^{s+p-1} x_{jt} \leq 1$$

for $1 \leq i \leq n$, $1 \leq t \leq p - 1$ and $1 \leq s \leq T - 2p - l + 2$. \quad (5.10)

These inequalities are introduced in Sousa and Wolsey (1989). Notice that the inequalities (5.2) are the special case of (5.10) for $l = 0$. However, for reasons of convenience, we maintain the distinction between these two classes. It is not difficult to see that the inequalities (5.10) are valid, but they are also facet-defining, as witnessed by the next theorem (due to Sousa and Wolsey (1989)).

**Theorem 5.6** The inequalities (5.10) define facets of $P$.

The validity of this theorem will also follow from the validity of the more general Theorem 5.8.

Observe that all (in)equalities (5.1), (5.2) and (5.10) are of the set-packing type, i.e. they only involve coefficients 0 or 1, and their right-hand side equals 1. In fact, the following holds if $T \geq p \cdot (n + 2) - 2$:

**Theorem 5.7** All facets of $P$ defined by set-packing inequalities are given by (5.2) and (5.10).

**Proof:**
Consider an arbitrary valid set-packing inequality $I$ and define

$$t^* = \max_{i} \max_{t} \{t_2 - t_1 : x_{jt_2} \text{ and } x_{jt_1} \text{ occur with coefficient 1 in } I\}.$$

Let $t$ be the job which realizes $t^*$. We will make use of the following observation: no two variables $x_{jt}$ and $x_{jt}$, with $k \neq j$, and $|s-t| \geq p$, can simultaneously occur with coefficient 1 in $I$.

Let us first consider the case $t^* \geq 2p - 1$. Then, it is easy to verify that no variable $x_{it}$, $i \neq j$, for any $t$, can occur in the inequality; thus $I$ is implied by equalities (5.1), and cannot represent a facet.

Next, suppose $p \leq t^* \leq 2p - 2$, i.e. $t^* = p + l - 1$ for some $1 \leq l \leq p - 1$. More specifically, suppose that $x_{il}$ and $x_{i,l+p+l-1}$ have coefficient 1 in $I$. From our previous observation, it easily follows that, for any $j \neq i$, $x_{jt}$ cannot occur in $I$ if either $t \leq s + l - 1$ or $t \geq s + p$. Hence, $I$ is implied by (5.10).
Finally, when \( t' \leq p - 1 \), let \( s \) be the smallest index such that, for some \( k \), \( x_{k, s} \) occurs in \( I \) with coefficient 1. It follows again from our observation that, for all \( j \) and for all \( t \geq s + p \), \( x_{jt} \) does not occur in \( I \). Hence, \( l \) is implied by (5.2). \( \square \)

(M. van den Akker has independently established that, for the more general scheduling problem \( S \) mentioned in Section 5.1, all facet-defining set-packing inequalities are given by Sousa and Wolsey (1989) (private communication). Theorem 5.7 does not hold if \( p \cdot (n + 1) \leq T < p \cdot (n + 2) - 2 \), as pointed out to us by H.-J. Bandelt (private communication).

In the following we investigate generalizations of (5.10). To start with, (5.10) can be generalized to the following inequalities:

\[
\sum_{j \in J} \sum_{t=s}^{s+k \cdot p + l - 1} x_{jt} + \sum_{j \in J} \sum_{r=0}^{k-1} \sum_{t=s+r \cdot p}^{s+(r+1) \cdot p - 1} x_{jt} \leq k,
\]

for \( J \subseteq \{1, \ldots, n\} \) with \( |J| = k > 0 \),

\( 1 \leq l \leq p-1 \) and \( 1 \leq s \leq T - (k+1) \cdot p - l + 2 \). \hspace{1cm} (5.11)

Notice that for \( J = \{i\} \), (5.11) is equivalent to (5.10). The inequalities (5.11) are valid and even facet-defining as witnessed by the following theorem:

**Theorem 5.8** The inequalities (5.11) define facets of \( P \).

**Proof:**
To facilitate the proof, we define subsets of time-units which occur in (5.11).

Let:

\[
A = [s, s + k \cdot p + l - 1] \quad \text{and} \\
B = \{t + r \cdot p : r = 0, \ldots, k-1; t = s + l, \ldots, s + p - 1\} \\
= [s + l, s + p - 1] \cup [s + p + l, s + 2p - 1] \cup \ldots \\
\ldots \cup [s + (k-1) \cdot p + l, s + k \cdot p - 1].
\]

(see Figure 5.2 for an illustration of the case \( p = 5, k = 3, l = 2 \)). With these notations, (5.11) can be rewritten as

\[
\sum_{j \in J} \sum_{t \in A} x_{jt} + \sum_{j \in J} \sum_{t \in B} x_{jt} \leq k. \hspace{1cm} (5.12)
\]
First we show that these inequalities are valid. Suppose that $k+1$ jobs start in the interval $[s, s + k \cdot p + l - 1]$. The only way to achieve this is to start exactly one job in each of the intervals $[s, s + l - 1], [s + p, s + p + l - 1], \ldots, [s + k \cdot p, s + k \cdot p + l - 1]$ (this is easily checked by induction on $k$), i.e. to start the jobs in $A \setminus B$. However, the time-units in $A \setminus B$ only occur in (5.12) for the $k$ jobs in $J$. This implies that (5.12) is valid. Let us show now that (5.12) is facet-defining.

Let $F = \{x \in P: \sum_{j \in J, t \in A} x_{jt} + \sum_{j \in J, t \in B} x_{jt} = k\}$ and suppose $\sum_{j=1}^{n} \sum_{t=1}^{n-p+1} \pi_{jt} z_{jt} = \pi_0$ for all $x \in F$.

Now, let $j \in J$ and $t \in A \setminus [s + k \cdot p + l - 1]$. Consider a solution with job $j$ started at time-unit $t$, and other jobs started at $t - p, t - 2p, \ldots$ and $t + p + 1, t + 2p + 1, \ldots$, in such a way that jobs in $J$ are started in $A$ (thus ensuring that $x \in F$). Shifting job $j$ one time-unit towards $t + 1$ proves

$$\pi_{jt} = \pi_{jt}^\text{in} \text{ for all } j \in J, \text{ for all } t \in A.$$  \hspace{1cm} (5.13)

Let now $i \notin J$ and $t \in [s + r \cdot p + l, s + (r+1) \cdot p - 2]$, where $r \in \{0, \ldots, k-1\}$ (this is assuming $l \leq p - 2$; else this step of the proof is not required). Consider the following solution: start job $i$ at time $t$, start $k - 1$ jobs from $J$ in $A$, at time-units $t - p, t - 2p, \ldots$, and $t + p + 1, t + 2p + 1, \ldots$, and start all other jobs outside $A$. Shifting job $i$ one time-unit proves

$$\pi_{it} = \pi^r_{it} \text{ for all } i \notin J, \text{ for all } t \in [s + r \cdot p + l, s + (r+1) \cdot p - 1],$$

$$r \in \{0, \ldots, k-1\}.$$  \hspace{1cm} (5.14)
Section 5.3

Also, interchanging job $i \not\in J$ and job $j \in J$ proves (for any $l \in \{1, \ldots, p - 1\}$):

$$\pi_j^{ln} + \pi_i^{r_1} = \pi_j^{ln} + \pi_i^{r_2}$$
for all $r_1, r_2 \in \{0, \ldots, k - 1\}$.

(5.15)

(5.14) and (5.15) together imply:

$$\pi_{tl} = \pi_i^{ln}$$
for all $i \not\in J$, for all $t \in B$.

(5.16)

Now, a similar reasoning as in Theorem 5.5 ensures that

$$\pi_{tl} = \pi_i^{out}$$
for all $i \not\in J$, for all $t \not\in A$.

(5.17)

Furthermore, consider a solution with the jobs from $J$ at $s$, $s + p, \ldots, s + (k - 1) \cdot p$ and job $i$, $i \not\in J$ at $s + k \cdot p$. Simple interchange arguments imply, together with (5.12) and (5.17),

$$\pi_{tl} = \pi_i^{out}$$
for all $i \not\in J$, $t \not\in B$.

(5.18)

Also, similar arguments imply

$$\pi_{jt} = \pi_j^{out}$$
for all $j \in J$, $t \not\in A$.

(5.19)

Moreover, it is easy to see that

$$\pi_j^{in} + \pi_i^{out} = \pi_j^{out} + \pi_i^{in}$$
for all $i, j \in \{1, 2, \ldots, n\}$,

(5.20)

or equivalently $\beta = \pi_j^{in} - \pi_j^{out}$ for all $j = 1, \ldots, n$.

Now (5.12), (5.16), (5.18), (5.19) and (5.20) imply:

$$\sum_{j=1}^n \sum_{t=1}^{T-p+1} \pi_{jt} x_{jt}$$

$$= \sum_{j \in J} \sum_{t \in A} \pi_j^{in} x_{jt} + \sum_{j \in J} \sum_{t \in B} \pi_j^{out} x_{jt} + \sum_{j \in J} \sum_{t \in A} \pi_j^{in} x_{jt} + \sum_{j \in J} \sum_{t \in B} \pi_j^{out} x_{jt}$$

$$= \sum_{j=1}^n \pi_j^{out} \sum_{t=1}^{T-p+1} x_{jt} + \beta \cdot (\sum_{j \in J} \sum_{t \in A} x_{jt} + \sum_{j \in J} \sum_{t \in B} x_{jt}),$$

proving the theorem.

Even though there is an exponential number of inequalities of type (5.11), the separation problem for this class of inequalities is polynomially solvable. Indeed, notice that (5.11) can be rewritten as:
\[
\sum_{j=1}^{n} \sum_{r=0}^{k-1} x_{j,t+r} + \sum_{j \in J} \sum_{r=0}^{R} \sum_{\ell = s}^{s+l-1} x_{j,t+r} \leq k. \tag{5.21}
\]

We want to check whether a given \( z^* \) violates one of these inequalities. Fix \( s, l \) and \( k \) (there are only \( O(pnT) \) choices for these three values). Then, the first term of (5.21) is a constant. Pick the \( k \) values of \( j \) which maximize the second term and put them in a set \( J^* \). If \( z^* \) violates (5.21) for any \( J \), then it does so for \( J^* \).

Another way of generalizing the inequalities (5.10) is the following. Choose a non-empty set \( J \subset \{1, \ldots, n\} \) and a non-empty set \( S \subset \{1, \ldots, T - 2p + 2\} \). For each \( s \in \{1, \ldots, T - p + 1\} \), define \( q_s = 1 \) if \( s \in S \) and \( q_s = 0 \) otherwise. Then, by adding the constraints (5.1) for \( j \in J \) and the constraints (5.2) for \( s \in S \), each with coefficient \( \frac{1}{2} \), we obtain the following valid inequality:

\[
\sum_{j \in J} \sum_{t=1}^{T-p+1} \left[ \frac{1}{2} + \frac{1}{2} \cdot \sum_{s=t-p+1}^{t} q_s \right] \cdot x_{jt} + \sum_{j \not\in J} \sum_{t=1}^{T-p+1} \left[ \frac{1}{2} \cdot \sum_{s=t-p+1}^{t} q_s \right] \cdot x_{jt} \leq \frac{1}{2}(|J| + |S|). \tag{5.22}
\]

We refer to these inequalities as \( (J, S) \) inequalities. The inequalities (5.10) are the special case of (5.22) obtained for \( J = \{i\} \) and \( S = \{s, s+l\} \). Of course, a more sophisticated choice for \( S \) could lead to other valid inequalities. Indeed, it is possible to generalize inequalities (5.10) by choosing \( S \) as \( k \) couples of time-units in the following way: for some \( k \geq 2 \) and \( l \in \{1, \ldots, p-1\} \),

\[
S = \{s, s+l, s+p, s+p+l, \ldots, s+(k-1) \cdot p, s+(k-1) \cdot p+l\}.
\]

However, the resulting \( (J, S) \) inequalities do not define facets of \( P \). In fact, (when \( k \geq 2 \) of course), they can be strengthened by lifting certain coefficients to 2. The following inequalities result:

\[
\sum_{j \not\in J} \sum_{t=s+1}^{s+k-1} x_{jt} + \sum_{t=s}^{s+k-1} x_{it} + \sum_{r=1}^{k-1} \sum_{t=s}^{s+l-1} x_{i,t+r} \leq k,
\]

for \( i, k, l, s \) with \( 1 \leq i \leq n, 1 \leq k \leq n, 1 \leq l \leq p - 1 \) and \( 1 \leq s \leq T - (k + 1) \cdot p - l + 2 \). \tag{5.23}

(observe that, when \( k \geq 2 \), then some variables occur with coefficient 2 in (5.23)). The following holds:
Theorem 5.9 The inequalities (5.29) define facets of $P$.

Proof:
We first introduce some notation. With $i, k, l, s$ as in (5.23), let

\[ A = [s, s + k \cdot p + l - 1], \]
\[ C = [s + l, s + k \cdot p - 1], \]
\[ D = \{ t + r \cdot p : r = 1, \ldots, k - 1; t = s, \ldots, s + l - 1 \} \]
\[ = [s + p, s + p + l - 1] \cup [s + 2p, s + 2p + l - 1] \cup \ldots \]
\[ \ldots \cup [s + (k - 1) \cdot p, s + (k - 1) \cdot p + l - 1]. \]

We can rewrite (5.23) as

\[ \sum_{j \neq i} \sum_{t \in C} x_{jt} + \sum_{t \in A \setminus D} x_{it} + 2 \cdot \sum_{t \in D} x_{it} \leq k \tag{5.24} \]

Let us first show that (5.24) is valid for $P$. Consider any feasible schedule.
It is easy to see that the only way to start $k$ jobs in $C$ is to start them in $C \setminus D = [s + l, s + p - 1] \cup \ldots \cup [s + (k - 1) \cdot p + l, s + k \cdot p - 1]$ (one job in each subinterval). But, if this is the case, then there is no room left to start job $i$ in $A$, and hence (5.24) is satisfied. So, the only way to violate (5.24) is to start $k - 1$ jobs in $C$, and job $i$ in $D$. Let us suppose job $i$ starts at $s + r \cdot p + q$, $r \in \{1, \ldots, k - 1\}$, $q \in \{0, \ldots, l - 1\}$. Then two intervals of consecutive time-units remain for placing $k - 1$ jobs in $C$:

\[ [s + l, s + (r - 1) \cdot p + q] \text{ and } [s + (r + 1) \cdot p + q, s + k \cdot p - 1]. \]

But it is easy to check that no $k - 1$ jobs can start in these intervals. This establishes the validity of (5.24). Let us show now that (5.24) is facet-defining.

Let $F = \{ x \in P : \sum_{j \neq i} \sum_{t \in C} x_{jt} + \sum_{t \in A \setminus D} x_{it} + 2 \cdot \sum_{t \in D} x_{it} = k \},$ and suppose \[ \sum_{j=1}^{n} \sum_{t=1}^{T-p+1} \pi_{jt} x_{jt} = \pi_0 \]
for all $x \in F$.
Consider a solution with job $j$, $j \neq i$, starting at time-unit $t \in C \setminus \{ s + k \cdot p - 1 \}$. Let the other jobs start at $t - p, t - 2p, \ldots$, and at $t + p + 1, t + 2p + 1, \ldots$, while ensuring that $x \in F$ (this is always possible). Shifting job $j$ towards time-unit $t + 1$ proves that

\[ \pi_{jt} = \pi_{jt}^{\text{in}} \text{ for all } j \neq i, \text{ for all } t \in C. \tag{5.25} \]
Now, consider a solution with job \( i \) placed at \( t \), with \( t \in [s, s + l - 1] \), and all other jobs at \( t - p, t - 2p, \ldots, t + p + 1, t + 2p + 1, \ldots \). This can be done in such a way that \( x \in F \), since \( t + p + 1, t + 2p + 1, \ldots, t + (k - 1)p + 1 \) are \( k - 1 \) time-units in \( C \). Now, shifting job \( i \) from \( t \) to \( t + 1 \) proves that

\[
\pi_{it} = \pi_i^{\text{in}} \quad \text{for all } t \in [s, s + l]. \tag{5.26}
\]

A similar argument shows:

\[
\pi_{it} = \pi_i^{\text{in}2} \quad \text{for all } t \in [s + k \cdot p, s + k \cdot p + l - 1]. \tag{5.27}
\]

Consider next a schedule \( x \) with job \( i \) starting at \( t, t \in [s + r \cdot p + l, s + (r + 1) \cdot p - 2] \), for some \( 0 \leq r \leq k - 1 \), and \( k - 1 \) other jobs starting at \( t - r \cdot p, t - (r - 1) \cdot p, \ldots, t - p, t + p + 1, \ldots, t + (k - r - 1) \cdot p + 1 \). Notice that the latter time-units are all in \( C \setminus D \), and hence \( x \in F \). Comparing \( x \) with another schedule in which job \( i \) starts at \( t + 1 \) shows that \( \pi_{it} = \pi_{i, t+1} \) for all \( t \in [s + r \cdot p + l, s + (r - 1) \cdot p - 2] \). Also exchanging job \( i \) with one of the other jobs which start in \( C \setminus D \) shows, in combination with our previous observations (5.25), (5.26) and (5.27) that:

\[
\pi_{it} = \pi_i^{\text{in}1} = \pi_i^{\text{in}2} = \pi_i^1 \quad \text{for all } t \in A \setminus D. \tag{5.28}
\]

Now, consider a solution with job \( i \) starting at \( t \), \( t \in D \) and place the other jobs at \( t - p, t - 2p, \ldots \) and \( t + p, t + 2p, \ldots \), ensuring that the solution is in \( F \) (notice that exactly \( k - 2 \) of these time-units are in \( D \), and hence in \( C \)). Interchanging job \( i \) and job \( j \neq i \) leads easily to

\[
\pi_{it} = \pi_i^2 \quad \text{for all } t \in D. \tag{5.29}
\]

To prove \( \pi_{jt} = \pi_j^\text{out} \) for all \( j \), for all \( t \not\in C \), we refer to the construction used in Theorem 5.5. Moreover, simple interchange arguments imply:

\[
\pi_{j_1j_2}^{\text{out}} + \pi_{j_2j_1}^{\text{in}} = \pi_{j_1j_2}^{\text{in}} + \pi_{j_2j_1}^{\text{out}} \quad \text{for all } j_1, j_2 \neq i,
\]

\[
\pi_i^{1} + \pi_j^{\text{out}} = \pi_i^{\text{out}} + \pi_j^{1} \quad \text{for all } j \neq i, \quad \text{and}
\]

\[
\pi_i^{2} + \pi_j^{\text{out}} = \pi_i^{\text{out}} + \pi_j^{2} \quad \text{for all } j \neq i.
\]

With these last equalities established and together with (5.25), (5.28) and (5.29) the theorem follows easily. \qed

Notice that there are \( O(pn^2T) \) inequalities in the class (5.23). Hence, the separation problem for this class of inequalities can be solved in polynomial time.
5.4 A cutting-plane algorithm for SEL

In this section, we describe an (unsophisticated) cutting-plane algorithm for SEL, based on the results of Section 5.3, and we report on its performance on a few randomly generated problems. The algorithm works as follows. We start with a model consisting solely of the constraints (5.1). This model is solved to optimality (we used the LP-package LINDO). Then the following five classes of inequalities are searched successively in order to find violated inequalities:

- constraints (5.2);
- constraints (5.10) with \( l = 1 \);
- constraints (5.11) and (5.23) with \( k > 1 \);
- constraints (5.22) with \( |J| = 2 \), and \( S = \{ s_1, s_2, s_2 + 1 \} \) for \( 1 \leq s_1 \leq T - 2p + 3 \) and \( 1 \leq s_2 \leq T - 2p + 2 \).

When violated inequalities are found, they are added to the model, the extended model is solved to optimality and the whole process is repeated. When no violated inequalities are detected or if an integral solution is found, the algorithm stops.

A few implementation issues are worth mentioning. First, if violated inequalities in one of the five classes are found, then subsequent classes are not checked. Secondly, at each iteration, only those inequalities are maintained whose slack is smaller than 0.1; all other inequalities are removed from the model. Observe also that, for all classes of valid inequalities used in this algorithm, the separation problem is polynomially solvable.

We generated 35 problem instances distributed over 7 categories. For each category, values of \( n, p \) and \( T \) are specified as well as a range from which the cost-coefficients are randomly drawn (using a uniform distribution); see the first column of Table 5.1. Notice that all cost-coefficients are integer, so that all lowerbounds computed can validly be rounded-up to the next integer. Regarding the choice of \( T \), preliminary experiments indicated that for relatively large values of \( T \) (\( T \geq (p + 1) \cdot n \)) as well as for minimal values of \( T \) (\( T = p \cdot (n + 1) \)), the LP-relaxation of model (5.1)–(5.3) almost always has an integral optimal solution. So, we tried to choose \( T \) in such a way that fractional LP-relaxations arise.

In Table 5.1, LP denotes the value of the LP-relaxation of model (5.1)–(5.3). CPA denotes the value found by the cutting-plane algorithm described
earlier and (i) indicates that the solution found is integral.

<table>
<thead>
<tr>
<th></th>
<th>LP</th>
<th>CPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p = 2 )</td>
<td>1</td>
<td>4 (i)</td>
</tr>
<tr>
<td>( n = 20 )</td>
<td>2</td>
<td>7 (i)</td>
</tr>
<tr>
<td>( T = 46 )</td>
<td>3</td>
<td>8.50 7 (i)</td>
</tr>
<tr>
<td>([0-25])</td>
<td>4</td>
<td>12.33 12.5</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>9 (i) 9 (i)</td>
</tr>
</tbody>
</table>

\[ p = 2 \]
\[ n = 30 \]
\[ T = 102 \]
\[ \text{[0-40]} \]
\[ 4 \] 9 (i) 9 (i)
\[ 5 \] 10.00 10.28

\[ p = 4 \]
\[ n = 20 \]
\[ T = 93 \]
\[ \text{[0-60]} \]
\[ 4 \] 23.00 24.06
\[ 5 \] 23.43 24.83

\[ p = 5 \]
\[ n = 20 \]
\[ T = 114 \]
\[ \text{[0-60]} \]
\[ 4 \] 22.14 22.25
\[ 5 \] 15.96 16 (i)

\[ p = 2 \]
\[ n = 40 \]
\[ T = 86 \]
\[ \text{[0-40]} \]
\[ 4 \] 20.00 20.33
\[ 5 \] 20.67 21.11

\[ p = 3 \]
\[ n = 20 \]
\[ T = 67 \]
\[ \text{[0-25]} \]
\[ 4 \] 8 (i) 8 (i)
\[ 5 \] 9.67 10 (i)

Table 5.1

The cutting-plane algorithm finds 11 times an integral solution (compared to six times for the LP-relaxation of (5.1)–(5.3)). In the other cases, it improves the (rounded-up) value of the lower bound 13 times.

Not surprisingly, the results indicate that the problems get harder when \( p \) and/or \( n \) increase. For the “easier” problems \((n = 20, p = 2, 3)\), the cutting-plane algorithm often finds integral optimal solutions. For the “hard” problems \((p = 2, n = 40 \text{ and } p = 4, 5)\) the algorithm usually improves the lower bound obtained from the LP-relaxation of (5.1)–(5.3). For the remaining problems \((n = 30, p = 2, 3)\) neither did the algorithm find integral solutions nor could it noticeably improve the lower bounds.
Acknowledgements
We are grateful to Antoon Kolen for pointing out the inequalities (5.22) to us.
Chapter 6

Minimizing the number of tool switches on a flexible machine

This chapter is joint work with Yves Crama, Antoon W.J. Kolen and Alwin G. Oerlemans; an article corresponding to this chapter is submitted to the International Journal of Flexible Manufacturing Systems.
6.1 Introduction

The importance of tool management for the efficient use of automated manufacturing systems has been recently stressed by several authors; we refer for instance to Gray, Seidmann and Stecke (1988) or Kiran and Krason (1988) for a thorough discussion of this issue. In particular, a central problem of tool management for flexible machines is to decide how to sequence the parts to be produced, and what tools to allocate to the machine, in order to minimize the number of tool setups. The problem becomes especially crucial when the time needed to change a tool is significant with respect to the processing times of the parts, or when many small batches of different parts must be processed in succession. These phenomena have been observed in the metal-working industry by Hirabayashi, Suzuki and Tsuchiya (1984), Finke and Kusiak (1987), Bard (1988), Tang and Denardo (1988), Bard and Feo (1989), etc. Blazewicz, Finke, Haupt and Schmidt (1988) describe for instance an NC-forging machine equipped with two tool magazines, each of which can handle eight tools. The tools are very heavy, and exchanging them requires a sizeable fraction of the actual forging time. Another situation where minimizing the number of tool setups may be important is described by Förster and Hirt (1989, p. 109). These authors mention that, when the tool transportation system is used by several machines, there is a distinct possibility that this system becomes overloaded. Then, minimizing the number of tool setups can be viewed as a way to reduce the strain on the tool transportation system. Bard (1988) mentions yet another occurrence of the same problem in the electronics industry. Suppose several types of printed circuit boards (PCBs) are produced by an automated placement machine (or a line of such machines). For each type of PCB, a certain collection of component feeders must be placed on the machine before boards of that type can be produced. As the machine can only hold a limited number of feeders, it is usually necessary to replace some feeders when switching from the production of one type of boards to that of another type. Exchanging feeders is a time-consuming operation and it is therefore important to determine a production sequence for the board types which minimizes the number of "feeder-setups". Identifying the feeders with tools, we see that this constitutes again an instance of the "job-sequencing and tool loading" problem evoked above.

This chapter deals with a particular formulation of this problem, due to Bard (1988) and Tang and Denardo (1988). Suppose that a batch of $N$ jobs have to be successively processed, one at a time, on a single flexible
machine. Each job requires a subset of tools, which have to be placed in the tool magazine of the machine before the job can be processed. The number of tools needed to produce all the jobs in the batch is denoted by $M$. We represent these data by an $M \times N$ tool-job matrix $A$, with:

\[ a_{ij} = \begin{cases} 1 & \text{if job } j \text{ requires tool } i, \\ 0 & \text{otherwise}, \end{cases} \]

for $i = 1, 2, \ldots, M$ and $j = 1, 2, \ldots, N$. Without loss of generality, $A$ has no zero row. The tool magazine has a limited capacity; it can accommodate at most $C$ tools, each of which fits in one slot of the magazine. To ensure feasibility of the problem, we assume that no job requires more than $C$ tools. We also assume that, while the jobs are in process, the tool magazine is always loaded at full capacity (as will be explained below, this is in fact a non-restrictive assumption for our problem). We thus call any subset of $C$ tools a loading of the magazine.

A job sequence is a permutation of $\{1, 2, \ldots, N\}$, or, equivalently, of the columns of $A$. As the number of tools needed to produce all jobs is generally larger than the capacity of the tool magazine (i.e., $M > C$), it is sometimes necessary to change tools between two jobs in a sequence. When this occurs, one or more tools are removed from the tool magazine and are replaced by a same number of tools retrieved from a storage area. We call setup the insertion of a tool in the magazine. A switch is the combination of a tool setup and a tool removal. Since each tool has to be set up at least once in order to process the whole batch of jobs, we will also pay attention to the extra setups of a tool, that is, to all setups of the tool other than the first one.

The tool switching problem is now defined as follows: determine a job sequence and an associated sequence of loadings for the tool magazine, such that all tools required by the $j$-th job are present in the $j$-th loading, and the total number of tool switches is minimized. In matrix terms, the tool switching problem translates as follows: determine an $M \times N$ 0–1 matrix $P = (p_{kj})$, obtained by permuting the columns of $A$ according to a given job sequence, and an $M \times N$ 0–1 matrix $T = (t_{kj})$ containing $C$ ones per column (each column of $T$ represents a tool loading), such that $t_{kj} = 1$ if $p_{kj} = 1$ (i.e., tool $k$ is placed in the $j$-th loading if it is needed for the $j$-th job in the sequence: $k = 1, \ldots, M$; $j = 1, \ldots, N$), and the following quantity is minimized:

\[ \sum_{j=2}^{N} \sum_{k=1}^{M} (1 - t_{k,j-1}) \cdot t_{kj} \]
(this quantity is exactly the number of switches required for the loading
sequence represented by $T$). Observe that minimizing the number of tool
switches is equivalent to minimizing the number of setups or of extra setups,
since the following relations hold:

\[
\text{number of setups} = \text{number of switches} + C \\
= \text{number of extra setups} + M.
\]

Let us now briefly discuss some of the (explicit and implicit) assumptions of
the tool switching model.

(1) As mentioned before, the assumption that the tool magazine is always
fully loaded does not affect the generality of the model. Indeed, since
no cost is incurred for tools staying in the magazine, one may consider
that the first $C$ tools to be used are all incorporated in the very first
loading; thereafter, a tool only needs to be removed when it is replaced
by another one.

(2) Each tool is assumed to fit in one slot of the magazine. Removing this
assumption would create considerable difficulties. For instance the
physical location of the tools in the magazine would then become rele-
vant, since adjacent slots would need to be freed in order to introduce
a tool requiring more than one slot.

(3) The time needed to remove or insert each tool is constant, and is
the same for all tools. This assumption is in particular crucial for
the correctness of the KTNS procedure (see Subsection 6.2.2) which
determines the optimal tool loadings for a given job sequence. Many
of our heuristic procedures, however, can easily be adapted in the case
where switching times are tool dependent.

(4) Tools cannot be changed simultaneously. This is a realistic assump-
tion in many situations, e.g. for the forging or for the PCB assembly
applications mentioned above.

(5) The subset of tools required to carry out each job is fixed in advance.
This assumption could be relaxed by assuming instead that, for each
job, a list of subsets of tools is given, and that the job can be executed
by any subset in the list; (i.e., several process plans are given for each
job; see e.g. Finke and Kusiak (1987)). Choosing the right subset
would then add a new dimension (and quite a lot of complexity) to the
problem.
Section 6.2

(6) Tools do not break down and do not wear out. This assumption is justified if the tool life is long enough with respect to the planning horizon. Otherwise, one may want to lift the assumption "deterministically", e.g. by assuming that tool \( k \) is worn out after the execution of \( w_k \) jobs, for a given value of \( w_k \). Alternatively, breakdowns and wear may also be modelled probabilistically. This would obviously result in a completely new model.

(7) The list of jobs is completely known. This assumption is realistic if the planning horizon is relatively short.

This chapter deals with various aspects of the tool switching problem. Section 6.2 contains some basic results concerning the computational complexity of this problem; in particular, we establish that the problem is already \( \mathcal{NP} \)-hard for \( C = 2 \), and we present a new proof of the fact that, for each fixed job sequence, an optimal sequence of tool loadings can be found in polynomial time. In Section 6.3, we describe several heuristics for the tool switching problem, and the performance of these heuristics on randomly generated problems is compared in Section 6.4. Section 6.5 discusses, in general terms, the difficult problem of computing good lower bounds for the optimal value of the tool switching problem. The Appendix contains some graph-theoretic definitions.

6.2 Basic results

We present in this section some results concerning the computational complexity of the tool switching problem. We assume that the reader is familiar with the basic concepts of complexity theory (see e.g. Nemhauser and Wolsey (1988)). Let us simply recall here that, loosely speaking, a problem is \( \mathcal{NP} \)-hard if it is at least as hard as the traveling salesman problem (see the Appendix).

6.2.1 \( \mathcal{NP} \)-hardness results

Tang and Denardo (1988) claim that the tool switching problem is \( \mathcal{NP} \)-hard. They do not present a formal proof of this assertion, but rather infer it from the observation that the problem can be modelled as a traveling salesman problem with variable edge lengths. Our immediate goal will be to establish the validity of two slightly stronger claims.
Consider first the following restricted version of the tool switching problem:

Input: an \( M \times N \) matrix \( A \) and a capacity \( C \).
Problem P1: is there a job sequence for \( A \) requiring exactly \( M \) setups (i.e., no extra setups)?

**Theorem 6.1** Problem P1 is \( \mathcal{NP} \)-hard.

**Proof:**
It is straightforward to check that P1 is precisely the decision version of the so-called matrix permutation problem, which has been extensively investigated in the VLSI design literature (see Möhring (1990) and references therein). Several equivalent versions of the matrix permutation problem have been shown to be \( \mathcal{NP} \)-hard (see Kashiwabara and Fujisawa (1979), Möhring (1990)), and hence P1 is \( \mathcal{NP} \)-hard.

In the description of problem P1, both \( A \) and \( C \) are regarded as problem data. But, from the viewpoint of our application, it may also be interesting to consider the situation where a specific machine, with fixed capacity, has to process different batches of jobs. The matrix \( A \) can then be regarded as the sole data of the tool switching problem. This observation leads us to define the following problem, where \( C \) is now considered as a fixed parameter:

Input: an \( M \times N \) matrix \( A \).
Problem P2: find a job sequence for \( A \) minimizing the number of setups required on a machine with capacity \( C \).

**Theorem 6.2** Problem P2 is \( \mathcal{NP} \)-hard for any fixed \( C \geq 2 \).

**Proof:**
Let \( G = (V, E, d) \) be a graph and \( H = (E, I, \delta) \) be its edge-graph (see Appendix). We consider the problem of finding a minimal length TS path in \( H \) (problem P3 in the Appendix). We are now going to prove Theorem 6.2 by showing that this \( \mathcal{NP} \)-hard problem can be formulated as a special case of problem P2, for any fixed \( C \geq 2 \). For simplicity, we first concentrate on a proof of Theorem 6.2 for \( C = 2 \). Let \( V = \{1, 2, \ldots, M\} \) and \( E = \{e_1, e_2, \ldots, e_N\} \). Define an \( M \times N \) matrix \( A \), with rows associated to the nodes of \( G \), columns associated to the edges of \( G \), and such that:

\[
a_{ij} = 1 \quad \text{if edge } e_j \text{ contains node } i,
= 0 \quad \text{otherwise.}
\]
Consider now $A$ as an instance of the tool switching problem, with capacity $C = 2$. A job sequence for this problem corresponds to a permutation of $E$, and hence to a $TS$ path in the edge-graph of $G$. Also, it is easy to see that the number of tool switches between two jobs $j$ and $k$, corresponding to the edges $e_j$ and $e_k$ of $G$, is:

- equal to 1 if $e_j$ and $e_k$ share a common node, that is, if $\delta(e_j, e_k) = 1$ in $H$;
- equal to 2 if $e_j$ and $e_k$ do not share a common node, that is, if $\delta(e_j, e_k) = +\infty$ in $H$.

This discussion immediately implies that an optimal job sequence for $A$ (with capacity 2) always corresponds to a minimal length $TS$ path in $H$. Hence, we can solve $P3$ by solving $P2$, and this entails that $P2$ is $\mathcal{NP}$-hard. To see that Theorem 6.2 is also valid for $C > 2$, it suffices to adapt the definition of $A$ in the previous argument, by adding $C - 2$ rows of 1's to it; that is, $A$ now has $(M + C - 2)$ rows, and $a_{ij} = 1$ if $i \geq M + 1$. The reasoning goes through with this modification.

6.2.2 Finding the minimum number of setups for a fixed job sequence

The tool switching problem naturally decomposes into two interdependent issues, namely:

1. **sequencing**: compute an (optimal) job sequence, and
2. **tooling**: for the given sequence, determine what tools should be loaded in the tool magazine at each moment, in order to minimize the total number of setups required.

In their paper, Tang and Denardo (1988) proved that the sequencing subproblem actually is the hard nut to crack, since the tooling problem can be solved in $O(MN)$ operations by applying a so-called Keep Tool Needed Soonest (KTNS) policy. A KTNS policy prescribes that, whenever a situation occurs where some tools should be removed from the magazine, so as to make room for tools needed for the next job, then those tools which are needed the soonest for a future job should be removed last (we refer to Tang and Denardo (1988) or Bard (1988) for a more precise description).

Tang and Denardo's proof of the correctness of KTNS relies on ad-hoc interchange arguments and is rather involved (as observed by Finke and
Roger - see Roger (1990) - the correctness of KTNS was already established by Mattson, Gecsei, Slutz and Traiger (1970) in the context of storage techniques for computer memory, in the case where each job requires exactly one tool; their proof is similar to Tang and Denardo’s).

We now look at the tooling subproblem from a different angle, and show that the problem can be modelled as a specially structured 0-1 linear programming problem, which can be solved by a greedy algorithm due to Hoffman, Kolen and Sakarovitch (1985) (see also Nemhauser and Wolsey (1988), pp. 562–573; Daskin, Jones and Lowe (1990) present another application of the same greedy algorithm in a flexible manufacturing context). When translated in the terminology of the tool switching problem, this algorithm precisely yields KTNS. Thus, this argument provides a new proof of correctness for KTNS.

The bulk of the work in our derivation of the KTNS procedure will simply consist in reformulating the tooling problem in an appropriate form. With this goal in mind, we first introduce some new notations and terminology. For the remainder of this section, assume that the job sequence σ is fixed. Let the $M \times N(0,1)$-matrix $P$ be defined by:

$$p_{ij} = \begin{cases} 1 & \text{if tool } i \text{ is required for the } j\text{-th job in } \sigma, \\ 0 & \text{otherwise} \end{cases}$$

(that is, $P$ is obtained by permuting the columns of $A$ according to the job sequence at hand). A tooling policy can now be described by flipping some entries of $P$ from 0 to 1, until each column of $P$ contains exactly $C$ ones. If we denote by $c_j$ the remaining capacity of column $j$, that is the quantity:

$$c_j = C - \sum_{i=1}^{M} p_{ij}$$

then a tooling policy must flip $c_j$ entries from 0 to 1 in the $j$-th column of $P$.

Let us next define a 0-block of $P$ as a maximal subset of consecutive zeroes in a row of $P$. More formally, a 0-block is a set of the form $\{(i, j), (i, j + 1), \ldots, (i, j + k)\}$, for which the following conditions hold:

1. $1 < j \leq j + k < N$,
2. $p_{ij} = p_{i,j+1} = \ldots = p_{i,j+k} = 0$,
3. $p_{i,j-1} = p_{i,j+k+1} = 1$.

Intuitively, a 0-block is a maximal time interval before and after which tool $i$ is needed, but during which it is not needed. It is easy to see that each
Section 6.2

0-block of $P$ is associated with an extra setup of tool $i$. Thus, flipping an element of $P$ from 0 to 1 can only reduce the number of extra setups if this element belongs to a 0-block, and if all other elements of this 0-block are also flipped. In other words, only flipping whole 0-blocks can help reducing the number of setups.

**Example 6.1** The matrix

$$
P = \begin{bmatrix}
0 & 1 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1
\end{bmatrix}
$$

contains three 0-blocks, namely $\{(1,3),(1,4)\}$, $\{(3,2)\}$ and $\{(3,5)\}$. They correspond to an extra setup of tool 1 in period 5, and two extra setups of tool 3, in periods 3 and 6. Assume that the capacity is $C = 2$. Then, the number of extra setups can be minimized by flipping the first and the third 0-blocks to 1, thus resulting in the matrix:

$$
T = \begin{bmatrix}
0 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 1
\end{bmatrix}
$$

From the previous discussion, it should now be clear that the tooling problem can be rephrased as follows: flip to 1 as many 0-blocks of $P$ as possible, while flipping at most $c_j$ entries in column $j$ ($j = 1, 2, \ldots, N$).

Denote by $B$ the number of 0-blocks in $P$, and, for $k = 1, 2, \ldots, B$, introduce the decision variables:

$$
x_k = \begin{cases}
1 & \text{if the } k-\text{th 0-block is flipped to 1,} \\
0 & \text{otherwise.}
\end{cases}
$$

For $j = 1, 2, \ldots, N$ and $k = 1, 2, \ldots, B$, let also:

$$
m_{jk} = \begin{cases}
1 & \text{if the } k-\text{th 0-block "meets" column } j \text{ in } P, \\
0 & \text{otherwise}
\end{cases}
$$

(formally, a 0-block meets column $j$ if it contains an element of the form $(i,j)$, for some $i$; for instance, in Example 6.1, the first 0-block meets columns 3 and 4).
Now, the tooling problem admits the following 0-1 linear programming formulation:

\[(TP) \quad \text{maximize } \sum_{k=1}^{B} x_k\]

subject to \[\sum_{k=1}^{B} m_{jk} x_k \leq c_j, \quad (j = 1, 2, \ldots, N)\]

\[x_k \in \{0, 1\},\]

Assume now that the 0-blocks of \(P\) have been ordered in non-decreasing order of their "endpoints": that is, the 0-blocks of \(P\) have been numbered from 1 to \(B\) in such a way that the index of the last column met by the \(k\)-th 0-block is smaller than or equal to the index of the last column met by the \((k+1)\)-st 0-block, for \(k = 1, \ldots, B-1\). Then, the matrix \((m_{jk})\) is a so-called greedy matrix, i.e. it does not contain the matrix \[
\begin{bmatrix}
1 & 1 \\
1 & 0
\end{bmatrix}
\] as a submatrix.

Hoffman et al. (1985) considered the following, more general problem on an \(N \times B\) greedy matrix:

\[(GP) \quad \text{maximize } \sum_{k=1}^{B} b_k x_k\]

subject to \[\sum_{k=1}^{B} m_{jk} x_k \leq c_j, \quad (j = 1, 2, \ldots, N)\]

\[0 \leq x_k \leq d_k, x_k \text{ integer }, \quad (k = 1, 2, \ldots, B)\]

where \(b_k, d_k (k = 1, 2, \ldots, B)\) and \(c_j (j = 1, 2, \ldots, N)\) are integers with \(b_1 \geq b_2 \geq \ldots \geq b_B\). They proved that, when the matrix \((m_{jk})\) is greedy, problem \((GP)\) can be solved by a greedy algorithm, in which each \(x_k (k = 1, 2, \ldots, B)\) is successively taken as large as possible while respecting the feasibility constraints. Reformulating this algorithm for \((TP)\), we see that we should successively flip 0-blocks to 1, in order of nondecreasing endpoints, as long as the remaining capacity of all columns met by the 0-block is at least one. We leave it to the reader to check that this procedure is precisely equivalent to a KTNS policy.

Remark. In a more general situation where the setup times are not identical for all tools, the tooling subproblem can still be formulated as a problem
Section 6.3

of the form \((GP)\), where \(t_k\) is now the time required to set up the tool associated with the \(k\)-th 0-block. Since the condition \(b_1 \geq b_2 \geq \ldots \geq b_h\) does not generally hold for these setup times, the greedy algorithm of Hoffman et al. (1985) and KTNS are no longer valid. However, the matrix \((m_{jk})\), being an interval matrix, is totally unimodular (see Subsection 6.3.4 and Nemhauser and Wolsey (1988) for definitions). It follows that the tooling subproblem can still be solved in polynomial time in that case, by simply solving the linear programming relaxation of the formulation \((GP)\).

6.3 Heuristics

The tool switching problem being \(NP\)-hard, and hence probably difficult to solve to optimality, we concentrate in the sequel on heuristic techniques for its solution. We propose hereunder six basic approaches, falling into two main categories (we adopt the terminology used by Golden and Stewart (1985) for the traveling salesman problem):

- **construction strategies**, which exploit the special structure of the tool switching problem in order to construct a single (hopefully good) job sequence (Subsections 6.3.1 to 6.3.4 below);

- **improvement strategies**, which iteratively improve a starting job sequence (Subsections 6.3.5 and 6.3.6 below).

**Composite strategies** will be obtained by combining construction and improvement procedures. A computational comparison of the resulting procedures will be presented in Section 6.4.

As explained in Section 6.1, the data of our problem consist of an \(M \times N\) tool-job matrix \(A\) and a capacity \(C\). We focus on the solution of the sequencing subproblem (see Subsection 6.2.2), since we already know that the tooling subproblem is easy to solve. Whenever we speak of the cost of a (partial) job sequence, we mean the minimal number of tool switches required by the sequence, as computed using KTNS.

6.3.1 Traveling salesman heuristics

These heuristics are based on an idea suggested by Tang and Denardo (1988). They consider a graph \(G = (V, E, lb)\) (see Appendix 1 for definitions), where \(V\) is the set of jobs, \(E\) is the set of all pairs of jobs, and the length \(lb(i, j)\) of edge \(\{i, j\}\) is an underestimate of the number of tool switches needed between
jobs \( i \) and \( j \) when these jobs are consecutively processed in a sequence. More precisely:

\[
lb(i,j) = \max(|T_i \cup T_j| - C, 0),
\]

where \( T_k \) is the set of tools required by job \( k \) (\( k = 1, 2, \ldots, N \)). Notice that, if each job requires exactly \( C \) tools (i.e. \( |T_k| = C \) for all \( k \)), then \( lb(i,j) \) is equal to the number of tool switches required between jobs \( i \) and \( j \) in any schedule.

Each traveling salesman (TS) path of \( G \) corresponds to a job sequence for the tool switching problem. So, as suggested by Tang and Denardo (1988), computing a short TS path in \( G \) constitutes a reasonable heuristic for the generation of a good sequence. As a matter of fact, when all jobs use full capacity, then the tool switching problem is precisely equivalent to the TS problem on \( G \).

In our computational experiments, we have considered the following procedures for constructing a short TS path in \( G \):

1. **Shortest Edge** heuristic: this is the heuristic used by Tang and Denardo (1988), and called "greedy feasible" in Nemhauser and Wolsey (1988); complexity: \( O(N^2 \log N) \);

2. **Nearest Neighbor** heuristic with all possible starting nodes: see Golden and Stewart (1985), Johnson and Papadimitriou (1985); complexity: \( O(N^3) \);

3. **Farthest Insertion** heuristic with all possible starting nodes: see Golden and Stewart (1985), Johnson and Papadimitriou (1985); complexity: \( O(N^4) \);

4. **B & B** algorithm: this is a state-of-the-art branch and bound code, which solves TS problems to optimality: see Volgenant and Jonker (1982); complexity: exponential in the worst-case.

Procedures (1), (2) and (3) are well-known heuristics for the traveling salesman problem. In addition to the complexity mentioned for each procedure, an overhead of \( O(MN^2) \) operations has to be incurred for the computation of the edge lengths \( lb(i,j) \).

### 6.3.2 Block minimization heuristics

We describe now another way of associating a traveling salesman instance to any given instance of the tool switching problem. We first introduce a
Section 5.3

directed graph \( D = (V^*, U, ub) \). Here, \( V^* \) is the set of all jobs, plus an additional node denoted by 0. Each ordered pair of nodes is an arc in \( U \).
The length \( ub(i, j) \) of arc \( (i, j) \) is given by:
\[
ub(i, j) = |T_i \setminus T_j|,
\]
where \( T_k \) is the set of tools required by job \( k (k = 1, 2, \ldots, N) \), and \( T_0 \) is the empty set. In words, \( ub(i, j) \) is the number of tools used by job \( i \) but not by job \( j \); hence, \( ub(i, j) \) is an upper-bound on the number of tool switches between jobs \( i \) and \( j \), for any sequence in which \( i \) and \( j \) must be consecutively processed. If every job requires exactly \( C \) tools, then \( ub(i, j) = ub(j, i) = lb(i, j) \) is equal to the number of switches between \( i \) and \( j \). But in general, \( ub(i, j) \) differs from \( ub(j, i) \).
Each TS path of \( D \) finishing at node 0 defines a sequence of jobs, and the length of the path is an upper-bound on the total number of switches entailed by the sequence. For reasons explained below, we refer to heuristics which attempt to construct a short TS path in \( D \) as block minimization heuristics. We have implemented two such heuristics:

(1) \textit{NN Block Minimization}, based on a nearest neighbor heuristic with all possible starting nodes; complexity: \( O(N^3) \);

(2) \textit{PI Block Minimization}, based on a farthest insertion heuristic with all possible starting nodes; complexity: \( O(N^4) \).

Let us mention another interesting interpretation of the block minimization approach. As in Subsection 6.2.2, consider the matrix \( P \) obtained after permuting the columns of \( A \) according to a job sequence \( \sigma \). We define a 1-block of \( P \) as a set of entries, of the form \( \{(i, j), (i, j + 1), \ldots, (i, j + k)\} \), for which the following conditions hold:

(1) \( 1 \leq j \leq j + k \leq N \),

(2) \( p_{ij} = p_{i, j+1} = \ldots = p_{i, j+k} = 1 \),

(3) either \( j = 1 \) or \( p_{i, j-1} = 0 \),

(4) either \( j + k = N \) or \( p_{i, j+k+1} = 0 \)

(this definition does not exactly mimic the definition of 0-blocks, but the difference is irrelevant here). Notice that, were it not for the possibility to carry out KTNS on \( P \), then each 1-block of \( P \) would induce a tool setup in
the job sequence \( \sigma \). Thus, the number of 1-blocks of \( P \) is an overestimate of the number of setups required by \( \sigma \).

We leave it to the reader to check that the number of 1-blocks in \( P \) is also equal to the length of the TS path associated with \( \sigma \) in \( D \) (and finishing at node 0). So, finding a shortest TS path in \( D \) is equivalent to determining a permutation of the columns of \( A \) which minimizes the number of 1-blocks in the permuted matrix. This observation is essentially due to Kou (1977). Kou (1977) also proved that finding a permutation which minimizes the number of 1-blocks is \( \mathcal{NP} \)-hard (our proof of Theorem 6.2 establishes the same result). This justifies the use of heuristics in our block minimization approach.

### 6.3.3 Greedy heuristics

One of the obvious drawbacks of the heuristics described in Subsections 6.3.1 and 6.3.2 is that they do not take a whole job sequence into account when estimating the number of tool switches required between pairs of jobs. For instance, \( lb(i, j) \) is in general only a lower-bound on the actual number of switches between jobs \( i \) and \( j \), and this lower-bound can sometimes be a quite poor estimate of the actual value. An extreme case would arise when no job requires more than \( C/2 \) tools; then, \( lb(i, j) = 0 \) for each pair \( (i, j) \), and any traveling salesman heuristic based on these edge-lengths picks a random job sequence! Similarly, \( ub(i, j) \) can also be a rough upper-bound on the number of switches required. In order to alleviate this difficulty, we propose now the following (Simple) Greedy heuristic:

**Step 1** start with the partial job sequence \( \sigma = (1) \); let \( Q = \{2, 3, \ldots, N\} \).

**Step 2** for each job \( j \) in \( Q \), let \( c(j) \) be the cost of the partial sequence \((\sigma, j)\) (i.e., the number of tool switches entailed by this partial sequence, disregarding the remaining jobs).

**Step 3** let \( i \) be a job in \( Q \) for which \( c(i) = \min_{j \in Q} c(j) \); let \( \sigma := (\sigma, i) \) and \( Q := Q \setminus \{i\} \).

**Step 4** if \( Q \) is not empty, then repeat Step 2; else, stop with the complete sequence \( \sigma \).

Greedy runs in time \( O(MN^3) \), since it requires \( O(N^2) \) applications of the KTNS procedure (in Step 2). Its empirical performance can be slightly improved by taking advantage of the fact that all the partial sequences considered in Step 2 share the same initial segment.
Section 6.3

Of course, there is no mandatory reason to select job 1 first in Step 1 of Greedy, rather than any other job. This observation suggests to consider the following, more elaborate *Multiple-Start Greedy* heuristic: run \( N \) times Greedy, once for each initial sequence \( \sigma = (j) \ (j = 1, 2, \ldots, N) \), and retain the best complete sequence found. This heuristic clearly dominates Greedy, in terms of the quality of the job sequence that it produces. Its worst-case complexity is \( O(MN^4) \).

As a final note on this approach, it may be interesting to observe that, if each job requires exactly \( C \) tools, then Multiple-Start Greedy is identical to the TS Nearest Neighbor heuristic (Subsection 6.3.1) or to the NN block minimization heuristic (Subsection 6.3.2).

6.3.4 Interval heuristic

In order to motivate our next heuristic, let us first consider a special situation: assume that the matrix \( P \) arising by permuting the columns of \( A \) according to some sequence \( \sigma \) has precisely one 1-block in each row. In other words, the ones in each row of \( P \) occur consecutively. When this is the case we say that \( A \) is an *interval matrix* (or that \( A \) has the *consecutive ones property*; see e.g. Fulkerson and Gross (1965), Booth and Lueker (1976), Nemhauser and Wolsey (1988)). Then, the job sequence \( \sigma \) requires only one setup per tool, and is obviously optimal.

Thus, every \( M \times N \) interval matrix admits an optimal sequence with \( M \) setups. Moreover, given an arbitrary matrix \( A \), one can decide in time \( O(MN) \) whether \( A \) is an interval matrix, and, in the affirmative, one can find within the same time bound a sequence entailing \( M \) setups for \( A \) (Booth and Lueker (1976)) (notice that this does not contradict Theorem 6.1: by applying KTNS, a sequence with \( M \) setups can sometimes be found for non-interval matrices). On the other hand, it is by no means clear that any of the heuristics described in Subsections 6.3.1, 6.3.2 or 6.3.3 would find an optimal job sequence for an interval matrix.

These observations suggest the implementation of the following *Interval heuristic*. The heuristic simultaneously builds a "large" interval submatrix of \( A \), and computes an optimal job sequence for the submatrix. This sequence is the solution returned by the heuristic. More precisely:

**Step 1** initialize \( I = \{\} \), \( i = 1 \).

**Step 2** determine whether the submatrix of \( A \) consisting of the rows with index in \( I \cup \{i\} \) is an interval matrix; if so, then let \( I := I \cup \{i\} \) and let \( \sigma \) be an optimal job sequence for the submatrix;
else, continue.

Step 3 if $i < M$, then let $i := i + 1$ and go to Step 2; else, continue.

Step 4 return the last job sequence found; stop.

The Interval heuristic has the attractive property that it produces an optimal job sequence for every interval matrix. The complexity of the heuristic is $O(MN)$ if the algorithm by Booth and Lueker (1976) is used. In our implementation, we have used a slower, but simpler recognition algorithm for interval matrices, due to Fulkerson and Gross (1965).

In the following subsections, we concentrate on improvement strategies. The input for each procedure is some initial job sequence $\sigma$, that we subsequently attempt to improve in an iterative way.

### 6.3.5 2-Opt strategies

This class of strategies is based on an idea that has been widely used for other combinatorial optimization problems: given a sequence $\sigma$, try to produce a better sequence by exchanging two jobs in $\sigma$ (if $i$ is the $k$-th job and $j$ is the $p$-th job in $\sigma$, then exchanging $i$ and $j$ means putting $i$ in $p$-th position and $j$ in $k$-th position). We have considered two versions of this basic approach. The first one, called Global 2-Opt, can be described as follows:

**Step 1** find two jobs $i$ and $j$ whose exchange results in an improved sequence; if there are no such jobs, then return $\sigma$ and stop; else, continue.

**Step 2** exchange $i$ and $j$; call $\sigma$ the resulting sequence; repeat Step 1.

Global 2-Opt has been proposed by Bard (1988) for the tool switching problem. Notice that each execution of Step 1 requires $O(N^2)$ applications of KTNS, i.e. $O(MN^3)$ operations. But the number of potential executions of this step does not appear to be trivially bounded by a polynomial in $N$ and $M$ (contrary to what is claimed by Bard (1988)). In order to reduce the computational effort by iteration of Global 2-Opt, the following Restricted 2-Opt procedure can also be considered.

**Step 1** find two consecutive jobs in $\sigma$, say the $k$-th and $(k+1)$-st ones, whose exchange results in an improved sequence; if there are no such jobs, then return $\sigma$ and stop.

**Step 2** exchange the jobs found in Step 1; call $\sigma$ the resulting sequence;
Section 6.3

repeat Step 1.

The complexity of Step 1 in Restricted 2-Opt is \(O(MN^2)\). This exchange strategy has also been proposed by Finke and Roger (see Roger (1990)).

6.3.6 Load-and-Optimize strategy

Consider again a job sequence \(\sigma\) and the matrix \(P\) obtained by permuting the columns of \(A\) according to \(\sigma\). Applying KTNS to \(P\) results in a new matrix \(T\), each column of which contains exactly \(C\) ones (the \(j\)-th column of \(T\) describes the loading of the tool magazine while the \(j\)-th job in \(\sigma\) is being processed). Suppose now that we look at \(T\) as defining a new instance of the tool switching problem (with capacity \(C\)). If we can find for \(T\) a better sequence than \(\sigma\), then this sequence will obviously be a better sequence than \(\sigma\) for the original matrix \(A\) as well. On the other hand, the problem instance \((T, C)\) is a little bit easier to handle than the instance \((A, C)\). Indeed, since each column of \(T\) contains \(C\) ones, the tool switching problem \((T, C)\) can be reformulated as a TS problem, as explained in Subsections 6.3.1, 6.3.2, 6.3.3. These observations motivate our Load-and-Optimize strategy:

**Step 1** permute the columns of \(A\) according to \(\sigma\) and apply KTNS; call \(T\) the resulting matrix.

**Step 2** compute an optimal sequence \(\sigma'\) for the tool switching instance \((T, C)\).

**Step 3** if \(\sigma'\) is a better sequence than \(\sigma\) for \(A\), then replace \(\sigma\) by \(\sigma'\) and repeat Step 1; else return \(\sigma\) and stop.

From a practical viewpoint, we have found it easier to slightly alter this basic strategy, in the following way. In Step 2, rather than computing an optimal sequence for \(T\) (which is computationally demanding), we simply use the farthest insertion heuristic to produce a good sequence \(\sigma'\) (as in Subsection 6.3.1). On the other hand, in Step 3, we accept the new sequence \(\sigma'\) even if it entails the same number of setups as \(\sigma\). We only stop when 10 iterations of the procedure have been executed without producing a strictly improved sequence. In the sequel, we also refer to this variant as "Load-and-Optimize".
6.4 Computational experiments

6.4.1 Generation of problem instances

We tested our heuristics on 160 random instances of the tool switching problem. Of course, tool-job matrices occurring in practice may have characteristics not present in the ones we generated. For instance, as pointed out by an anonymous referee, realistic matrices are likely to display inter-row and inter-column correlations, as well as "tool clusters". However, in the absence of real-world data or even of detailed statistical information about these, we decided to follow a procedure similar to the one proposed by Tang and Denardo (1988) in generating our test problems.

Each random instance falls into one of 16 instance types, characterized by the size \((M, N)\) of the tool-job matrix and by the value \(C\) of the capacity. Accordingly, we denote the type of an instance by a triple \((M, N, C)\). There are 10 instances of each type. The tool-job matrices are \(M \times N\) matrices, where \((M, N)\) is either \((10,10)\), \((20,15)\), \((40,30)\) or \((60,40)\). For each size \((M, N)\), we also define a pair \((\text{Min}, \text{Max})\) of parameters with the following interpretation:

- \(\text{Min}\) = lower-bound on the number of tools per job,
- \(\text{Max}\) = upper-bound on the number of tools per job.

The specific values of these parameters are displayed in Table 6.1.

<table>
<thead>
<tr>
<th>Problem size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10,10)</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>(20,15)</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>(40,30)</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>(60,40)</td>
<td>7</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 6.1

For each problem size \((M, N)\), 10 random matrices \(A\) were generated. For each \(j = 1, 2, \ldots, N\), the \(j\)-th column of \(A\) was generated as follows. First, an integer \(t_j\) was drawn from the uniform distribution over \([\text{min}, \text{max}]\): this number denotes the number of tools needed for job \(j\), i.e. the number of 1's in the \(j\)-th column of \(A\). Next, a set \(T_j\) of \(t_j\) distinct integers were drawn from the uniform distribution over \([1, M]\): these integers denote the tools required by job \(j\), i.e. \(a_{kj} = 1\) if and only if \(k\) is in \(T_j\). Finally, we checked
whether $T_j \subseteq T_i$ or $T_i \subseteq T_j$ held for any $i < j$. If any of these inclusions was found to hold, then the previous choice of $T_j$ was cancelled, and a new set $T_j$ was generated (Tang and Denardo (1988) and Bard (1988) have observed that any column of $A$ contained in another column can be deleted without affecting the optimal solution of the problem; thus, we want to make sure that our problem instances actually involve $N$ columns, and cannot be reduced by this simple trick). Notice that this generation procedure does not a priori prevent the occurrence of null rows in the matrix. In practice, only two of the 40 matrices that we generated contained null rows (these were two (20,15) matrices, containing respectively one and three null rows).

A problem instance of type $(M,N,C)$ is now obtained by combining an $M \times N$ tool-job matrix $A$ with one of the four capacities $C_1$, $C_2$, $C_3$ and $C_4$ displayed in Table 6.2.

<table>
<thead>
<tr>
<th>Problem size</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
<th>$C_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10,10)</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>(20,15)</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>(40,30)</td>
<td>15</td>
<td>17</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>(60,40)</td>
<td>20</td>
<td>22</td>
<td>25</td>
<td>30</td>
</tr>
</tbody>
</table>

**Table 6.2**

We will see that the performance of some heuristics strongly depends on the value of the ratio $\max / C$. We call sparse those problem instances for which $\max / C$ is small, and dense those for which the ratio is close to 1. Notice, in particular, that all instances of type $(M,N,C_1)$ have $\max / C_1 = 1$. Varying the capacity as indicated in Table 6.2 will allow us to examine the behavior of our heuristics under different sparsity conditions. Let us mention here that, according to the empirical observation of many real-world systems described by Förster and Hirt (1989), sparse instances are probably more "realistic" than dense ones. But of course, this conclusion is very much system-dependent.

### 6.4.2 Computational results

All heuristics described in Section 6.3 have been implemented in Turbo Pascal and tested on the problem instances described above. The experiments were run on an AT personal computer equipped with an 80286 microprocessor and an additional 80287 coprocessor. Since our primary goal was
to compare the quality of the solutions produced by the heuristics, no systematic attempts were made to optimize the running time of the codes. Accordingly, we will not report here on precise computing times, but simply give some rough indication of the relation between the times required by the various methods.

The performance of heuristic $H$ on problem instance $I$ is measured in terms of "percentage above the best solution found", namely, by the quantity:

$$\delta_H(I) = \left( \frac{H(I) - \text{Best}(I)}{\text{Best}(I)} \right) \cdot 100,$$

where $H(I)$ is the number of tool setups required by the job sequence produced by heuristic $H$, and Best($I$) is the number of setups required by the best sequence found by any of our heuristics.

For information, Table 6.3 indicates the evolution of Best($I$) as a function of the problem type (average of Best($I$) over all ten instances of each type). All subsequent tables (Tables 6.4, 6.5, 6.6 report averages and (in brackets) standard deviations of $\delta_H(I)$ over all instances $I$ of a given type.

<table>
<thead>
<tr>
<th>Problem size</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
<th>$C_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10,10)</td>
<td>13.2</td>
<td>11.2</td>
<td>10.3</td>
<td>10.1</td>
</tr>
<tr>
<td>(20,15)</td>
<td>26.5</td>
<td>21.6</td>
<td>20.0</td>
<td>19.6</td>
</tr>
<tr>
<td>(40,30)</td>
<td>113.6</td>
<td>95.9</td>
<td>76.8</td>
<td>56.8</td>
</tr>
<tr>
<td>(60,40)</td>
<td>211.6</td>
<td>189.7</td>
<td>160.5</td>
<td>127.4</td>
</tr>
</tbody>
</table>

Table 6.3
Table 6.4 compares the behavior of the four traveling salesman heuristics described in Subsection 6.3.1. We will see later that TS heuristics perform best on dense instances, and tend to behave very badly on sparse instances. Therefore, we limit ourselves here to a comparison of these heuristics on the densest instances, that is, those instances where $C = C_1 = \text{max}$.  

From Table 6.4, it appears that on average, and mostly for large instances, Farthest Insertion yields better solutions than the other TS heuristics. Farthest Insertion is also a very fast heuristic, which produces solutions in a matter of seconds (about 30 seconds for the largest instances). The Shortest Edge and Nearest Neighbor heuristics are even faster, but Farthest Insertion presents in our view the best quality vs. efficiency trade-off. Thus, we will select Farthest Insertion as our “winner” among TS heuristics, and no longer report on the other TS heuristics in the sequel.

A similar comparison between the two block minimization heuristics presented in Subsection 6.3.2 would lead to similar conclusions. Here again, FI is slightly better and slightly slower than NN. In the remainder of this section, we only report on the performance of FI, and no longer of NN.

Tables 6.5 displays the performance of “constructive” and “improvement” heuristics over our complete sample of problem instances. The results (averages and standard deviations) for each heuristic are given in different columns.

The results presented under the labels “2-Opt” or “Load-and-Optimize” have been obtained by first picking a random job sequence, and then applying the corresponding improvement strategies to it. The columns labelled “Random” provide, for the sake of comparison, the number of tool setups entailed by the initial random job sequence.
<table>
<thead>
<tr>
<th>(M,N,C)</th>
<th>Farthest Insertion</th>
<th>FI Block Minimization</th>
<th>Simple Greedy</th>
<th>Multiple-Start Greedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10,10,4)</td>
<td>12.1 (9.8)</td>
<td>14.3 (7.7)</td>
<td>12.3 (6.3)</td>
<td>4.6 (3.8)</td>
</tr>
<tr>
<td>(10,10,5)</td>
<td>19.0 (7.8)</td>
<td>13.6 (7.6)</td>
<td>8.1 (6.0)</td>
<td>3.7 (4.6)</td>
</tr>
<tr>
<td>(10,10,6)</td>
<td>17.8 (10.8)</td>
<td>9.7 (6.4)</td>
<td>5.7 (4.7)</td>
<td>2.9 (4.4)</td>
</tr>
<tr>
<td>(10,10,7)</td>
<td>11.7 (10.3)</td>
<td>3.9 (4.8)</td>
<td>1.0 (3.0)</td>
<td>0.0 (0.0)</td>
</tr>
<tr>
<td>(20,15,6)</td>
<td>15.5 (8.6)</td>
<td>12.0 (4.2)</td>
<td>13.7 (7.6)</td>
<td>4.6 (3.5)</td>
</tr>
<tr>
<td>(20,15,8)</td>
<td>37.3 (10.8)</td>
<td>13.9 (8.4)</td>
<td>11.0 (7.3)</td>
<td>4.6 (3.0)</td>
</tr>
<tr>
<td>(20,15,10)</td>
<td>30.5 (5.8)</td>
<td>8.3 (6.2)</td>
<td>5.6 (4.3)</td>
<td>1.5 (2.3)</td>
</tr>
<tr>
<td>(20,15,12)</td>
<td>15.3 (5.5)</td>
<td>2.1 (3.5)</td>
<td>1.0 (2.1)</td>
<td>0.0 (0.0)</td>
</tr>
<tr>
<td>(40,30,15)</td>
<td>9.4 (5.3)</td>
<td>8.6 (4.4)</td>
<td>11.4 (4.8)</td>
<td>6.2 (3.1)</td>
</tr>
<tr>
<td>(40,30,17)</td>
<td>16.3 (7.5)</td>
<td>9.4 (3.8)</td>
<td>9.8 (3.5)</td>
<td>5.5 (2.2)</td>
</tr>
<tr>
<td>(40,30,20)</td>
<td>33.8 (9.1)</td>
<td>12.1 (3.6)</td>
<td>9.8 (4.2)</td>
<td>3.2 (2.0)</td>
</tr>
<tr>
<td>(40,30,25)</td>
<td>39.4 (6.6)</td>
<td>15.0 (2.7)</td>
<td>8.3 (4.9)</td>
<td>2.6 (2.3)</td>
</tr>
<tr>
<td>(60,40,20)</td>
<td>6.9 (2.7)</td>
<td>9.7 (2.4)</td>
<td>10.2 (2.6)</td>
<td>5.8 (1.5)</td>
</tr>
<tr>
<td>(60,40,22)</td>
<td>9.9 (2.7)</td>
<td>8.7 (2.6)</td>
<td>7.9 (3.1)</td>
<td>3.3 (1.7)</td>
</tr>
<tr>
<td>(60,40,25)</td>
<td>21.8 (5.7)</td>
<td>10.5 (3.1)</td>
<td>8.2 (2.8)</td>
<td>2.8 (2.0)</td>
</tr>
<tr>
<td>(60,40,30)</td>
<td>36.7 (4.0)</td>
<td>13.1 (3.7)</td>
<td>6.5 (2.4)</td>
<td>1.7 (1.4)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(M,N,C)</th>
<th>Interval 2-opt</th>
<th>Restricted 2-opt</th>
<th>Global 2-opt</th>
<th>Load-and-Optimize</th>
<th>Random 2-opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10,10,4)</td>
<td>22.6 (12.2)</td>
<td>26.0 (7.7)</td>
<td>8.7 (4.7)</td>
<td>5.8 (5.3)</td>
<td>41.2 (18.9)</td>
</tr>
<tr>
<td>(10,10,5)</td>
<td>14.1 (14.1)</td>
<td>24.3 (10.1)</td>
<td>7.4 (7.1)</td>
<td>10.1 (7.2)</td>
<td>33.8 (16.2)</td>
</tr>
<tr>
<td>(10,10,6)</td>
<td>9.7 (11.8)</td>
<td>18.3 (7.7)</td>
<td>3.0 (4.6)</td>
<td>6.7 (4.4)</td>
<td>26.3 (9.1)</td>
</tr>
<tr>
<td>(10,10,7)</td>
<td>3.0 (6.4)</td>
<td>9.8 (7.5)</td>
<td>0.0 (0.0)</td>
<td>3.0 (6.4)</td>
<td>13.8 (7.9)</td>
</tr>
<tr>
<td>(20,15,6)</td>
<td>25.7 (9.7)</td>
<td>33.6 (7.2)</td>
<td>10.0 (4.3)</td>
<td>12.3 (6.8)</td>
<td>45.9 (8.8)</td>
</tr>
<tr>
<td>(20,15,8)</td>
<td>20.4 (9.2)</td>
<td>35.7 (10.8)</td>
<td>9.7 (4.1)</td>
<td>23.8 (8.5)</td>
<td>42.2 (11.8)</td>
</tr>
<tr>
<td>(20,15,10)</td>
<td>10.4 (8.2)</td>
<td>24.3 (9.2)</td>
<td>6.4 (7.3)</td>
<td>25.6 (11.7)</td>
<td>30.1 (12.3)</td>
</tr>
<tr>
<td>(20,15,12)</td>
<td>3.5 (5.9)</td>
<td>13.6 (8.3)</td>
<td>1.0 (2.0)</td>
<td>16.6 (9.6)</td>
<td>18.1 (11.3)</td>
</tr>
<tr>
<td>(40,30,15)</td>
<td>30.5 (4.3)</td>
<td>30.3 (5.0)</td>
<td>6.0 (4.0)</td>
<td>16.6 (5.3)</td>
<td>42.9 (6.1)</td>
</tr>
<tr>
<td>(40,30,17)</td>
<td>31.2 (5.4)</td>
<td>31.0 (4.6)</td>
<td>4.5 (3.3)</td>
<td>27.5 (4.3)</td>
<td>44.6 (6.4)</td>
</tr>
<tr>
<td>(40,30,20)</td>
<td>30.4 (6.0)</td>
<td>33.0 (6.6)</td>
<td>6.0 (2.9)</td>
<td>35.1 (6.4)</td>
<td>45.5 (8.9)</td>
</tr>
<tr>
<td>(40,30,25)</td>
<td>27.8 (6.6)</td>
<td>34.5 (7.4)</td>
<td>8.1 (3.7)</td>
<td>37.8 (7.0)</td>
<td>40.5 (7.1)</td>
</tr>
<tr>
<td>(60,40,20)</td>
<td>30.6 (2.7)</td>
<td>25.8 (3.8)</td>
<td>4.8 (2.4)</td>
<td>20.0 (3.8)</td>
<td>37.1 (3.6)</td>
</tr>
<tr>
<td>(60,40,22)</td>
<td>29.3 (4.1)</td>
<td>25.4 (2.9)</td>
<td>3.7 (2.6)</td>
<td>25.4 (4.1)</td>
<td>36.5 (3.5)</td>
</tr>
<tr>
<td>(60,40,25)</td>
<td>30.2 (3.6)</td>
<td>29.7 (3.0)</td>
<td>2.1 (1.9)</td>
<td>35.5 (4.3)</td>
<td>38.0 (3.6)</td>
</tr>
<tr>
<td>(60,40,30)</td>
<td>28.8 (3.4)</td>
<td>30.1 (3.3)</td>
<td>4.5 (2.7)</td>
<td>36.7 (4.4)</td>
<td>37.6 (3.8)</td>
</tr>
</tbody>
</table>

Table 6.5 Average (and standard deviation) of \( \delta_H(I) \).
Let us now try to sketch some of the conclusions that emerge from this table. Consider first the case of dense instances, that is, the instances of type (10, 10, 4), (20, 15, 6), (40, 30, 15) and (60, 40, 20). As the size of these instances increases, the ranking of the solutions delivered by the various heuristics seems to become more or less stable. Namely, Multiple-Start Greedy and Global 2-Opt produce (on the average) the best results. Next comes a group made up of Farthest Insertion, Simple Greedy and FI Block Minimization, which usually yield solutions of slightly lower quality. Finally, the worst solutions are produced by Load-and-Optimize, Restricted 2-Opt and Interval (and, as expected, the random procedure).

We get a somewhat different ranking of the heuristics when we look at sparse instances. Consider e.g. the instances of type (10, 10, 7), (20, 15, 12), (40, 30, 25), (60, 40, 30). Multiple-Start Greedy, Global 2-Opt, Simple Greedy and FI Block Minimization remain, in that order, the best heuristics. But Farthest Insertion performs now almost as badly as the random procedure! As a matter of fact, for larger instances, it appears that the performance of Farthest Insertion deteriorates very systematically as sparsity increases. This behavior is matched by all other TS heuristics (Shortest Edge, Nearest Neighbor, and B& B). It can be explained by observing that, for sparse instances, the bounds \( l_o(i, j) \) tend to be poor estimates of the number of switches required between jobs \( i \) and \( j \) (see Subsections 6.3.1 and 6.3.3).

Our conclusion at this point would be that, if we are only concerned with the quality of the solution produced by each heuristic, then Multiple-Start Greedy and Global 2-Opt come out the winners, while Simple Greedy and FI Block Minimization are good contenders. For dense problems, Farthest Insertion also is a very good technique.

This first picture becomes more nuanced when we also take computing times into account. Indeed, the various heuristics run at very different speeds. For instance, solving an instance of type (10, 10, 4) takes about 0.30 seconds by Farthest Insertion, FI Block Minimization or by Simple Greedy, 2 seconds by Global 2-opt and 3 seconds by Multiple-Start Greedy. More strikingly, the instances of type (60, 40, 20) require about 30 seconds by Farthest Insertion or by FI Block Minimization, 1.5 minutes by Simple Greedy, 30 minutes by Global 2-Opt, and 1 hour by Multiple-Start Greedy (these times are rather stable, for a given method, over all instances of the same type). Even though some of these procedures could certainly be accelerated by implementing them more carefully, it is probably safe to say that the first three heuristics are fast, while the latter two are computationally more demanding. Therefore, for those applications where a solution of high quality
has to be found quickly, FI Block Minimization and Simple Greedy seem to be perfectly adequate procedures (as well as Farthest Insertion, for dense instances). On the other hand, when computing time does not matter too much, and the thrust is instead on the quality of the solution, Multiple Start Greedy and Global 2-Opt could be considered.

Table 6.6 contains the results of our experiments with composite heuristics. The idea is here to quickly compute a good job sequence using one of the constructive heuristics, and to subsequently improve it by relying on some improvement strategy. In view of our previous experiments, we consider five ways to produce an initial solution (namely, by Farthest Insertion, FI Block Minimization, Simple Greedy, Interval and by a random procedure), and we choose Global 2-Opt as improvement strategy.

<table>
<thead>
<tr>
<th>(M,N,C)</th>
<th>Farthest Insertion</th>
<th>FI Block Minimization</th>
<th>Simply Greedy</th>
<th>Interval</th>
<th>Global 2-opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10,10,4)</td>
<td>5.0 (5.5)</td>
<td>8.7 (6.9)</td>
<td>5.4 (3.6)</td>
<td>6.9 (4.5)</td>
<td>8.7 (4.7)</td>
</tr>
<tr>
<td>(10,10,5)</td>
<td>8.3 (5.3)</td>
<td>7.3 (7.1)</td>
<td>5.3 (5.5)</td>
<td>3.6 (4.4)</td>
<td>7.4 (7.1)</td>
</tr>
<tr>
<td>(10,10,6)</td>
<td>4.9 (4.9)</td>
<td>2.9 (4.4)</td>
<td>1.9 (3.8)</td>
<td>2.0 (4.0)</td>
<td>3.0 (4.6)</td>
</tr>
<tr>
<td>(10,10,7)</td>
<td>2.0 (4.0)</td>
<td>1.0 (3.0)</td>
<td>0.0 (0.0)</td>
<td>1.0 (3.0)</td>
<td>0.0 (0.0)</td>
</tr>
<tr>
<td>(20,15,8)</td>
<td>6.3 (5.1)</td>
<td>6.4 (3.8)</td>
<td>6.6 (3.8)</td>
<td>4.7 (2.9)</td>
<td>10.0 (4.3)</td>
</tr>
<tr>
<td>(20,15,8)</td>
<td>12.3 (6.4)</td>
<td>6.2 (4.9)</td>
<td>7.1 (3.4)</td>
<td>8.9 (5.5)</td>
<td>9.7 (4.1)</td>
</tr>
<tr>
<td>(20,15,10)</td>
<td>5.0 (5.5)</td>
<td>3.6 (4.0)</td>
<td>3.9 (3.0)</td>
<td>3.9 (5.2)</td>
<td>6.4 (7.3)</td>
</tr>
<tr>
<td>(20,15,12)</td>
<td>1.5 (3.2)</td>
<td>0.0 (0.0)</td>
<td>0.5 (1.5)</td>
<td>1.0 (3.0)</td>
<td>1.0 (2.0)</td>
</tr>
<tr>
<td>(40,30,15)</td>
<td>2.5 (3.1)</td>
<td>2.8 (2.0)</td>
<td>5.3 (4.3)</td>
<td>5.3 (3.1)</td>
<td>6.0 (4.0)</td>
</tr>
<tr>
<td>(40,30,17)</td>
<td>3.1 (1.3)</td>
<td>3.0 (2.5)</td>
<td>5.0 (2.4)</td>
<td>6.5 (2.6)</td>
<td>4.5 (3.3)</td>
</tr>
<tr>
<td>(40,30,20)</td>
<td>6.6 (4.1)</td>
<td>3.4 (2.1)</td>
<td>5.3 (2.7)</td>
<td>6.6 (2.9)</td>
<td>6.0 (2.9)</td>
</tr>
<tr>
<td>(40,30,25)</td>
<td>7.7 (3.0)</td>
<td>3.9 (2.2)</td>
<td>4.6 (3.4)</td>
<td>9.1 (5.1)</td>
<td>6.1 (3.7)</td>
</tr>
<tr>
<td>(60,40,20)</td>
<td>1.5 (1.6)</td>
<td>2.2 (1.8)</td>
<td>5.2 (1.5)</td>
<td>5.0 (1.5)</td>
<td>4.8 (2.4)</td>
</tr>
<tr>
<td>(60,40,22)</td>
<td>2.0 (2.4)</td>
<td>2.6 (2.1)</td>
<td>2.5 (2.3)</td>
<td>2.7 (2.0)</td>
<td>3.7 (2.6)</td>
</tr>
<tr>
<td>(60,40,25)</td>
<td>3.7 (1.7)</td>
<td>2.7 (2.0)</td>
<td>2.3 (2.5)</td>
<td>4.1 (3.4)</td>
<td>2.1 (1.9)</td>
</tr>
<tr>
<td>(60,40,30)</td>
<td>3.2 (2.7)</td>
<td>1.6 (2.0)</td>
<td>2.4 (2.0)</td>
<td>3.7 (1.5)</td>
<td>4.5 (2.7)</td>
</tr>
</tbody>
</table>

Table 6.6 Average (and standard deviation) of $\delta_H(I)$ for composite heuristics.

We see from Table 6.6 that, for dense instances, Farthest Insertion usually provides a very good initial solution, while FI Block Minimization always performs among the best for sparser instances. But in fact, surprisingly enough, all initialization procedures for Global 2-Opt (including the random one) come extremely close to each other, in terms of the quality of the
solution produced. Also, their running times do not differ significantly.

### 6.5 Lower bounds

In order to judge of the quality of the heuristics described above, it would have been desirable to know tight and easily computed lower bounds on the cost of an optimal job sequence. The knowledge of such lower bounds would also be a prerequisite for the development of an exact optimization procedure (e.g. of the branch-and-bound type) for the tool switching problem. At this moment, unfortunately, we do not have very good lower-bounding procedures for our problem. We now briefly discuss some of the directions which may be worth exploring in this regard. In this discussion, we denote by \( \text{cost}(A, C) \) the total number of setups required by an optimal sequence for the problem instance \((A, C)\).

#### 6.5.1 Traveling salesman paths

Since the quantity \( lb(i, j) \) introduced in Subsection 6.3.1 is a lower bound on the number of tool switches incurred between job \( i \) and job \( j \) in any sequence, the length of a shortest TS path in the graph \( G = (V, E, lb) \) certainly is a lower bound for the total number of switches in the optimal sequence (see Subsection 6.3.1). In other words, denoting by \( L(A, C) \) the length of such an optimal path, we see that \( L(A, C) + C \) is a lower bound on \( \text{cost}(A, C) \) (Tang and Denardo (1988)). Our computational experiments indicate that this bound is generally extremely weak.

The lower bound \( L(A, C) + C \) can sometimes be improved by relying on the following observations. It is obvious that, if \((A', C)\) is a new problem instance obtained by deleting some jobs from \( A \) (i.e., the columns of \( A' \) form a subset of the columns of \( A \)), then the number of setups required for \((A', C)\) is never larger than the number of setups required for \((A, C)\), i.e. \( \text{cost}(A', C) \leq \text{cost}(A, C) \). Thus, in particular, \( L(A', C) + C \) is a lower bound on \( \text{cost}(A, C) \). But it may happen that \( L(A, C) < L(A', C) \), in which case \( L(A', C) + C \) is a sharper bound than \( L(A, C) + C \).

**Example 6.2** Consider the instance \((A, C)\) described in Tang and Denardo (1988). After some reductions, the instance involves 6 jobs, and the matrix \( lb(i, j) \) is given by:
\[ \begin{bmatrix} - & 2 & 3 & 2 & 2 & 1 \\ 2 & - & 3 & 1 & 0 & 1 \\ 2 & 3 & - & 3 & 2 & 2 \\ 2 & 1 & 3 & - & 2 & 2 \\ 2 & 0 & 2 & 2 & - & 2 \\ 1 & 1 & 2 & 2 & 2 & - \end{bmatrix} \]

The path \((3, 6, 1, 4, 2, 5)\) is a shortest TS path with respect to \(lb\). Its length is \(L(A, C) = 6\). On the other hand, deleting the second job from this instance results in an instance \((A', C)\) for which the shortest Hamiltonian path \((3, 6, 1, 4, 5)\) has length \(L(A', C) = 7\). Since the sequence \((3, 6, 1, 2, 4, 5)\) requires precisely 7 switches (see Tang and Denardo (1988)), we conclude that this sequence is optimal for the instance \((A, C)\).

An interesting question is how the partial instance \((A', C)\) should be (heuristically) picked in order to raise as much as possible the lower bound \(L(A', C) + C\). This question has not been investigated yet.

### 6.5.2 Structures implying extra setups

Another approach for obtaining lower bounds on cost\((A, C)\) is to identify subsets of tools for which extra setups are needed in any sequence. This can for instance be done as follows. Let \(K\) be a subset of rows (viz. tools), and \(J\) a subset of columns (viz. jobs) of \(A\). Say that a job \(j \in J\) is heavy (with respect to \(J\) and \(K\)) if, for every partition of \(J\) into \(J_1 \cup \{j\} \cup J_2\) (\(J_1\) and \(J_2\) nonempty),

\[
|\{k \in K : a_{kj} = 1\}| + |\{k \in K : a_{kj} = 0\text{ and }a_{kr} = a_{ks} = 1\text{ for some }r \in J_1, s \in J_2\}| > C. \tag{6.1}
\]

The idea behind this definition is easy to grasp: the left-hand side of (6.1) represents the number of tools required to process job \(j\) \((a_{kj} = 1)\), plus the number of tools which are not required by \(j\) \((a_{kj} = 0)\), but which are used before and after \(j\) in a sequence of the form \((J_1, j, J_2)\) \((a_{kr} = a_{ks} = 1)\). Now, we have:

**Theorem 6.3** If \(J\) contains three heavy jobs with respect to \(J\) and \(K\), then, in any sequence, at least one tool of \(K\) incurs an extra setup.

**Proof:**
Consider any sequence. We can always find a partition \(J\) into \(J_1 \cup \{j\} \cup J_2\)
such that \( j \) is heavy, \( J_1 \) and \( J_2 \) are nonempty, all the jobs in \( J_1 \) occur before \( j \) in the sequence, and all the jobs in \( J_2 \) occur after \( j \). It follows directly from (1) that, among all the tools of \( K \) which are needed both in \( J_1 \) and in \( J_2 \), some will not be present in the magazine when job \( j \) is processed (since this would exceed the magazine capacity). Hence, an extra setup will be necessary for these tools.

The statement of Theorem 6.3 is probably too general to be of direct interest. But it can nevertheless be used to identify some substructures in the tool-job matrix \( A \) which imply extra setups in any sequence. Two illustrations of such structures are now given.

1) Assume there exist three jobs (say, without loss of generality, \( j = 1, 2, 3 \)) and a subset \( K \) of tools such that:

(i) each tool in \( K \) is used by exactly two of the jobs 1, 2, 3;

(ii) each of the jobs 1, 2, 3 needs (strictly) more than \( C - K \) tools among those not in \( K \).

Under these conditions, one verifies that the jobs 1, 2, 3 are heavy with respect to \( K \); hence, the conclusion of Theorem 6.3 applies.

2) Suppose that \( J \) and \( K \) are subsets of jobs and tools respectively, such that:

(i) \( |J| = |K| \geq 3 \);

(ii) the submatrix of \( A \) with column-set \( J \) and row-set \( K \) is the adjacency matrix of a cycle (see Nemhauser and Wolsey (1988));

(iii) at least three jobs in \( J \) require \( C \) tools.

Then, the three tools mentioned under (iii) are heavy, and Theorem 6.3 applies again.

Consider now \( p \) subsets of tools \( K_1, K_2, \ldots, K_p \) for which we know (e.g., using Theorem 6.3) that at least one tool in \( K_i \) incurs an extra setup in any sequence \( (i = 1, 2, \ldots, p) \). Let \( K = \cup_i K_i \). Then, a lower bound on the total number of extra setups is provided by the optimal value of the following set-covering problem:
\[ Z = \minimize \sum_{k \in K} t_k \] (6.2)

subject to \[ \sum_{k \in K} t_k \geq 1 \quad (i = 1, 2, \ldots, p) \] (6.3)
\[ t_k \in \{0, 1\} \quad (k \in K). \] (6.4)

Hence, \( Z + M \) is a lower bound on \( \text{cost}(A, C) \) (where \( M \) is, as always, the total number of tools).

6.5.3 Valid inequalities

Tang and Denardo (1988) propose the following formulation of the tool switching problem (see also Bard (1988)). Let \( x_{ij} = 1 \) if job \( i \) is the \( j \)-th job in a sequence, and \( x_{ij} = 0 \) otherwise; let \( t_{kj} = 1 \) if tool \( k \) is on the machine when the \( j \)-th job is processed, and 0 otherwise; let \( y_{kj} = 1 \) if tool \( k \) is setup just before processing the \( j \)-th job of the sequence, and 0 otherwise \((k = 1, 2, \ldots, M; i, j = 1, 2, \ldots, N)\). Then,

\[
\text{cost}(A, C) = \minimize \sum_{k} \sum_{j} y_{kj} \\
\text{subject to} \sum_{i} x_{ij} = 1 \quad (j = 1, 2, \ldots, N) \] (6.5)
\[ \sum_{j} x_{ij} = 1 \quad (i = 1, 2, \ldots, N) \] (6.6)
\[ \sum_{i} a_{ki} x_{ij} \leq t_{kj} \quad (k = 1, 2, \ldots, M; j = 1, 2, \ldots, N) \] (6.7)
\[ \sum_{k} t_{kj} = C \quad (j = 1, 2, \ldots, N) \] (6.8)
\[ y_{kj} \geq t_{kj} - t_{k,j-1} \quad (k = 1, 2, \ldots, M; j = 2, \ldots, N) \] (6.9)
\[ y_{k1} \geq t_{k1} \quad (k = 1, 2, \ldots, M) \] (6.10)
\[ x_{ij} \in \{0, 1\} \quad (i, j = 1, 2, \ldots, N) \] (6.11)
\[ t_{kj}, y_{kj} \in \{0, 1\} \quad (k = 1, 2, \ldots, M; j = 1, 2, \ldots, N). \] (6.12)

Call SW this 0-1 programming problem. The linear programming relaxation of SW provides a lower bound on \( \text{cost}(A, C) \). But this bound is extremely weak (Tang and Denardo (1988)). One way of improving it would be to add
more valid constraints to the formulation of SW. For instance, the following inequalities are valid for the tool switching problem:

$$\sum_j y_{kj} \geq 1 \quad (k = 1, 2, \ldots, M)$$

(notice that these trivial inequalities are not even satisfied, in general, by an optimal solution to the continuous relaxation of SW). Another family of valid inequalities can be derived as follows. Let $K$ be a subset of tools for which we know that at least one extra setup is required in the optimal sequence (see Theorem 6.3). Then,

$$\sum_{k \in K} \sum_j y_{kj} \geq |K| + 1$$

is valid. More generally, if $Z$ is the optimal value of (6.2)–(6.4), then a valid constraint is given by:

$$\sum_{k \in K} \sum_j y_{kj} \geq |K| + Z.$$

More work is needed on the strengthening of the formulation SW.

Still another possible approach would be to replace SW by a formulation of the tool switching problem using different variables. For instance, one may want to consider the “disaggregated” variables $t_{kij}$, with the interpretation that $t_{kij} = 1$ if tool $k$ is set up just after finishing the $i$-th job and is removed just after finishing the $j$-th job. It is easy to write a formulation of our problem involving only the variables $x_{ij}$, $t_{kij}$ and $y_{kj}$. It is also relatively easy to derive exponentially many valid inequalities using these variables, which can in principle be added to the initial formulation in order to strengthen it. But our preliminary computational experiments along these lines were quite disappointing, in that they did not allow us to noticeably improve our previous lower bounds on the optimal value of the problem.

### 6.5.4 Lagrangian relaxation

Lagrangian relaxation is a classical tool in deriving bounds on the optimal value of an integer programming problem (see Nemhauser and Wolsey (1988)). For problem SW, one may for instance try to relax the groups of constraints (6.7) and (6.8). Indeed, as observed by Bard (1988), the resulting subproblems are then easy to solve (Bard (1988) uses this relaxation scheme in order to produce a sequence of heuristic solutions for the tool switching
problem). But it is easy to prove that the optimal value of the Lagrangian dual problem obtained in this way is equal to the optimal value of the linear relaxation of SW (this is because all extreme points of the system defined by (6.5), (6.6), (6.9), (6.10) and the relaxation of (6.11), (6.12) are integral; see Nemhauser and Wolsey (1988)).

The possibility of deriving good lower bounds on \( \text{cost}(A,C) \) using Lagrangian relaxation is an avenue that should be further explored.
Appendix: Graph-theoretic definitions

In this chapter, a graph $G$ is a triple of the form $(V, E, d)$, where:

- $V$ is a finite set; the elements of $V$ are the nodes of $G$;
- $E$ is a set of pairs of nodes, called edges;
- $d$ is a function which assigns a nonnegative length to each pair of nodes;
  we assume that $d(u, v) = +\infty$ when $\{u, v\}$ is not an edge.

A path in a graph is a sequence of nodes, i.e. a permutation of a subset of $V$. A traveling salesman path (or TS path) is a permutation of $V$. The length of a path $(u_1, \ldots, u_k)$ is by definition:

$$d(u_1, u_2) + d(u_2, u_3) + \ldots + d(u_{k-1}, u_k).$$

Notice, in particular, that the length of such a path is infinite if some pair $\{u_i, u_{i+1}\}$ is not an edge of the graph. The traveling salesman problem on a graph $G$ can be stated as follows: find a TS path of minimal length in $G$.

With a graph $G = (V, E, d)$, we can associate another graph $H = (E, I, \delta)$, called the edge-graph of $G$, and defined as follows:

- each node of $H$ is an edge of $G$;
- a pair $\{e, f\}$, with $e, f \in E$, is an edge of $H$ if and only if the edges $e$ and $f$ share a common node in $G$;
- $\delta(e, f) = 1$ if $\{e, f\}$ is an edge of $H$, and $\delta(e, f) = +\infty$ otherwise.

Observe that, in an edge-graph, every TS path has length either $|E| - 1$ or $+\infty$. Consider now the restriction of the traveling salesman problem to edge-graphs, that is:

Input: a graph $G$.
Problem P3: find a TS path of minimal length in the edge-graph of $G$.

Equivalently, P3 asks whether there exists a TS path of finite length in the edge-graph of $G$. Bertossi (1981) proved that this problem is NP-hard.

We also deal in this chapter with directed graphs. A directed graph is a triple $(V, U, d)$, where $V$ is defined as for a graph, and:

- $U$ is a set of ordered pairs of nodes, called arcs; i.e., $U \subseteq V \times V$;
- \( d \) is a (nonnegative) length function defined on \( V \times V \), with the property that \( d(u, v) = +\infty \) when \((u, v)\) is not an arc.

So, in a directed graph, \( d(u, v) \) may differ from \( d(v, u) \). The definitions of a TS path and of the TS problem extend in a straightforward way for directed graphs.
References


Ahmadi, J., R. Ahmadi, H. Matsuo and D. Trupati (1990), *Component fixture positioning for printed circuit board assembly with concurrent operations*, Working Paper 90-03-01, ICC Institute, University of Texas, Austin, Texas.


References


Möhring, R.H. (1990), Graph problems related to gate matrix layout and PLA folding, in: Computational Graph Theory, G. Tinhofer et al. (eds), Springer-Verlag, Wien, 17–51.


Roger, C. (1990), La gestion des outils sur machines à commande numérique, Mémoire DEA de Recherche Opérationnelle, Université Joseph Fourier, Grenoble, France.


Vliet, M. van, and L.N. van Wassenhove (1989), Operational research techniques for analyzing flexible manufacturing systems, Research Memorandum series, No. Ti-1989/16, Tinbergen Institute, Erasmus University Rotterdam, Rotterdam.

Samenvatting

Dit proefschrift gaat over wiskundige problemen die zich kunnen voordoen bij geautomatiseerde productie-processen. In Hoofdstuk 1 wordt, naast een overzicht over het proefschrift, een algemene beschrijving gegeven van een geautomatiseerd productie-proces in de electronica industrie. Het gaat hierbij om de productie van zogenaamde printplaten. Zo'n productie-proces ziet er in het algemeen als volgt uit. Een lijn van machines is gewijd aan de productie van printplaten, die elk met een automatische transportband van de een machine naar de volgende machine worden vervoerd. De assemblage van een printplaat bestaat uit het plaatsen van electronische componenten (zoals weerstanden en condensatoren) van bepaalde types op vooraf bepaalde locaties in de printplaat. Om deze componenten te kunnen hanteren, is elke machine uitgerust met een zogenaamde arm. Deze arm pakt componenten op van zogenaamde voeders, beweegt naar de geschikte locaties, plaatst de componenten daar in de printplaat en beweegt terug naar de voeders om nieuwe componenten te pakken. Deze voeders leveren componenten van een bepaald type (één type per voeder) en ze kunnen aan de machine bevestigd worden middels zogenaamde voederklemmen. Elke machine heeft een rij van dergelijke klemmen beschikbaar. Afhankelijk van het type, neemt een voeder 1 of 2 (en soms zelfs meer dan 2) aaneengesloten klemmen in beslag (zie Figuur 2.1). Het pakken van componenten door de arm, het bewegen naar de juiste locaties en het plaatsen van de componenten in de printplaat zullen we een slag van de machine noemen.

We nemen aan dat het een doelstelling van de producent is om zoveel mogelijk printplaten per tijdseenheid te produceren. Aangezien deze productiensnelheid afhankelijk is van de kwaliteit van de machine-instructies, is het van belang om goede optimaliseringsprogramma's te hebben. Echter, de assemblage van een 'gemiddelde' printplaat vergt de plaatsing van honderden componenten door ingewikkelde machines. Vandaar dat het maximaliseren van de productiesnelheid voor verschillende type's printplaten met een
lijn bestaande uit meerdere machines, een bijzonder moeilijk planningsprobleem is. Een gebruikelijke aanpak bij dit soort planningsproblemen is om het probleem op hierarchische wijze te decomponeren. Er worden dan een aantal (relatief makkelijkere) deelproblemen onderscheiden, op een zodanige wijze dat een oplossing van het ene deelprobleem als gegeven dient voor het volgende deelprobleem. In dit proefschrift worden een aantal algorithmen ontwikkeld en geanalyseerd voor bepaalde deelproblemen die in zo'n hierarchische probleemdecompositie naar voren komen.

Hoofdstuk 2 gaat in op een specifiek assemblage-systeem voor printplaten. Eén van de additionele kenmerken van dit systeem is dat de arm van de machine is uitgerust met drie koppen. Dat betekent dat in één slag van de machine drie componenten in de printplaat geplaatst kunnen worden. Bovendien moet zo'n kop, om een component te kunnen pakken, voorzien worden van een bepaald gereedschap. Een component type kan maar door bepaalde soorten gereedschap gepakt worden. Verder wordt er in Hoofdstuk 2 uitgegaan van de productie van één type printplaat. Het resulterende planningsprobleem wordt dan in de volgende, hierarchisch gekoppelde deelproblemen, opgedeeld:

(A) bepaal hoeveel componenten elke machine moet plaatsen en met welk gereedschap;
(B) wijs voedertypes toe aan de machines;
(C) bepaal welke componenten door welke kop van de machine in de printplaat geplaatst worden;
(D) groepen de locaties in groepjes van drie, die in één slag van de machine bediend worden;
(E) bepaal in welke slag welke groepjes van locaties bediend worden;
(F) wijs voeders toe aan de klemmen.

Deelproblemen (A) en (B) bepalen de werklast van een machine. Deelproblemen (C)-(F) moeten voor elke machine één keer opgelost worden.

In Hoofdstuk 2 is beschreven hoe elk van deze deelproblemen als een wiskundig model geformuleerd is, en welke algoritmen gebruikt zijn om deze modellen op te lossen. Vervolgens is deze aanpak getest op een praktijkinstantie, waarbij 258 componenten van 39 types in een printplaat gezet moeten worden door een lijn van drie machines. Het blijkt dat onze aanpak de maximale werklast aanzienlijk terugbrengt.
Hoofdstuk 3 van dit proefschrift houdt zich bezig met deelprobleem (D) uit de eerder gegeven decompositie. Het blijkt dat dit probleem een speciaal geval is van het drie-dimensionale toewijzingsprobleem (3DA), dat als volgt beschreven kan worden. Gegeven zijn drie disjuncte verzamelingen van \( n \) punten, en tevens zijn er niet-negatieve kosten bekend voor elk drietal punten bestaande uit één punt van elke verzameling. Het probleem is om \( n \) drietallen te vinden zodanig dat elk punt in één van die drietallen zit en dat de som van de kosten van die drietallen minimaal is. In deelprobleem (D) corresponderen de drie disjuncte verzamelingen punten met locaties waar componenten door kop 1, kop 2 of kop 3 respectievelijk geplaatst moeten worden. Instanties van deelprobleem (D) hebben wat meer structuur dan die van het algemene 3DA, omdat de kosten van een drietal in subprobleem (D) afhangen van afstanden tussen de punten in het drietal. Bovendien voldoet die afstandsfunctie aan de driehoeksongelijkheid. We bewijzen dat deelprobleem (D) tot de klasse der zogenaamde \( \mathcal{NP} \)-moeilijke problemen behoort. Verder presenteren we twee algoritmen gebaseerd op het oplossen van een aantal gerelateerde twee-dimensionale toewijzingsproblemen. Deze algoritmen blijken altijd een oplossing te vinden met kosten maximaal \( 1.5 \), respectievelijk \( 4/3 \), keer de kosten van de optimale oplossing. Experimenten laten zien dat de prestaties van deze algoritmen op willekeurig gegeven probleeminstanties excellent is.

In Hoofdstuk 4 worden generalisaties van deelprobleem (D) behandeld. Gegeven zijn nu \( k \) disjuncte verzamelingen punten, \( k \geq 3 \), en een afstand tussen elk tweetal punten in verschillende verzamelingen. Laat een klikt een deelverzameling van \( k \) punten zijn met één punt uit elke verzameling. Neem verder aan dat de kosten van een klikt functies zijn van de afstanden tussen de punten in de klikt. Het probleem is nu om een partitie van de verzameling punten in klikt te vinden met minimale kosten. Verscheidene kosten-functies worden geïntroduceerd, en er worden verschillende algoritmen voorgesteld. Voor elk van deze algoritmen worden ‘worst-case’ grenzen afgeleid. Bijvoorbeeld, er wordt bewezen dat voor bepaalde kosten-functies er een algoritme bestaat dat, in polynomiale tijd, altijd een oplossing levert met kosten maximaal twee keer de kosten van de optimale oplossing, voor elke waarde van \( k \).

Hoofdstuk 5 is gewijd aan het volgende probleem. Gegeven zijn \( n \) taken die door 1 machine verricht moeten worden in de tijdspanne \( \{1, 2, \ldots, T\} \). De processies van elke job bedraagt \( p, p \in \mathbb{N} \). De productie-kosten van elke taak zijn een functie van de starttijd van die taak. Het probleem is om zulke starttijden voor de taken te vinden, dat de som van de productie-kosten
minimaal is. We bewijzen dat, voor $p \geq 2$, dit probleem tot de klasse der \textit{\text{NP}}-moeilijke problemen behoort. Aan de andere kant, voor $T = n \cdot p + c$, waar $c$ een constante voorstelt, is het probleem oplosbaar in polynomiale tijd. Verder wordt dit probleem onderzocht vanuit een polyhedrale invalshoek. Er worden onder meer twee klassen van facet-definierende ongelijkheden afgeleid. Dit probleem blijkt overigens sterk verwant te zijn aan deelprobleem (F) in de eerder gegeven decompositie.

Tenslotte wordt in Hoofdstuk 6 het volgende probleem behandeld. Een aantal taken moet door een flexibele machine bewerkt worden. Voor de bewerking van elke taak moet er een bepaalde verzameling gereedschappen in het magazijn van de machine aanwezig zijn. Aangezien de capaciteit van dit magazijn beperkt is, en bovendien het aantal gereedschappen nodig ter bewerking voor alle taken groter is dan de capaciteit van het magazijn, moet er soms tussen 2 opeenvolgende taken gereedschappen gewisseld worden. Wij beschouwen in dit hoofdstuk het probleem om een zodanige volgorde van taken te vinden dat het aantal gereedschapswisselingen wordt geminimaliseerd. Er wordt in Hoofdstuk 6 een toepassing van dit probleem in de assemblage van printplaten beschreven. Verder blijkt dat het minimaliseren van het aantal gereedschapswisselingen al een \textit{\text{NP}}-moeilijk probleem is voor een vaste magazijn capacity $C \geq 2$. Tenslotte worden een aantal algoritmen voorgesteld en getest op willekeurig gegenereerde probleeminstanties.
Curriculum vitae