DEPARTMENT OF OPERATIONS RESEARCH
UNIVERSITY OF AARHUS

Working Paper no. 2006 / 4 2006 / 09 / 19

Hybrid IP/CP Methods for Solving Sports Scheduling Problems

Rasmus Vinther Rasmussen
PhD Thesis

ISSN 1600-8987

Department of Mathematical Sciences
Telephone: +45 8942 1111
E-mail: institut@imf.au.dk

Building 1530, Ny Munkegade
DK-8000 Aarhus C, Denmark
URL: www.imf.au.dk

Hybrid IP/CP Methods for Solving Sports Scheduling Problems

Rasmus Vinther Rasmussen

PhD dissertation

July 2006

Hybrid IP/CP Methods for Solving Sports Scheduling Problems

Rasmus Vinther Rasmussen,
Department of Operations Research,
University of Aarhus,
Denmark.

PhD dissertation, July 2006.

Dissertation committee:

- Jørgen Aase Nielsen, University of Aarhus.
- Dominique de Werra, Ecole Polytechnique Fédérale de Lausanne.
- Jørgen Bang-Jensen, University of Southern Denmark.

Advisor:

- Kim Allan Andersen, Aarhus School of Business.

Rasmus Vinther Rasmussen
Department of Operations Research
University of Aarhus
Ny Munkegade
Building 1530
DK-8000 Aarhus C
Denmark
vinther@imf.au.dk

Subject classification:

- *MSC2000*: Primary: 90B35; secondary: 90C10, 90C27, 90C90.
- *OR/MS*: Scheduling, Combinatorics, Integer programming.

Preface

This dissertation presents the outcome of the research I have done during my PhD program at the Department of Operations Research at the University of Aarhus. The objective of my work has been to develop solution methods integrating integer programming (IP) and constraint programming (CP) and to find practical applications for which such methods can be successfully applied.

At the beginning of my PhD programme a Danish net-operator Sonofon presented a job scheduling problem. Since hybrid IP/CP methods have previously performed very well on similar applications, I started working on the problem together with Christian Roed Pedersen and Kim Allan Andersen. Unfortunately, we discovered that the size of the problem prevented the use of an exact solution method and instead a tabu search was developed. A paper [67] describing the solution method has subsequently been accepted for publication in *Computers & Operations Research*.

During the work for Sonofon I discovered the field of sports scheduling, which is a research area containing numerous small, highly constrained and very hard optimization problems. These problems turn out to be well suited for hybrid IP/CP methods since they often contain a very hard feasibility aspect and at the same time have an objective function. For this reason solution methods for sports scheduling problems became the main focus in the rest of my work.

In Denmark, soccer is the number one sport and developing a solution method for scheduling the best Danish soccer league quickly became a natural goal. However, before this goal could be achieved a number of preliminary steps had to be taken.

During a three months visit to Carnegie Mellon University at the beginning of 2005 I worked together with Michael A. Trick. At first we developed a hybrid IP/CP method for mirrored double round robin tournaments taking advantage of a known decomposition approach. Compared to previous methods our approach decreases computation time by following an iteration scheme instead of solving one step at a time. The solution method gave a lot of insight into how the strengths of IP and CP can be utilized, however, the real breakthrough came when we combined the solution method with a pattern generating approach. This second

approach is now known as the *pattern generating Benders approach* and a paper [72] describing this approach has been accepted for publication in the *European Journal of Operational Research*.

In the fall of 2005 the approach was further developed in order to handle the numerous constraints present for the best Danish soccer league. This research has led to a solution method capable of finding high quality solutions for the tournament and the Danish Football Association has used it for scheduling the 2006/2007 season. A paper [70] presenting the application and the solution method has been submitted to an international journal of operations research.

In addition to the work relating to the Danish soccer league, I have also considered sports scheduling problems in which the travel distance must be minimized. This problem applies for many leagues in the USA, since the teams travel from one away game to the next without returning home. In a paper [73] presented at the CP-AI-OR 2006 conference and published in *Lecture Notes of Computer Science*, Michael A. Trick and I define a new problem called the *timetable constrained distance minimization problem* (TCDMP) and we evaluate a number of solution methods for this problem. Furthermore, an extended version of the paper has been submitted to *Annals or Operations Research* in which we present a new solution method denoted the *circular traveling salesman approach* for a problem called the *traveling tournament problem*.

Finally, I have used the knowledge on sports scheduling obtained during my PhD programme to write a survey paper on round robin tournament scheduling. Again this is a joint work with Michael A. Trick and the paper [74] will be submitted to an international journal of operations research.

Acknowledgements

During my PhD programme several people have contributed with valuable assistance and suggestions. First of all I would like to thank my advisor Kim Allan Andersen for countless meetings at which his insightful comments and suggestions enabled continuing progress.

At the beginning of 2005 I had the pleasure of visiting Michael A. Trick at Carnegie Mellon University for three months. This visit led to some of the major contributions presented in this dissertation and I am deeply indebted for his support and guidance - not only during my stay in Pittsburgh - but during the rest of my PhD programme.

I would also like to thank my colleagues Trine Krogh Kristoffersen and Christian Roed Pedersen. They have provided an ideal working atmosphere and their ability to come up with qualified suggestions and ideas when things have seemed stuck has been a huge help. In the writing process Randi Mosegaard has provided invaluable linguistic support for which I am very thankful.

During my work I have found it important to relate the theoretical contributions to practical applications. Therefore I would like to thank Benny Olsen, Peter Ebbesen, Allan G. Hansen (the Danish Football Association), Morten Bech Kristensen, Lars Jørgensen (Sonofon) and Lars Grynderup (DM-Data) for making this possible. Without their data supply and patience when explaining the problems, it would have been impossible to test the solution methods on practical applications.

I thank the Department of Mathematical Sciences for sponsoring a number of conferences. These conferences have made it possible to establish contact with researches from outside University of Aarhus.

Finally, a very special thanks goes to Line Fiil Tarp for her love, support and friendship. Without her and her incredible faith in me this work would never have been possible.

Århus, June, 2006

Rasmus Vinther Rasmussen

Contents

Chapter 1

Introduction

The field of sports scheduling comprises a challenging research area with a great variety of problems and applications. Most of the problems are very hard combinatorial optimization problems providing a perfect platform for developing and testing all kinds of solution methods. During the last 30 years, these problems have encouraged a great amount of research on building effective solution methods leading to a huge repertoire of approaches including column generation, Benders decomposition, constraint programming methods, hybrid methods combining integer programming and constraint programming, and meta heuristic methods such as tabu search, genetic algorithms and simulated annealing. The individual methods have become more and more effective but still problems with just eight teams remain unsolvable.

Furthermore, the area provides lots of practical applications since individual sports leagues face different constraints and have different objectives. The applications are normally characterized by a huge amount of conflicting requirements coming from teams, fans, etc. To solve these problems, specialized solution methods capable of integrating the application specific requirements into state-of-the-art algorithms are required. In the literature various approaches are presented, but as the algorithms improve, the number of constraints grow. The ability to obtain high quality solutions for sports leagues does not only provide schedules satisfying team requests but can result in huge earnings to the sports leagues. The schedule of a given sports league may constitute a significant factor when the price of TV rights are negotiated with TV networks.

In Chapter 2 we give a short introduction to the standard solution methods for combinatorial optimization problems. The basic concepts of IP, CP, and meta-heuristic algorithms are explained and we discuss hybrid algorithms combining IP and CP. All the solution methods have been used within the sports scheduling society and the solution methods presented later in this dissertation rely on the

standard approaches.

Chapter 3 is based on Rasmussen and Trick [74] and gives a comprehensive survey of the sports scheduling literature. It introduces the sports scheduling terminology and presents the main results obtained in the literature. The papers on sports scheduling are partitioned into two categories: papers on break minimization and papers on distance minimization. Within each category an almost chronological review of the current papers are presented.

After the survey, the following three chapters focus on the *constrained minimum break problem* which is a classical sports scheduling problem. We must find a schedule for a round robin tournament facing some application specific constraints and at the same time minimize the total number of breaks. Various solution methods have been proposed to solve the problem and a general decomposition scheme has been adopted in the literature.

Chapter 4 presents the first approach using Benders decomposition to solve the constrained minimum break problem. The algorithm is capable of outperforming traditional IP and CP models and it demonstrates how Benders decomposition can be applied to sports scheduling problems.

The technique from Chapter 4 is further developed in Chapter 5 which is based on Rasmussen and Trick [72]. The solution method presented here is known as the *pattern generating Benders approach* and it employs a decomposition scheme widely used in the sports scheduling literature. The enhanced algorithm leads to significant reductions in computation times compared to previously known methods and it is capable of solving previously unsolved problem instances.

The pattern generating Benders approach is applied to the Danish soccer league in Chapter 6 which is based on Rasmussen [70]. This is a triple round robin tournament facing a large number of conflicting constraints. The pattern generating Benders approach presented in Chapter 5 is developed for a double round robin tournament with place constraints but in this chapter we show how to modify the algorithm to deal with the additional constraint types present in the application and to solve a triple round robin tournament instead of a double. The modified algorithm is capable of producing high quality schedules and it has been used to find the schedule for the 2006/2007 season.

In Chapter 7 based on Rasmussen and Trick [73] we shift attention towards tournaments concerning distance minimization. Such tournaments are normal in American sports leagues since teams often travel from one away game to the next without returning home as they often do in European leagues. We define a new problem called the *timetable constrained distance minimization problem* (TCDMP) and compare four solution methods for solving the problem. Furthermore, we present a new heuristic solution method called the *circular traveling salesman approach* (CTSA) for a well known sports scheduling problem named the *traveling tournament problem* (TTP). The CTSA are able to find good solutions quickly and the TCDMP can be used to obtain further improvements.

Finally, in Chapter 8 we consider a practical job scheduling problem faced by Sonofon, a Danish net operator. This is a large-scale precedence constrained scheduling problem with time windows in which the makespan must be minimized. Furthermore, the jobs are elastic such that the capacity and time consumption are elastic. We present a tabu search algorithm for solving the problem and the computational results show a significant reduction in makespan compared to the strategy implemented by Sonofon. This chapter is based on Pedersen et al. [67].

Chapter 2

Solution Methods

Before turning towards the sports scheduling literature, let us give a brief introduction to the solution methods used in the succeeding chapters. Sports scheduling problems and scheduling problems in general are combinatorial optimization or feasibility problems and the common solution methods include *integer programming* (IP), *constraint programming* (CP), hybrid IP/CP algorithms and *metaheuristic algorithms*. These four general frameworks solve combinatorial problems in different ways but they all provide powerful tools for finding optimal or near optimal solutions. We discuss the basic concepts of each method and outline the algorithms used in this dissertation.

2.1 Integer Programming

Within the operations research community, discrete problems have been modelled as IP problems or mixed integer linear programming (MILP) problems. Both types of problems are formulated as linear programming (LP) problems containing a linear objective function which must be minimized or maximized and a number of linear inequality or equality constraints which must be satisfied. The difference between LP problems and IP/MILP problems are the integrality requirements enforced on the variables in an IP problem and on a subset of the variables in a MILP problem. Although they reduce the feasible region, these constraints make the problems substantially harder to solve and specialized solution techniques are necessary.

In the following sections we will give a short introduction to some of the solution methods which have proven effective at solving IP problems and which have been relevant to this research.

2.1.1 Branch and Bound

Branch and bound algorithms constitute the backbone of integer programming and to date the most effective solution methods are built on a branch and bound framework. It is a tree search algorithm which uses variable branching to obtain integrality and bounds to avoid total enumeration.

Initially, the *branching tree* consists of a single node called the root node but additional nodes are added as the search proceeds. Each of the nodes has an associated LP problem and the problem associated with the root node is the LP relaxation of the IP or MILP problem considered.

During the search, the algorithm keeps track of the nodes which have not been considered previously and nodes are chosen from this set according to some *node selection strategy*, see Nemhauser and Wolsey [65]. When a node has been chosen, the associated LP problem is solved and one of the following four situations will occur: i) the problem is infeasible, ii) the problem is feasible with an integer solution, iii) the problem is feasible with a fractional solution and a solution value worse than the current best integer solution value or iv) the problem is feasible with a fractional solution and a solution value better than the current best integer solution value. If situation i) or iii) occurs, the node can be pruned without further notice. If situation ii) occurs, the node can also be pruned, however, in case the solution value is better than the current best integer solution value, the new solution is stored as the current best. Finally, if situation iv) occurs, the algorithm chooses an integer variable with a fractional value according to a *branching strategy*, see [65], and it branches on the value of this variable.

Branching takes place by creating two child nodes of the current node. Both of these nodes are associated with an LP problem similar to the problem from the parent node. The first node contains an additional constraint saying that the value of the branching variable must be no greater than the current value rounded down, while the second node contains a similar constraint saying that the value must be no less than the current value rounded up.

The algorithm continues from node to node until all nodes have been visited. When this happens, we have either found the optimal solution or proven that the problem is infeasible.

2.1.2 Branch and Cut

Branch and cut is a generalization of branch and bound which was developed in the 1980's. It combines the divide and conquer philosophy used in branch and bound with the idea of improving the LP relaxation by adding cuts which cuts off infeasible solutions.

Originally, such cuts were used in *cutting plane algorithms* proposed by Gomory [38] in the beginning of the 1960's. However, although finite convergence of the

cutting plane algorithms has been proven, the algorithms often perform poorly. This led to a lack of confidence in the gains from using cutting planes until the method was combined with branch and bound. The merger of cutting plane algorithms and branch and bound algorithms took place in the 1980's and the results have moved focus back to cutting planes.

A branch and cut algorithm uses the branch and bound framework but it does not automatically branch when a fractional solution is obtained. Instead, valid cuts violated by the current solution are added to the LP relaxation and the problem is solved again. New cuts are generated for a number of iterations and then the algorithm branches and continues to the next node.

The cuts help the LP relaxation by letting the feasible region approximate the convex hull of the feasible integer solutions. The main problem of cutting plane algorithms is slow convergence towards this convex hull but branch and cut algorithms are able to branch when the convergence slows down. The combination has proven to be very effective since the improved LP relaxations result in smaller search trees.

In addition, polyhedral theory has provided strong cutting planes for a variety of problems such as the traveling salesman problem and knapsack problems. The use of these cutting planes in branch and cut has lead to huge improvements for the particular problems. Furthermore, branch and cut algorithms are also very effective at solving IP problems in general. For the general problems, families of general inequalities based on knapsack problems [24], Gomory cutting planes [3, 38] and lift and project cutting planes [2] have proven efficient.

We refer to Nemhauser and Wolsey [65] for a thorough description of cutting plane methods and Mitchell [60] for a survey on branch and cut algorithms.

2.1.3 Branch and Price

Branch and Price algorithms are another generalization of branch and bound. In this kind of algorithm additional columns are added to the LP relaxation, in contrast, to the rows added in branch and cut.

At the root node of the branching tree we solve a *restricted master problem* where most of the columns are omitted. This means that the solution may not be optimal but a *pricing problem* is used to find improving columns if any exist. The pricing problem uses the dual variables of the master problem to calculate the reduced costs of the omitted columns. If improving columns exist, they are added to the restricted master problem and the restricted master problem is re-optimized. The algorithm keeps iterating between the restricted master problem and the pricing problem until no profitable columns are found in the pricing problem. When this happens the algorithm proceeds to a new node as described in the branch and bound algorithm. The iterations between the restricted master problem and the pricing problem are repeated in each node of the search tree.

Branch and price algorithms are applicable to problems with a huge number of constraints or problems where a reformulation with a huge number of constraints may present some advantages. These advantages could for instance be a tighter LP relaxation or a formulation where the columns implicitly satisfy some of the constraints. The underlying idea is that, since most of the variables will be zero in an optimal solution and the entire problem cannot be solved efficiently, the extra time used to find improving columns will be offset by the time reduction obtained by solving the reduced master problem instead of the original LP relaxation.

The branch and price technique has successfully been applied to a number of practical applications including sports scheduling problems which we will get back to. However, the technique is perhaps best known from the results obtained for the *airline crew scheduling problem*. The problem consists of assigning crews to flights such that costs are minimized and a large number of regulations are satisfied. Due to the regulations and a complex cost structure, it is undesirable to formulate the problem using 0-1 variables assigning a single crew to a single flight. Instead, all feasible flight *sequences* are enumerated and the problem is formulated by using variables which assign a crew to a flight sequence. The obvious problem in this formulation is the number of feasible flight sequences which is extremely large for practical applications but branch and price algorithms have been used to overcome this problem. We refer to Savelsbergh [80] and Barnhart, Johnson, Nemhauser, Savelsbergh, and Vance [8] for further description of branch and price.

2.1.4 Benders Decomposition

Benders [12] proposed a decomposition method for solving mixed integer linear programming (MILP) problems. It partitions the problem into a master problem containing all the integer variables and a subproblem containing the rest of the variables. When the master problem is solved it corresponds to finding trial values for the integer variables and the subproblem is afterwards used to obtain the best solution consistent with the trial values. From this solution a *Benders cut* is derived which gives a valid bound on the solution value. The cut is added to the master problem and the master problem is re-solved.

The solution method is best explained by looking at an example and therefore we introduce the following MILP problem

$$\begin{aligned} \min\ & cx + dy \\ \text{s.t.}\ & A^1 x \geq b^1 \\ & A^2 x + By \geq b^2 \\ & x \in \mathbb{Z}_+^{n_1} \\ & y \in \mathbb{R}_+^{n_2} \end{aligned}$$

where $c \in \mathbb{R}^{n_1}$, $d \in \mathbb{R}^{n_2}$, $A^1 \in \mathbb{R}^{m_1} \times \mathbb{R}^{n_1}$, $A^2 \in \mathbb{R}^{m_2} \times \mathbb{R}^{n_1}$, $B \in \mathbb{R}^{m_2} \times \mathbb{R}^{n_2}$, $b^1 \in \mathbb{R}^{m_1}$ and $b^2 \in \mathbb{R}^{m_2}$. This problem is decomposed into the following master problem and subproblem.

Master problem:	Subproblem:
min z	min $dy + c\bar{x}$
s.t. $z \geq cx$	s.t. $By \geq b^2 - A^2\bar{x}$
$A^1x \geq b^1$	$y \in \mathbb{R}^{n_2}_+$
$x \in \mathbb{Z}^{n_1}_+$	

where $\bar{x}$ in the subproblem refers to the values assigned to x in the master problem. The use of z in the master problem will become clear when we derive the Benders cut.

The Benders cut is obtained by using the dual problem of the subproblem

$$\begin{aligned} \max\ & c\bar{x} + (b^2 - A^2\bar{x})u \\ \text{s.t.}\ & uB \leq d \\ & u \in \mathbb{R}^{m_2}_+ \end{aligned}$$

Since $\bar{x}$ does not occur in the constraints of the dual problem, any $\bar{x}$ will be feasible and the value of the subproblem will be greater than $c\bar{x} + (b^2 - A^2\bar{x})u$ for any $\bar{x}$. When we let $\bar{u}$ denote the optimal solution to the dual problem, we obtain the cut

$$z \geq cx + (b^2 - A^2x)\bar{u}$$

which can be added to the master problem. The algorithm continues until the solution value of the master problem is no less than the solution value of the subproblem. In this case an optimal solution has been found.

The essential part of Benders decomposition is the Benders cuts which make it capable of learning from mistakes. When a Benders cut is added, it does not only give information about the current solution to the master problem, but it gives information about all the feasible solutions. In this way the algorithm is able to avoid enumerating through all the solutions like the branch and bound algorithm is able to avoid searching through the entire search tree because of the bounds which are obtained during the search. Benders decomposition is discussed by Hooker and Ottosson [50], Nemhauser and Wolsey [65] and Schrijver [84].

2.2 Constraint Programming

CP is an alternative technique for solving combinatorial optimization problems, which has evolved from research done in the artificial intelligence community back

in the 1960's and 1970's. The technique has been developed in parallel to IP but although the two techniques are both used to solve combinatorial problems, almost no interaction has taken place before the 1990's.

In CP, a problem is modelled by using a finite set of variables, a finite domain for each variable and a finite set of constraints. The task is to find an *instantiation* (assignment) of each variable to a value from the corresponding domain such that all constraints are satisfied. This sounds very similar to the description of an IP problem, however, in IP all constraints must be formulated using linear functions, since the LP relaxation should be available. CP, on the other hand, offers a much richer modelling language which allows a huge variety of constraint types. In this way it is easier to formulate a problem using an intuitive model and the solution technique becomes easier to use for people without a deeper understanding of the underlying mechanisms.

In the following sections, we give a brief introduction to some of the solution methods used to solve CP problems but first we will discuss the constraints used in later chapters. For a detailed description of CP we refer to Hooker [48] and Marriott and Stuckey [57]. Wallace [92] gives a survey of applications and Barták [9] gives an introduction to CP in general.

2.2.1 Constraints

In addition to linear equality and inequality constraints known from IP, CP also allows *logic constraints* and *global constraints*.

Logic constraints consist of a logic statement such as

$$(x = 0) \Rightarrow (y = 0) \qquad \text{or} \qquad \sum_i (x_i = 0) = 10.$$

and they are satisfied if the statements are true. For the first example, this means that the constraint is satisfied if both x and y equal 0 but it is also satisfied if $x \neq 0$ no matter which value y takes. In the second example, the constraint is satisfied if exactly 10 of the x variables equal 0, since the expression $(x_i = 0)$ corresponds to 1 if it is true and 0 otherwise.

Global constraints are constraints representing a set of constraints with a special structure. Often they are used to gather a group of similar constraints to help the solution methods. We outline the syntax of the following global constraints, since they are applied in subsequent chapters

- $\mathit{alldifferent}(x_1, \ldots, x_n)$.
  As the name suggests the constraint is satisfied when all the variables $x_1, \ldots, x_n$ are instantiated to different values.

- $\mathit{sequence}(nbMin, nbMax, width, vars, values, card)$
  In this constraint $nbMin$, $nbMax$ and $width$ are integers, $vars$ is an array of

variables and *values* and *card* are arrays of integers having the same index set I. The constraint is satisfied when the following holds true. For each index $i \in I$ the value $value_i$ occurs exactly $card_i$ times in the vector $vars$ and in any subsequence of length $width$ of $vars$ the value $value_i$ occurs at least $nbMin$ and at most $nbMax$ times.

- $atmost(card, values, vars)$
  The syntax of this constraint is like the syntax of *sequence* and it is satisfied if the value $value_i$ occurs at most $card_i$ times in the array $vars$ for all $i \in I$.

- $one\text{-}factor(x_1, \ldots, x_n)$
  The *one-factor* constraint is satisfied when the variables $x_1, \ldots, x_n$ are paired two and two such that $x_i \neq i$ and $x_i = j$ if and only if $x_j = i$.

The following example shows how a combinatorial problem can be formulated as a CP problem by using logic and global constraints.

Example 1 Consider a tournament with 6 teams. The tournament is played over a period of 5 days, all teams must play one game each day and all teams must meet all other teams once. This problem can be formulated using the global constraints *alldifferent* and *one-factor* leading to the following CP model

$$alldifferent(x_{i1}, \ldots, x_{i5}) \qquad \forall i \in T \tag{2.2.1}$$
$$one\text{-}factor(x_{1s}, \ldots, x_{6s}) \qquad \forall s \in S \tag{2.2.2}$$
$$x_{is} \in T \setminus \{i\} \qquad \forall i \in T, \ \forall s \in S \tag{2.2.3}$$

where T is the set of teams, S is the set of days and the variable x_{is} gives the opponent of team i at day s. □

2.2.2 Backtracking

CP problems can be solved by a systematic search algorithm known as *backtracking*. This is a tree search algorithm like the branch and bound algorithm but in this setup the nodes are referred to as *choice points*. At each choice point, a variable is instantiated to a value consistent with the values of the already instantiated variables. In this way, the algorithm keeps extending a partial solution towards a complete solution until an inconsistency is detected or a feasible solution has been obtained. In the first case, the algorithm backtracks in the search tree until it reaches the most recent instantiated variable with alternative values. From this choice point it once again starts instantiating variables.

The backtracking algorithm performs better than total enumeration but it suffers from the following drawbacks:

1. Trashing
   Trashing is when the algorithm repeatedly instantiates variables to values which are infeasible for the same reason. This happens because the algorithm does not identify conflicting values.

2. Redundant work
   The backtracking algorithm does not remember conflicting instantiations and this means that the same variables can be assigned to the same conflicting values in different branches of the search tree.

3. Late conflict detection
   Conflicts are not detected before an inconsistency occurs.

The first and second of these drawbacks can be handled by *backjumping* and *backmarking* described by Barták [9]. The third drawback can be handled by consistency checks discussed in the following section. In Example 1 we show why late conflict detection is a problem.

Example 1 (Continued) The variables x_{is} can naturally be represented in matrix form where each row represents a team and each column represents a day. Now, imagine that we use backtracking to solve the problem, we instantiate variables from the upper left corner of the matrix and we try to assign as high values as possible to the variables. In this case we would reach the partial solution displayed in Figure 2.1. Already at this point we can see that the partial solution is inconsistent since the four instantiations imply that team 3 and team 4 meet each other in day 1 and day 2. However, this is not detected by the backtracking algorithm since it is able to continue much deeper in the search tree before constraints are violated. Obviously, this leads to a lot of work which could have been avoided if the inconsistency had been detected immediately. □

Day:	1	2	3	4	5
Team 1:	6	5	-	-	-
Team 2:	5	6	-	-	-
Team 3:	-	-	-	-	-
Team 4:	-	-	-	-	-
Team 5:	-	-	-	-	-
Team 6:	-	-	-	-	-

Figure 2.1: Partial solution.

2.2.3 Consistency

A set of constraints is said to be *consistent* when all assignments which cannot be part of a feasible solution are explicitly ruled out. This means that the variables can be instantiated one by one without backtracking. Unfortunately, such a degree of consistency is hard to obtain and instead a lesser degree is used in practice.

Before discussing the alternative degrees of consistency, let us introduce the *constraint graph* also known as the *dependency graph*. The constraint graph contains a node for each variable in the problem and two nodes are connected by an edge if the corresponding variables occur in the same constraint. The constraint graph leads to the notions of *node consistency*, *arc consistency* and *k-consistency*.

Node consistency is the simplest form of consistency and is obtained when all values which violate unary constraints are removed from the domains of the variables.

Arc consistency is obtained when all values which are inconsistent with binary constraints are removed. This means that any consistent instantiation of one variable can be extended to a consistent instantiation of an arbitrary additional variable.

Finally, k-consistency is obtained when any consistent instantiation of $k-1$ variables can be extended to a consistent instantiation of an arbitrary additional variable. We say that a constraint set is *strongly k-consistent* when it is l-consistent for all $l \leq k$. Node consistency corresponds to strong 1-consistency while arc consistency corresponds to strong 2-consistency.

Example 1 (Continued) Since the constraints (2.2.3) are unary constraints node consistency removes the value i from the domain of variable x_{is} for all $i \in T$ and all $s \in S$. The constraints (2.2.1) and (2.2.2) enforce binary constraints but arc consistency is not able to obtain further reductions in the domains. □

Consistency techniques can be used to solve CP problems but the task of obtaining n-consistency for a problem with n variables is often greater than solving the problem using backtrack. Instead, the consistency techniques are combined with backtracking.

2.2.4 Constraint Propagation

The combination of backtracking and consistency techniques are known as *constraint propagation* and constitutes to date the most effective method for solving CP problems. Constraint propagation uses the strengths of both techniques and avoids some of the drawbacks, since the search tree can be reduced and we can limit the consistency techniques to obtain k-consistency for a small k. Often arc consistency is used but the trade-off between the reduction in the search tree and the reduction in the time used to obtain consistency is problem specific.

At each choice point in the search tree, consistency techniques are used to reduce the domains of the variables. Each time we move to a new choice point, a variable has been instantiated to a fixed value and the consistency techniques may be able to reduce the domains of variables occurring in the same constraints as the fixed variables. This may lead to reductions in the domains of other variables and in this way inconsistencies can be detected long before the backtracking algorithm reaches an inconsistency.

Since the domain of one variable may affect the domain of other variables, it is not enough to use the consistency techniques once at each node. The techniques have to be used repeatedly until no further domain reductions are possible.

Example 1 (Continued) Consider once again the instantiation $x_{11} = 6$, $x_{21} = 5$, $x_{12} = 5$ and $x_{22} = 6$. When using arc consistency we can reduce the domains of x_{31}, x_{41}, x_{51}, x_{61}, x_{32}, x_{42}, x_{52} and x_{62} to:

$$D_{x_{31}} = \{4,5,6\}, \quad D_{x_{41}} = \{3,5,6\}, \quad D_{x_{51}} = \{2\}, \quad D_{x_{61}} = \{1\},$$

$$D_{x_{32}} = \{4,5,6\}, \quad D_{x_{42}} = \{3,5,6\}, \quad D_{x_{52}} = \{1\}, \quad D_{x_{62}} = \{2\},$$

by using constraints (2.2.2). The variables with a domain reduced to a singleton can be instantiated immediately and arc consistency can be applied again. This leads to the domains:

$$D_{x_{31}} = \{4\}, \quad D_{x_{41}} = \{3\}, \quad D_{x_{32}} = \{4\}, \quad D_{x_{42}} = \{3\},$$

and the variables x_{31}, x_{41}, x_{32} and x_{42} can be instantiated. Applying arc consistency once more detects an inconsistency since $x_{31} = x_{32}$ and $x_{41} = x_{42}$. In this way the inconsistency is observed without using additional choice points and redundant branching is avoided. □

2.3 Combining IP and CP

Hooker [49] notes that, although both CP and IP are very effective methods for solving combinatorial problems, they use different strategies to obtain solutions. IP uses the linear relaxation to find a good but often infeasible solution while the branching tree is used to obtain feasibility. CP maintains feasibility along the search by starting with a partial feasible solution which is extended to a complete feasible solution during the search.

The differences in the solution methods lead to complementary strengths. IP is in general very good at solving optimization problems since the LP relaxation automatically optimizes the objective value and gives a lower bound. The CP framework does not provide such a bound but the propagation techniques have proven to be successful at solving highly constrained feasibility problems such as planning and scheduling problems.

A natural question arises: Can CP and IP successfully be combined into a hybrid method which outperforms the state of the art versions of CP and IP? The answer is yes for some problem classes.

Sellmann, Zervoudakis, Stamatopoulos, and Fahle [85] and Fahle, Junker, Karisch, Kohl, Sellmann, and Vaaben [29] present hybrid methods for solving the airline crew assignment problem. As discussed in Section 2.1.3, branch and price is well suited for this type of problems but hybrid approaches are able to exploit the strengths of both CP and IP. Some very strict regulations for European airlines make CP advantageous for finding feasible flight sequences while IP provides the best framework for solving the master problem. In this way the techniques are combined and leads to a superior algorithm.

Another application where hybrid methods have proven efficient is machine scheduling on parallel machines. This setup has been used to show considerable reductions in CPU time obtained by a general framework for combining IP and CP known as *logic-based Benders decomposition*. Since this decomposition technique plays an important role in the following chapters, we will give a general introduction to the concept.

2.3.1 Logic-Based Benders Decomposition

Logic-based Benders decomposition was introduced by Hooker and Yan [51] who used it for logic circuit verification. Later Hooker and Ottosson [50] formally developed the idea and illustrated the technique by applying it to propositional satisfiability and to 0-1 programming problems. Jain and Grossmann [53] used it for solving a minimum-cost scheduling problem on dissimilar parallel machines where release and due dates must be satisfied and they showed significant reductions in computation times when compared to CP and IP models. In the scheduling problem used by Jain and Grossmann, the subproblems split into one-machine disjunctive feasibility problems but Hooker [47] shows how logic-based Benders decomposition can be applied to scheduling problems where the subproblems split into one-machine cumulative optimization problems.

To explain logic-based Benders decomposition we use the problem

$$\begin{aligned} \min\ & f(x,y) \\ \text{s.t.}\ & C_1(x) \\ & C_2(x,y) \\ & x \in D_x,\ \ y \in D_y \end{aligned}$$

where x and y are vectors of variables and D_x and D_y are the associated domains. C_1 and C_2 are sets of constraints but in contrast to traditional Benders decomposition, we do not restrict these constraints to be equality or inequality constraints. The constraints discussed in Section 2.2.1 are also applicable.

The partitioning of the problem is similar to the partitioning used in traditional Benders decomposition and we obtain the following master problem and subproblem:

Master problem:	Subproblem:
min z	min $f(\bar{x}, y)$
s.t. $C_1(x)$	s.t. $C_2(\bar{x}, y)$
$x \in D_x$	$y \in D_y$
$z \in \mathbb{R}_+$	

where $\bar{x}$ denotes a solution to the master problem.

The next step is to obtain a cut which can be added to the master problem when the subproblem has been solved. This cut should be able to prevent the master problem from choosing the same solution again if it leads to an infeasible subproblem and it should impose a lower bound on the objective value, if the master problem finds a feasible solution which has been found before.

The traditional Benders cut obtained from the dual solution to the subproblem satisfies both of these criteria and constitutes a very elegant way of transferring information from the subproblem to the master problem. However, the broader framework of logic-based Benders decomposition does not provide an LP subproblem and a new kind of cut must be derived. To obtain such a cut we use an *inference dual* which is the problem of obtaining the strongest bound for a problem. In this case we use the inference dual for the subproblem which can be stated as

$$\begin{aligned} &\max \beta \\ &\text{s.t.} \quad (C_2(\bar{x}, y) \wedge y \in D_y) \Rightarrow (f(\bar{x}, y) \geq \beta). \end{aligned}$$

The optimal value of the inference dual β^* gives a valid lower bound on the optimal value of the subproblem and this lower bound can be used to derive a cut for the master problem. The cut

$$(x = \bar{x}) \Rightarrow (z \geq \beta^*)$$

forces the value of the master problem to be greater than β^* when $\bar{x}$ is chosen. Unfortunately, this cut is not very strong since it only applies when x equals $\bar{x}$. Instead, we need to extend the lower bound to a function $\beta_{\bar{x}}(x)$ which gives a valid lower bound on the master problem for all $x \in D_x$ and is equal to β^* when x equals $\bar{x}$. Such a function leads to the cut

$$z \geq \beta_{\bar{x}}(x)$$

which can be added to the master problem. This cut is known as a *logic-based Benders cut* and, as the name suggests, it is the counterpart of the Benders cut

from traditional Benders decomposition. This cut is formulated as an optimization cut but feasibility cuts can also be added. In case the subproblem is infeasible, we need to find a valid cut which cuts of the current solution or even better which cuts of a set of infeasible solutions.

How the logic-based Benders cuts are derived depends on the given application. In [50] cuts for propositional satisfiability and 0-1 programming problems are derived and in [47] cuts for a disjunctive machine scheduling problem are derived. In the Chapters 4 - 6 we show how cuts can be derived for sports scheduling problems.

2.4 Metaheuristic Solution Methods

The solution methods discussed so far have all been exact methods certain to obtain an optimal solution in a finite amount of time. However, for many combinatorial problems and especially for many scheduling problems, the amount of time needed to find an optimal solution exceeds the time available since an exponential number of solutions exists. When this is the case heuristic methods can be used instead of exact methods since the ability to find near optimal solutions in reasonable time is more important than proving optimality.

The heuristic solution methods can be divided into two broad classes of algorithms: constructive algorithms and local search algorithms. The constructive algorithms build the solution by extending a partial solution until a complete solution has been obtained. This kind of algorithm is often very fast but they may be of varying quality. The local search algorithms, on the other hand, start from an initial solution and try to find improving solutions by using some iteration scheme. They are often very efficient at finding improving solutions but the basic algorithms face the problem of getting stuck in a local optimum.

During the last 20 years the metaheuristic solution methods have evolved from the basic local search algorithms and they provide a higher level framework capable of searching the solution space in an efficient way. There is no exact definition of a metaheuristic but Blum and Roli [16] outline the fundamental properties characterizing a metaheuristic algorithm:

- Metaheuristics are strategies that "guide" the search process.
- The goal is to efficiently explore the search space in order to find (near-) optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.

- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics permit an abstract level description.
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- Today's more advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.

Within the class of metaheuristic algorithms, we distinguish between *trajectory algorithms* and *population based algorithms*. The trajectory algorithms start from a single initial solution and travers through the search space by iteratively replacing the current solution. This group of algorithms include tabu search, simulated annealing, iterated local search and variable neighbourhood search. The population based algorithms are based on a learning process which identifies attractive areas of the search space by learning from the solutions obtained earlier. Instead of focusing on a single solution in each iteration, they focus on a population of solutions and new solutions can be obtained by mutating and combining solutions from this population. This kind of metaheuristics include evolutionary computation and ant colony optimization.

We will not discuss all the metaheuristic methods but concentrate on tabu search since this method is used in Chapter 8. For a thorough discussion of metaheuristic methods we refer to Blum and Roli [16].

2.4.1 Tabu Search

Tabu search introduced by Glover [34] is a trajectory metaheuristic with the ability to guide the search of a descent heuristic using intelligent strategies. It benefits from combining long- and short-term memory which helps the algorithm to remember good solutions and avoid cycling.

Before going into details, some basic definitions are needed. Given a minimization problem: $\min \{f(x)|x \in X\}$ and a feasible solution, $\bar{x}$, a *move* is defined to be some modification of $\bar{x}$ which turns it into a new solution. The exact definition of a move is problem specific but it is usually restricted to minor modifications such that the new solution resembles the original solution. From the definition of a move, we can define the *neighbourhood* $N(\bar{x})$ of solution $\bar{x}$ to be the set of solutions which can be obtained by performing a single move from $\bar{x}$. A local optima is defined to be a solution $\bar{x}$ for which $f(\bar{x}) \leq f(x)$ for all $x \in N(\bar{x})$.

In this setup a simple descent algorithm can be used to solve the minimization problem by starting from an initial feasible solution and keep moving to an

improving solution in the neighbourhood until no improving solutions exist. The descent algorithm may lead to good solutions but it obviously faces a problem when it encounters a local minima.

Tabu search overcomes this problem by allowing the descent algorithm to perform a non-improving move. This makes the descent algorithm capable of escaping a local minima but it also opens for the possibility of cycling. In order to prevent cycling, a *tabu list* is introduced to represent the short-term memory of a tabu search. The tabu list contains a number of forbidden moves which may result in cycling and the moves stay in the tabu list for a specified number of iterations. In this way the tabu search is able to guide the search away from a local optimum and hopefully find better solutions in the long term.

During the search the algorithm may encounter a tabu move leading to a desirable solution and in this case the algorithm is allowed to make an exception and perform the tabu move. To evaluate when a solution is good enough to violate the tabu restriction, an *aspiration criterion* is used and in case this criterion is satisfied a tabu move is allowed.

The long-term memory of a tabu search algorithm is implemented using an *intensification* and a *diversification strategy*. An intensification strategy keeps track of promising moves or solutions such as moves leading to great reductions in the objective value. The algorithm then makes sure that solutions or moves with similar characteristics are considered to see if further improvements can be obtained. By doing this, the algorithm is able to explore some regions of the solution space very thoroughly in case they seem attractive.

The diversification strategy complements the intensification strategy by guiding the search away from explored regions in the solution space. This is achieved by for instance an escape function performing a number of random moves in case the search gets stuck in some region. Alternatively, the objective function can be modified during the search to make new regions more attractive compared to regions explored earlier.

In addition to the standard features described above specialized features can be adopted to enhance computational results for some problems. For a general introduction to tabu search, we refer to Glover [34, 35] and for a thorough description Glover and Laguna [36].

Chapter 3

Round Robin Scheduling

As long as there has been competitive sports, there has been a need for sports schedules. During the last 30 years, sports scheduling has turned into a research area of its own within the operations research and computer science communities. While it may seem trivial to schedule a tournament, and combinatorial mathematics has methods for scheduling simple tournaments, when additional requirements are added the problem becomes a very hard combinatorial optimization problem. In fact, for many types of problems, instances with more than 20 teams are considered large-scale and heuristic solution methods are often necessary in order to find good schedules.

The challenging problems and the practical applications provide a perfect area for developing and testing solution methods. In the literature we find methods ranging from pure combinatorial approaches to every aspect of discrete optimization, including IP, CP, metaheuristic approaches, and various combinations thereof. The solution methods have evolved over time and today methods exist capable of finding optimal or near-optimal solutions for hard practical instances.

In addition to the theoretical gains from developing efficient solution methods, sports scheduling has an economic aspect. Professional sports are big business and the revenue of a sports league may be affected by the quality of the schedule since a substantial part of the revenue often comes from TV networks. The TV networks buy the rights to broadcast the games but in return they want the most attractive games to be scheduled at certain dates.

In this chapter we give a comprehensive survey of the sports scheduling literature concerned with scheduling round robin tournaments. The literature is partitioned into papers on break minimization and papers on distance minimization and, for both parts, we present the main contributions and outline the development. In order to keep the paper within reasonable size we have restricted ourselves to papers on round robin tournaments in which teams are associated

with a particular venue. This means that the problem of finding balanced tournament designs is not considered but for readers interested in this subject we refer to [15, 17, 40, 41, 54, 58, 79, 89, 98].

3.1 Terminology

In this section we will explain the sports scheduling terminology. It is important to stress that the terminology is far from consistent in the literature since notions have multiple meanings. However, to avoid misunderstandings, we will use the definitions from this section throughout the paper although it may conflict with papers to which we refer.

A *round robin tournament* is a tournament where all teams meet all other teams a fixed number of times. Most sports leagues play a double round robin tournament where teams meet twice but single, triple and quadruple round robin tournaments do also occur.

When scheduling a tournament, the games must be allocated to a number of *time slots* (slots) in such a way that each team plays at most one game in each slot. When the number of teams n is even at least $(n-1)$ slots are required and when n is uneven at least n slots are required to schedule a single round robin tournament. In case the number of available slots equals the lower bound, we say that the tournament is *compact* while it is *relaxed* when more slots are available. Note that these terms correspond to the terms *temporally constrained* and *temporally relaxed* defined in [66].

The allocation of games to slots can be presented as a *timetable*. Each row of the timetable corresponds to a team while the columns correspond to slots. The entry of row i and column s is the opponent of team i in slot s. Figure 3.1 shows a timetable for a compact single round robin tournament with 6 teams and a timetable for a corresponding tournament with 7 teams.

Slots	1	2	3	4	5
Team 1	6	3	5	2	4
Team 2	5	6	4	1	3
Team 3	4	1	6	5	2
Team 4	3	5	2	6	1
Team 5	2	4	1	3	6
Team 6	1	2	3	4	5

Slots	1	2	3	4	5	6	7
Team 1	7		5	2	4	6	3
Team 2	5	6		1	3	7	4
Team 3	4	7	6		2	5	1
Team 4	3	5	7	6	1		2
Team 5	2	4	1	7	6	3	
Team 6		2	3	4	5	1	7
Team 7	1	3	4	5		2	6

Figure 3.1: Examples of timetables for tournaments with 6 and 7 teams.

In the literature teams often have an associated *venue* and when they play at their own venue, they play *home games* while they play *away games* at all other venues. It is assumed that each time two teams meet one of the teams plays home while the other plays away. If a team does not play in a slot it is said to have a *bye*. The sequence of home games, away games and byes according to which a team plays during the tournament is known as a *home away pattern* (pattern). If byes occur in the tournament, a pattern is normally represented by a vector with an entry for each slot containing either an H, an A, or a B. In compact tournaments with an even number of teams, all teams play in each slot and the B is omitted. In this case H and A are often replaced by 1 and 0, respectively. In many tournaments it is considered attractive to have an alternating pattern of home and away games and a pattern is said to have a *break* in slots differing from such an alternating sequence. This means that a break corresponds to two consecutive home games or two consecutive away games. Two patterns are said to be *complementary* if the first pattern has an away game when the second pattern has a home game and vice versa. Figure 3.2 (a) shows 2 complementary patterns for a compact single round robin tournament with 6 teams. Notice that both patterns have a break in slot 3.

1	0	0	1	0
0	1	1	0	1

(a)

Slots	1	2	3	4	5
Team 1	1	0	1	0	1
Team 2	1	0	0	1	0
Team 3	0	1	1	0	1
Team 4	1	0	1	0	0
Team 5	0	1	0	1	1
Team 6	0	1	0	1	0

(b)

Figure 3.2: (a) Two complementary patterns, (b) Example of a pattern set for a tournament with 6 teams.

To represent the assignments of home and away games for a tournament with n teams we use a *home away pattern set* (pattern set). This is a set of exactly n patterns and each pattern is associated with one of the teams. Figure 3.2 (b) shows an example of a pattern set for a tournament with 6 teams. Notice that this pattern set exclusively consists of pairs of complementary patterns. When this is the case, the pattern set satisfies the *complementary property* and it is said to be *complementary*. If all teams have the same number of breaks it is an *equitable* pattern set and we say that a pattern set for a single round robin tournament is *equilibrated* when the number of home games for each team varies with no more than one.

Furthermore, the pattern set can be associated with the timetable for 6 teams displayed in Figure 3.1 since, in every game, one of the opponents plays home while the other plays away. A pattern set for which a corresponding timetable exists is said to be *feasible*. Figure 3.3 gives an example of three patterns which would make a pattern set *infeasible* since the three mutual games can only be played in slots 1 and 2.

1	0	0	1	0
1	1	0	1	0
0	1	0	1	0

Figure 3.3: Example of an infeasible subset of patterns.

The combination of a pattern set and a corresponding timetable constitutes a *schedule* for a tournament. A schedule is *mirrored* when the first and the second half are identical except the home games and away games are exchanged. Furthermore, we say that a schedule is *irreducible* when at most one opponent in each game has a break. A schedule can be represented as in Figure 3.4 showing a mirrored double round robin schedule. In the figure, a + denotes a home game while a − denotes an away game. A sequence of consecutive away games is called a *trip* while a sequence of consecutive home games is called a *home stand*. An entire row of the schedule defines a *tour* for the corresponding team.

Slots	1	2	3	4	5	6	7	8	9	10
Team 1	+6	−3	+5	−2	+4	−6	+3	−5	+2	−4
Team 2	+5	−6	−4	+1	−3	−5	+6	+4	−1	+3
Team 3	−4	+1	+6	−5	+2	+4	−1	−6	+5	−2
Team 4	+3	−5	+2	−6	−1	−3	+5	−2	+6	+1
Team 5	−2	+4	−1	+3	+6	+2	−4	+1	−3	−6
Team 6	−1	+2	−3	+4	−5	+1	−2	+3	−4	+5

Figure 3.4: Example of a mirrored double round robin schedule.

When solving a sports scheduling problem it may be advantageous to postpone the assignment of games until a schedule has been obtained. In that case *placeholders* are used to represent the teams in the pattern set and in the timetable until a schedule has been found.

Since round robin tournaments have a correspondence to graphs, we also introduce a few graph theoretical concepts. Consider a graph $G = (V, E)$ where

V is a finite set of nodes and E is the set of edges in G. A *matching* in G is a set of independent edges (non-adjacent edges) and a matching in which all the nodes in V are incident to an edge is called a *complete matching*. The graph induced by a complete matching is a 1-regular graph (the degree is 1 for all nodes). This is called a 1-*factor* and a partitioning of the graph G into factors is called a 1-*factorization*.

In the rest of the paper, we let n denote the number of teams, T the set of teams and S the set of slots. Since most of the sports scheduling literature focuses on compact tournaments with an even number of teams this is assumed to be the case unless otherwise stated.

3.2 Constraints

Practical sports scheduling applications are very often characterized by a large number of conflicting constraints arising from teams, TV networks, sports associations, fans and local communities. Consequently, a section discussing the various constraints applicable to a particular sports league has become a standard part of papers considering practical applications since each league has their own special requirements. In this section we will give a short outline of the most typical constraints and we give references to papers facing these constraints.

Place constraints ([10, 11, 18, 22, 23, 25, 30, 45, 70, 72, 75, 81, 83, 87, 91, 96, 100])
Constraints ensuring that a team plays home or away in a certain slot. This kind of constraint is normally imposed when a venue is unavailable due to other events.

Top team and bottom team constraints ([10, 23, 25, 43, 45, 66, 70, 81, 83])
In some leagues special considerations are taken for teams which have just qualified for the league and teams which are known to be strong.

Break constraints ([10, 25, 43, 45, 66, 70, 97])
Often leagues want to avoid a schedule where teams have a break in slot 2 or a break in the last slot.

Game constraints ([10, 18, 25, 28, 43, 46, 62–64, 66, 69, 70, 86, 100])
These are constraints fixing a certain game to a particular time slot. The constraints are normally imposed by TV networks who want "big" games at certain dates.

Complementary constraints ([10, 18, 23, 25, 70, 81, 97])
When two teams share a venue, a complementary constraint is used to make

sure that the two teams play home at different slots. Of course both teams play home in some sense when they meet but playing the official home game may be important since revenue is often earned by the home team.

Geographical constraints ([10, 23, 25, 70, 83, 93])
To avoid slots in which many home games are gathered in a small area, games should be scattered throughout the region in which the tournament is played.

Pattern constraints ([4, 10, 11, 18, 19, 22, 23, 25, 30, 43, 45, 62, 66, 70, 75, 78, 81, 87, 91, 100])
Some applications have special requirements on the patterns such as restrictions on the number of consecutive breaks or certain sequences of home games, away games and byes which should be avoided. They also include requests for equitable pattern sets saying that all teams must have the same number of breaks.

Separation constraints ([10, 22, 30, 43, 66, 70, 72, 100])
When we consider tournaments where teams meet more than once and the schedule is non-mirrored, most leagues have a lower bound on the number of slots between two games with the same opponents. This constraint is not relevant to mirrored schedules since such schedules always have at least $n-2$ slots between such games.

In many applications the constraints are separated into hard constraints and soft constraints. All the hard constraints must be satisfied in a feasible solution, while the soft constraints are penalized such that penalties are incurred if the constraints are violated. In addition to minimizing the number of violated soft constraints the objective of a sports scheduling problem is normally to minimize either the number of breaks or the travel distance. In the following two sections we will discuss the papers on minimizing breaks and minimizing travel distance, respectively.

3.3 Minimizing Breaks

When teams return home after each away game instead of travelling from one away game to the next, an alternating pattern of home and away games are usually preferred. Such patterns consider the fans by avoiding long periods without home games and they ensure regular earnings from home games. Furthermore, the strength of a team is better reflected by its position throughout the tournament when all teams alternate between home and away games since a team starting with a long sequence of home games might do relatively better than a team starting with many away games.

The need for alternating patterns has led to a large amount of research originating from graph theoretical approaches for minimizing the number of breaks in a pattern set and leading to highly sophisticated solution methods for practical applications facing numerous constraints. During the last 30 years, focus has moved from constructive methods applicable for general tournaments without specific constraints to decomposition methods capable of handling all the constraints applicable for a certain sports league.

3.3.1 Constructive Methods

In the 1980's, Rosa and Wallis [77], de Werra [93, 94, 95, 96], de Werra et al. [97] and Schreuder [82] published a number of papers on the relationship between graphs and tournaments and used the relationship to obtain results for schedules. De Werra [94] presents the relationship between a tournament and a graph in the following way.

Consider a compact single round robin tournament with an even number of teams n - in case the number of teams is uneven a dummy team can be added. This tournament can be associated with the complete graph K_n by letting each node correspond to a team and letting each edge correspond to the game between the teams associated with the end nodes. A factorization $F = (F_1, \ldots, F_{n-1})$ of K_n where $F_1, \ldots, F_{n-1}$ are 1-factors then corresponds to a partitioning of the games into $n-1$ slots since each node will be incident to exactly one edge in each 1-factor.

The home away assignments can be represented by orienting the edges and letting an edge from node i to node j correspond to a game where team i visits team j. An oriented 1-factorization $\vec{F} = (\vec{F}_1, \ldots, \vec{F}_{n-1})$ or equivalently an oriented $(n-1)$-coloring then characterizes a schedule for the single round robin tournament. Figure 3.5 shows an oriented 1-factorization of K_6 and Figure 3.6 shows the associated schedule. We refer to Mendelsohn and Rosa [59] for a survey on 1-factorizations.

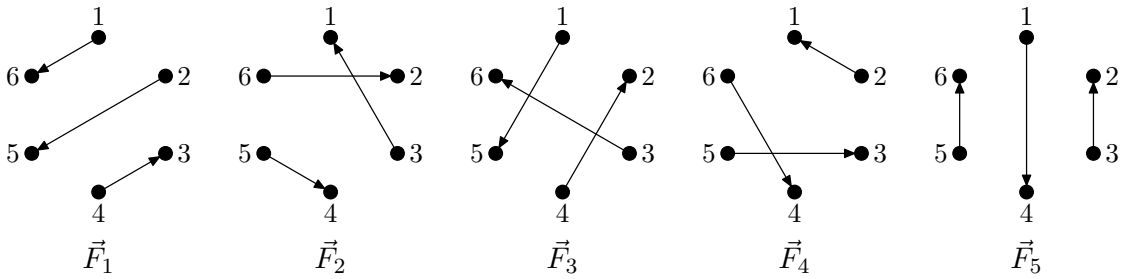

Figure 3.5: Oriented 1-factorization of K_6.

Slots	1	2	3	4	5
Team 1	-6	$+3$	-5	$+2$	-4
Team 2	-5	$+6$	$+4$	-1	$+3$
Team 3	$+4$	-1	-6	$+5$	-2
Team 4	-3	$+5$	-2	$+6$	$+1$
Team 5	$+2$	-4	$+1$	-3	-6
Team 6	$+1$	-2	$+3$	-4	$+5$

Figure 3.6: Schedule corresponding to the 1-factorization from Figure 3.5.

We present some of the most important results obtained from the relationship between graphs and schedules. The first and most basic result is the following.

Proposition 1 (De Werra [94]) *In any oriented coloring of K_n, there are at least $n-2$ breaks.*

The proof is straightforward when observing that at most two teams can have a pattern without breaks. However, the result is very important since it gives a lower bound on the number of breaks in a single round robin tournament. Furthermore, de Werra was also able to show that the lower bound was obtainable by constructing a 1-factorization with exactly $n-2$ breaks. The 1-factorization is called the *canonical 1-factorization* and it is defined as follows.

Definition 1 (De Werra [94]) *The canonical 1-factorization satisfies that for $i = 1, \ldots, n-1$*

$$F_i = \{(n, i)\} \cup \{(i+k, i-k) : k = 1, \ldots, n/2 - 1\}$$

where $i+k$ and $i-k$ are expressed as one of the numbers $1, \ldots, n-1$ (mod $n-1$).

To obtain a schedule with exactly $n-2$ breaks, the canonical factorization is oriented such that the edge (i, n) is oriented from i to n if i is odd and from n to i if i is even and the edge $(i+k, i-k)$ in F_i is oriented from $i+k$ to $i-k$ if k is odd and the other way if k is even.

Proposition 2 (De Werra [94]) *There exists an oriented coloring of K_n with exactly $n-2$ breaks.*

The canonical 1-factorization has subsequently been widely used in the literature and the associated schedule is referred to as the *canonical schedule*. The factorization and schedule shown in Figure 3.5 and Figure 3.6 are the canonical factorization and the canonical schedule for a tournament with 6 teams.

The canonical schedule can also be used for tournaments with an uneven number of teams by using a dummy node and in this way de Werra was able to construct a tournament without breaks.

Corollary 1 (De Werra [94]) *K_{n+1} has an oriented coloring without breaks.*

Notice, that the removal of team 6 in Figure 3.6 produces a schedule for 5 teams without breaks.

Multi-period schedules were also considered and the following two results were obtained.

Proposition 3 (De Werra [94]) *A mirrored double round robin tournament has at least $3n-6$ breaks.*

This can be proved by noting that teams with 1 break in the first half have a corresponding break in the second half and a third break at the beginning of the second half.

Proposition 4 (De Werra [94]) *A mirrored double round robin tournament with exactly $3n-6$ breaks exists and if $n \neq 4$ no team has two consecutive breaks.*

Again the canonical schedule was used for constructing a mirrored double round robin tournament with exactly $3n-6$ breaks although small modifications were necessary to avoid consecutive breaks. The resulting schedule is known as the *modified canonical schedule*.

In [93] de Werra gives a characterization of canonically feasible break sequences and he considers tournaments facing geographical constraints. The geographical constraints require that, when teams are located close to each other, they should have complementary patterns if possible. De Werra [93] treats a number of specific problems occurring when geographical constraints are considered and presents constructive methods for obtaining schedules.

Schreuder [82] formulates necessary and sufficient conditions for a tournament by using 0-1 variables $x_{i_1 i_2 s}$ which is 1 if team i_1 plays home against team i_2 in slot s. The conditions look as follows:

$$\sum_{i_1 \in T} (x_{i_1 i_2 s} + x_{i_2 i_1 s}) = 1 \qquad \forall i_2 \in T,\ \forall s \in S$$

$$\sum_{s \in S} (x_{i_1 i_2 s} + x_{i_2 i_1 s}) = 1 \qquad \forall i_1, i_2 \in T,\ i_1 \neq i_2$$

Rosa and Wallis [77] raise an interesting problem regarding the timetable. They define a *premature set* to be a partial timetable (only the first k slots are determined) which cannot be extended to a full timetable and asks the question: How much can go wrong if we assign games one slot at a time without looking ahead? In other words do premature sets exist? Indeed, they do exist and Rosa and Wallis prove the following corollary.

Corollary 2 (Rosa and Wallis [77]) *For n even, there is a premature set of k one-factors in K_n whenever $\frac{n}{2} \leq k \leq n-3$ and $\frac{n}{2}$ is odd, and whenever $\frac{n}{2} < k \leq n-3$ and $\frac{n}{2}$ is even.*

They also show that when the tournament is big enough nothing can go wrong in the first slots.

Corollary 3 (Rosa and Wallis [77]) *If $n \geq 8$ and even, there exists no premature set of three 1-factors in K_n.*

This corollary is followed by a conjecture which is still an open question.

Conjecture 1 (Rosa and Wallis [77]) *For any positive integer k, there exists $n(k)$ such that if $n > n(k)$, then any premature set of 1-factors of K_n contains more than k one-factors.*

In [95] de Werra concentrates on irregularities in schedules. He notices that when no breaks occur between two time slots s_1 and s_2 the edges of the oriented graph K_n, corresponding to the games played in the slots $s_1, \ldots, s_2$, will form a regular bipartite graph. This property is used to obtain schedules minimizing the number of *irregular slots* (slots containing a break) and to distribute the irregular slots evenly.

De Werra summarizes most of the previous results in [96] where, for the first time, place constraints are considered. In order to solve the scheduling problem with place constraints, schedules with placeholders are generated and, for each schedule, teams are assigned to placeholders by constructing a factor in a bipartite graph. The bipartite graph contains a node for each team, a node for each placeholder, and an edge between a team i and a placeholder j if the pattern of placeholder j satisfies the place constraints of team i. If a factor can be constructed in the bipartite graph we have a feasible solution and otherwise we move on to the next schedule. This is the first step towards the decomposition methods presented in the following section.

However, before moving to the decomposition methods, let us mention de Werra et al. [97] facing a problem with 2 leagues A and B. League A plays a double round robin while league B plays a single round robin before it is partitioned into two leagues C and C' which both play an additional single round robin. The partitioning of league B is not known in advance since it depends on the outcome of the games. The objective is to spread breaks evenly and minimize the total number of breaks. The problem is constrained by teams from different leagues using the same venue, and breaks in the last slot is not allowed. Since team specific requirements are not considered, it is possible to construct an optimal solution for the problem.

3.3.2 The Constrained Minimum Break Problem

In the beginning of the 1990's focus moved from the graph theoretical results to practical applications. This change meant that the constraints outlined in Section 3.2 were taken into account and solution methods capable of handling these

constraints had to be developed. The problem of finding a schedule minimizing the number of breaks and at the same time take additional constraints into account is known as the *constrained minimum break problem*. However, the problem may change significantly from one application to another since different constraints are considered.

To solve the problem two metaheuristic approaches were applied by Willis and Terrill [99] who use simulated annealing and Wright [101] who uses tabu search for scheduling cricket tournaments. However, the majority of the papers use a decomposition approach. A sports scheduling problem naturally decomposes into four steps and, although the order of the steps vary and some steps are combined, these four steps are used in almost all solution methods for solving variations of the constrained minimum break problem. The four steps are:

Step 1 Generate patterns.

Step 2 Find a pattern set for placeholders.

Step 3 Find a timetable for placeholders.

Step 4 Allocate teams to placeholders.

Schreuder [83] solves a mirrored double round robin problem for the Dutch professional football league and uses a 2-phase approach which resembles the method used by de Werra [96]. In this method Phase 1 combines Steps 1 to 3 by constructing the canonical schedule for placeholders and Phase 2 corresponds to Step 4 and allocates teams to placeholders. The problem of assigning teams to placeholders is formulated as a quadratic assignment problem and a heuristic solution method is presented for solving the problem.

In 1998 Nemhauser and Trick [66] schedules the basketball tournament for the Atlantic Coast Conference consisting of nine university teams from the United States. In their approach all four steps are used but instead of using a combinatorial design, as seen in the earlier approaches, they use IP combined with enumeration techniques to obtain pattern sets. In Step 1 they generate mirrored patterns having a reasonable chance of being used in a feasible pattern set and in order to satisfy a specific constraint, slots 8 and 10 are interchanged. After the patterns have been generated, an IP model is used in Step 2 to generate pattern sets. The IP model chooses 9 patterns which minimize the number of breaks and it requires that in each slot, 4 patterns have a home game, 4 patterns have an away game and 1 pattern has a bye. All feasible solutions to the model are generated and it leads to 17 pattern sets. For each pattern set all feasible timetables are generated using another IP model and this leads to 826 timetables. Finally, teams are allocated to placeholders by enumerating through the 9! possible allocations. Almost 300 million schedules had to be considered but only 17 were feasible and from these schedules a final schedule was chosen.

After the IP/enumeration approach by Nemhauser and Trick, Schaerf [81], Henz [43, 45], and Régin [75] introduced CP approaches for solving sports scheduling problems.

Schaerf [81] considers the problem of scheduling a mirrored double round robin tournament with complementary constraints, place constraints, geographic constraints and top team constraints. The constraints are split into hard constraints which must be satisfied and soft constraints enforcing a penalty when violated. To solve the problem, he uses the 2-phase approach known from de Werra [96] and Schreuder [83] in which Phase 1 combines Steps 1, 2 and 3 while Phase 2 corresponds to Step 4. Phase 1 is handled by using the modified canonical schedule since this schedule minimizes the number of breaks and avoids consecutive breaks but it is noted that Phase 2 is independent of the schedule chosen in Phase 1. The assignment problem considered in Phase 2 is solved using CP. The variables and constraints used to formulate the problem is outlined and computational results are presented. The CP model takes longer time than the heuristic method presented by Schreuder [83] but in return it gives the optimal solution.

In contrast to Schaerf [81], Henz [45] uses CP to solve all four steps. The individual steps are solved in the order 1, 2, 3, 4 and in the order 1, 2, 4, 3. Henz reports that, in most cases, the best performances are obtained by solving Step 4 before Step 3. CP models are presented for each of the four steps and a generic constraint-based round robin planning tool known as Friar Tuck is presented. Friar Tuck uses the finite domain constraint programming system Mozart 1.0 and allows the user to fine-tune the solution process and the constraints. In [43] Henz uses the CP approach explained in [45] to solve the Basketball league considered by Nemhauser and Trick and shows that the CP approach clearly outperforms the combined IP and enumeration technique used previously. Henz is able to find all solutions to the problem in less than one minute while Nemhauser and Trick used more than 24 hours.

Régin [75] also presents CP approaches for solving sports scheduling problems. At first he gives a general discussion of symmetry breaking constraints, the use of implicit constraints, global constraints and pertinent and redundant constraints. This discussion is followed by a CP model for generating a single round robin schedule with a minimal number of breaks when no additional constraints are present. Régin shows how symmetry breaking is able to enhance performance significantly and the problem size solvable in approximately 1 minute increases from 6 to 60 teams. Next Régin considers a problem which is later known as *the break minimization problem*.

Definition 2 *Given a timetable, the break minimization problem consists of finding a feasible pattern set which minimizes the number of breaks.*

For this problem most of the symmetry breaking constraints added to the first model becomes invalid but Régin is able to derive new constraints and again

significant improvements can be obtained. In this case a problem with 16 teams can be solved in approximately 1 minute.

Subsequently, Trick [86] motivates the use of the break minimization problem by discussing the order of the four solution steps. He argues that the steps should be ordered such that the most critical aspects of the schedule are considered early in the solution process. Solving Steps 1 and 2 before Steps 3 and 4 makes sense when for instance many place constraints are considered. On the other hand, when game constraints or other constraints associated with the timetable become more important, Steps 3 and 4 should be solved before Steps 1 and 2. Trick presents a 2-phase solution method which solves Steps 3 and 4 in Phase 1 and solves the break minimization problem corresponding to Steps 1 and 2 in Phase 2. The method combines CP and IP by using CP for Phase 1 and IP for Phase 2. Two CP models for solving Phase 1 are discussed and both models are able to find a 20-team schedule in less than 1 second and able to find 500 20-team schedules in around one minute. In Phase 2 the symmetry breaking constraints presented by Régin [75] are used in an IP model and the computation times show improvements for large instances (more than 16 teams) compared to the CP model presented by Régin.

The papers by Régin [75] and Trick [86] were followed by a number of papers focusing on the break minimization problem alone. Elf et al. [28] show that solving the break minimization problem is equivalent to a maximum cut problem in an undirected graph G. Given a timetable, the graph G is constructed by adding a node v_{is} for each team i and each slot s such that v_{is} corresponds to entry (i, s) in the timetable. An example is shown in Figure 3.7. For each team i and each

Slots	1	2	3	4	5
Team 1	6	3	5	2	4
Team 2	5	6	4	1	3
Team 3	4	1	6	5	2
Team 4	3	5	2	6	1
Team 5	2	4	1	3	6
Team 6	1	2	3	4	5

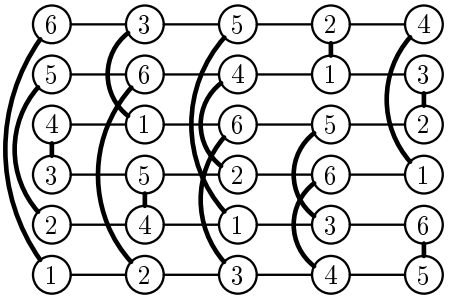

Figure 3.7: Timetable and corresponding maximum cut graph.

slot s in $2, \ldots, |S|$ the nodes v_{is-1} and v_{is} are connected by an edge corresponding to the horizontal edges in Figure 3.7. The vertical edges combine nodes $v_{i_1 s}$ and $v_{i_2 s}$ when team i_1 plays against team i_2 in slot s. By assigning a weight of 1 to all the horizontal edges and a weight M to the vertical edges, Elf et al. are able to show that a maximum cut in G corresponds to an optimal solution to the

break minimization problem when $M \geq n(n-2)+1$. The reasoning behind the argument is that a cut separates the vertices into two sets. One of the sets will correspond to home games and the other to away games. In order to obtain a feasible home-away assignment we must ensure that, when two nodes play against each other, one belongs to the set of home games while the other belongs to the set of away games. This is handled by assigning big weights to all the vertical edges in G. Maximizing the number of horizontal edges in the cut, corresponds to minimizing the number of breaks since an edge which is not part of the cut leads to a break.

After the graph G has been constructed, Elf et al. show how to transform this graph into a smaller graph by contracting the vertical edges one by one and changing the signs of some of the horizontal edges. This leads to a graph with $\frac{n(n-1)}{2}$ nodes and $n(n-1)$ edges. The modified graph speeds up the solution process since a maximum cut for the modified graph can be directly transformed to a maximum cut for the original graph G. A maximum cut is found by applying a branch and cut algorithm described by Barahona, Grötschel, Jünger, and Reinelt [7]. The computational tests show great reductions in computation times compared to the CP and IP approaches presented by Régin and Trick, respectively, and instances with up to 26 teams can be solved within reasonable time (1215.9 seconds).

A similar idea is used by Miyashiro and Matsui [63] who also consider the break minimization problem. They use two graphs G_1 and G_2 both having a node set equal to the node set of G. The edges of G_1 correspond to the horizontal edges of G while the edges of G_2 correspond to the vertical edges of G. Instead of using weights equal to M, they notice that the problem is a special case of MAX RES CUT discussed by Goemans and Williamson [37] and therefore solvable by an approximation algorithm based on positive semidefinite programming proposed by Goemans and Williamson [37]. The problem can also be stated as a special case of MAX 2SAT but solving the MAX 2SAT problem is equivalent to solving the MAX RES CUT problem when the algorithm of Goemans and Williamson is applied. In contrast to the previous methods on the break minimization problem, this is an approximative method and this makes it capable of finding solutions for problems with up to 40 teams compared to the 26 teams considered by Elf et al. [28].

At the end of the paper by Elf et al. [28], it is conjectured that instances with only $n-2$ breaks are solvable in polynomial time and that the break minimization problem in general is NP-hard.

The first conjecture was proved affirmatively in [61] where Miyashiro and Matsui consider the problem of finding a pattern set with exactly $n-2$ breaks for a given timetable or showing that such a pattern set does not exist. The problem is reduced into n decision problems $\mathrm{P1}_k$ for $k = 1, \ldots, n$ where $\mathrm{P1}_k$ is similar to the original problem except for an extra constraint requiring that team k has a pattern

without breaks and starts with a home game. If a pattern set exists for one of the problems $P1_1, \ldots, P1_n$, we have a solution and otherwise no feasible pattern set exists with $n-2$ breaks. Since 2SAT problems can be solved in polynomial time, Miyashiro and Matsui are now able to show that the original problem can be solved in polynomial time by transforming each of the problems $P1_1, \ldots P1_n$ into a 2SAT problem. The transformation is accomplished by constructing a pattern set and using boolean variables x_{is} which are true when team i plays according to the constructed pattern set in slot s and false otherwise. The conclusion is that, for a given timetable, it is possible to find a feasible pattern set with $n-2$ breaks in polynomial time or show that such a pattern set does not exist.

Corollary 4 (Miyashiro and Matsui [61]) *The following problem is solvable in polynomial time.*

Instance: *A timetable with n teams where n is even.*

Task: *Find a pattern set with at most $n-2$ breaks that is consistent with the given timetable if it exists and return "infeasible" otherwise.*

Furthermore, Miyashiro and Matsui [62] show that the result is also valid for pattern sets with n breaks.

Corollary 5 (Miyashiro and Matsui [62]) *The following problem is solvable in polynomial time.*

Instance: *A timetable with n teams where n is even.*

Task: *Find a pattern set with at most n breaks that is consistent with the given timetable if it exists and return "infeasible" otherwise.*

The procedure is very similar to the procedure used in [61]. Again the problem is transformed to a number of 2SAT problems and since they can be solved in polynomial time, it is possible to solve the original problem in polynomial time. In addition to the corollary an interesting property combining break minimization and break maximization is presented. Given a pattern set H represented by a h_{is} for each team i and each slot s, the pattern set $\tilde{H}$ is defined such that $\tilde{h}_{is} = h_{is}$ if s is uneven and $\tilde{h}_{is} \neq h_{is}$ if s is even. Due to the construction, each team has a break in each slot s, $s \geq 2$, in exactly one of the pattern sets and this leads to the following lemma.

Lemma 1 (Miyashiro and Matsui [62]) *Let H be a pattern set for a tournament with n teams where n is even. Then the number of breaks in H plus the number of breaks in $\tilde{H}$ equals $n(n-2)$.*

Lemma 1 implies the following theorem.

Theorem 1 (Miyashiro and Matsui [62]) *Given a timetable, then a feasible pattern set minimizes the number of breaks if and only if $\tilde{H}$ maximizes the number of breaks.*

This implies that minimizing and maximizing the number of breaks for a given timetable is equivalent.

The second conjecture by Elf et al. [28] regarding NP-hardness of the break minimization problem is considered by Post and Woeginger [69]. They consider *partial timetables* for single round robin tournaments. The partial timetables only contain a subset of the normal $n-1$ slots and they satisfy that two teams do not meet more than once. By using a polynomial time reduction from an NP-hard version of the Max-Cut problem Post and Woeginger are able to show the following theorem.

Theorem 2 (Post and Woeginger [69]) *Break minimization in partial timetables with n teams and three slots is NP-hard.*

The theorem leads to the following corollary.

Corollary 6 (Post and Woeginger [69]) *Break minimization in partial timetables with n teams and a fixed number $r \geq 4$ of slots is NP-hard.*

Post and Woeginger also consider lower and upper bounds on the solution values for the break minimization problem. Let $B_{min}(TT_n)$ be the optimal solution value to the break minimization problem given the timetable TT_n with n teams. They are able to obtain a lower bound on $\max_{TT_n} B_{min}(TT_n)$ when $n = 4^k$ for some $k \geq 1$.

Theorem 3 (Post and Woeginger [69]) *For $n = 4^k$ teams with $k \geq 1$, there exists a timetable TT_n^* with $B_{min}(TT_n^*) \geq \frac{1}{6}n(n-1)$.*

An upper bound on $B_{min}(TT_n)$ for an arbitrary timetable TT_n is also derived.

Theorem 4 (Post and Woeginger [69]) *Each timetable TT_n for n teams satisfies*

$$B_{min}(TT_n) \leq \begin{cases} \frac{1}{4}n(n-2), & \text{if } n \text{ is of the form } 4k; \\ \frac{1}{4}(n-2)^2, & \text{if } n \text{ is of the form } 4k+2. \end{cases}$$

Furthermore, a corresponding pattern set can be computed in polynomial time.

In Table 3.1 the lower bounds obtained by Elf et al. [28] are denoted LB-EJR, the lower bounds for schedules with $n = 4^k$ are denoted LB-PW and the upper bounds are stated UB-PW according to the table from [69].

Finally, Post and Woeginger conjecture that the upper bound on $B_{min}(TT_n)$ for any even n and any timetable TT_n can be improved to $\frac{1}{6}n(n-1)$. However, we are able to obtain a counterexample with $n = 8$ to this conjecture by using

Table 3.1: Upper and lower bounds for $\max_{TT_n} B_{min}(TT_n)$ with $n \leq 26$.

n	4	6	8	10	12	14	16	18	20	22	24	26
LB-EJR	2	4	8	12	18	26	32	44	54	64	74	90
LB-PW	2	–	–	–	–	–	40	–	–	–	–	–
UB-PW	2	4	12	16	30	36	56	64	90	100	132	144

a simple 2 phase approach. Phase 1 consists of a basic CP model for generating timetables [46] and in Phase 2 we solve the break minimization problem by using the IP model presented in [86]. The procedure iterates between the two phases and, for each number of teams n, we are able to obtain the lower bounds displayed in Table 3.2 within 15 minutes of computation time. Table 3.2 also displays the upper bounds conjectured by Post and Woeginger and we see that, for $n = 8$, our lower bound exceeds the conjectured upper bound.

Table 3.2: Lower bound for $\max_{TT_n} B_{min}(TT_n)$ and conjectured upper bound.

n	4	6	8	10	12	14	16	18	20	22	24	26
LB-RT	2	4	12	14	20	26	32	40	–	–	–	–
UB-Conj	2	5	9	15	22	30	40	51	63	77	92	108

The minimum break problem was motivated by the scheduling approach used by Régin [75] and Trick [86] but it only solves half the problem since it requires a given timetable. The first part of this approach regarding the timetabling problem has been considered by Henz et al. [46]. They use variables o_{is} to represent the opponent of team i in slot s and they formulate the problem using the global CP constraints *alldifferent* and *one-factor*.

$$\begin{aligned} &\textit{alldifferent}(o_{i1}, \ldots, o_{in-1}) && \forall i \in 1, \ldots, n, \\ &\textit{one-factor}(o_{1s}, \ldots, o_{ns}) && \forall s \in 1, \ldots, n-1. \end{aligned}$$

The two constraints make sure that any feasible solution constitutes a timetable for a single round robin tournament. However, since this is CP constraints they can be implemented in more than one way and the choice of propagation technique used for each of the constraints may have a great impact on the size of the search tree and the computation time used to solve the problem. Henz et al. provide an extensive analysis of propagation techniques to obtain guidelines for choosing the most effective solution method. In the analysis it is concluded that the propagation techniques used for the *alldifferent* constraint should obtain arc-consistency since this will reduce both the search tree and the runtime when compared to

weaker consistency techniques. For the *one-factor* constraint, three propagation techniques are considered.

1. Arc-consistent propagation with respect to the constraints:
$$o_{is} \neq i \qquad i = 1, \ldots, n$$
$$o_{o_{is}s} = i \qquad i = 1, \ldots, n$$

2. Arc-consistent propagation with respect to the constraints:
$$o_{is} \neq i \qquad i = 1, \ldots, n, \;\; s = 1, \ldots, n-1$$
$$o_{o_{is}s} = i \qquad i = 1, \ldots, n, \;\; s = 1, \ldots, n-1$$
$$\mathit{alldifferent}(o_{1s}, \ldots, o_{ns}) \qquad s = 1, \ldots, n-1$$

3. Arc-consistent propagation with respect to the constraints:
$$\mathit{one\text{-}factor}(o_{1s}, \ldots, o_{ns}) \qquad \forall s \in 1, \ldots, n-1.$$

The first of the three propagation techniques leads to poor performances but, by adding the redundant *alldifferent* constraint in the second technique, much better results are obtained. The computational tests show that, when a pattern set is given, the second technique obtains the best results. The additional time used to obtain arc consistency for the *one-factor* constraint in the third technique outweights the time reduction achieved by the reduction in the search tree. However, when no pattern set is given, the third propagation technique obtains the best results.

Trick [87] is able to show why the second propagation technique works better than the third when the pattern set is given. Let D_i be the set of feasible opponents for team i in a given slot s. Then D_i is said to be bipartite if the set of teams can be divided into two sets X and Y such that
$$|X| = |Y| = \frac{n}{2}$$
$$i \in X \Rightarrow D_i \subseteq Y$$
$$i \in Y \Rightarrow D_i \subseteq X$$

Theorem 5 (Trick [87]) *If the D_i are bipartite, then arc-consistency for the constraints*
$$o_{is} \neq i \qquad i = 1, \ldots, n$$
$$o_{o_{is}s} = i \qquad i = 1, \ldots, n$$
$$\mathit{alldifferent}(o_{1s}, \ldots, o_{ns})$$
implies arc-consistency for the constraint
$$\mathit{one\text{-}factor}(o_{1s}, \ldots, o_{ns})$$

This result means that, when the pattern set is determined before we find a timetable, there is no point in using the third propagation technique since arc-consistency for the *one-factor* constraint is obtained by the second technique and it requires less computation time. In the paper Trick provides numerous comparisons of CP and IP models for solving sports scheduling problems. These include tightly constrained timetables, schedules with home-away restrictions and schedules for more than one division. The conclusion is that IP in general performs best when an objective value is considered, while CP is best at handling the feasibility problems. However, at a few feasibility problems, IP outperforms the CP model since the propagation techniques were unable to recognize infeasibility.

Although much work has concentrated on the break minimization problem, some of the recent papers on practical sports scheduling applications find pattern sets before timetables. This approach relies on good pattern sets in the first phase but finding a characterization of feasible pattern sets is still an open problem. However, Miyashiro et al. [64] present a necessary condition for feasible pattern sets and show that the condition characterizes feasible pattern sets with a minimum number of breaks for schedules with up to 26 teams. For a subset of teams $\hat{T} \subseteq T$ they let the functions $A(\hat{T}, s)$ and $H(\hat{T}, s)$ return the number of away games and home games $\hat{T}$ plays in slot s. The necessary condition can then be stated as follows.

$$\sum_{s \in S} \min\{A(\hat{T}, s), H(\hat{T}, s)\} \geq \frac{|\hat{T}|(|\hat{T}| - 1)}{2} \qquad \forall \hat{T} \subseteq T \tag{3.3.1}$$

The reasoning behind the condition is that any subset of teams $\hat{T}$ must play $\frac{|\hat{T}|(|\hat{T}|-1)}{2}$ games in a single round robin tournament and, in any slot s, they cannot play more than $\min\{A(\hat{T}, s), H(\hat{T}, s)\}$ mutual games. Miyashiro et al. also show that, for pattern sets with a minimum number of breaks and no more than 26 teams the condition is both necessary and sufficient. Furthermore, whether a given pattern set with a minimum number of breaks satisfies the condition can be checked in polynomial time. and it can be checked in polynomial time.

Croce and Oliveri [23] schedules the Italian soccer league and again this is a problem with a lot of additional constraints. Each team is assigned to one of two concurrent TV networks and the chosen TV network holds the rights to all the home games of the particular team. This means that the schedule should be balanced with respect to TV coverage such that both TV networks have a proportional part of the home games in each slot. Furthermore, the league contains teams sharing stadium and therefore complementary constraints must be imposed. The problem is solved by a 3-phase approach but all four decomposition steps are actually used since all patterns with no more than 4 breaks are generated before solving Phase 1. Phase 1 corresponds to Step 2, Phase 2 corresponds to Step 3 and Phase 3 corresponds to Step 4. All phases are solved by IP models and to

obtain a good solution, the phases are solved iteratively according to the following scheme.

1. 200 pattern sets are generated.

2. For each generated pattern set a feasible timetable is found if possible.

3. For each generated feasible timetable, teams are allocated to placeholders.

The solution method is able to generate a number of high quality schedules and the authors note that preliminary contacts with the Italian Football (soccer) League is ongoing.

Rasmussen and Trick [72] propose another iterative approach using logic-based Benders decomposition called a pattern generating Benders approach (PGBA). This is a 4-phase approach in which Phase 1 generates patterns, Phase 2 uses an IP model to find a pattern set from the generated patterns, Phase 3 checks feasibility of the pattern set and assigns teams to placeholders and, finally, Phase 4 generates a timetable using a CP model. The resemblance to Benders decomposition comes from a number of feasibility checks in Phase 3. In case one of these checks prove the pattern set to be infeasible, a logic-based Benders cut is added to the IP model from Phase 2 and the algorithm returns to Phase 2. This iterative process continues until a feasible pattern set has been found or the IP model from Phase 2 becomes infeasible. In the first case, a corresponding timetable is found in Phase 4 and the algorithm stops. In the second case, we return to Phase 1 and generate additional patterns since Phase 1 only generates a subset of the feasible patterns initially. The algorithm continues until an optimal solution has been found or infeasibility has been proved. The computational results show that the PGBA leads to significant reductions in computation times for hard instances. The details of the PGBA will be outlined in Chapter 5.

Subsequently, Rasmussen [70] has used the PGBA to schedule a triple round robin tournament for the best Danish soccer league. In the original presentation of the PGBA, only place constraints were considered but numerous constraints are present in the practical application. These include constraints relating to the timetable, which makes the problem harder to solve since the subproblem becomes an optimization problem instead of a feasibility problem. Therefore not only feasibility cuts but also optimality cuts must be added to the master problem. However, the modified PGBA is able to obtain very good solutions in short time and it has been used for scheduling the 2006/2007 season of the Danish soccer league. Chapter 6 gives a detailed description of this solution method.

Recently, Bartsch et al. [10] have presented a new approach based on renewable resources for solving sports scheduling problems. They consider the problems of scheduling the German and the Austrian soccer leagues. Again various constraints must be taken into account but in contrast to previous methods this is done by

using partially renewable resources. Bartsch et al. present models based on this technique for both the German and the Austrian leagues and they develop a specialized heuristic 3-phase approach for solving the problem. In this approach, Phase 1 generates both a pattern set and a timetable with placeholders, Phase 2 assigns teams to placeholders and Phase 3 determines the exact date for each team since each slot covers more than one day. The approach has been used in practice in both Germany and Austria.

A general approach for using resource-based models are presented by Drexl and Knust [25]. In their paper they show how various constraints can be modelled using resources and they are working on corresponding solution methods.

3.4 Minimizing Travel Distance

The minimization of travel distance becomes relevant when teams travel from one away game to the next without returning home. In this setup huge savings can be obtained when long trips are applied and teams located close together are visited on the same trip.

The interest in minimizing travel distances arose from the increasing travel costs due to the oil crises in the 1970's. This led to a request for efficient solution methods capable of finding good solutions for practical applications and a number of papers on distance minimization has appeared since 1976. However, in 2001 Easton, Nemhauser, and Trick [26] proposed the *traveling tournament problem* and this problem has received most of the attention concerned with minimizing travel distances since then. In the following two sections we will give an outline of the papers applied for practical applications and the papers focusing on the traveling tournament problem, respectively.

3.4.1 Practical Applications

Campbell and Chen [19] presented the first paper considering the problem of scheduling a basketball conference of ten teams. This is a relaxed double round robin tournament and the teams are allowed to play at most two consecutive away games without returning home. To solve the problem, a 2-phase approach is applied. In Phase 1, the optimal trips for each team are found and the authors show that, for a tournament with an even number of teams, it is equivalent to pairing the teams two and two such that the distances between the paired teams are minimized. Figure 3.8 shows why this holds true. Each node in the graph corresponds to a team and we want to minimize the total travel distance for team i. Team i travels at least once from or to all other teams (the dotted edges) and hence these edges can be discarded. Furthermore, the number of trips of length 2 must be maximized when minimizing the travel distance. This means

that the optimal solution corresponds to a pairing of the teams (the remaining edges) which minimizes the total distance between the paired teams. This pairing is independent of the team for which we minimize the travel distance.

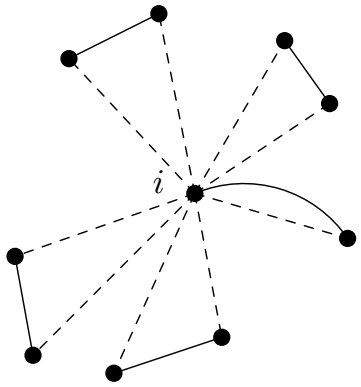

Figure 3.8: Graph showing the optimal trips for team i [19].

In Phase 2, the optimal pairing is translated into a number of feasible sequences using a constructive approach. This approach takes all the constraints into account and the result is a feasible schedule which minimizes the total travel distance.

Ball and Webster [4] solve a similar scheduling problem for a basketball conference in their paper from 1977. They first model the problem using an IP formulation but the problem is too large to solve and, instead, a heuristic solution method very similar to the method by Campbell and Chen is developed.

The same year Cain, Jr. [18] presents a heuristic approach for scheduling major league baseball. The league consists of 12 teams partitioned into two divisions and each team plays 162 games making this problem one of the largest in the literature. Furthermore, a very large number of constraints and considerations are described which makes the problem even harder. The solution method is a constructive approach decomposing the season into three phases. For each phase, a pattern set is generated and, given the pattern set, an optimal timetable is subsequently found by using a computer.

In 1980, Bean and Birge [11] return to a basketball instance since they schedule the tournament for the national basketball association (NBA). As Ball and Webster, they first formulate the problem using IP but again the problem becomes too large to solve in reasonable time. Instead, they use a heuristic 2-phase approach resembling the approaches used by Campbell and Chen and Ball and Webster. However, in this problem the teams are allowed to play five consecutive away games and this relaxation makes the problem substantially harder. Furthermore, a large number of place constraints are present since the venues are used for other purposes. In Phase 1, a heuristic method is used to minimize the travel

distance for each team individually. Due to the longer trips, it is no longer possible to use the "pairing approach" from the earlier methods. In Phase 2, the trips are scheduled one by one starting with the longest travel distance. The trips are scheduled in order to cover most of the home game requests and, in case a trip cannot be scheduled, it is divided into partial trips. After a feasible solution has been obtained, a switching algorithm is applied to improve the solution.

In 1991, Ferland and Fleurent [30] present a support system to help scheduling the National Hockey League (NHL). This is a relaxed tournament with 21 teams, it is divided into 2 conferences and each conference is divided into 2 divisions. The problem contains a number of constraints such as place constraints, restrictions on how often teams can play, restrictions on the minimum time between two games with the same opponents and restrictions on the traveling distances. The problem is modelled mathematically but the size of the problem makes it impossible to solve. Instead, a number of procedures are presented which can be used while the schedule is created manually. After this paper, the NHL decided to expand the league from 21 teams to 24 teams and Fleurent and Ferland [32] presented an IP model for deciding the number of games played between the four divisions.

Russell and Leung [78] considered a baseball league in 1994 with eight teams divided into two divisions. The problem is a compact scheduling problem consisting of three segments: first a double round robin tournament for each division, then a double round robin tournament for the entire league and, finally, another double round robin tournament for each division. They apply a 2-Phase approach generating schedules for placeholders in Phase 1 and assigning teams to placeholders in Phase 2. Due to the structure of the tournament, it is possible to solve Phase 2 using total enumeration within reasonable time and Phase 1 is solved using an exchange heuristic. A feasible schedule is obtained and from this schedule new schedules are obtained by exchanging the slots. Furthermore, the number of consecutive away slots is limited to two, which means that the pairing technique from [4, 19] can be applied. They use this method to obtain a new kind of schedule with more variation compared to the traditional schedule format. However, the new schedule is rejected since it allows byes and it increases travel distance.

In the paper, they note that minimizing travel distance is correlated to maximizing the number of breaks and they prove the following theorem.

Theorem 6 (Russell and Leung [78]) *For a round robin tournament with an even number of teams $n \geq 6$ where each team can play no more than two consecutive home or two consecutive away games, the maximum number of breaks is strictly less than $n(\frac{n}{2} - 1)$.*

The first metaheuristic solution method is applied by Costa [22] in 1995. It is an evolutionary tabu search algorithm combining the mechanisms of genetic algorithms and tabu search and it is used to schedule the NHL also considered

by Ferland and Fleurent [30]. The algorithm consists of three phases which are used repeatedly after initial schedules have been obtained. The initial population of schedules is generated by an algorithm similar to the one used in [30] and the road trips are built sequentially. The reproduction phase assigns a probability for each schedule to be reproduced. The probability is monotonically decreasing with respect to the number of violated constraints. The crossover phase contains the evolutionary part of the algorithm since it generates new schedules from existing schedules and the tabu search face contains a traditional tabu. The neighbourhood of the tabu search consists of all the schedules that can be obtained by moving a single game from one day to another.

Recently, two papers have appeared on minimizing travel distance for a practical application. The first is by Voorhis [91] and once more college basketball is considered. The application is a double round robin tournament with 10 teams allowing trips of length 2 (called *travel swings*). The problem is formulated as an IP model assigning games to slots and it is solved using a depth first branching algorithm. The algorithm starts with assigning trips of length two to slots and afterwards the remaining games are scheduled. For comparison the IP model is also solved using CPLEX but no feasible solutions were obtained within 15 hours of CPU time. In contrast, the developed algorithm found 9 schedules in 1.33 hours of CPU time.

The second paper, by Wright [100], considers the national basketball league of New Zealand. This is a relaxed double round robin tournament with 10 teams and trips of length two are allowed. To solve the problem, a subcost-guided simulated annealing algorithm and the objective function reflects the number of violated requests. The paper gives a thorough comparison of variations of the algorithm and concludes that it is advantageous to keep a certain structure at the beginning of the search but relaxing the structural constraints during the search.

3.4.2 The Traveling Tournament Problem

Easton et al. [26] presented the *traveling tournament problem* (TTP) in 2001. The problem is motivated by the problem of scheduling major league baseball and it is formulated to capture the fundamental difficulties of minimizing the travel distance for a sports league. By using the TTP as benchmark problems, it is possible to develop and compare solution methods which, afterwards, can be specialized for the various constraints present in practical applications. The TTP can be formulated as follows.

Definition 3 (Easton et al. [26]) *The traveling tournament problem is as follows:*

Input: *n, the number of teams; D an n by n integer distance matrix; L, U integer parameters.*

Output: *A double round robin tournament on the n teams such that*
- The number of consecutive home games and consecutive away games are between L and U inclusive, and
- The total distance travelled by the teams is minimized.

Furthermore, two additional requirements are mentioned. The first is a mirroring constraint requiring that the schedule is mirrored and the second is a no-repeater constraint requiring that two teams cannot play two games against each other in two consecutive slots. Notice that at most one of the two requirements is relevant since the no-repeater constraint is always satisfied in a mirrored schedule.

In the paper two instance classes are presented:

Circle instances (circular distance):
An instance of the circular distance TTP with n teams is obtained by generating an n-node circle graph with unit distances (distance of 1 between all adjacent nodes). The distance between two teams i and j with $i > j$ is then equal to the length of the shortest path between i and j and it equals the minimum of $i - j$ and $j - i + n$.

National league instances (NL):
The MLB consists of two leagues called the National League and the American League. In order to create small instances reflecting the actual structure of the MLB the teams of the National League was used to obtain benchmark problems with 4 to 16 teams called NL4, NL6,. . . ,NL16.

Later Ribeiro and Urrutia [76] have presented a third instance class:

Constant distance (CTTP):
The constant distance instances are characterized by a distance of 1 between all teams and Ribeiro and Urrutia [76] show that, for this instance class, minimizing travel distance is equivalent to maximizing the number of breaks.

All the instance classes are presented at [88] together with the current best upper and lower bounds.

Various solution methods have been presented for solving the TTP. Easton et al. [26] present a method based on the *independent lower bound* (IB), which they define to be the sum of the minimum travel distances for each team when they are considered independently. The solution method generates pattern sets with as many trips as possible and a corresponding timetable minimizing the travel distance is found afterwards. In this setup, a strengthening of the IB can be used to check optimality and, as long as this bound is below the best solution, the algorithm continues. This method is able to solve the NL4 and NL6 to optimality.

Benoist, Laburthe, and Rottembourg [13] apply a hybrid algorithm combining Lagrange relaxation and CP. The algorithm has a hierarchical architecture consisting of three components. The main component is a CP model capturing the

entire problem and capable of solving the problem by itself. However, a global constraint is introduced in order to improve the bounds during the search. This global constraint corresponds to the second component and it contains a Lagrange controller using either sub-gradient or modified gradient techniques to adjust the lagrange multipliers for the third component consisting of a perturbated subproblem for each team. The subproblem for a given team i schedules all the games associated with team i such that team i's travel distance is minimized.

Subsequently, Easton, Nemhauser, and Trick [27] present another hybrid IP/CP solution method. This is a branch and price (column generation) algorithm in which the columns correspond to tours for the teams. The master problem is a linear programming problem assigning teams to tours, while the pricing problem for generating tours is a CP problem. A parallel version of the algorithm is implemented and it is to date the only solution method which has been able to prove optimality of an instance of NL8. However, the no-repeater constraint was not imposed which means that the solution value can only be used as a lower bound for the instance found at [88].

The next approach for the TTP was a simulated annealing algorithm by Anagnostopoulos, Michel, Van Hentenryck, and Vergados [1] called TTSA. From an initial schedule found by a simple backtrack search TTSA searches for improving solutions using five kinds of moves: *SwapHomes, SwapRounds, SwapTeams, PartialSwapRounds* and *PartialSwapTeams*. By applying these moves, the structure of the schedule is destroyed but for each move a corresponding ejection chain is able to restore the structure. In this way the algorithm is able to satisfy all hard constraints during the search, whereas the soft constraints may be violated. The hard constraints include the round robin constraints while the no-repeater is considered a soft constraint. The number of violated soft constraints is incorporated in the objective function to force the algorithm towards feasible solutions. TTSA randomly selects a move and it is performed with probability 1 if it leads to an improving solution and otherwise the probability depends on the resulting increase in travel distance plus the current "temperature". The TTSA were able to improve all the current best known upper bounds for the NL instances with more than 10 teams and, in a recent paper by Hentenryck and Vergados [42], the TTSA are further refined to handle mirrored tournaments. In this paper they also use a randomized version of a hill climbing algorithm to obtain better initial schedules.

The first paper focussing solely on mirrored TTP instances is by Urrutia and Ribeiro [90] and they present a heuristic 3-phase approach for generating mirrored schedules quickly. In Phase 1 they first use the canonical schedule to obtain a timetable with placeholders and afterwards they construct a matrix of consecutive opponents. Each entry (i, j) of the matrix gives the number of times another team meets i and j consecutively and this is used in Phase 2 when teams are assigned to placeholders. A simple heuristic assigns teams located close together to placeholders who are met consecutively by many teams. Finally, Phase 3 uses two

steps to obtain a pattern set. In Step 1 a constructive method generates an initial pattern set and afterwards Step 2 performs local search to improve the pattern set. Urrutia and Ribeiro also presents a heuristic method combining GRASP and iterated local search (ILS) which they call *GRILS-mTTP*. The GRILS-mTTP performs a number of iterations all starting with the algorithm explained above for generating an initial schedule. Afterwards, a local search is applied to obtain a locally optimal solution and then GRILS-mTTP iterates between a perturbation procedure and a local search until some re-initialization criterion is satisfied.

Henz [44] proposes to combine large neighbourhood search and CP to overcome the problem of getting away from local optima. He uses five types of moves which all relax a substantial part of the given schedule. For instance the move called *Relax rounds* does not only exchange two slots but it relaxes all variables associated with a number of slots. CP is then applied to obtain a new schedule given the partial schedule which has not been relaxed. In the paper it is noted that only preliminary results have been obtained and they are not competitive to the conventional local search techniques applied earlier.

As mentioned above Ribeiro and Urrutia [76] present the instance class with constant distances and show that minimizing travel distance for these instances is equivalent to maximizing the number of breaks. In the paper they also derive upper bounds on the number of breaks for unconstrained single round robin tournaments, equilibrated single round robin tournaments, unconstrained double round robin tournaments and double round robin tournaments with a maximum of three consecutive home games and three consecutive away games. The limit on consecutive home games and away games in the last kind resembles the bounds from the benchmark TTP instances. By separating these instances into three classes ($(n-1) \bmod 3 = 0$, $(n-1) \bmod 3 = 1$ and $(n-1) \bmod 3 = 2$) the following bounds were obtained.

$$UB_{\text{MTTP}} = \begin{cases} 14, & \text{if } n = 4, \\ 4(n^2-n)/3 - 4n + 20, & \text{if } ((n-1) \bmod 3 = 0 \text{ and } n \neq 4, \\ 4(n^2-2n)/3, & \text{if } ((n-1) \bmod 3 = 1, \\ 4(n^2/3-n), & \text{if } ((n-1) \bmod 3 = 2. \end{cases}$$

The corresponding mirrored constant distance TTP is solved by the GRILS-mTTP presented in [90] and the algorithm is able to solve the instances with 4, 6, 8, 10, 12 and 16 teams to optimality by obtaining solutions which reach the upper bound stated above.

The constant distance TTP were also considered by Rasmussen and Trick [72] who used the PGBA discussed in Section 3.3.2 to solve the problem. They were able to prove optimality for all the mirrored instances with 18 teams or less and all the non-mirrored instances with 16 teams or less by using the algorithm for maximizing breaks instead of minimizing breaks. Hentenryck and Vergados [42] have also used their TTSA approach and improved the best solution for mirrored

instance with 20 teams and the best solutions for the non-mirrored instances with 18-24 teams.

Lim et al. [54] apply a hybrid metaheuristic algorithm combining simulated annealing and hill-climbing for the TTP. After having found an initial schedule using beam search the algorithm iterates between two components for improving the current schedule. The first component searches for improving schedules by using simulated annealing. The moves in this component, called *conditional local jumps*, exchange sets of matches in such a way that all the constraints are satisfied. The second component applies hill-climbing for finding a better team assignment. This is done by means of local exchanges and the algorithm moves in the direction of decreasing travel distance. The fundamental idea of the overall approach is to improve the schedule when a good team assignment has been obtained and to search for a better team assignment when the schedule seems promising. The algorithm continues until no improvements have been obtained for a fixed number of iterations or until a time limit is reached. The computational results show that the algorithm is able to improve the best solutions for all the non-mirrored circular TTP instances with 10 teams or more.

As a generalization of the break minimization problem when distances are considered instead of breaks Rasmussen and Trick [73] define the *timetable constrained distance minimization problem* (TCDMP). The problem is defined as follows:

Definition 4 (Rasmussen and Trick [73]) *Given a timetable for a double round robin tournament with n teams, a distance matrix specifying the distances between the venues and an upper bound UB on the number of consecutive home and consecutive away games, find a feasible pattern set which minimizes the total distance traveled by all teams.*

In the paper four solution methods for the problem are presented and evaluated. The method showing the best performances is a 2-phase hybrid IP/CP approach which generates all feasible patterns in Phase 1 using CP and assigns teams to patterns in Phase 2 using IP. In an extended version of the paper Rasmussen and Trick also present a new heuristic approach called the *circular traveling salesman approach* (CTSA) to solve the TTP. The CTSA first solves the traveling salesman problem containing all the teams in a given tournament. Afterwards an instance of the circular distance TTP is then formulated with the teams ordered according to the TSP solution. To solve the circular distance TTP the solutions obtained by Lim et al. [54] are used and this gives a solution to the original TTP instance. In spite of the simpleness the CTSA is capable of obtaining solutions comparable to the beam search used in [54] for obtaining initial solutions. The four solution methods used to solve the TCDMP are presented in Chapter 7 together with computational results and a more detailed description of the CTSA. In this chapter we also discuss how the TCDMP can be used to enhance the solutions obtained by the CTSA.

Chapter 4

A Benders Approach for Sports Scheduling

In this chapter we consider the constrained minimum break problem for a mirrored double round robin tournament with an even number of teams n. As explained in the previous chapter, this is the problem of finding a schedule which minimizes the number of breaks and at the same time satisfies some additional constraints. We consider tournaments facing place constraints and in order to guarantee acceptable patterns for all teams, we do not allow consecutive breaks.

To solve the problem, we present a hybrid IP/CP algorithm using Benders decomposition. The algorithm uses logic-based Benders cuts to enhance performance, and since the subproblem is discrete, it has been necessary to make some modifications compared to traditional Benders decomposition.

To show the effectiveness of the algorithm, we compare the computation times with two CP models and an IP model. The computational tests show that the hybrid method performs much better than the three basic models and it is capable of solving problems in few seconds, which cannot be solved by any of the other models in 30 minutes.

4.1 The Algorithm

In this chapter we only consider the first half of the tournament, since the second half of the schedule is determined implicitly from the first half in mirrored double round robin tournaments. Furthermore, the problem is decomposed into finding a pattern set and finding a timetable.

The algorithm has a structure like a traditional Benders decomposition with a master problem and a subproblem. The master problem is associated with the pattern set and minimizes the number of breaks while satisfying the place constraints.

The subproblem, on the other hand, searches for a timetable corresponding to the pattern set found in the master problem. In case a feasible solution to the subproblem is found, we have an optimal solution to the problem and the algorithm stops.

Unfortunately, the subproblem cannot be modelled as an LP problem and, without an LP formulation, we are unable to obtain the dual variables needed for generating a traditional Benders cut. Instead, the subproblem is modelled as an IP problem (called (IPS)) and the linear relaxation of this problem (called (LPS)) is then used to obtain Benders cuts. However, when (LPS) is used as the subproblem instead of (IPS), we face the risk of finding a pattern set which makes (LPS) feasible although (IPS) would have been infeasible.

In order to take this scenario into account, we introduce an additional feasibility check. In case the pattern set found by the master problem results in a feasible solution to (LPS), we solve a CP subproblem (called (CPS)). (CPS) is also relaxed compared to (IPS), since the relaxation makes it possible to add logic-based Benders cuts to the master problem in case the pattern set is infeasible.

Finally, we have added a separation procedure capable of generating additional feasibility cuts for the master problem. This procedure checks some basic requirements of the pattern set before (LPS) is solved and it adds cuts to the master problem if the pattern set turns out to be infeasible.

The solution method will be referred to as *CPBD* and it iterates as follows. When a pattern set is found by the master problem, the problem of checking feasibility and finding a feasible game assignment is divided into three levels. At first the separation procedure checks for infeasibility, and logic-based Benders cuts may be added to the master problem. Next (LPS) is solved and in case the pattern set is proved infeasible, a traditional Benders cut is added to the master problem. After (LPS) has been solved, the master problem is re-solved if cuts have been added. Otherwise, (CPS) is solved and the algorithm either stops with a feasible solution or a logic-based Benders cut is added to the master problem. Figure 4.1 illustrates the iteration scheme of CPBD.

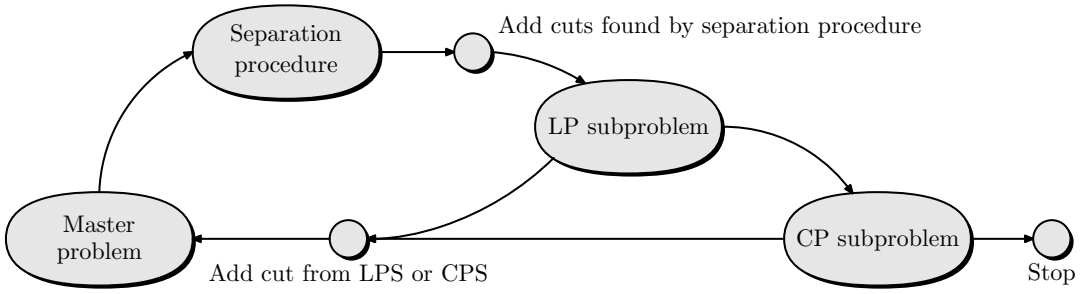

Figure 4.1: Iteration scheme of CPBD.

4.1.1 Master Problem

The master problem is an IP minimization problem finding a pattern set for the first half of the tournament. The objective is to minimize the number of breaks and the solution must satisfy a number of constraints which will be explained beneath the model. In the following we let $T = \{1, \ldots, n\}$ and $S = \{1, \ldots, n-1\}$ denote the sets of teams and slots, respectively. The place constraints are represented by the sets I_i^1 and I_i^0, holding the slots in which team i must play home and away. The binary variable h_{is} is 1 (0) if team i plays home (away) in slot s and the binary variable b_{is} is 1 if team i has a break in slot s. A team cannot have a break in slot 1 but b_{i1} is 1 if team i has a break in the first slot of the second half. LB is a lower bound on the objective value, which helps to reduce the computation time. Initially LB is set to the lower bound $3(n-2)$ stated by de Werra [94] and in the following iterations it is equal to the solution value of the preceding iteration.

$$\min \quad \sum_{i=1}^{n} b_{i1} + \sum_{i=1}^{n} \sum_{s=2}^{n-1} 2b_{is} \tag{4.1.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} b_{i1} + \sum_{i=1}^{n} \sum_{s=2}^{n-1} 2b_{is} \geq LB \tag{4.1.2}$$

$$\sum_{s_1=s-2}^{s} h_{is_1} \leq 2 \qquad \forall i \in T,\ \forall s \in S\ \ s \geq 3 \tag{4.1.3}$$

$$\sum_{s_1=s-2}^{s} h_{is_1} \geq 1 \qquad \forall i \in T,\ \forall s \in S\ \ s \geq 3 \tag{4.1.4}$$

$$h_{i1} + h_{i2} - h_{in-1} \leq 1 \qquad \forall i \in T \tag{4.1.5}$$

$$h_{i1} + h_{i2} - h_{in-1} \geq 0 \qquad \forall i \in T \tag{4.1.6}$$

$$h_{i1} - h_{in-2} - h_{in-1} \leq 0 \qquad \forall i \in T \tag{4.1.7}$$

$$h_{i1} - h_{in-2} - h_{in-1} \geq -1 \qquad \forall i \in T \tag{4.1.8}$$

$$1 - h_{is} - h_{is-1} \leq b_{is} \qquad \forall i \in T,\ \forall s \in S,\ \ s \geq 2 \tag{4.1.9}$$

$$h_{is} + h_{is-1} - 1 \leq b_{is} \qquad \forall i \in T,\ \forall s \in S,\ \ s \geq 2 \tag{4.1.10}$$

$$h_{i1} - h_{in-1} \leq b_{i1} \qquad \forall i \in T \tag{4.1.11}$$

$$-h_{i1} + h_{in-1} \leq b_{i1} \qquad \forall i \in T \tag{4.1.12}$$

$$\sum_{i=1}^{n} h_{is} = n \qquad \forall s \in S \tag{4.1.13}$$

$$\sum_{s=1}^{n-1} h_{is} \leq n - 1 - \left\lfloor \frac{n-1}{3} \right\rfloor \qquad \forall i \in T \tag{4.1.14}$$

$$\sum_{s=1}^{n-1} h_{is} \geq \left\lfloor \frac{n-1}{3} \right\rfloor \qquad \forall i \in T \tag{4.1.15}$$

$$h_{is} = 1 \qquad \forall i \in T,\ \forall s \in I_i^1 \tag{4.1.16}$$

$$h_{is} = 0 \qquad \forall i \in T,\ \forall s \in I_i^0 \tag{4.1.17}$$

$$h_{is},\ b_{is} \in \{0,1\} \qquad \forall i \in T,\ \forall s \in S \tag{4.1.18}$$

The objective function (4.1.1) minimizes the total number of breaks for both halves of the double round robin schedule and constraint (4.1.2) gives a lower bound on the objective value. The constraints (4.1.3) - (4.1.8) limit the number of consecutive home and away games to be no more than two. Notice that constraints (4.1.5) - (4.1.8) are used to limit the number of consecutive home and away games in the transition from the first half to the second half of the tournament. A break variable b_{is} is forced to 1 by the constraints (4.1.9)-(4.1.12) whenever $h_{is-1} = h_{is}$, and again special constraints are used in the transition from the first to the second half of the tournament. The constraints (4.1.13) ensure that half the teams play home in each slot. Since the number of consecutive home and away games are limited to two, we can obtain upper and lower bounds on the number of home games for each team. These limits are presented in constraints (4.1.14) and (4.1.15). Constraints (4.1.16) and (4.1.17) state the place constraints.

The master problem is solved in each iteration and the current solution will be denoted $\bar{h}$ in the following. Furthermore, we update the two index sets $J_1 = \{(i,s) \in T \times S | \bar{h}_{is} = 1\}$ and $J_0 = \{(i,s) \in T \times S | \bar{h}_{is} = 0\}$ in each iteration.

4.1.2 Separation Procedure

In order to check the feasibility of a given pattern set $\bar{h}$, we use the necessary condition (3.3.1) stated by Miyashiro et al. [64]. Since each subset of teams $\hat{T}$ must play exactly $\frac{|\hat{T}|(|\hat{T}|-1)}{2}$ mutual games, it must also satisfy the condition

$$\sum_{s=1}^{n-1} \left(\min \left\{ \sum_{i \in \hat{T}} \bar{h}_{is}, \sum_{i \in \hat{T}} (1 - \bar{h}_{is}) \right\} \right) \geq \frac{|\hat{T}|(|\hat{T}|-1)}{2} \tag{4.1.19}$$

where $\sum_{i \in \hat{T}} \bar{h}_{is}$ and $\sum_{i \in \hat{T}} (1 - \bar{h}_{is})$ give the number of home games and the number of away games played by teams from $\hat{T}$ in slot s.

In case a subset of teams $\hat{T}$ violates (4.1.19) by a value v, the following two

logic-based Benders cuts can be added to the master problem.

$$\sum_{(i,s)\in J_1^{\hat{T}}} (1 - h_{is}) + \sum_{(i,s)\in J_0^{\hat{T}}} h_{is} \geq v \tag{4.1.20}$$

$$\sum_{(i,s)\in J_1^{\hat{T}}} h_{is} + \sum_{(i,s)\in J_0^{\hat{T}}} (1 - h_{is}) \geq v \tag{4.1.21}$$

where $J_1^{\hat{T}} = \{(i,s) \in \hat{T} \times S | \bar{h}_{is} = 1\}$ and $J_0^{\hat{T}} = \{(i,s) \in \hat{T} \times S | \bar{h}_{is} = 0\}$.

The cut (4.1.20) makes sure that, if (4.1.19) is violated by v, then at least v changes are made in the patterns corresponding to $\hat{T}$ in the following pattern sets. Due to symmetry, (4.1.21) gives a valid upper bound on the number of changes.

However, since the number of subsets is exponential, it is not practical to check all subsets. Instead, the separation procedure checks all subsets with cardinality two and adds cuts for all subsets violating (4.1.19). If cuts have been found, the procedure stops. Otherwise, subsets with cardinality three are checked but now the procedure stops when the first cut is found. As long as no cuts are found, the procedure continues to increase the cardinality of the subsets until it exceeds a parameter *subsetLimit*. When this happens the procedure stops.

4.1.3 Benders Cuts

The problem of finding a feasible timetable (IPS) can be formulated as the following IP problem where the binary variable $x_{i_1 i_2 s}$ is 1 if team i_1 plays home against team i_2 in slot s and 0 otherwise.

$$\text{min } 0 \tag{4.1.22}$$

$$\text{s.t. } x_{iis} = 0 \qquad \forall i \in T,\ \forall s \in S \tag{4.1.23}$$

$$\sum_{i_2=1}^{n} (x_{i_1 i_2 s} + x_{i_2 i_1 s}) = 1 \qquad \forall i_1 \in T,\ \forall s \in S \tag{4.1.24}$$

$$\sum_{s=1}^{n-1} (x_{i_1 i_2 s} + x_{i_2 i_1 s}) = 1 \qquad \forall i_1, i_2 \in T,\ i_1 < i_2 \tag{4.1.25}$$

$$\sum_{i_2=1}^{n} x_{i_1 i_2 s} = \bar{h}_{i_1 s} \quad \forall (i_1, s) \in J_1 \tag{4.1.26}$$

$$\sum_{i_2=1}^{n} x_{i_1 i_2 s} = \bar{h}_{i_1 s} \quad \forall (i_1, s) \in J_0 \tag{4.1.27}$$

$$x_{i_1 i_2 s} \in \{0, 1\} \qquad \forall i_1, i_2 \in T,\ \forall s \in S \tag{4.1.28}$$

This is a feasibility problem and hence we minimize a constant objective. Constraints (4.1.23) restrict teams from playing against themselves and constraints

(4.1.24) and (4.1.25) require that each team plays against exactly one opponent in each slot and meets all other teams once. The two constraints (4.1.26) and (4.1.27) link the timetable to the given pattern set $\bar{h}$ by restricting the teams to play according to the pattern set.

To obtain a Benders cut in case no feasible timetable exists, we use the linear relaxation of (IPS). Furthermore, we relax the problem by adding an artificial 0-1 variable a_{is} to the sum in constraint (4.1.26) and subtracting an artificial 0-1 variable a_{is} from the sum in constraint (4.1.27). These variables make sure that the problem is feasible. When a_{is} is set to 1, it corresponds to changing the home-away assignment of team i in slot s. In this way the entire pattern set can be changed but a penalty is incurred every time a change is made. This gives the following LP problem (LPS) where the u's to the right denote the dual variables.

$$\min \quad \sum_{i=1}^{n}\sum_{s=1}^{n-1} a_{is} \tag{4.1.29}$$

$$\text{s.t.} \quad x_{iis} = 0 \qquad \forall i \in T,\ \forall s \in S \qquad u^1_{is} \tag{4.1.30}$$

$$\sum_{i_2=1}^{n}(x_{i_1 i_2 s} + x_{i_2 i_1 s}) = 1 \qquad \forall i_1 \in T,\ \forall s \in S \qquad u^2_{i_1 s} \tag{4.1.31}$$

$$\sum_{s=1}^{n-1}(x_{i_1 i_2 s} + x_{i_2 i_1 s}) = 1 \qquad \forall i_1, i_2 \in T,\ i_1 < i_2 \qquad u^3_{i_1 i_2} \tag{4.1.32}$$

$$\sum_{i_2=1}^{n} x_{i_1 i_2 s} + a_{i_1 s} = \bar{h}_{i_1 s} \qquad \forall (i_1, s) \in J_1 \qquad u^4_{i_1 s} \tag{4.1.33}$$

$$\sum_{i_2=1}^{n} x_{i_1 i_2 s} - a_{i_1 s} = \bar{h}_{i_1 s} \qquad \forall (i_1, s) \in J_0 \qquad u^5_{i_1 s} \tag{4.1.34}$$

$$x_{i_1 i_2 s},\ a_{i_1 s} \in \mathbb{R}_+ \qquad \forall i_1, i_2 \in T,\ \forall s \in S \tag{4.1.35}$$

The optimal objective value of the dual problem to (LPS) is:

$$\sum_{i=1}^{n}\sum_{s=1}^{n-1} u^{2*}_{is} + \sum_{i_1=1}^{n-1}\sum_{i_2=i_1+1}^{n} u^{3*}_{i_1 i_2} + \sum_{(i,s)\in J_1} \bar{h}_{is} u^{4*}_{is} + \sum_{(i,s)\in J_0} \bar{h}_{is} u^{5*}_{is}$$

where u^{1*}, u^{2*}, u^{3*}, u^{4*} and u^{5*} denote the optimal dual variables.

(LPS) is always feasible and the optimal solution value of (LPS) is therefore equal to the optimal solution value of the dual problem. This means that the optimal value of the dual problem must be zero whenever a game assignment exists. However, since the pattern set $\bar{h}$ does not appear in the constraints of the dual problem, u^{1*}, u^{2*}, u^{3*}, u^{4*} and u^{5*} are always feasible in the dual problem no

matter which pattern set is chosen. A pattern set h for which a game assignment exists must therefore satisfy the Benders cut:

$$\sum_{i=1}^{n}\sum_{s=1}^{n-1} u_{is}^{2*} + \sum_{i_1=1}^{n-1}\sum_{i_2=i_1+1}^{n} u_{i_1 i_2}^{3*} + \sum_{(i,s)\in J_1} h_{is} u_{is}^{4*} + \sum_{(i,s)\in J_0} h_{is} u_{is}^{5*} \leq 0$$

This cut is added to the master problem whenever the optimal solution value of (LPS) is greater than zero.

4.1.4 CP Subproblem

When a pattern set satisfies all the constraints from the separation procedure and leads to an optimal solution value of zero in (LPS), we use a CP problem (CPS) to find a game assignment if one exists. Otherwise, a cut is added to the master problem.

(CPS) could be formulated as a feasibility problem which was feasible when a game assignment existed and infeasible otherwise. However, this approach would lead to a cut, which would only cut off the current pattern set from the master problem and we are searching for a stronger cut. Instead a relaxation is used which minimizes the number of violations. If the optimal solution value is equal to zero, a feasible game assignment has been found and otherwise a cut is added to the master problem.

To formulate the problem we use the opponent variables o_{is} which give the opponent of team i in slot s, and to relax the problem the artificial variables a_{is} are used. The basic constraints of (CPS) are from Henz et al. [46].

$$\min \sum_{i=1}^{n}\sum_{s=1}^{n-1} a_{is} \tag{4.1.36}$$

$$\text{s.t.} \quad o_{is} \neq i \qquad \forall i \in T,\ \forall s \in S \tag{4.1.37}$$

$$\mathit{alldifferent}(o_{i1}, \ldots, o_{in-1}) \qquad \forall i \in T \tag{4.1.38}$$

$$\mathit{alldifferent}(o_{1s}, \ldots, o_{ns}) \qquad \forall s \in S \tag{4.1.39}$$

$$(o_{i_1 s} = i_2) \Leftrightarrow (o_{i_2 s} = i_1) \qquad \forall i_1, i_2 \in T,\ i_1 < i_2,\ \forall s \in S \tag{4.1.40}$$

$$(\bar{h}_{i_1 s} = \bar{h}_{i_2 s}) \wedge (o_{i_1 s} = i_2) \Rightarrow (a_{i_1 s} = 1) \vee (a_{i_2 s} = 1) \qquad \forall i_1, i_2 \in T,\ i_1 < i_2,\ \forall s \in S \tag{4.1.41}$$

$$\sum_{i:(i,s)\in J_1} a_{is} - \sum_{i:(i,s)\in J_0} a_{is} = 0 \qquad \forall s \in S \tag{4.1.42}$$

$$o_{is} \in T,\ a_{is} \in \{0,1\} \qquad \forall i \in T,\ \forall s \in S \tag{4.1.43}$$

In this model the objective function (4.1.36) minimizes the sum of the artificial variables. The constraints (4.1.37) state that a team cannot meet itself and con-

straints (4.1.38) make sure that each team meets all other teams. Constraints (4.1.39) require that all teams meet different opponents in each slot and constraints (4.1.40) make sure that teams meet pairwise. The last two constraints involve the artificial constraints. Constraints (4.1.41) force an artificial variable to one if two opponents both play home or both play away and constraints (4.1.42) state that the number of positive artificial variables in a given slot corresponding to home games must equal the number of variables corresponding to away games.

Since the objective function minimizes the sum of the artificial variables, the optimal objective value can be used as a lower bound on the number of changes needed in the pattern set to make it feasible.

Lemma 2 *Given a pattern set $\bar{h}$, then the optimal objective value of (CPS) is a lower bound on the number of $\bar{h}_{is}$ variables which need to change to obtain a feasible game assignment.*

Proof. Let (o^*, a^*) be an optimal solution to (CPS) corresponding to pattern set $\bar{h}$ and assume that a pattern set $\hat{h}$ with a feasible game assignment $\hat{o}$ exists for which

$$\sum_{i=1}^{n}\sum_{s=1}^{n-1} |\bar{h}_{is} - \hat{h}_{is}| < \sum_{i=1}^{n}\sum_{s=1}^{n-1} a^*_{is}$$

We show that this leads to a contradiction. Let $\hat{a}_{is} = |\bar{h}_{is} - \hat{h}_{is}| \ \ \forall i \in T, \ \ \forall s \in S$ and notice that

$$\hat{h}_{is} = \begin{cases} \bar{h}_{is} - \hat{a}_{is} & \text{if } (i,j) \in J_1 \\ \bar{h}_{is} + \hat{a}_{is} & \text{if } (i,j) \in J_0 \end{cases}$$

This means that $(\hat{o}, \hat{a})$ is a solution to (CPS) corresponding to the pattern set $\bar{h}$ since $\hat{a}$ transforms $\bar{h}$ to $\hat{h}$ for which $\hat{o}$ is a solution. However, this leads to a contradiction since (o^*, a^*) was assumed to be an optimal solution and

$$\sum_{i=1}^{n}\sum_{s=1}^{n-1} \hat{a}_{is} < \sum_{i=1}^{n}\sum_{s=1}^{n-1} a^*_{is}$$ □

The following theorem gives two cuts which can be added to the master problem in case (CPS) has an optimal solution greater than zero.

Theorem 7 *Let (o^*, a^*) be an optimal solution to (CPS) corresponding to the pattern set $\bar{h}$. Then the following constraints can be added to the master problem*

$$\sum_{(i,s)\in J_1} h_{is} - \sum_{(i,s)\in J_0} h_{is} \leq n(n-1) - \sum_{i=1}^{n}\sum_{s=1}^{n-1} a^*_{is} \tag{4.1.44}$$

$$\sum_{(i,s)\in J_0} h_{is} - \sum_{(i,s)\in J_1} h_{is} \leq n(n-1) - \sum_{i=1}^{n}\sum_{s=1}^{n-1} a^*_{is} \tag{4.1.45}$$

where $J_0 = \{(i,s)|\bar{h}_{is} = 0\}$, $J_1 = \{(i,s)|\bar{h}_{is} = 1\}$.

Proof. From Lemma 2 we know that at least $\sum_{is} a^*_{is}$ of the h variables must be changed compared to $\bar{h}$ in order to obtain a feasible game assignment. This gives the constraint

$$\sum_{(i,s)\in J_1} (1 - h_{is}) + \sum_{(i,s)\in J_0} h_{is} \geq \sum_{i=1}^{n} \sum_{s=1}^{n-1} a^*_{is}$$

which leads to (4.1.44). The constraint (4.1.45) is valid due to the symmetry of the problem. □

4.1.5 Cut Pool

To avoid redundant cuts in the master problem, we use a cut pool. A cut is moved from the master problem to the cut pool if the number of consecutive non-binding iterations exceed a parameter *iterBeforePool*. On the other hand, if a solution obtained in the master problem violates a cut from the cut pool, then the cut is added to the master problem again and it is re-solved.

4.2 Computational Results

In order to examine the efficiency of the solution method, we compare the performance of CPBD with the performance of two CP models and one IP model.

- The first CP model (CP1) corresponds to a combination of the master problem and (CPS) apart from minor modifications implemented to use the extra options available in CP.
- The second CP model (CP2) has variables for all $2(2n-1)$ time slots. Again the home-away patterns and breaks of each team are determined by the variables h_{is} and b_{is} but the game assignment is determined by *venue variables* v_{is}. These variables determine the venue at which team i plays in slot s.
- The IP model (IP) corresponds to a combination of the master problem and (IPS).

All tests have been performed on an Intel Xeon 2.67 GHz processor with 4 GB RAM. All models are implemented in OPL Studio by ILOG [52] using default settings and all computation times are reported in seconds. We use a time limit of 1800 seconds.

The solution method has been tested on several instances with and without place constraints and with varying number of teams. To get a general idea of the efficiency when place constraints are present, 10 random instances have been generated for each number of teams and the average solution time is reported.

Table 4.1: Computation times for instances without place constraints.

	Solution time (s)			
$2n$	CP1	CP2	IP	CPBD
4	0.01	0.01	0.04	0.10
6	0.01	0.32	0.35	0.72
8	1.46	–	13.29	2.76
10	–	–	–	26.55
12	–	–	–	24.79
14	–	–	–	–

Table 4.2: Average computation times for instances with 5 place constraints.

	Problems solved				Average solution time (s)			
$2n$	CP1	CP2	IP	CPBD	CP1	CP2	IP	CPBD
4	10	10	10	10	0.00	0.00	0.00	0.00
6	10	10	10	10	0.11	0.28	0.31	0.06
8	10	8	10	10	219.12	127.87	11.29	0.83
10	0	0	6	9	–	–	244.06	34.26
12	0	0	0	3	–	–	–	508.69

Table 4.1 gives the solution times for instances without place constraints and the number of teams ranging from 4 to 14. Instances which cannot be solved within the time limit are marked with –. Since consecutive breaks are not allowed, all of the instances with only 4 teams are infeasible. In the table we see that CPBD is much faster than the CP and IP models when 10 and 12 teams are considered but the instance with 14 teams is intractable to all 4 methods.

Tables 4.2 and 4.3 present the results for instances with 5 and 15 place constraints respectively. Since 10 instances are solved for each row, we report the number of instances solved (or proved infeasible) and the average time used for these instances. Instances which could not be solved or proved infeasible within the time limit are not included in the average. Again we see that the computation times are quite similar for the small instances but CPBD is much faster at instances with 10 and 12 teams and it is able to solve instances unsolvable for any of the other methods.

Notice that proving infeasibility is often faster than solving the feasible instances. An example of this is seen in Table 4.3 where the two CP models seem to be very fast at solving instances with 10 teams. However, all they do is proving infeasibility of a single instance.

Table 4.3: Average computation times for instances with 15 place constraints.

	Problems solved				Average solution time (s)			
$2n$	CP1	CP2	IP	CPBD	CP1	CP2	IP	CPBD
4	10	10	10	10	0.00	0.01	0.00	0.00
6	10	10	10	10	0.00	0.01	0.01	0.01
8	8	8	10	10	133.06	125.58	1.30	0.18
10	1	1	8	10	0.01	0.02	444.57	31.73
12	0	0	0	4	–	–	–	479.86

Chapter 5

The Pattern Generating Benders Approach

Although the CPBD algorithm from Chapter 4 performs significantly better than basic CP and IP models, stronger methods are needed to handle practical applications. In this chapter we show how the underlying idea of the CPBD algorithm combined with pattern generation can lead to a much faster solution method.

Inspired by the work of Hooker and Ottosson [50], we present a hybrid IP/CP approach using logic-based Benders decomposition. The method adopts the decomposition steps used in earlier papers (see Section 3.3.2) but it contains two essential differences compared to the previous approaches. It obtains speedups by limiting the number of patterns generated initially and it reduces the number of infeasible pattern sets found in the master problem by using logic-based Benders cuts. The approach will be referred to as the *pattern generating Benders approach* (PGBA).

Computational tests show that the PGBA leads to substantial speedups for most instances when compared to the three-phase approach used by Henz [45]. The speedups are small for small instances but increase with the number of teams. These speedups make it possible to consider non-mirrored tournaments of realistic size.

We extend this work to the circular distance TTP (CTTP) discussed in Section 3.4.2. Urrutia and Ribeiro [90] show that, for the CTTP, the problem of minimizing the total travel distance is equivalent to maximizing the number of breaks. With some modifications of the PGBA, it is possible to solve a number of previously unsolved benchmark problems for CTTP to optimality.

5.1 Problem Formulation

As in Chapter 4 we consider the constrained minimum break problem for a double round robin tournament with an even number of teams n. Again, the tournament must satisfy place constraints and consecutive breaks are not allowed but, in contrast to Chapter 4, we extend the problem to include non-mirrored schedules. Non-mirrored schedules are sometimes preferred for highly constrained practical applications since their flexibility makes them capable of satisfying more constraints than mirrored schedules. However, when non-mirrored schedules are considered, we need to add an additional constraint saying that two games with the same opponents must be separated by at least k time slots for some given value of k.

The sets of teams and time slots are denoted T and S respectively and I_i^1 and I_i^0 hold the slots in which team i must play home and away due to place constraints. To formulate the problem we introduce the following variables.

$$h_{is} = \begin{cases} 1 \text{ if team } i \text{ plays home in slot } s \\ 0 \text{ else} \end{cases}$$

$$b_{is} = \begin{cases} 1 \text{ if team } i \text{ has a break in slot } s\text{: } h_{is-1} = h_{is} \\ 0 \text{ else} \end{cases}$$

$x_{i_1 i_2} \in S$ gives the slot in which team i_2 visits team i_1.

Since we consider non-mirrored tournaments, we have to schedule both halves of the tournament and this problem can be modelled as the following CP problem referred to as (CPP).

$$\min \sum_{i \in T} \sum_{s \in S} b_{is} \tag{5.1.1}$$

$$\text{s.t. } \mathit{sequence}(1, 2, 3, \mathit{all}(s \in S)\ h_{is}, 1, n-1) \qquad i \in T \tag{5.1.2}$$

$$(b_{is} = 1) \Leftrightarrow (h_{is-1} = h_{is}) \qquad i \in T,\ s \in S \setminus \{1\} \tag{5.1.3}$$

$$b_{i1} = 0 \qquad i \in T \tag{5.1.4}$$

$$\sum_{i \in T} h_{is} = \frac{n}{2} \qquad s \in S \tag{5.1.5}$$

$$h_{is} = 1 \qquad i \in T,\ s \in I_i^1 \tag{5.1.6}$$

$$h_{is} = 0 \qquad i \in T,\ s \in I_i^0 \tag{5.1.7}$$

$$(h_{is} = 0) \vee (h_{js} = 1) \Rightarrow (x_{ij} \neq s) \qquad i, j \in T,\ i \neq j \tag{5.1.8}$$

$$\mathit{alldifferent}(\mathit{all}(j \in T \setminus i)\ x_{ij}, \mathit{all}(j \in T \setminus i)\ x_{ji}) \qquad i \in T \tag{5.1.9}$$

$$(x_{ij} - x_{ji} < -k) \vee (x_{ij} - x_{ji} > k) \qquad i, j \in T,\ i < j \qquad (5.1.10)$$

$$h_{is},\ b_{is} \in (0, 1) \qquad i \in T,\ s \in S \qquad (5.1.11)$$

$$x_{ij} \in S \qquad i, j \in T,\ i \neq j \qquad (5.1.12)$$

The sequence constraints (5.1.2) require that each team plays exactly $n-1$ home games and has a pattern without consecutive breaks since 3 consecutive slots must contain at least 1 and at most 2 home games. Constraints (5.1.3) define a break and since breaks are impossible in the first slot, constraints (5.1.4) set all break variables for slot 1 to zero. Constraints (5.1.5) state that in each time slot, exactly n teams play home and constraints (5.1.6) and (5.1.7) make sure that the place requirements are satisfied. Constraints (5.1.8) make sure that team j does not visit team i in a slot where team i plays away or team j plays home. The *alldifferent* constraints (5.1.9) make sure that a team does not play more than a single game in each time slot. Finally, constraints (5.1.10) separate two games played by the same teams with at least k time slots.

Notice, that the constraints (5.1.2)-(5.1.4) are all constraints for individual patterns, (5.1.5) is a constraint for the set of patterns and (5.1.8)-(5.1.10) consider assignments of games to the pattern set. This partitioning will be used in the solution method.

5.2 Methodology

To solve the problem defined in Section 5.1, we present a pattern generating Benders approach. The approach decomposes the problem into the following four components: generate feasible patterns, find pattern sets from the patterns generated, check feasibility of the pattern sets, and find a timetable with a feasible team allocation. However, instead of solving these components one by one, the algorithm iterates between the four components.

Contrary to previous approaches, the PGBA only generates patterns with a small number of breaks at first. Then, from among these patterns, a minimization problem (the master problem) finds a pattern set with a minimal number of breaks. This pattern set is a good candidate for an optimal solution if a corresponding timetable and a corresponding team allocation exist. The check for optimality is discussed in Section 5.6.

Furthermore, the PGBA introduces the third component for checking feasibility of the pattern sets. The component heuristically determines infeasibility and, when successful, adds logic-based Benders cuts to the master problem. A number of models are used to perform this check and they are presented in Section 5.5.

The fourth component is used to find a timetable and a team allocation for pattern sets which have not been proved infeasible. If the pattern set turns out to

be infeasible anyway, a Benders cut is added to the master problem. Otherwise, a feasible solution is found and optimality must be proved.

In case the master problem is infeasible the algorithm tries to generate additional patterns. If extra patterns exist the algorithm continues. Otherwise an optimal solution has been found or infeasibility of the problem has been proved and the algorithm stops. The chart in Figure 5.1 gives an overview of how the algorithm works.

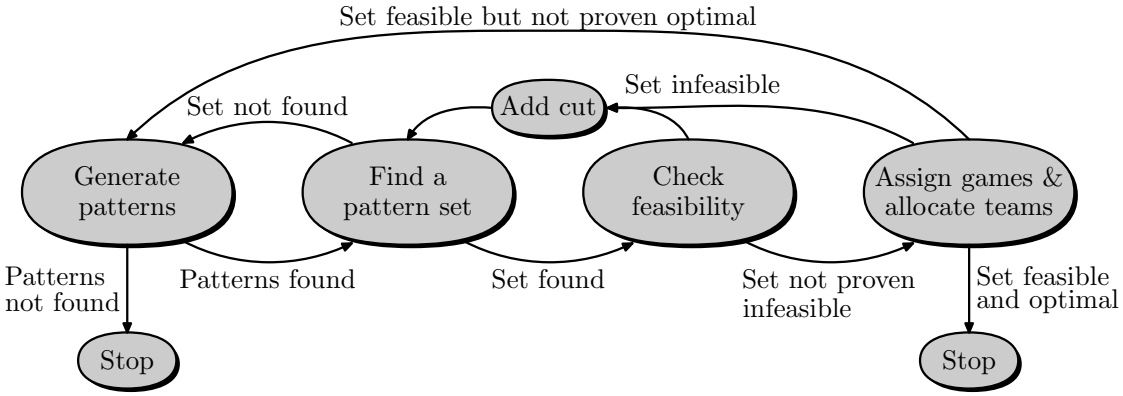

Figure 5.1: Overview of the algorithm.

The algorithm is applicable for both mirrored and non-mirrored schedules but some of the models are modified according to the given problem. Both IP and CP models are used in the algorithm since the decomposition makes it possible to use IP for the optimization problems and CP for feasibility problems. The models are outlined in the following three sections and a discussion of each of the four components from Figure 5.1 is given in Section 5.6.

5.3 Generating Patterns

A CP model is used for generating patterns. The model generates all patterns which begin with an away game, satisfy the constraint (5.1.2) and have exactly c breaks, where c is a parameter given to the model. Notice that, for each pattern found by the model, the complementary pattern, where home and away games are reversed, is available as well.

By using the variables h_s for all $s \in S$, where h_s equals 1 (0) if the pattern has a home (away) game in slot s, the CP model referred to as (PM) looks as follows:

$$sequence(1, 2, 3, all(s \in S)\ \ h_s, 1, n-1) \tag{5.3.1}$$

$$h_1 = 0 \tag{5.3.2}$$

$$\sum_{s=1}^{|S|-1} (h_s = h_{s+1}) = c \tag{5.3.3}$$

$$h_s + h_{s+n-1} = 1 \qquad s = 1, \ldots, n-1 \tag{5.3.4}$$

$$h_s \in (0, 1) \qquad s \in S \tag{5.3.5}$$

Constraint (5.3.1) corresponds to (5.1.2) from (CPP) presented in Section 5.1. Constraint (5.3.2) restricts the search to patterns which begin with an away game and constraint (5.3.3) limits the search to patterns with c breaks. Constraints (5.3.4) require that the pattern is mirrored and is only used when a mirrored tournament is considered.

5.4 Pattern Sets

Given a set of patterns found by (PM) we want to find a subset of n patterns with a minimal number of breaks. Since all the patterns satisfy the constraint (5.1.2), and the game assignment and team allocation are postponed, the only constraint from (CPP) which should be considered is (5.1.5). However, restricting the model to find pattern sets for which all teams can be allocated to at least one pattern based on the place constraints helps to avoid obviously infeasible sets. Finally a lower and an upper bound (LB & UB) on the number of breaks are added. The lower bound is used to reduce computation time while the upper bound is used to prove optimality when a feasible schedule has been found. The adjustments of these bounds will be described in Section 5.6.

The set of generated patterns is denoted P and a binary variable p for all j in P is used to determine whether pattern j is included in the subset ($p_j = 1$) or not ($p_j = 0$). c_j denotes the number of breaks for pattern j and h_{js} is a parameter telling whether pattern j plays home ($h_{js} = 1$) or away ($h_{js} = 0$) in slot s. The set $P_i = \{j \in P | h_{js} = 1 \ \forall s \in I_i^1 \wedge h_{js} = 0 \ \forall s \in I_i^0\}$ is used to store the patterns which satisfy all place constraints of team i.

Since the problem of finding pattern sets is a minimization problem, the following IP model called (PSM) is used.

$$\min \sum_{j \in P} c_j p_j \tag{5.4.1}$$

$$\text{s.t.} \sum_{j \in P} p_j \qquad = n \tag{5.4.2}$$

$$\sum_{j \in P} h_{js} p_j \quad = \frac{n}{2} \qquad s \in S \tag{5.4.3}$$

$$\sum_{j \in P_i} p_j \qquad \geq 1 \qquad i \in T \tag{5.4.4}$$

$$\sum_{j \in P} c_j p_j \quad \geq LB \tag{5.4.5}$$

$$\sum_{j \in P} c_j p_j \quad \leq UB \tag{5.4.6}$$

$$p_j \in (0,1) \qquad\qquad j \in P \tag{5.4.7}$$

Constraint (5.4.2) makes sure that exactly n patterns are chosen, constraints (5.4.3) correspond to (5.1.5), constraints (5.4.4) require that all teams can be allocated to at least one pattern and constraints (5.4.5) and (5.4.6) enforce a lower and an upper bound on the number of breaks.

5.5 Feasibility Check and Benders Cuts

A pattern set found by (PSM) is feasible if all games can be assigned to slots and teams can be allocated to patterns. If this is not the case, a Benders cut is added to (PSM) to cut off the current and similar solutions. However, the Benders cuts used in this method are not obtained from a linear subproblem as in traditional Benders decomposition. Instead a logic-based Benders decomposition is used, as defined by Hooker and Ottosson [50], where the Benders cuts are obtained from inference duals. When the subproblem is a feasibility problem, the inference dual is a condition which, when satisfied, implies that the master problem is infeasible. This condition can then be used to obtain a Benders cut for cutting off infeasible solutions.

In this section we give a necessary and sufficient condition for a team allocation to exist and necessary conditions for a game assignment to exist. Furthermore, we present a logic-based Benders cut for each of the conditions. The pattern set which is currently checked is denoted P^C.

5.5.1 Team Allocation

Let $G = (A, B)$ be a bipartite graph where the node sets A and B correspond to the set of teams and the set of patterns respectively. Furthermore, connect node $i \in A$ and node $j \in B$ by an edge if $j \in P_i \cap P^C$. The allocation of teams to patterns corresponds to a matching between the two node sets A and B from G. Figure 5.2 gives an example of such a bipartite graph. This means that Hall's theorem gives a necessary and sufficient condition for a team allocation to exist.

Theorem 8 (Hall's theorem [55]) *Let $G = (A, B)$ be a bipartite graph. Then G has a matching of A into B if and only if $|\Gamma(X)| \geq |X|$ for all $X \subseteq A$.*

$\Gamma(X)$ is the set of nodes from B incident to a node in X and in our case this means that $\Gamma(X) = \cup_{i \in X}(P_i \cap P^C)$. We can use the Hungarian method to find a

	Patterns			
	1	2	3	4
P_1:	1	2		
P_2:	1			
P_3:	1		3	4
P_4:		2		

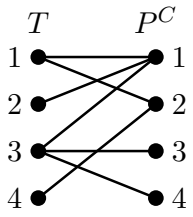

Figure 5.2: Example of bipartite graph used to find a team allocation.

set of teams X which violates the condition from Hall's theorem if any such set exists. Otherwise the Hungarian method finds a feasible team allocation.

If a violating set of teams X is found the following Benders cut can be used.

$$\sum_{j \in \cup_{i \in X} P_i} p_j \geq |X| \tag{5.5.1}$$

5.5.2 Diversity of Patterns

In case a subset of patterns from the current pattern set P^C is very similar it might be difficult or even impossible to schedule the mutual games between these patterns. In this case we use the necessary condition by Miyashiro et al. [64] stating that all subsets of patterns $\bar{P}$ must satisfy

$$\sum_{s \in S} \left(\min \left\{ \sum_{j \in \bar{P}} h_{js}, \sum_{j \in \bar{P}} (1 - h_{js}) \right\} \right) \geq |\bar{P}|(|\bar{P}| - 1) \tag{5.5.2}$$

where parameter h_{js} tells whether pattern j plays home or away in slot s. Notice that this condition is modified to be valid for a double round robin tournament.

Instead of checking each subset, we formulate an IP model called (UBM) which, for a given subset size Z, finds the subset of teams with the smallest left hand side in (5.5.2). To formulate this problem a variable α_j for each j in P^C is used to determine whether pattern j is in the subset ($\alpha_j = 1$) or not ($\alpha_j = 0$). A variable δ_s for each s in S is set to one (zero) if the home (away) games are counted in slot s and a variable β_s for each s in S counts the number of home (away) games from slot s.

$$\min \sum_{s \in S} \beta_s \tag{5.5.3}$$

$$\text{s.t.} \sum_{j \in P^C} \alpha_j = Z \tag{5.5.4}$$

$$\beta_s - \sum_{j \in P^C} h_{js}\alpha_j + Z(1-\delta_s) \geq 0 \qquad s \in S \tag{5.5.5}$$

$$\beta_s - \sum_{j \in P^C} (1-h_{js})\alpha_j + Z\delta_s \geq 0 \qquad s \in S \tag{5.5.6}$$

$$\alpha_j,\ \delta_s \in (0,1) \qquad j \in P^C,\ s \in S \tag{5.5.7}$$

$$\beta_s \in \mathbb{R}_+ \qquad s \in S \tag{5.5.8}$$

In this problem constraint (5.5.4) ensures that exactly Z patterns are chosen and constraints (5.5.5) and (5.5.6) make sure that the number of home games are counted in slot s if $\delta_s = 1$ and away games are counted if $\delta_s = 0$.

(UBM) can be used to formulate a necessary condition for P^C to be feasible since (5.5.2) is satisfied for all subsets of size Z if and only if the optimal solution of (UBM) is no less than $Z(Z-1)$.

The pattern diversity condition: *Given a pattern set P^C and a subset size Z then the pattern set is feasible only if the optimal solution of (UBM) is no less than $Z(Z-1)$.*

If this condition is violated the Benders cut

$$\sum_{\substack{j \in P: \\ \alpha_j = 1}} p_j \leq Z - 1 \tag{5.5.9}$$

can be added to (PSM).

When the tournament is mirrored, (UBM) can be modified to only consider the first half of the slots. In that case (5.5.9) can be added to (PSM) if the objective of (UBM) is strictly less than $\frac{Z(Z-1)}{2}$.

5.5.3 Game Separation

When a non-mirrored tournament with $k > 0$ is considered two additional necessary conditions can be stated. One for subsets of patterns with cardinality two and one for subsets with cardinality greater than two.

Subsets of Two Patterns.

Since $k > 0$ the pattern set might contain two patterns without possibility for playing the two required games. See Figure 5.3 for an example when $k \geq 2$.

By letting the parameters $s^f_{j_1 j_2}$ and $s^l_{j_1 j_2}$ denote the first respectively last slot in which j_2 can visit j_1 the following lemma can be established.

Lemma 3 *The two games between patterns j_1 and j_2 can be assigned to two slots in S separated by k slots if and only if*

$$(s^l_{j_1 j_2} - s^f_{j_2 j_1} > k) \vee (s^l_{j_2 j_1} - s^f_{j_1 j_2} > k). \tag{5.5.10}$$

Slots:	1	2	3	4	5	6	7	8	9	10
Pattern 1:	1	1	0	0	1	0	1	0	1	0
Pattern 2:	1	0	1	1	0	0	1	0	1	0
		s^f_{12}	s^f_{21}	s^l_{21}	s^l_{12}					

Figure 5.3: Example of two patterns which cannot meet when $k \geq 2$.

In Figure 5.3 we see that $s^f_{12} = 2$, $s^f_{21} = 3$, $s^l_{12} = 5$ and $s^l_{21} = 4$. This means that (5.5.10) is violated when $k \geq 2$.

Lemma 3 implies the necessary condition.

The pair separation condition: *Given a pattern set P^C and two patterns j_1, j_2 from this set, then the pattern set is feasible only if j_1, j_2 satisfy 5.5.10.*

All pairs j_1, j_2 violating this condition lead to the Benders cut:

$$p_{j_1} + p_{j_2} \leq 1 \tag{5.5.11}$$

Subsets of More Than Two Patterns.

Let $\bar{P}$ be a subset of P^C containing more than two patterns. To check if this subset can play the required number of mutual games, a CP model is used. The constraints are similar to the constraints (5.1.8) - (5.1.10) but they only consider the patterns from $\bar{P}$. This gives the following CP model referred to as (GAM).

$$(h_{j_1 s} = 0) \vee (h_{j_2 s} = 1) \Rightarrow (x_{j_1 j_2} \neq s) \qquad j_1, j_2 \in \bar{P},\ j_1 \neq j_2,\ s \in S \tag{5.5.12}$$

$$\mathit{alldifferent}(\mathit{all}(j_2 \in \bar{P} \setminus j_1)\ x_{j_1 j_2},\ \mathit{all}(j_2 \in \bar{P} \setminus j_1)\ x_{j_2 j_1}) \qquad j_1 \in \bar{P} \tag{5.5.13}$$

$$(x_{j_1 j_2} - x_{j_2 j_1} < -k) \vee (x_{j_1 j_2} - x_{j_2 j_1} > k) \qquad j_1, j_2 \in \bar{P},\ j_1 < j_2 \tag{5.5.14}$$

$$x_{j_1 j_2} \in S \qquad j_1, j_2 \in \bar{P},\ j_1 \neq j_2 \tag{5.5.15}$$

The fact that the mutual games between patterns in $\bar{P}$ can be assigned to time slots if and only if (GAM) is feasible leads to the second condition.

The multiple pattern separation condition: *Given a pattern set P^C and a subset of patterns $\bar{P}$ from this set, then the pattern set is feasible only if $\bar{P}$ is a feasible solution to (GAM).*

For all subsets $\bar{P}$ which are infeasible to (GAM) the following Benders cut can be added to (PSM)

$$\sum_{j \in \bar{P}} p_j \leq |\bar{P}| - 1 \tag{5.5.16}$$

Since the number of subsets is exponential the algorithm only checks subsets with cardinality less than a bound $maxCard_{GAM}$.

5.5.4 Game Assignment

When the pattern set P^C satisfies all the necessary conditions outlined above, a team allocation has already been found by the Hungarian method but a feasible game assignment is not guaranteed.

To find a game assignment if any exists, the CP model (GAM) from Section 5.5.3 is used. If a non-mirrored tournament with $k > 0$ is considered the model can be used as it is on the entire pattern set P^C. Otherwise, the separation constraint (5.5.14) is redundant and can be removed.

In case (GAM) is feasible a feasible game assignment is found and otherwise the Benders cut (5.5.16) is added to (PSM).

5.6 The Algorithm

This section presents pseudo code for each of the four components illustrated in Figure 5.1 and discusses how the components work.

Initialization.

Before the first patterns are generated a number of parameters must be initialized. The parameter c used in (PM) is initialized to zero, LB and UB used in (PSM) are initialized to $2n-2$ and $2n(2n-2)$ respectively and the parameters *nbPatterns* and *cutAdded* are initialized to zero. Furthermore, the parameters $maxCard_{GAM}$ and $maxCard_{UBM}$ are used to limit the size of the subsets for which (GAM) and (UBM) are used. These parameters must be initialized to numbers between three and $2n$.

Generating Patterns.

The procedure *Generate Patterns* is shown in Figure 5.4. Since the maximum number of breaks for a single pattern is $2n-1$ the algorithm stops if $c > 2n-1$ when the procedure is called. Otherwise it generates all patterns with exactly c breaks, increments c by one and calls the procedure *Find Pattern Set*. However, if the number of patterns is less than the number of teams, the procedure repeats itself.

Finding Pattern Sets.

Figure 5.5 outlines the procedure *Find Pattern Set* which solves (PSM). In case of a feasible solution, the lower bound is set equal to the objective value and the procedure *Check Feasibility* is called. Otherwise, *Generate Patterns* is called to generate additional patterns.

```
1  procedure Generate Patterns
2      if (c > 2n − 1) then
3          Stop
4      end if
5      else
6          Find all solutions to (PM)
7          Update nbPatterns
8          Let c = c + 1
9          if (nbPatterns < 2n) then
10             Generate Patterns
11         end if
12         else
13             Find Pattern Set
14         end else
15     end else
16 end procedure
```

Figure 5.4: Procedure for generating patterns.

```
1  procedure Find Pattern Set
2      Solve (PSM)
3      if ((PSM) is feasible) then
4          Update P^C
5          Update LB
6          Check Feasibility
7      end if
8      else
9          Generate Patterns
10     end else
11 end procedure
```

Figure 5.5: Procedure for finding pattern sets.

Feasibility.

The procedure *Check Feasibility*, is used to check if any of the necessary conditions from Section 5.5 is violated. Computational tests have shown that the algorithm performs best when the pattern diversity condition is omitted in case of non-mirrored tournaments and the pair separation condition and the multiple pattern separation condition is omitted in case of mirrored tournaments. Figure 5.6 displays the procedure for the non-mirrored case.

```
1  procedure Check Feasibility
2      cutAdded = 0, card = 2
3      Use the Hungarian method
4      if (∃X ⊆ T :  Γ(X) < X) then
5          Add (5.5.1) to (PSM), cutAdded = 1
6      end if
7      if (cutAdded = 0) then
8          for all (i, j ∈ P^C :  i < j) do
9              if ((5.5.10) is violated) then
10                 Add (5.5.11) to (PSM), cutAdded = 1
11             end if
12         end for all
13     end if
14     while ((card < maxCard_GAM) ∧ (cutAdded = 0)) do
15         card = card + 1
16         for all (P̄ ⊆ P^C :  |P̄| = card) do
17             if (cutAdded = 0) then
18                 Solve (GAM) for P̄
19                 if ((GAM) is infeasible) then
20                     Add (5.5.16) to (PSM), cutAdded = 1
21                 end if
22             end if
23         end for all
24     end while
25     if (cutAdded = 0) then
26         Find Timetable
27     end if
28     else
29         Find Pattern Set
30     end else
31 end procedure
```

Figure 5.6: Procedure for checking feasibility.

The procedure starts by using the Hungarian method to find a team allocation. In case no allocation exists the method finds a subset of teams which violates the necessary and sufficient condition stated in Hall's theorem and the cut (5.5.1) is added to (PSM). If a team allocation is found the algorithm adds the cut (5.5.11) to (PSM) for all pairs of patterns which violate (5.5.10). In case all pairs satisfy (5.5.10) it solves (GAM), starting with subsets of size three, and continues until a subset is found for which (GAM) is infeasible, or all subsets with cardinality less than or equal to $maxCard_{GAM}$ have been checked. In the first case the cut

(5.5.16) is added to (PSM). Finally, if no cut has been added, *Find Timetable* is called and otherwise *Find Pattern Set* is called.

```
1  procedure Find Timetable
2      Solve (GAM) for P^C
3      if ((GAM) is infeasible) then
4          Add (5.5.16) to (PSM)
5          Find Pattern Set
6      end if
7      else
8          Let UB = LB − 2
9          Let LB = 2n − 3 + c
10         if (c ≤ UB − 2n + 3) then
11             for all (i ∈ {c, ..., max{2n − 1, UB − 2n + 3}}) do
12                 Generate Patterns
13             end for all
14             Let c = 2n
15             Find Pattern Set
16         end if
17         else
18             Stop, optimal solution found
19         end else
20     end else
21 end procedure
```

Figure 5.7: Procedure for finding a timetable.

Timetable.

The procedure for finding a timetable is shown in Figure 5.7. In this procedure (GAM) is solved for P^C and in case of infeasibility (5.5.16) is added to (PSM) and *Find Pattern Set* is called. Otherwise a timetable has been found. This gives us a feasible solution to the problem with value LB and to prove optimality the algorithm starts searching for a better solution. This is done by generating new patterns and updating the upper and lower bounds used in (PSM). We let $UB = LB - 2$ since we are only searching for improving solutions. Furthermore, an improving solution must contain at least one pattern with no less than c breaks (otherwise we would have found it all ready) which means that we can let $LB = 2n - 3 + c$ (at most 2 patterns without breaks, 1 with at least c breaks and $2n - 3$ with 1 break). When generating extra patterns UB can be used to limit the number of patterns which is necessary to prove optimality. Since no pattern can have more than $UB - (2n - 3)$ breaks in a solution with value UB we generate

all patterns with at most $UB - (2n - 3)$ breaks. Afterwards, *Find Pattern Set* is called and in case of a new feasible solution this is the optimal, otherwise the first solution is optimal.

5.7 Computational results

In order to examine the performances of the PGBA, we have tested the algorithm on numerous instances of mirrored and non-mirrored tournaments with and without place constraints.

For comparison we use an algorithm denoted TPA which corresponds to the three phase approach used by Nemhauser and Trick [66] and Henz [43]. However, since our problem consists of finding only one schedule, TPA stops when the first feasible schedule has been generated. This is implemented by checking feasibility of each pattern set before the next is generated.

Computation times reported in the following tables are in seconds and all tests have been performed on a 2.53 GHz pentium 4 processor with 512 MB RAM. The algorithms are implemented by using a script in ILOG OPL studio [52] and CPLEX and SOLVER are called for solving IP and CP problems respectively. A time limit of 1800 seconds are enforced and "–" is used when this limit is violated. The instances are named using the following abbreviations: np/pl (no place constraints/place constraints), mi/nm (mirrored tournament/non-mirrored), ki ($k = i$), pi (i is total number of place constraints), ni (i teams).

Table 5.1 shows the computation times for mirrored instances without place constraints. The instance with 4 teams is infeasible since consecutive breaks are forbidden.

In Table 5.2 the computation times for non-mirrored instances without place constraints are shown. For each number of teams the problem is solved with k equal to 0, 1, 2 and 3.

Table 5.2: Non-mirrored instances without place constraints.

Instance	Breaks	TPA	PGBA
np-nm-k0-n4	2	0.02	0.02
np-nm-k0-n6	4	0.03	0.09
np-nm-k0-n8	6	0.28	0.17
np-nm-k0-n10	8	1.45	0.36
np-nm-k0-n12	10	29.78	1.41
np-nm-k0-n14	12	885.80	3.95
np-nm-k0-n16	14	–	1.70
np-nm-k0-n18	16	–	4.36

* Problem is infeasible.

Table 5.2: (Continued).

Instance	Breaks	TPA	PGBA
np-nm-k0-n20	18	–	7.16
np-nm-k0-n22	20	–	6.17
np-nm-k0-n24	22	–	9.31
np-nm-k0-n26	24	–	17.27
np-nm-k0-n28	26	–	129.00
np-nm-k0-n30	?	–	–
np-nm-k1-n4	6	0.06	0.09
np-nm-k1-n6	10	546.88	20.70
np-nm-k1-n8	8	19.48	0.56
np-nm-k1-n10	10	90.03	0.94
np-nm-k1-n12	12	–	0.91
np-nm-k1-n14	14	–	1.92
np-nm-k1-n16	16	–	3.99
np-nm-k1-n18	18	–	5.00
np-nm-k1-n20	20	–	14.70
np-nm-k1-n22	?	–	–
np-nm-k2-n4 *	–	0.16	0.14
np-nm-k2-n6	10	555.11	15.14
np-nm-k2-n8	8	19.61	0.55
np-nm-k2-n10	10	91.03	0.89
np-nm-k2-n12	12	–	0.92
np-nm-k2-n14	14	–	2.14
np-nm-k2-n16	16	–	3.48
np-nm-k2-n18	18	–	4.97
np-nm-k2-n20	20	–	14.75
np-nm-k2-n22	?	–	–
np-nm-k3-n4 *	–	0.16	0.13
np-nm-k3-n6	12	–	70.02
np-nm-k3-n8	12	–	137.00
np-nm-k3-n10	?	–	–
np-nm-k3-n12	16	–	8.55
np-nm-k3-n14	18	–	8.86
np-nm-k3-n16	20	–	17.55
np-nm-k3-n18	?	–	–

* Problem is infeasible.

Since different place constraints affect the complexity of an instance differently we have tested the algorithms on 10 sets of randomly generated place constraints

Table 5.1: Mirrored instances without place constraints.

Instance	Breaks	TPA	PGBA
np-mi-n4 *	–	0.00	0.02
np-mi-n6	12	0.02	0.02
np-mi-n8	18	0.02	0.03
np-mi-n10	24	0.13	0.09
np-mi-n12	30	0.09	0.08
np-mi-n14	36	0.94	0.19
np-mi-n16	42	3.27	0.31
np-mi-n18	48	9.44	0.63
np-mi-n20	54	88.14	1.34
np-mi-n30	84	–	2.33
np-mi-n38	108	–	4.83
np-mi-n40	114	–	–

* Problem is infeasible

for each instance. The place constraints include both home and away requirements and they may result in infeasible problems. The constraints can be found at [71].

Tables 5.3 and 5.4 show the results for the mirrored and the non-mirrored instances respectively. The second column tells how many of the instances that are feasible, the third and fourth tell how many instances the two algorithms were able to solve within the time limit and the rest of the columns give the minimum, the maximum and the average computation time for the two algorithms. Notice, that the average time only includes instances which were solved within the time limit.

The computational tests for both the mirrored and non-mirrored instances, with and without place constraints show that PGBA is superior to TPA and it is able to solve problems in few seconds which cannot be solved by TPA in half an hour. In the easy instances the time is almost the same, but PGBA performs significantly better than TPA when hard instances are considered. PGBA also proves to be extremely stable when random place constraints are considered. In Table 5.4 the maximum time used by PGBA is 8.19 seconds while TPA is unable to solve several of the instances in less than half an hour.

5.7.1 The Constant Distance Traveling Tournament Problem

Urrutia and Ribeiro [90] denote the special version of the TTP where all distances are equal to 1 the Constant Distance Traveling Tournament Problem (CTTP) and they show that maximizing the number of breaks is equivalent to solving the

Table 5.3: Mirrored instances with place constraints.

Instance	# Feas.	# Solved		Min. Time		Max. Time		Avg. Time	
		TPA	PGBA	TPA	PGBA	TPA	PGBA	TPA	PGBA
pl-mi-n12-p5	10	10	10	0.17	0.11	0.67	0.33	0.31	0.19
pl-mi-n12-p10	10	10	10	0.13	0.19	1.02	0.47	0.49	0.29
pl-mi-n12-p15	9	10	10	0.03	0.20	3.36	0.73	0.65	0.32
pl-mi-n12-p20	9	10	10	0.03	0.25	0.81	0.59	0.40	0.38
pl-mi-n12-p25	7	10	10	0.03	0.23	7.58	3.44	0.86	0.69
pl-mi-n12-p30	5	9	10	0.03	0.38	31.20	1.34	3.63	0.71
pl-mi-n16-p5	10	10	10	2.49	0.11	21.24	0.69	6.83	0.37
pl-mi-n16-p10	10	10	10	1.11	0.14	13.16	28.47	6.46	3.17
pl-mi-n16-p15	9	10	10	0.42	0.11	1306.03	14.48	137.51	2.15
pl-mi-n16-p20	9	10	10	0.72	0.11	233.03	204.38	33.72	24.96
pl-mi-n16-p25	9	8	10	0.72	0.20	57.86	29.20	12.98	6.52
pl-mi-n16-p30	9	9	10	0.42	0.11	1793.05	21.24	215.67	8.11

Table 5.4: Non-mirrored instances with place constraints.

Instance	# Feas.	# Solved		Min. Time		Max. Time		Avg. Time	
		TPA	PGBA	TPA	PGBA	TPA	PGBA	TPA	PGBA
pl-nm-k0-n8-p5	10	10	10	0.08	0.08	0.33	0.27	0.14	0.20
pl-nm-k0-n8-p10	10	10	10	0.08	0.06	0.25	0.61	0.15	0.21
pl-nm-k0-n8-p15	8	10	10	0.06	0.08	398.58	1.64	40.75	0.50
pl-nm-k0-n8-p20	5	8	10	0.08	0.28	38.27	1.91	5.34	0.83
pl-nm-k0-n8-p25	7	7	10	0.06	0.25	195.64	4.38	32.39	1.38
pl-nm-k0-n8-p30	3	10	10	0.06	0.63	76.58	0.99	8.65	0.77
pl-nm-k1-n8-p5	10	10	10	2.95	0.17	8.78	0.47	7.01	0.28
pl-nm-k1-n8-p10	10	10	10	1.11	0.17	258.76	0.94	33.17	0.50
pl-nm-k1-n8-p15	8	9	10	0.06	0.16	531.89	3.08	66.85	0.93
pl-nm-k1-n8-p20	5	8	10	0.06	0.38	424.81	5.66	69.83	1.75
pl-nm-k1-n8-p25	7	5	10	0.06	0.64	79.48	8.19	16.19	2.68
pl-nm-k1-n8-p30	3	10	10	0.06	0.61	619.63	4.00	69.75	1.16
pl-nm-k2-n8-p5	10	10	10	2.98	0.17	8.80	0.50	7.03	0.29
pl-nm-k2-n8-p10	10	10	10	1.09	0.17	258.64	0.67	33.14	0.46
pl-nm-k2-n8-p15	8	9	10	0.06	0.16	529.65	2.59	66.59	0.86
pl-nm-k2-n8-p20	5	8	10	0.06	0.38	429.67	5.75	70.40	2.06
pl-nm-k2-n8-p25	7	5	10	0.06	0.69	79.34	5.67	16.16	2.19
pl-nm-k2-n8-p30	3	10	10	0.06	0.61	619.58	3.06	69.79	1.07

CTTP. Miyashiro and Matsui [62] have shown that maximizing breaks is equivalent to minimizing breaks and the result means that the CTTP can be solved by minimizing breaks. However, we have used the PGBA to solve the CTTP by changing (PSM) to maximizing the number of breaks instead of minimizing it and allowing up to three consecutive home and away games. This means that the

Table 5.5: Solution values for Mirrored CTTP.

# Teams	Distance			# Breaks	Time
	UB	LB	PGBA		
4	17	17	17	14	0.02
6	48	48	48	24	0.03
8	80	80	80	64	0.13
10	130	130	130	100	0.28
12	192	192	192	144	0.23
14	256	252	253	222	35.50
16	342	342	342	276	3.02
18	434	432	432	360	2.53
20	526	520	–	–	–

PGBA finds feasible pattern sets for the TTP with a maximal number of breaks and as shown in the following two tables it is able to solve benchmark problems previously unsolved. The Benchmark problems plus bounds can be seen at [88].

Table 5.5 shows the results for the mirrored CTTP. The first column gives the number of teams, columns two and three give the upper and lower bounds reported by Urrutia and Ribeiro [90] while column four gives the distances obtained by PGBA. Columns five and six give the number of breaks and the solution time used by PGBA respectively. Notice, that since PGBA is an exact solution method it provides both lower and upper bounds.

In Table 5.6 the corresponding columns are presented for the non-mirrored CTTP. No lower bounds have been obtained prior to this work and the upper bounds are reported by Schaerf and Di Gaspero at [88].

Table 5.6: Solution values to Non-mirrored CTTP.

# Teams	Distance			# Breaks	Time
	UB	LB	PGBA		
4	17	–	17	14	0.02
6	43	–	43	34	0.14
8	80	–	80	64	0.94
10	124	–	124	112	6.91
12	182	–	181	166	327.94
14	253	–	252	224	23.63
16	331	–	327	306	43.42
18	423	–	–	–	–
20	525	–	–	–	–

Chapter 6

Scheduling SAS Ligaen

In this chapter we use the methodology of the PGBA to find a seasonal schedule for the best Danish soccer league called SAS Ligaen. This is a triple round robin tournament with 12 teams, it is partitioned into 33 slots and each team plays one game in each slot.

As always the major challenge, when creating the schedule for a sports league, is to satisfy the constraints arising from teams, spectators, TV stations, other tournaments, etc. The constraints are often conflicting and call for a solution method capable of ranking schedules with respect to the number of broken constraints instead of methods searching for schedules which satisfy all constraints.

Due to the promising computational results of the previous chapter, we want to use the strength of the PGBA to schedule SAS Ligaen. However, a number of modifications is necessary to deal with the additional constraints and the triple round robin structure. One of the main difficulties when scheduling SAS Ligaen compared to scheduling tournaments facing only place constraints is constraints concerning the timetable. In the previous chapter all constraints concerned the pattern set and the timetabling problem could be modelled as a feasibility problem. When the additional constraints are added, the timetabling problem becomes an optimization problem and we have to use optimality cuts in addition to the feasibility cuts used before. The algorithm will be outlined in Section 6.2 and the details are discussed in Section 6.3 but, before going deeper into the solution method, let us present the constraints.

6.1 Constraints for SAS Ligaen

The constraints of the tournament are partitioned into *hard constraints* which must be satisfied and *soft constraints* which incur a penalty in case they are violated. Below is an outline of the constraints for SAS Ligaen 2005/2006.

The structure of SAS Ligaen gives rise to a triple round robin tournament with 33 slots and 6 games in each slot. Furthermore, the tournament consists of three single round robin tournaments such that the slots from 1 to 11, the slots from 12 to 22 and the slots from 23 to 33 all form a single round robin tournament. In the rest of the chapter the single round robin tournament in the slots 1 - 11 will be referred to as Part 1 and the double round robin tournament in slots 12 - 33 will be referred to as Part 2. In Part 2 all teams must meet all other teams once at home and once at the opponent's venue.

Consecutive constraints limit the number of consecutive home games and consecutive away games to be less than or equal to 2. This is a hard constraint.

Separation constraints give a lower limit k on the number of slots between two games with the same opponents. If $k = 0$ it means that *repeater games* are allowed. But when $k = 1$ there must be at least one slot between slots where the two teams meet. This is a hard constraint.

Best half constraints are hard constraints stating that the 6 teams which finished in the best half of the tournament the preceding year get an extra home game in Part 1. This means that these teams play 6 home games in Part 1, while the other teams play 5 home games.

Ending constraints are hard constraints saying that teams cannot have a break at the last slot.

Place constraints are constraints saying that a specific team wants to play home in a certain slot or away in a certain slot. These constraints are hard or soft depending on the reason. If for instance a stadium is unavailable due to reconstruction or a concert it would be a hard constraint but in case the requirement is imposed due to nearby arrangements it would be a soft constraint.

Game constraints state that at least one game between a specific pair of teams (i_1, i_2) must be played in a certain set of slots. The game constraints are soft constraints and the set of constraints are denoted C^{Ga}. For a game constraint l, the set T_l^{Ga} denotes the pair of teams (i_1, i_2) and S_l^{Ga} denotes the set of slots in which a game between i_1 and i_2 must be played.

Top team constraints make sure that all non-top teams play at least one home game against one of the top teams in Part 1. In SAS Ligaen two teams are categorized as top teams. This is partly due to good results but also due to a large number of fans which means that revenue from spectators increase when a top team is visiting. Since the revenue goes to the home team, all teams want to play top teams home in Part 1. This is only a concern in Part 1 since all teams meet all other teams once home and once away in Part 2. The top team constraints are soft constraints and the set of non-top teams is denoted T^{To}.

Home constraints are used when a team i_1 has played many away games against another team i_2 in Part 1 in the preceding years. In this case a soft constraint can be added to make sure that team i_1 plays home against i_2 in Part

1. The set of home constraints are denoted C^{Ho} and for each $l \in C^{Ho}$ we use T_l^{Ho} to denote the set of teams (i_1, i_2) where i_1 must play home.

Beginning constraints are soft constraints which state that all teams must have a home game and an away game in the first two slots.

Geographic constraints require that at least one team from a certain area plays home in each slot or at least one team plays away in each slot. These constraints are used to avoid slots where some areas experience a large number of games while others are without games. We let C_H^{Ge} and C_A^{Ge} denote the set of constraints where at least one team must play home and the set of constraints where at least one team must play away while $C^{Ge} = C_H^{Ge} \cup C_A^{Ge}$ denotes the total set of geographic constraints. For each $l \in C^{Ge}$ the set T_l^{Ge} denotes the set of teams from the given area.

Break constraints say that the teams must alternate between home and away games. These are soft constraints and they are broken each time a team has a break.

The objective is to minimize the total penalty imposed by the violated constraints. For each of the soft constraints a coefficient represents the penalty which is added to the objective value if the constraint is violated. The coefficients can be seen in Table 6.1, which gives an overview of the constraints. For each constraint it shows whether the constraint is hard, soft, has influence on Part 1, Part 2, the pattern set or the timetable.

Table 6.1: Constraints for SAS Ligaen 2005/2006

Constraint	Hard	Soft	Part 1	Part 2	Patt. Set	TT.	Coef.	Con. Set
Structure	×		×	×	×	×	–	–
Consecutive	×		×	×	×		–	–
Separation	×		×	×		×	–	–
Best Half	×		×		×	×	–	–
Ending	×			×	×		–	–
Place	×	×	×	×	×		c^{Pl}	C^{Pl}
Game		×	×	×		×	c^{Ga}	C^{Ga}
Top Team		×	×			×	c^{To}	T^{To}
Home		×	×			×	c^{Ho}	C^{Ho}
Beginning		×	×		×		c^{Be}	–
Geographic		×	×	×	×		c^{Ge}	C^{Ge}
Break		×	×	×	×		c^{Br}	–

6.2 Methodology

To solve the problem we use a logic-based Benders decomposition strategy similar to the PGBA. The master problem consists of finding a pattern set and allocate teams to patterns while the subproblem finds a timetable if any exists. If a timetable is found, an optimality cut is added to the master problem and, otherwise, a feasibility cut is added. Since the subproblem is an IP problem we use logic-based Benders cuts instead of traditional Benders cuts. Furthermore, in order to limit the number of feasible solutions to the master problem, we use a column generation strategy to solve the master problem.

This leads to a solution method which decomposes the problem into four steps. In Step 1, we generate patterns, in Step 2, we find a pattern set and allocate teams to patterns, in Step 3, we check feasibility of the pattern set found in Step 2 and finally, in Step 4, we find a timetable. The four steps are visited iteratively during the process.

The algorithm uses a set containing all patterns which have been generated. Initially this set is empty but each time the algorithm goes to Step 1, additional patterns are added unless all feasible patterns have been generated already. In that case, the algorithm stops. The first time Step 1 is used, it generates all patterns with 0 breaks and each time the algorithm returns to Step 1, the number of breaks is increased by one. In this way the best patterns, with respect to the number of breaks, are considered first.

When patterns have been generated in Step 1, we solve an IP problem in Step 2 to find a pattern set and allocate the teams to the patterns. This IP problem is referred to as the master problem, since it resembles the master problem from Benders decomposition. In case the master problem is infeasible, we return to Step 1 where additional patterns are generated and otherwise we go to Step 3.

Step 3 is used to detect infeasible pattern sets and generate logic-based Benders cuts which can be added to the master problem. The strength of a logic-based Benders cut depends on the number of infeasible pattern sets it is able to cut off, and in order to find strong cuts, we need to know why the pattern sets are infeasible. If we have an infeasible pattern set but no knowledge of why it is infeasible we can only prevent the master problem from finding the same solution again. On the other hand, if we know that a pattern set is infeasible because it contains two patterns which cannot be in the same pattern set, we can add a cut which prevents the master problem from finding any pattern set containing both of these two patterns. Therefore, Step 3 contains a number of feasibility checks which are used to determine why a pattern set if infeasible. If infeasibility is detected, we return to Step 2 and otherwise, we proceed to Step 4. However, the feasibility checks in Step 3 are not exhaustive meaning that the pattern set might be infeasible anyway.

The problem of finding an optimal timetable for the pattern set found in Step 2

is formulated as an IP model and is referred to as the subproblem. In case the problem is infeasible, it means that the pattern set is infeasible and a logic-based Benders cut is added to the master problem. Otherwise we have found a feasible schedule and an optimality cut can be added to the master problem. In both cases we return to Step 2 after the cut has been added.

The algorithm keeps iterating until the master problem eventually becomes infeasible and all feasible patterns have been generated. When this happens we have either found an optimal solution or proved that the problem is infeasible. Figure 6.1 displays a flowchart showing how the algorithm iterates between the four steps.

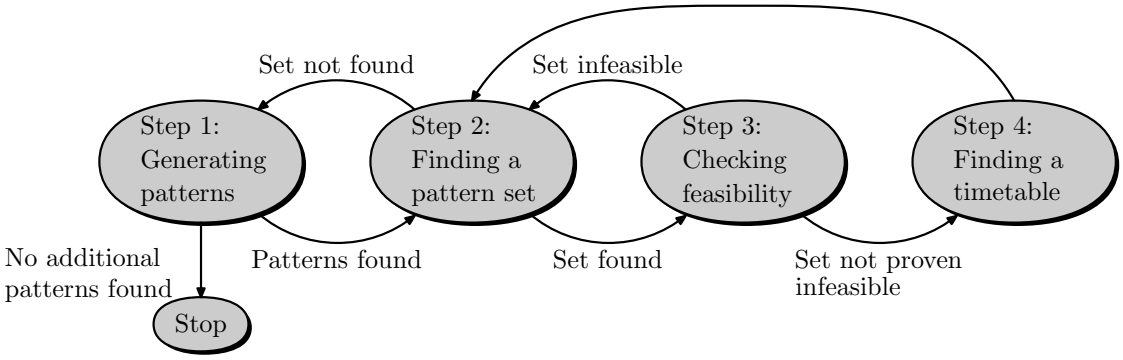

Figure 6.1: Flowchart for the algorithm.

6.3 The Algorithm

In this section we give a more detailed description of the four steps discussed in Section 6.2 but before we do that we need some notation. In the rest of the paper we let n denote the number of teams while T denotes the set of teams. The set $\mathcal{P} = \{1, 2\}$ represents the two parts discussed in Section 6.1 and S, S^1 and S^2 denote the set of all slots, the set of slots in Part 1 and the set of slots in Part 2, respectively.

6.3.1 Generating Patterns

The partitioning of the tournament into Part 1 and Part 2 is used to reduce the total number of patterns which must be generated. Instead of generating patterns covering all slots, we generate patterns for Part 1 and Part 2 separately.

In addition to reducing the number of patterns, the partitioning also makes the algorithm more flexible. If one part is infeasible, we can generate additional patterns for this part alone without generating extra patterns for the other part.

We let P^1 and P^2 denote the sets of patterns which have been generated for Part 1 and Part 2 respectively while B^1 and B^2 denote the current upper bounds on the number of breaks. When additional patterns are generated for Part p, the associated bound B^p is increased by one and all feasible patterns for Part p with exactly B^p breaks are generated. In order to limit the number of breaks per team, B^1 and B^2 cannot exceed 3 but still this allows up to seven breaks per team.

The generation of patterns has been implemented in OPL Script but the feasible patterns for Part 1 with B^1 breaks correspond to the feasible solutions of the following CP model where the variable h_s is 1 if the pattern has a home game in slot s and 0 if it has an away game.

$$\sum_{s=1}^{n-1} h_s \leq n/2 \tag{6.3.1}$$

$$\sum_{s=1}^{n-1} h_s \geq n/2 - 1 \tag{6.3.2}$$

$$\sum_{s=\hat{s}}^{\hat{s}+2} h_s \leq 2 \qquad \forall \hat{s} \in \{1, \ldots, n-3\} \tag{6.3.3}$$

$$\sum_{s=\hat{s}}^{\hat{s}+2} h_s \geq 1 \qquad \forall \hat{s} \in \{1, \ldots, n-3\} \tag{6.3.4}$$

$$\sum_{s=1}^{n-2} (h_s = h_{s+1}) = B^1 \tag{6.3.5}$$

$$h_s \in \{0, 1\} \qquad \forall s \in \{1, \ldots, n-1\} \tag{6.3.6}$$

The constraints (6.3.1) and (6.3.2) make sure that the pattern has at most $n/2$ home games and at most $n/2$ away games. Constraints (6.3.3) and (6.3.4) limit the number of consecutive home and consecutive away games to be less than or equal to 2, and constraint (6.3.5) makes sure that the number of breaks equals B^1.

Similarly, the feasible patterns for Part 2 with B^2 breaks correspond to the feasible solutions of the CP model below.

$$sequence(1, 2, 3, [h_n, \ldots, h_{3n-3}], [0, 1], [n-1, n-1]) \tag{6.3.7}$$

$$\sum_{s=n}^{3n-4} (h_s = h_{s+1}) = B^2 \tag{6.3.8}$$

$$h_{3n-4} \neq h_{3n-3} \tag{6.3.9}$$

$$h_s \in \{0, 1\} \qquad \forall s \in \{n, \ldots, 3n-3\} \tag{6.3.10}$$

Here constraint (6.3.7) makes sure that the pattern has $n-1$ home games, $n-1$ away games and no more than two consecutive home games or two consecutive away games. Constraint (6.3.8) determines the number of breaks, and constraint (6.3.9) avoids a break in the last slot.

In addition to the two sets P^1 and P^2, we let P_i^p denote the set of patterns which satisfy the hard place constraints of team i for each part $p \in \mathcal{P}$ and each team $i \in T$. We also need some coefficients to evaluate the patterns in the master problem. For each pattern $j \in P^p$, $p \in \mathcal{P}$, the coefficient c_j^{Br} denotes the break penalty and is equal to the number of breaks B^p times the break coefficient c^{Br}. The coefficient c_{ij}^{Pl} denotes the place penalty incurred if team i uses pattern j and is equal to the sum of place coefficients c^{Pl} associated with the violated place constraints. Furthermore, for each pattern $j \in P^1$ the coefficient c_j^{Be} denotes the beginning penalty which is equal to the beginning coefficient c^{Be}, if j has a break in the second slot and 0 otherwise. By letting $c_{ij}^1 = c_j^{Br} + c_{ij}^{Pl} + c_j^{Be}$ for each $i \in T$ and $j \in P^1$ and $c_{ij}^2 = c_j^{Br} + c_{ij}^{Pl}$ for each $i \in T$ and $j \in P^2$, we can let c_{ij}^p denote the coefficient of assigning team i to pattern j in Part p. These coefficients are used in the objective function of the master problem.

6.3.2 Pattern Set

When patterns have been generated for both parts we use an IP model to find a pattern set and assign teams to patterns. In the model, the parameters h_{js} for all $j \in P^p$, $s \in S^p$ and $p \in \mathcal{P}$ represent the patterns and h_{js} is 1 if pattern j has a home game in slot s and 0 if it has an away game. We use a binary variable x_{ij}^p for each team $i \in T$, each pattern $j \in P^p$ and each part $p \in \mathcal{P}$ to assign teams to patterns. The variable x_{ij}^p is 1 if team i uses pattern j in Part p, and it is 0 otherwise.

In addition to the assignment variables we use two kinds of penalty variables π_i^{Br} and π_{ls}^{Ge}. The first variable π_i^{Br} is 1 if team i has a break in the first slot in Part 2, and it is 0 otherwise. The second variable π_{ls}^{Ge} is 1 if the geographic constraint l is violated in slot s, and 0 otherwise.

Finally, we need a variable v which is used for optimality cuts to be explained in Section 6.3.4. This gives the following IP model.

$$\min \sum_{p\in\mathcal{P}} \sum_{i\in T} \sum_{j\in P_i^p} c_{ij}^p x_{ij}^p + \sum_{i\in T} c^{Br}\pi_i^{Br} + \sum_{l\in C^{Ge}} \sum_{s\in S} c^{Ge}\pi_{ls}^{Ge} + v \tag{6.3.11}$$

$$\text{s.t.} \sum_{p\in\mathcal{P}} \sum_{i\in T} \sum_{j\in P_i^p} c_{ij}^p x_{ij}^p + \sum_{i\in T} c^{Br}\pi_i^{Br} + \sum_{l\in C^{Ge}} \sum_{s\in S} c^{Ge}\pi_{ls}^{Ge} + v \geq LB \tag{6.3.12}$$

$$\sum_{p\in\mathcal{P}} \sum_{i\in T} \sum_{j\in P_i^p} c_{ij}^p x_{ij}^p + \sum_{i\in T} c^{Br}\pi_i^{Br} + \sum_{l\in C^{Ge}} \sum_{s\in S} c^{Ge}\pi_{ls}^{Ge} + v \leq UB \tag{6.3.13}$$

$$\sum_{j \in P_i^p} x_{ij}^p = 1 \qquad \forall i \in T,\ \forall p \in \mathcal{P} \tag{6.3.14}$$

$$\sum_{i \in T} x_{ij}^p \leq 1 \qquad \forall j \in P^p,\ \forall p \in \mathcal{P} \tag{6.3.15}$$

$$\sum_{i \in T} \sum_{j \in P_i^p} h_{js} x_{ij}^p = n/2 \qquad \forall s \in S^p,\ \forall p \in \mathcal{P} \tag{6.3.16}$$

$$\sum_{i \in T_l^{Ge}} \sum_{j \in P_i^p} h_{js} x_{ij}^p + \pi_{ls}^{Ge} \geq 1 \qquad \forall l \in C_H^{Ge},\ \forall s \in S^p,\ \forall p \in \mathcal{P} \tag{6.3.17}$$

$$\sum_{i \in T_l^{Ge}} \sum_{j \in P_i^p} (1 - h_{js}) x_{ij}^p + \pi_{ls}^{Ge} \geq 1 \qquad \forall l \in C_A^{Ge},\ \forall s \in S^p,\ \forall p \in \mathcal{P} \tag{6.3.18}$$

$$\sum_{j \in P_i^1} h_{jn-1} x_{ij}^1 + \sum_{j \in P_i^2} h_{jn} x_{ij}^2 + \pi_i^{Br} \geq 1 \qquad \forall i \in T \tag{6.3.19}$$

$$\sum_{j \in P_i^1} (1 - h_{jn-1}) x_{ij}^1 + \sum_{j \in P_i^2} (1 - h_{jn}) x_{ij}^2 + \pi_i^{Br} \geq 1 \qquad \forall i \in T \tag{6.3.20}$$

$$\sum_{j \in P_i^1} (h_{jn-2} + h_{jn-1}) x_{ij}^1 + \sum_{j \in P_i^2} h_{jn} x_{ij}^2 \leq 2 \qquad \forall i \in T \tag{6.3.21}$$

$$\sum_{j \in P_i^1} (h_{jn-2} + h_{jn-1}) x_{ij}^1 + \sum_{j \in P_i^2} h_{jn} x_{ij}^2 \geq 1 \qquad \forall i \in T \tag{6.3.22}$$

$$\sum_{j \in P_i^1} h_{jn-1} x_{ij}^1 + \sum_{j \in P_i^2} (h_{jn} + h_{jn+1}) x_{ij}^2 \leq 2 \qquad \forall i \in T \tag{6.3.23}$$

$$\sum_{j \in P_i^1} h_{jn-1} x_{ij}^1 + \sum_{j \in P_i^2} (h_{jn} + h_{jn+1}) x_{ij}^2 \geq 1 \qquad \forall i \in T \tag{6.3.24}$$

$$x_{ij}^p \in \{0,1\} \qquad \forall i \in T,\ \forall j \in P^p,\ \forall p \in \mathcal{P} \tag{6.3.25}$$

$$\pi_i^{Br} \in \{0,1\} \qquad \forall i \in T \tag{6.3.26}$$

$$\pi_{ls}^{Ge} \in \{0,1\} \qquad \forall l \in C^{Ge},\ \forall s \in S \tag{6.3.27}$$

$$v \in \mathbb{R}_+ \tag{6.3.28}$$

The objective function (6.3.11) includes break penalties, beginning penalties, place penalties, geographic penalties and the coefficient v used for optimality cuts. v gives a lower bound on the objective value of the subproblem but it has no effect before feasibility cuts have been added. The feasibility cut will be explained in Section 6.3.4 Constraints (6.3.12) and (6.3.13) give upper and lower bounds on the objective value. Constraints (6.3.14) make sure that all teams are assigned to a pattern in each part, and constraints (6.3.15) say that a pattern cannot be assigned to more than one team. Constraints (6.3.16) require that exactly half

the teams play home in each slot and constraints (6.3.17) and (6.3.18) state the geographic constraints. Constraints (6.3.19) and (6.3.20) make sure that π_i^{Br} is 1 if team i has a break in slot n, and the constraints (6.3.21) – (6.3.24) are used to avoid more than 2 consecutive home games or more than 2 consecutive away games in the transition between Part 1 and Part 2.

In case the problem is infeasible, we return to Step 1 and generate additional patterns. Otherwise, we let $(\bar{x}, \bar{\pi}^{Br}, \bar{\pi}^{Ge})$ denote an optimal solution to the problem, we let $\bar{P}^p$ denote the set of patterns used in Part p, and in case $LB < \bar{v}$, we update LB to be equal to $\bar{v}$. Finally, we let the parameters $\bar{h}_{is}^1$ for each $s \in S^1$ represent the pattern team i uses in Part 1 and $\bar{h}_{is}^2$ for each $s \in S^2$ represent the pattern team i uses in Part 2.

6.3.3 Feasibility Checks

After having found a pattern set, we need to check feasibility. We start by checking feasibility of Part 1 and Part 2 separately. In case one or both of these are infeasible, cuts are added to the master problem and we return to Step 2. Otherwise we continue with examining feasibility of the combined pattern set for both parts.

Feasibility of Part 1

Once again we use the necessary condition by Miyashiro et al. [64] stating that any subset of patterns $\hat{P}^1 \subseteq \bar{P}^1$ with cardinality Z must satisfy the condition

$$\sum_{s \in S^1} \left(\min \left\{ \sum_{j \in \hat{P}^1} h_{js}, \sum_{j \in \hat{P}^1} (1 - h_{js}) \right\} \right) \geq \frac{Z(Z-1)}{2}$$

if the pattern set $\bar{P}$ should be feasible.

In Chapter 5 we presented a MILP model to check if the condition is satisfied for all subsets of cardinality Z. The model finds the subset of patterns with fewest possible mutual games. The binary variable α_j is 1 if pattern j is included in the subset, and 0 otherwise. The binary variable δ_s is 1 if home games are counted in slot s, and 0 if away games are counted. Finally, the variable β_s counts the number of home or away games in slot s. The model is known as (UBM) and looks as follows:

$$\min \sum_{s \in S^1} \beta_s \tag{6.3.29}$$

$$\text{s.t.} \sum_{j \in \bar{P}^1} \alpha_j = Z \tag{6.3.30}$$

$$\beta_s - \sum_{j \in \bar{P}^1} h_{js}\alpha_j + Z(1-\delta_s) \geq 0 \qquad s \in S^1 \tag{6.3.31}$$

$$\beta_s - \sum_{j \in \bar{P}^1} (1-h_{js})\alpha_j + Z\delta_s \geq 0 \qquad s \in S^1 \tag{6.3.32}$$

$$\alpha_j,\ \ \delta_s \in \{0,1\} \qquad j \in \bar{P}^1,\ \ s \in S^1 \tag{6.3.33}$$

$$\beta_s \in \mathbb{R}_+ \qquad s \in S^1 \tag{6.3.34}$$

The objective function (6.3.29) minimizes the number of possible mutual games. The constraint (6.3.30) makes sure that Z teams are included in the subset and constraints (6.3.31) and (6.3.32) require that β_s is equal to the number of home games if $\delta_s = 1$ and equal to the number of away games if $\delta_s = 0$.

We can now use the pattern diversity condition to perform the first feasibility check.

> **The pattern diversity condition:** *Given a pattern set $\bar{P}^1$ and a subset size Z, then the pattern set is feasible only if the optimal solution of (UBM) is no less than $\frac{Z(Z-1)}{2}$.*

Notice that the condition was stated for a double round robin tournament in Chapter 5 but we have changed the threshold value $Z(Z-1)$ to $\frac{Z(Z-1)}{2}$. If the condition is violated, we add the following logic-based Benders cut to the master problem:

$$\sum_{i \in T} \sum_{j \in \hat{P}^1} x_{ij}^1 \leq Z - 1 \tag{6.3.35}$$

where $\hat{P}^1 = \{j \in \bar{P}^1 | \alpha_j = 1\}$.

Feasibility of Part 2

To check feasibility of Part 2, we use both the pattern diversity condition and the *multiple pattern separation condition* presented in Chapter 5. However, both feasibility checks can be strengthened compared to the original presentation due to the additional constraints considered in this application.

Since all teams must meet once in both halves of Part 2 we can apply the pattern diversity condition on both halves of Part 2 in the same way we applied it on Part 1. This would lead to cuts similar to (6.3.35). However, we can apply a much stronger cut when we use the fact that multiple patterns in Part 2 can have identical first halves or identical second halves.

Assume that a set of patterns $\hat{P}^2$ cannot meet in the first half of Part 2. This gives us the cut

$$\sum_{i \in T} \sum_{j \in \hat{P}^2} x_{ij}^2 \leq Z - 1 \tag{6.3.36}$$

but instead we add the cut

$$\sum_{i \in T} \sum_{j \in \hat{P}_E^2} x_{ij}^2 \leq Z - 1 \tag{6.3.37}$$

where $\hat{P}_E^2$ is the extended set of patterns consisting of all patterns which have a first half identical to one of the patterns in $\hat{P}^2$. A cut for the last half can be strengthened similarly.

The multiple pattern separation condition checks if the mutual games between the teams using a subset of patterns $\hat{P}^2 \subseteq \bar{P}^2$ can be assigned to slots. The following CP model is used to find a feasible assignment of games to slots if any exists, and otherwise we know that $\hat{P}^2$ is an infeasible subset. The model uses the variable $\sigma_{j_1 j_2}$ and sets it equal to the slot where the team using pattern j_1 plays home against the team using pattern j_2.

$$(h_{j_1 s} = 0) \vee (h_{j_2 s} = 1) \Rightarrow$$
$$(\sigma_{j_1 j_2} \neq s) \qquad \forall j_1, j_2 \in \hat{P},\ j_1 \neq j_2,\ \forall s \in S^2 \tag{6.3.38}$$
$$\textit{alldifferent}\big(\textit{all}(j_2 \in \hat{P}^2 \setminus j_1)\sigma_{j_1 j_2},$$
$$\textit{all}(j_2 \in \hat{P}^2 \setminus j_1)\sigma_{j_2 j_1}\big) \qquad \forall j_1 \in \hat{P}^2 \tag{6.3.39}$$
$$(\sigma_{j_1 j_2} - \sigma_{j_2 j_1} < -k) \vee (\sigma_{j_1 j_2} - \sigma_{j_2 j_1} > k) \qquad \forall j_1, j_2 \in \hat{P}^2,\ j_1 < j_2 \tag{6.3.40}$$
$$(\sigma_{j_1 j_2} \leq 2n - 2) \Leftrightarrow (\sigma_{j_2 j_1} \geq 2n - 1) \qquad \forall j_1, j_2 \in \hat{P}^2,\ j_1 < j_2 \tag{6.3.41}$$
$$\sigma_{j_1 j_2} \in S^2 \qquad \forall j_1, j_2 \in \hat{P}^2,\ j_1 \neq j_2 \tag{6.3.42}$$

The constraints (6.3.38) make sure that the team using pattern j_1 has a home game and the team using pattern j_2 has an away game when the latter team visits the first. Constraints (6.3.39) require that all games involving the same pattern are scheduled in different slots and constraints (6.3.40) state the separation constraints between games with the same two opponents. Finally, the constraints (6.3.41) state that all pairs of teams must play a game in both halves of Part 2.

> **The multiple pattern separation condition:** *Given a pattern set $\bar{P}^2$ and a subset of patterns $\hat{P}^2$ from this set, then the pattern set is feasible only if the above CP model has a feasible solution.*

If a subset of patterns $\hat{P}^2$ with cardinality Z makes the CP model infeasible, we add the logic-based Benders cut

$$\sum_{i \in T} \sum_{j \in \hat{P}^2} x_{ij}^2 \leq Z - 1 \tag{6.3.43}$$

to the master problem.

In Chapter 5 all subsets of patterns with cardinality less than a lower bound were checked in order to find infeasible subsets. Instead, we use a heuristic approach for finding candidate subsets in this approach since it is faster and it allows us to check subsets with larger cardinality. The price we pay is the risk of missing an infeasible subset with small cardinality.

The idea is to find the subset of teams which are most likely to make the CP model infeasible. We do that by finding a subset of patterns $\hat{P}^2$ which can only play a small number of mutual games and where the games must be played close to the middle of Part 2 since this will conflict with the separation constraints.

For a given cardinality Z, we let $\hat{P}^2$ be equal to the subset of $\bar{P}^2$ which has cardinality Z and minimizes

$$\sum_{j_1 \in \hat{P}^2} \sum_{j_2 \in \hat{P}^2} v_{j_1 j_2}$$

where

$$v_{j_1 j_2} = \sum_{s=n}^{2n-2-k} |\hat{h}^2_{j_1 s} - \hat{h}^2_{j_2 s}| + \sum_{s=2n-1-k}^{2n-2+k} \frac{1}{2} |\hat{h}^2_{j_1 s} - \hat{h}^2_{j_2 s}| + \sum_{s=2n-1+k}^{3(n-1)} |\hat{h}^2_{j_1 s} - \hat{h}^2_{j_2 s}|$$

The parameter $v_{j_1 j_2}$ increases with the number of slots in which j_1 and j_2 can meet, and games close to the middle of Part 2 contribute less than games in the beginning or in the end of Part 2.

The multiple pattern separation condition is used for increasing cardinalities until we find an infeasible subset or until we have checked all cardinalities.

Feasibility of the Combined Pattern Set

We use two kinds of feasibility checks for the combined pattern set. First all pairs of patterns are checked to see if the required number of games can be scheduled without violating the separation constraints. If this is not the case, we add a cut for each pair of teams which violates the constraints.

Assume that we have two patterns which violate the separation constraints and the first pattern consists of patterns j_1^1 and j_1^2 while the second pattern consists of patterns j_2^1 and j_2^2. If the first pattern is assigned to team i_1 and the second to team i_2, we add the following logic-based Benders cut to the master problem.

$$x^1_{i_1 j_1^1} + x^2_{i_1 j_1^2} + x^1_{i_2 j_2^1} + x^2_{i_2 j_2^2} \leq 3.$$

If all pairs of patterns satisfy the separation constraints, we use the multiple pattern separation condition on the combined pattern set. Again we need a CP model to check feasibility but this time we consider a subset of teams $\hat{T}$, and for each team i, we let j_i^1 and j_i^2 be the patterns assigned to team i in Part 1 and

Part 2, respectively. In order to ease notation let $\bar{h}_{is} = \bar{h}^1_{j_i^1 s}$ for all $s \in S^1$ and $\bar{h}_{is} = \bar{h}^2_{j_i^2 s}$ for all $s \in S^2$ in the following CP model. We can then formulate the CP model by letting the variable $\sigma^1_{i_1 i_2}$ denote the slot where the teams i_1 and i_2 meet in Part 1 and by letting $\sigma^2_{i_1 i_2}$ denote the slot where team i_2 visits team i_1 in Part 2.

$$(\bar{h}_{i_1 s} = \bar{h}_{i_2 s}) \Rightarrow (\sigma^1_{i_1 i_2} \neq s) \qquad \forall i_1, i_2 \in \hat{T},\ i_1 < i_2,\ \forall s \in S^1 \tag{6.3.44}$$

$$\sigma^1_{i_1 i_2} = \sigma^1_{i_2 i_1} \qquad \forall i_1, i_2 \in \hat{T},\ i_1 < i_2 \tag{6.3.45}$$

$$\textit{alldifferent}\left(\textit{all}(i_2 \in \hat{T} \setminus i_1)\sigma^1_{i_1 i_2}\right) \qquad \forall i_1 \in \hat{T} \tag{6.3.46}$$

$$\sigma^1_{i_1 i_2} < \sigma^2_{i_1 i_2} - k \qquad \forall i_1, i_2 \in \hat{T},\ i_1 \neq i_2 \tag{6.3.47}$$

$$(\bar{h}_{i_1 s} = 0) \vee (\bar{h}_{i_2 s} = 1) \Rightarrow (\sigma^2_{i_1 i_2} \neq s) \qquad \forall i_1, i_2 \in \hat{T},\ i_1 \neq i_2,\ \forall s \in S^2 \tag{6.3.48}$$

$$\textit{alldifferent}\left(\textit{all}(i_2 \in \hat{T} \setminus i_1)\ \sigma^2_{i_1 i_2},\ \textit{all}(i_2 \in \hat{T} \setminus i_1)\sigma^2_{i_2 i_1}\right) \qquad \forall i_1 \in \hat{T} \tag{6.3.49}$$

$$(\sigma^2_{i_1 i_2} < \sigma^2_{i_2 i_1} - k) \vee (\sigma^2_{i_2 i_1} < \sigma^2_{i_1 i_2} - k) \qquad \forall i_1, i_2 \in \hat{T},\ i_1 < i_2 \tag{6.3.50}$$

$$(\sigma^2_{i_1 i_2} \leq 2n - 2) \Leftrightarrow (\sigma^2_{i_2 i_1} \geq 2n - 1) \qquad \forall i_1, i_2 \in \hat{T},\ i_1 < i_2 \tag{6.3.51}$$

$$\sigma^1_{i_1 i_2} \in S^1 \qquad \forall i_1, i_2 \in \hat{T},\ i_1 \neq i_2 \tag{6.3.52}$$

$$\sigma^2_{i_1 i_2} \in S^2 \qquad \forall i_1, i_2 \in \hat{T},\ i_1 \neq i_2 \tag{6.3.53}$$

In this model constraints (6.3.44) make sure that, in Part 1, two teams can only meet in a slot where one of the teams plays home and the other plays away. Constraints (6.3.45) say that the model does not distinguish between home and away games in Part 1 and constraints (6.3.46) require that a team does not play more than one game in the same slot in Part 1. Constraints (6.3.47) make sure that the required separation between games with the same opponents is satisfied between games from Part 1 and Part 2. The constraints (6.3.48) - (6.3.51) are similar to the constraints (6.3.38) - (6.3.41).

In case a subset of teams $\hat{T}$ makes this CP model infeasible, we add the following logic-based Benders cut to the master problem.

$$\sum_{i \in \hat{T}} (x^1_{i j_i^1} + x^2_{i j_i^2}) \leq 2|\hat{P}| - 1 \tag{6.3.54}$$

We check subsets of teams with cardinality less than or equal to 4 and return to Step 2 when we find an infeasible subset.

6.3.4 Timetable

If no cuts have been generated in Step 3, we use an IP model to find an optimal timetable for the given pattern set or to prove that the pattern set is infeasible.

In this model we use a binary variable $y_{i_1 i_2 s}$ for all $i_1, i_2 \in T$ and for all $s \in S$. For a slot s in Part 1 the variable $y_{i_1 i_2 s}$ is 1, if the teams i_1 and i_2 meet in slot s, and for slots in Part 2 it is 1, if team i_1 plays home against team i_2 in slot s.

In the model we consider the game, home and top team constraints and for each of these constraints we associate a penalty variable which becomes 1 if the constraint is violated. The penalty variables are denoted π^{Ga}, π^{Ho} and π^{To}, respectively.

Before we state the model, recall that T_l^{Ga} is the pair of teams involved in game constraint l and S_l^{Ga} is the set of slots at which the game must be played. T_l^{Ho} is the pair of teams (i_1, i_2) involved in home constraint l and the team i_1 must play home to satisfy the constraint. To ease notation we use a parameter $\bar{h}_{is}$ for all $s \in S$ to represent the pattern assigned to team i, we let $S^1_{i_1 i_2}$ denote the slots in Part 1 where teams i_1 and i_2 can meet and we let $S^2_{i_1 i_2}$ denote the set of slots where team i_2 can visit team i_1. Furthermore, we let S_k^p denote the set of slots $\{p(n-1)+1-k, \ldots, p(n-1)\}$ for all $p \in \mathcal{P}$, since this set is used in the separation constraints.

$$\min \quad \sum_{i \in T^{To}} c^{To} \pi_i^{To} + \sum_{l \in C^{Ga}} c^{Ga} \pi_l^{Ga} + \sum_{l \in C^{Ho}} c^{Ho} \pi_l^{Ho} \tag{6.3.55}$$

$$\text{s.t.} \quad y_{i_1 i_2 s} = 0 \qquad \forall i_1, i_2 \in T, i_1 > i_2 \;\; s \in S^1 \tag{6.3.56}$$

$$\sum_{i_2 \in T \setminus i_1} (y_{i_1 i_2 s} + y_{i_2 i_1 s}) = 1 \qquad \forall i_1 \in T, \;\; \forall s \in S^p, \;\; p \in \mathcal{P} \tag{6.3.57}$$

$$\sum_{s \in S^1_{i_1 i_2}} y_{i_1 i_2 s} = 1 \qquad \forall i_1, i_2 \in T, \;\; i_1 < i_2 \tag{6.3.58}$$

$$\sum_{s \in S^2_{i_1 i_2}} y_{i_1 i_2 s} = 1 \qquad \forall i_1, i_2 \in T, \;\; i_1 \neq i_2 \tag{6.3.59}$$

$$\sum_{s=l(n-1)+1}^{(l+1)(n-1)} (y_{i_1 i_2 s} + y_{i_2 i_1 s}) = 1 \qquad \forall i_1, i_2 \in T, \;\; i_1 < i_2, \;\; \forall l \in \{1, 2\} \tag{6.3.60}$$

$$\sum_{s=\bar{s}}^{\bar{s}+k} (y_{i_1 i_2 s} + y_{i_2 i_1 s}) \leq 1 \qquad \forall i_1, i_2 \in T, \;\; i_1 < i_2, \;\; \forall \bar{s} \in S_k^p, \;\; \forall p \in \mathcal{P} \tag{6.3.61}$$

$$\sum_{s \in S_l^{Ga}} (y_{i_1 i_2 s} + y_{i_2 i_1 s}) + \pi_l^{Ga} \geq 1 \qquad (i_1, i_2) = T_l^{Ga}, \;\; \forall l \in C^{Ga} \tag{6.3.62}$$

$$\sum_{s \in S^1} \bar{h}_{i_1 s} (y_{i_1 i_2 s} + y_{i_2 i_1 s}) + \pi_l^{Ho} \geq 1 \qquad (i_1, i_2) = T_l^{Ho}, \;\; \forall l \in C^{Ho} \tag{6.3.63}$$

$$\sum_{i_2 \in T \setminus C^{To}_{i_1}} \sum_{s \in S^1} \bar{h}_{i_1 s}(y_{i_1 i_2 s} + y_{i_2 i_1 s}) + \pi^{To}_{i_1} \geq 1 \qquad \forall i_1 \in T^{To} \tag{6.3.64}$$

$$y_{i_1 i_2 s} \in \{0,1\} \qquad \forall i_1, i_2 \in T, \;\; s \in S \tag{6.3.65}$$

$$\pi^{To}_i \in \{0,1\} \qquad \forall i \in T^{To} \tag{6.3.66}$$

$$\pi^{Ga}_l \in \{0,1\} \qquad \forall l \in C^{Ga} \tag{6.3.67}$$

$$\pi^{Ho}_l \in \{0,1\} \qquad \forall l \in C^{Ho} \tag{6.3.68}$$

The objective function (6.3.55) minimizes the penalties associated with the timetable. The constraints (6.3.56) require that $y_{i_1 i_2 s}$ is zero if $i_1 > i_2$ for all $s \in S^1$, and constraints (6.3.57) make sure that all teams play exactly one game in each slot. Constraints (6.3.58) and (6.3.59) make sure that all pairs of teams meet once in Part 1 and both teams play a home game against the other team in Part 2. Constraints (6.3.60) make sure that both halves of Part 2 constitute a single round robin tournament, since they require that all pairs of teams meet in both halves. The separation constraints are satisfied due to constraints (6.3.61) which require that only one game with the same opponents can be played within $k+1$ consecutive slots. The constraints (6.3.62), (6.3.63) and (6.3.64) require that the penalty variables for the game, home and top constraints are 1 if the corresponding constraints are violated.

In case the model is infeasible, we add the following logic-based Benders cut to the master problem.

$$\sum_{i \in T} (x^1_{ij^1_i} + x^2_{ij^2_i}) < 2n - 1.$$

Otherwise, the optimal solution of the model gives an optimal timetable for the pattern set found in the master problem. This means that we have a feasible schedule and the value of the schedule is the value of the pattern set plus the value of the timetable.

After a feasible schedule has been found, we start searching for a better schedule by setting UB equal to the value of the current best schedule and by adding the following optimality cut to the master problem.

$$v \geq v^{TT} \left(\sum_{i \in T} (x^1_{ij^1_i} + x^2_{ij^2_i}) - 2n \right)$$

where v^{TT} is the value of the timetable which has just been found.

6.4 Computational Results

The performance of the algorithm and the quality of the schedules it obtains have been tested by comparing with the real schedule for the 2005/2006 season and

Table 6.2: Results for SAS Ligaen 2005/2006.

Instance	Feasible sol. found	Optimality proven	Time to find best solution	Time to prove optimality	Best value
$k0$-nComp	yes	no	707.94	–	41
$k0$-Comp	yes	no	188.11	–	42
$k1$-nComp	yes	no	544.09	–	41
$k1$-Comp	yes	yes	604.67	634.32	41
$k2$-nComp	yes	no	419.05	–	41
$k2$-Comp	yes	no	361.34	–	42
$k3$-nComp	yes	no	458.20	–	42
$k3$-Comp	yes	no	167.55	–	42
$k4$-nComp	yes	no	882.48	–	42
$k4$-Comp	yes	no	1310.81	–	42

by solving a number of randomly generated instances. The algorithm has been implemented as an OPL Script in Ilog OPL Studio [52] which uses Ilog CPLEX to solve IP problems and Ilog SOLVER to solve CP problems. All the computational tests presented in this section have been performed on a 2.53 GHz Pentium 4 processor with 512 MB RAM and we have used a time limit of 1800 seconds.

The Danish Football Association prefers a k value (separation) of at least 3 but in the schedule for the 2005/2006 season they had to let k equal 0 in order to satisfy team requirements. We have solved the problem with k ranging from 0 to 4 and the algorithm is able to satisfy more team requirements than the real schedule in all instances. We have also tested a complementary constraint requiring that the pattern set is complementary, since this constraint may be able to help the algorithm in some cases.

The results for the instances with k ranging from 0 to 4 ($k0 - k4$) with and without the complementary constraint (Comp/nComp) are reported in Table 6.2. It states whether a feasible solution has been obtained, whether optimality has been proven, the time used to find the best feasible solution, the time used to prove optimality and the value of the best schedule when all penalty coefficients equal 1.

Although the algorithm has problems with proving optimality we see that it generates very good solutions within a short amount of time and, in practical applications, optimal solutions may not even be the goal. In fact a number of good feasible schedules may be just as good or even better than one optimal schedule since it can be very hard to determine the right values for the penalty coefficients.

The exact constraints for the 2005/2006 season are confidential but in Table 6.3 we show the number of constraints, the number of violated constraints for the

Table 6.3: Number of violated constraints for the 2005/2006 season.

Instance	Place (29)	Game (1)	Top (10)	Home (1)	Beg. (10)	Geo. (3)	Break (372)	Total (426)
Real schedule	3	0	2	0	2	8	46	61
k0-nComp	0	0	1	1	2	1	36	41
k0-Comp	0	1	1	0	2	0	38	42
k1-nComp	0	0	1	1	2	1	36	41
k1-Comp	0	0	1	0	2	0	38	41
k2-nComp	0	1	1	0	2	1	36	41
k2-Comp	0	0	1	1	2	0	38	42
k3-nComp	0	0	1	0	2	1	38	42
k3-Comp	0	0	1	1	2	0	38	42
k4-nComp	0	0	1	0	2	1	38	42
k4-Comp	0	1	1	0	2	0	38	42

real schedule and the number of violated constraints for each of the 10 instances we have solved.

In all instances we are able to reduce the number of violated constraints with more than 30 percent and at the same time we can increase the separation from 0 to 4. Both top teams have only 5 home games in Part 1 and this means we will always violate at least 1 top team constraint. We also violate 2 beginning constraints in each instance but this is because one of the teams wants to begin with 2 away games. When k is less than 3, the price of using the complementary constraint is 2 additional breaks but, for greater k, there is no difference in the solution values.

In addition to the tests for the 2005/2006 season, we have tested the solution method on 10 randomly generated instances which resembles the real problem. Each of the instances have 30 randomly distributed place constraints, they have 2 top teams, 1 geographic home constraint with three teams, beginning constraints and best half constraints besides the break minimization constraint. Again we have tested the instances with k ranging from 0 to 4, with and without the complementary constraint.

The computational results are displayed in Table 6.4 and show the number of instances in which a feasible schedule has been found, the number of instances in which optimality has been proven, the average time to find the best schedule, the average time to prove optimality and the average value of the best schedule obtained.

The algorithm is very efficient at solving the problems with k less than three but, with k equal to three and four, the problems are getting harder and without

Table 6.4: Results for randomly generated instances.

Instance	Feasible solution	Proven optimal	Avg. time to find best solution	Avg. time to prove optimality	Avg. value
$k0$-nComp	10	10	119.41	185.26	33.50
$k0$-Comp	10	10	62.56	115.34	33.70
$k1$-nComp	10	10	136.13	196.47	33.50
$k1$-Comp	10	10	74.19	139.03	33.70
$k2$-nComp	10	10	147.70	215.17	33.50
$k2$-Comp	10	10	98.42	172.70	33.70
$k3$-nComp	5	3	637.03	859.08	34.20
$k3$-Comp	10	10	264.37	400.21	33.70
$k4$-nComp	2	2	995.43	1141.81	33.00
$k4$-Comp	9	8	561.05	672.84	33.77

the complementary constraint we are only able to solve 5 of the instances with k equal to 3 and 2 of the instances with k equal to 2 within the time limit. On the other hand when the complementary constraint is used, we can solve all instances but 1 and the objective value does not increase significantly. This makes the complementary constraint a good option when hard instances are considered.

The solution method were applied for finding a schedule for the 2006/2007 season and Table 6.5 displays the number of constraints and the number of violated constraints for the chosen $k4$-Comp instance.

Table 6.5: Number of violated constraints for the 2006/2007 season.

Instance	Place (38)	Game (3)	Top (10)	Home (9)	Beg. (10)	Geo. (8)	Break (372)	Total (426)
$k4$-Comp	0	0	1	0	4	2	44	51

In the schedule the four violated beginning constraints and the relatively high number of breaks are due to place constraints. One of the top team constraints will always be violated and the two violated geographic constraints are of second priority. The schedule is displayed in Figure 6.2.

In the schedule FCK has three consecutive home games in slots 9, 10 and 11, however, this is a consequence of their own place constraints and therefore it has been accepted.

Slot:	1	2	3	4	5	6	7	8	9	10	11
BIF:	+VB	+FCN	-FCM	+ACH	-RFC	+VIB	+SIF	-AAB	+EFB	-OB	-FCK
FCK:	-ACH	-SIF	+RFC	-VIB	+OB	-VB	+AAB	-EFB	+FCN	+FCM	+BIF
FCN:	+RFC	-BIF	+SIF	-EFB	+VB	-OB	-FCM	+ACH	-FCK	+VIB	-AAB
VB:	-BIF	+OB	-AAB	+FCM	-FCN	+FCK	-VIB	-RFC	+SIF	-EFB	+ACH
ACH:	+FCK	-RFC	+VIB	-BIF	+EFB	-SIF	+OB	-FCN	-FCM	+AAB	-VB
FCM:	-OB	-VIB	+BIF	-VB	+AAB	-EFB	+FCN	-SIF	+ACH	-FCK	+RFC
RFC:	-FCN	+ACH	-FCK	-OB	+BIF	-AAB	+EFB	+VB	-VIB	+SIF	-FCM
SIF:	-EFB	+FCK	-FCN	+AAB	-VIB	+ACH	-BIF	+FCM	-VB	-RFC	+OB
VIB:	-AAB	+FCM	-ACH	+FCK	+SIF	-BIF	+VB	-OB	+RFC	-FCN	+EFB
AAB:	+VIB	-EFB	+VB	-SIF	-FCM	+RFC	-FCK	+BIF	+OB	-ACH	+FCN
EFB:	+SIF	+AAB	-OB	+FCN	-ACH	+FCM	-RFC	+FCK	-BIF	+VB	-VIB
OB:	+FCM	-VB	+EFB	+RFC	-FCK	+FCN	-ACH	+VIB	-AAB	+BIF	-SIF

Slot:	12	13	14	15	16	17	18	19	20	21	22
BIF:	+FCM	-ACH	+AAB	-OB	-FCK	+SIF	-RFC	+FCN	-VIB	+VB	-EFB
FCK:	-RFC	+VIB	-OB	+VB	+BIF	-AAB	+FCM	-EFB	+FCN	-ACH	+SIF
FCN:	+SIF	-VB	+FCM	-EFB	+AAB	-OB	+ACH	-BIF	-FCK	+VIB	-RFC
VB:	-VIB	+FCN	-RFC	-FCK	+OB	-EFB	+AAB	-FCM	+SIF	-BIF	+ACH
ACH:	-OB	+BIF	+EFB	-FCM	-VIB	+RFC	-FCN	+SIF	-AAB	+FCK	-VB
FCM:	-BIF	+OB	-FCN	+ACH	-SIF	+VIB	-FCK	+VB	-RFC	+EFB	-AAB
RFC:	+FCK	-AAB	+VB	-VIB	+EFB	-ACH	+BIF	-OB	+FCM	-SIF	+FCN
SIF:	-FCN	+EFB	+VIB	-AAB	+FCM	-BIF	+OB	-ACH	-VB	+RFC	-FCK
VIB:	+VB	-FCK	-SIF	+RFC	+ACH	-FCM	+EFB	-AAB	+BIF	-FCN	+OB
AAB:	-EFB	+RFC	-BIF	+SIF	-FCN	+FCK	-VB	+VIB	+ACH	-OB	+FCM
EFB:	+AAB	-SIF	-ACH	+FCN	-RFC	+VB	-VIB	+FCK	+OB	-FCM	+BIF
OB:	+ACH	-FCM	+FCK	+BIF	-VB	+FCN	-SIF	+RFC	-EFB	+AAB	-VIB

Slot:	23	24	25	26	27	28	29	30	31	32	33
BIF:	+RFC	-AAB	+VIB	-VB	+EFB	-FCN	+FCK	-FCM	+OB	-SIF	+ACH
FCK:	-FCM	+OB	-FCN	+ACH	-SIF	+AAB	-BIF	+RFC	-VIB	+EFB	-VB
FCN:	+EFB	-FCM	+FCK	+OB	-ACH	+BIF	-AAB	+VB	-SIF	+RFC	-VIB
VB:	-OB	+RFC	-SIF	+BIF	-AAB	+VIB	+EFB	-FCN	+FCM	-ACH	+FCK
ACH:	+VIB	-EFB	+FCM	-FCK	+FCN	+OB	-SIF	+AAB	-RFC	+VB	-BIF
FCM:	+FCK	+FCN	-ACH	+RFC	-OB	+SIF	-VIB	+BIF	-VB	+AAB	-EFB
RFC:	-BIF	-VB	+AAB	-FCM	+VIB	-EFB	+OB	-FCK	+ACH	-FCN	+SIF
SIF:	+AAB	-VIB	+VB	-EFB	+FCK	-FCM	+ACH	-OB	+FCN	+BIF	-RFC
VIB:	-ACH	+SIF	-BIF	+AAB	-RFC	-VB	+FCM	-EFB	+FCK	-OB	+FCN
AAB:	-SIF	+BIF	-RFC	-VIB	+VB	-FCK	+FCN	-ACH	+EFB	-FCM	+OB
EFB:	-FCN	+ACH	-OB	+SIF	-BIF	+RFC	-VB	+VIB	-AAB	-FCK	+FCM
OB:	+VB	-FCK	+EFB	-FCN	+FCM	-ACH	-RFC	+SIF	-BIF	+VIB	-AAB

Figure 6.2: Schedule for SAS Ligaen 2006/2007.

Chapter 7

Minimizing Travel Distance

In this chapter we shift attention from break minimization towards minimization of travel distance. We consider a generalization to the break minimization problem when distances are considered instead of breaks. The problem is denoted the *timetable constrained distance minimization problem* (TCDMP) and it applies for example when leagues face many requests from TV networks. In order to increase the revenue earned from TV networks the sports leagues must be able to accommodate this kind of requests. Typically, these requests concern high-quality games since, the TV networks want to maximize the number of viewers. When many of these requests are present it may be necessary to determine all the opponents before finding a home-away assignment and in this case a good solution for the TCDMP becomes important.

To solve the TCDMP, we present and compare four solution methods. The problem is formulated as an IP model and a CP model which are solved using CPlex and Solver, respectively. Furthermore, we present a hybrid IP/CP solution method utilizing the strengths of both techniques by using CP for solving feasibility problems and IP for solving an optimization problem. Finally, the problem is solved by using a branch and price algorithm. It would be appealing to try to formulate this problem as a weighted maximum cut problem, mimicking the work of Elf et al. [28] and Miyashiro and Matsui [63], but it is not possible to accurately model the distance travelled through just the cut values.

In addition to the TCDMP we also present a new heuristic solution method for the TTP. This method called the *circular traveling salesman approach* (CTSA) takes advantage of already obtained solutions for one instance class of the TTP to obtain solutions for the general TTP problem. In this way, it is able to obtain solutions almost instantaneously since it only needs to solve a traveling salesman problem with up to 16 nodes. The solutions obtained are close to the known upper bounds and besides, being used as final schedules, they can be used as

initial solutions for some of the metaheuristic solution methods for the TTP.

7.1 The Timetable Constrained Distance Minimization Problem

First let us give a formal definition of the TCDMP.

Definition 5 (TCDMP) *Given a timetable for a double round robin tournament with n teams, a distance matrix specifying the distances between the venues and an upper bound UB on the number of consecutive home and consecutive away games, find a feasible pattern set which minimizes the total distance travelled by all teams.*

We have modelled the TCDMP using both an IP formulation and a CP formulation and the models are presented in the following two subsections. In the rest of the chapter, we let n denote the number of teams, while T denotes the set of teams. The set of slots is denoted S, and we let $S^0 = S \cup \{0\}$. The distance matrix is represented by D, and entrance $D_{i_1 i_2}$ contains the distance between the venue of team i_1 and the venue of team i_2. TT denotes the timetable and entrance TT_{is} gives the opponent of team i in slot s. Notice that $D_{TT_{is}TT_{is+1}}$ is the travel distance of team i between slots s and $s+1$ if team i plays away in both slots.

7.1.1 Integer Programming Formulation

To formulate the problem as an IP model, we use a binary variable h_{is} for each $i \in T$ and each $s \in S$. h_{is} equals 1 if team i plays home in slot s and it equals 0 if it plays away. To calculate the total travel distance, we use an integer variable, d_{is} for each $i \in T$ and each $s \in S^0$, which is equal to the distance team i travels between slot s and slot $s+1$. We use the dummy slots 0 and $2n-1$ to make sure that all teams start and end at home. This gives the following IP model.

$$\min \sum_{i \in T} \sum_{s \in S^0} d_{is} \tag{7.1.1}$$

$$\text{s.t. } d_{is} \geq (1 - h_{is} - h_{is+1}) D_{TT_{is}TT_{is+1}} \qquad \forall i \in T,\ \forall s \in S^0 \tag{7.1.2}$$

$$d_{is} \geq (h_{is} - h_{is+1}) D_{iTT_{is+1}} \qquad \forall i \in T,\ \forall s \in S^0 \tag{7.1.3}$$

$$d_{is} \geq (-h_{is} + h_{is+1}) D_{TT_{is}i} \qquad \forall i \in T,\ \forall s \in S^0 \tag{7.1.4}$$

$$h_{i0} = 1 \qquad \forall i \in T \tag{7.1.5}$$

$$h_{i2n-1} = 1 \qquad \forall i \in T \tag{7.1.6}$$

$$\sum_{s \in S} h_{is} = n - 1 \qquad \forall i \in T \tag{7.1.7}$$

$$h_{i_1 s} + h_{i_2 s} = 1 \qquad \forall i_1, i_2 \in T,\ i_1 < i_2,\ \forall s \in S,\ TT_{i_1 s} = i_2 \tag{7.1.8}$$

$$h_{is_1} + h_{is_2} = 1 \qquad \forall i \in T,\ \forall s_1, s_2 \in S,\ s_1 < s_2,\ TT_{is_1} = TT_{is_2} \tag{7.1.9}$$

$$\sum_{s=\hat{s}}^{\hat{s}+UB} h_{is} \leq UB \qquad \forall i \in T,\ \forall \hat{s} \in \{1, \ldots, 2(n-1) - UB\} \tag{7.1.10}$$

$$\sum_{s=\hat{s}}^{\hat{s}+UB} h_{is} \geq 1 \qquad \forall i \in T,\ \forall \hat{s} \in \{1, \ldots, 2(n-1) - UB\} \tag{7.1.11}$$

$$h_{is} \in \{0, 1\} \qquad \forall i \in T,\ \forall s \in \{0, \ldots, 2n-1\} \tag{7.1.12}$$

$$d_{is} \in \mathbb{Z}_+ \qquad \forall i \in T,\ \forall s \in S^0 \tag{7.1.13}$$

Constraints (7.1.2) - (7.1.4) give lower bounds on the distance team i travels between slots s and $s+1$ when i plays away in the two slots, when it plays home and away and when it plays away and home, respectively. Constraints (7.1.5) and (7.1.6) ensure that all teams start and end at home and constraints (7.1.7) make sure that all teams have exactly $n-1$ home games. Constraints (7.1.8) require that, when teams i_1 and i_2 meet in slot s, one of the teams must play home and the other must play away, while constraints (7.1.9) make sure that team i plays one home game and one away game in two slots with the same opponent. Finally, the constraints (7.1.10) and (7.1.11) give upper bounds on the number of consecutive home games and the number of consecutive away games.

7.1.2 Constraint Programming Formulation

When formulating the problem as a CP model, we use variables similar to the variables used in the IP model, but the CP model allows us to reformulate the constraints. In particular, we are able to formulate the constraints (7.1.7), (7.1.10) and (7.1.11) as a single constraint called *sequence* and we can use logical expressions to determine the travel distance. This gives the following CP model.

$$\min \sum_{i \in T} \sum_{s \in S^0} d_{is} \tag{7.1.14}$$

$$\text{s.t.}\ (h_{is} = 1) \wedge (h_{is+1} = 1) \Rightarrow (d_{is} = 0) \qquad \forall i \in T,\ \forall s \in S^0 \tag{7.1.15}$$

$$(h_{is} = 0) \wedge (h_{is+1} = 1) \Rightarrow (d_{is} = D_{TT_{is}i}) \qquad \forall i \in T,\ \forall s \in S^0 \tag{7.1.16}$$

$$(h_{is} = 1) \wedge (h_{is+1} = 0) \Rightarrow (d_{is} = D_{iTT_{is+1}}) \qquad \forall i \in T,\ \forall s \in S^0 \tag{7.1.17}$$

$$(h_{is} = 0) \wedge (h_{is+1} = 0) \Rightarrow (d_{is} = D_{TT_{is}TT_{is+1}}) \qquad \forall i \in T,\ \forall s \in S^0 \tag{7.1.18}$$

$$sequence(1, UB, UB+1, [h_{i1}, \ldots, h_{i2n-2}], [1], [n-1]) \qquad \forall i \in T \tag{7.1.19}$$

$$h_{is_1} \neq h_{is_2} \qquad \forall i \in T,\ \forall s_1, s_2 \in S,\ s_1 < s_2,\ TT_{is_1} = TT_{is_2} \tag{7.1.20}$$

$$h_{is} \neq h_{TT_{is}s} \qquad \forall i \in T,\ \forall s \in S \tag{7.1.21}$$

$$h_{i0} = 1 \qquad \forall i \in T \tag{7.1.22}$$

$$h_{i2n-1} = 1 \qquad \forall i \in T \tag{7.1.23}$$

$$h_{is} \in \{0,1\} \qquad \forall i \in T,\ \forall s \in \{0,\ldots,2n-1\} \tag{7.1.24}$$

$$d_{is} \in \mathbb{Z}_+ \qquad \forall i \in T,\ \forall s \in S^0 \tag{7.1.25}$$

The constraints (7.1.15) - (7.1.18) determine the travel distance of team i between slots s and $s+1$, depending on whether team i plays home or away in the two slots. The sequence constraints (7.1.19) say that team i must play exactly $n-1$ home games and in $UB+1$ consecutive slots it cannot play less than one home game or more than UB home games. Constraints (7.1.20) state that team i must play one home game and one away game in two slots where it meets the same opponent and constraints (7.1.21) require that the opponent of team i plays home (away) if team i plays away (home). Constraints (7.1.22) and (7.1.23) make sure that all teams start and end home.

In the following sections, we present a hybrid IP/CP approach and a branch and price algorithm for solving the TCDMP. A Benders decomposition approach similar to the method presented in previous chapter has also been implemented but the reduction in time used to solve the master problem could not offset the additional iterations which are required compared to the hybrid IP/CP approach.

7.2 Hybrid IP/CP Approach

The first of the specialized solution methods is a hybrid IP/CP approach which decomposes the problem into two phases. Phase 1 generates all feasible patterns for each team in the tournament and Phase 2 finds the optimal pattern set by assigning each team to one of the patterns found in Phase 1. Due to the complementary strengths of CP and IP we use a CP model for finding feasible patterns in Phase 1 while IP is used to solve the optimization problem in Phase 2. The details of the two phases are explained below.

7.2.1 Phase 1

In order to generate all feasible patterns, we use a CP model for each team and find all feasible solutions to each of the models. Each pattern must contain exactly $n-1$ home games and satisfy the upper bound on the number of consecutive home games and consecutive away games. Furthermore, a pattern for a specific team i must satisfy that, for all pairs of slots s_1 and s_2 where $TT_{is_1} = TT_{is_2}$, the pattern has both a home game and an away game.

To formulate a CP model for finding all feasible patterns of team i, we use a binary variable h_s for each $s \in S$. As in the earlier sections, $h_s = 1$ implies a home game in slot s and $h_s = 0$ implies an away game. The CP model for team i looks

as follows.

$$sequence(1, UB, UB+1, [h_1, \ldots, h_{2(n-1)}], [1], [n-1]) \tag{7.2.1}$$
$$h_{s_1} \neq h_{s_2} \qquad \forall s_1, s_2 \in S,\ s_1 < s_2,\ TT_{is_1} = TT_{is_2} \tag{7.2.2}$$
$$h_s \in \{0,1\} \qquad \forall s \in S \tag{7.2.3}$$

Constraint (7.2.1) corresponds to constraint (7.1.19) in the CP formulation and makes sure that the number of home games is correct and the upper bound on consecutive home games and consecutive away games is satisfied. Constraints (7.2.2) correspond to the constraints (7.1.20) and make sure that the patterns are feasible with respect to the timetable.

For each team $i \in T$, we let P_i denote the set of feasible patterns and for each $j \in P_i$, we let h_{js} represent the entry h_s of pattern j. We also calculate the distance team i must travel, if it uses pattern j, and denote it d_{ij}. The distance can be calculated since the timetable gives us the opponent of each slot and the pattern tells if team i plays home or away.

7.2.2 Phase 2

In Phase 2, we must find an optimal allocation of each team i to a pattern $j \in P_i$, such that the total travel distance is minimized and the pattern set is feasible with respect to the timetable.

To formulate an IP model for this problem, we use a binary variable x_{ij} for each $i \in T$ and each $j \in P_i$. The variable is 1 if team i is assigned to pattern j and 0 otherwise. We also use the home-away parameter h_{js} for each pattern j and each slot s and the distance parameter d_{ij} for each team i and each pattern $j \in P_i$.

$$\min \sum_{i \in T} \sum_{j \in P_i} d_{ij} x_{ij} \tag{7.2.4}$$
$$\text{s.t.} \sum_{j \in P_i} x_{ij} = 1 \qquad \forall i \in T \tag{7.2.5}$$
$$\sum_{i \in \{i_1, i_2\}} \sum_{j \in P_i} h_{js} x_{ij} = 1 \qquad \forall i_1, i_2 \in T,\ i_1 < i_2,\ \forall s \in S,\ TT_{i_1 s} = i_2 \tag{7.2.6}$$
$$x_{ij} \in \{0,1\} \qquad \forall i \in T,\ \forall j \in P_i \tag{7.2.7}$$

Constraints (7.2.5) are the assignment constraints saying that all teams must be assigned to a feasible pattern and constraints (7.2.6) make sure that when 2 teams meet, one of the teams play home and the other plays away. These two constraints are enough to ensure a feasible pattern set with respect to the timetable since we know from Phase 1 that all the teams play one home game and one away game in two slots where they meet the same opponent.

7.3 Branch and Price

In addition to the hybrid IP/CP approach, we present a branch and price algorithm (see Barnhart et al. [8] for a detailed description of branch and price) to solve the TCDMP. This method has successfully been applied to the TTP and it is to date the best exact solution method for the TTP.

The branch and price algorithm assigns teams to patterns by solving a linear programming (LP) problem which is restricted to only contain a subset of the feasible patterns instead of all the feasible patterns. This problem is known as the master problem.

The solution of the master problem may be fractional since we use an LP problem and it might not be optimal since we consider only a subset of the patterns. The optimality issue is handled by solving a pricing problem which finds patterns to the master problem with negative reduced costs. These patterns are added to the master problem and it is re-solved. If no patterns with negative reduced costs exist and the solution is fractional, the algorithm uses branch and bound to obtain an integer solution.

Before describing the details of the algorithm, let us present an outline with references to the relevant sections. UB^M denotes an upper bound on the master problem and N is the node set of the branch and bound tree.

Branch and price algorithm

Initialization Find a feasible pattern set (Section 7.3.1).

Initialize UB^M to the solution value of the initial feasible solution.

Initialize N to a single node.

Step 1 If $N \neq \emptyset$, choose $\hat{\eta}$ from N and let $N = N \setminus \hat{\eta}$ (Section 7.3.2).

Otherwise, stop.

Step 2 Solve the master problem and go to Step 3 (Section 7.3.3).

Step 3 Solve the pricing problem for each team (Section 7.3.4). If no patterns with negative reduced costs exist, go to Step 4.
Otherwise, add patterns to the master problem and go to Step 2.

Step 4 If the solution value is greater than UB^M, go to Step 1.
Otherwise, if the solution is fractional, go to Step 5.
Otherwise, update UB^M to the solution value and go to Step 1.

Step 5 branch, add the new nodes to N and go to Step 1 (Section 7.3.5).

7.3.1 Initial Feasible Pattern Set

In order to find an initial feasible pattern set, we use a CP model. The model corresponds to the CP model presented in Section 7.1.2 but, in this model, we ignore the travel distance, since we are only looking for a feasible solution. This gives the following CP model where the variable h_{is}, for each $i \in T$ and each $s \in S$, is 1 if team i plays home in slots s and 0 if it plays away.

solve:

$$sequence(1, UB, UB+1, [h_{i1}, \cdots, h_{i2(n-1)}], [1], [n-1]) \qquad \forall i \in T \tag{7.3.1}$$

$$h_{is_1} \neq h_{is_2} \qquad \forall i \in T,\ \forall s_1, s_2 \in S,\ s_1 < s_2,\ TT_{is_1} = TT_{is_2} \tag{7.3.2}$$

$$h_{is} \neq h_{TT_{is}s} \qquad \forall i \in T,\ \forall s \in S \tag{7.3.3}$$

$$h_{is} \in \{0, 1\} \qquad \forall i \in T,\ \forall s \in S \tag{7.3.4}$$

The constraints (7.3.1) - (7.3.3) are similar to the constraints (7.1.19) - (7.1.21) from the CP model in Section 7.1.2. The rest of the constraints from the CP model in Section 7.1.2 can be ignored, since they are all related to the travel distance.

If the model is infeasible, it means that no feasible pattern set exists and we are done. Otherwise, we store the patterns used by each team and calculate the travel distances. As in Section 7.2 we let P_i denote patterns which are feasible for team i but in this context P_i does not necessarily contain all the feasible patterns for team i. When an initial feasible solution has been found, P_i is initialized to contain the pattern team i uses and the travel distance is calculated.

We have tested the effect of generating an additional number of good patterns initially. In this way fewer patterns have to be generated during the search and less time is spend on solving the pricing problem. However, the overall computation time increased when this approach were used since the computation time of the master problem increased and additional branching took place before the optimal solution were found.

In the rest of the section, we use the notation that, for each team $i \in T$, the parameter h_{js} for each pattern $j \in P_i$ and each slot $s \in S$ is 1 if pattern j has a home game in slot s and 0 if it has an away game. We also let d_{ij} denote the travel distance of team i if it uses pattern j from P_i.

7.3.2 Node Selection Strategy

We have implemented two node selection strategies which are used to choose nodes from the branch and bound tree. The first strategy is the well-known *depth first strategy*. This strategy chooses one of the child nodes of the current strategy if any exists and otherwise it backtracks. The strategy corresponds to a *last in, first out* (LIFO) strategy since it always chooses the last node which has been added.

The second strategy is a *best lower bound strategy* which chooses the node with the lowest lower bound. In our case, we use the objective value of the parent node as lower bound and therefore the strategy chooses the node with the smallest parent value. Ties are broken arbitrarily.

7.3.3 Master Problem

The master problem of the branch and price algorithm is almost identical to the linear relaxation of the problem solved in Phase 2 of the hybrid IP/CP approach. The only differences are that the problem is restricted since P_i does not contain all the feasible patterns of team i, and that a number of branching constraints are added. The number of branching constraints corresponds to the level of the current node in the branch and bound tree.

The branching strategy and the branching constraints will be further discussed in Section 7.3.5 but we need some notation to formulate the master problem. We let B^η denote the set of branching constraints present in node η and we let i_b, s_b and v_b denote the team, the slot and the value of branching constraint $b \in B^\eta$. If v_b equals 1, it means that team i_b must play home in slot s_b and it must play away if v_b equals 0.

Since we use the linear relaxation, the variable x_{ij} gives the fraction of team i which is assigned to pattern j from P_i. Now, we can state the master problem corresponding to node $\eta \in N$ as follows.

$$\min \sum_{i \in T} \sum_{j \in P_i} d_{ij} x_{ij} \tag{7.3.5}$$

$$\text{s.t.} \sum_{j \in P_i} x_{ij} = 1 \qquad \forall i \in T \tag{7.3.6}$$

$$\sum_{i \in \{i_1, i_2\}} \sum_{j \in P_i} h_{js} x_{ij} = 1 \qquad \forall i_1, i_2 \in T,\ i_1 < i_2,\ \forall s \in S,\ TT_{i_1 s} = i_2 \tag{7.3.7}$$

$$\sum_{j \in P_{i_b}} h_{j s_b} x_{i_b j} = v_b \qquad \forall b \in B^\eta \tag{7.3.8}$$

$$x_{ij} \in R_+ \qquad \forall i \in T,\ \forall j \in P_i \tag{7.3.9}$$

The constraints (7.3.6) - (7.3.7) are similar to constraints (7.2.5) - (7.2.6) from Section 7.2.2 and constraints (7.3.8) are the branching constraints.

In the following, we refer to the optimal solution of the master problem as $\bar{x}$ and we let $\bar{P}_i = \{j \in P_i : \bar{x}_{ij} > 0\}$ denote the set of patterns to which a fraction of team i is assigned.

In case the master problem is infeasible, we need to check if this is because of missing patterns or because the branching constraints make the problem infeasible. To do this, we solve a CP model similar to the model for finding an initial feasible

solution presented in Section 7.3.1 - but with the branching constraints added. If this model has a feasible solution, we add the patterns used by each of the teams and re-solve the master problem. Otherwise, we return to Step 1 of the algorithm and choose a new node in the search tree.

7.3.4 Pricing Problem

When the master problem has been solved we use a pricing problem for finding patterns with negative reduced costs. For a general optimization problem

$$\begin{aligned} \min\ & cx \\ \text{s.t.}\ & Ax = b \\ & x \geq 0 \end{aligned}$$

where c and x are n-vectors, A is an $m \times n$ matrix and b is an m-vector, the reduced cost of a variable x_i is $c_i - \bar{u}A_i$ when $\bar{u}$ is an optimal dual solution.

We let $\bar{u}^1_i$ for all $i \in T$, $\bar{u}^2_{i_1 i_2 s}$ for all $i_1, i_2 \in T$, $i_1 < i_2$ and $s \in S$ where $TT_{i_1 s} = i_2$ and $\bar{u}^3_b$ for all $b \in B^\eta$ denote optimal dual variables corresponding to constraints (7.3.6), (7.3.7) and (7.3.8) from the master problem, respectively. Furthermore, we let $\bar{u}^2_{i_2 i_1 s} = \bar{u}^2_{i_1 i_2 s}$ for all $i_1, i_2 \in T$ with $i_1 < i_2$ and $s \in S$ where $TT_{i_1 s} = i_2$. Then the reduced cost of a pattern $\hat{j}$ used by team $\hat{i}$ in node η can be written as follows.

$$d_{\hat{i}\hat{j}} - \bar{u}^1_{\hat{i}} - \sum_{i \in T} \sum_{\substack{s \in S \\ TT_{\hat{i}s} = i}} \bar{u}^2_{i\hat{i}s} h_{\hat{j}s} - \sum_{b \in B^\eta} \bar{u}^3_b h_{js_b}$$

In order to find patterns with negative reduced costs, we use a pricing problem for each team. The pricing problem finds the pattern with the smallest reduced cost and the pattern is added to the master problem if the reduced cost is negative.

To solve the pricing problem, we use an IP model since we want to minimize the reduced cost. Alternatively, a CP model could be used to generated patterns with negative reduced costs but we have obtained the best results when the reduced cost is minimized.

To formulate the IP model for a team $\hat{i}$, we use a binary variable h_s for each $s \in \{0, \ldots, 2n-1\}$ and an integer variable d_s for each $s \in S^0$. The variable h_s is 1 if the pattern has a home game in slot s and 0 if it has an away game, while d_s is the travel distance of team $\hat{i}$ between slot s and $s+1$. The IP model for team $\hat{i}$ is presented below.

$$\min \sum_{s \in S^0} d_s - \bar{u}^1_{\hat{i}} - \sum_{i \in T} \sum_{\substack{s \in S \\ TT_{\hat{i}s} = i}} \bar{u}^2_{i\hat{i}s} h_s - \sum_{b \in B^\eta} \bar{u}^3_b h_{s_b} \tag{7.3.10}$$

$$\text{s.t.} \quad d_s \geq (1 - h_s - h_{s+1}) D_{TT_{\hat{i}s} TT_{\hat{i}s+1}} \qquad \forall s \in S^0 \tag{7.3.11}$$

$$d_s \geq (h_s - h_{s+1}) D_{\hat{i} TT_{\hat{i}s+1}} \qquad \forall s \in S^0 \tag{7.3.12}$$

$$d_s \geq (-h_s + h_{s+1}) D_{TT_{\hat{i}s} \hat{i}} \qquad \forall s \in S^0 \tag{7.3.13}$$

$$h_0 = 1 \tag{7.3.14}$$

$$h_{2n-1} = 1 \tag{7.3.15}$$

$$\sum_{s \in S} h_s = n - 1 \tag{7.3.16}$$

$$h_{s_1} + h_{s_2} = 1 \qquad \forall s_1, s_2 \in S, \;\; s_1 < s_2, \;\; TT_{\hat{i}s_1} = TT_{\hat{i}s_2} \tag{7.3.17}$$

$$\sum_{s=\hat{s}}^{\hat{s}+UB} h_s \leq UB \qquad \forall \hat{s} \in \{1, \ldots, 2(n-1) - UB\} \tag{7.3.18}$$

$$\sum_{s=\hat{s}}^{\hat{s}+UB} h_{is} \geq 1 \qquad \forall \hat{s} \in \{1, \ldots, 2(n-1) - UB\} \tag{7.3.19}$$

$$h_s \in \{0, 1\} \qquad \forall s \in \{0, \ldots, 2n-1\} \tag{7.3.20}$$

$$d_s \in \mathbb{Z}_+ \qquad \forall s \in S^0 \tag{7.3.21}$$

The objective function calculates the reduced cost of the pattern given by the h_s variables. Constraints (7.3.11) - (7.3.13) are used to calculate the distance, constraints (7.3.14) and (7.3.15) make sure that $\hat{i}$ starts and ends home and constraints (7.3.16) - (7.3.19) state the general constraints for a pattern. We refer to the IP model presented in Section 7.1.1 for further explanation of the constraints.

The pricing problem is solved for each team $\hat{i} \in T$ and in case the solution value is less than zero, we add the pattern given by the h_s variables to $P_{\hat{i}}$. When the pricing problem has been solved for all teams, we re-solve the master problem if patterns have been added and otherwise, we go to Step 4 of the algorithm.

Instead of using the optimization problem outlined above we have also tested the effect of using a CP feasibility model to find patterns. In this approach we were able to find a number of patterns with negative reduced costs instead of a single pattern. However, the CP model were not able to ensure that the pattern with the lowest reduced cost were found and this lead to a higher number of iterations. In addition the extra patterns made the master problem larger and the overall performance of the algorithm decreased.

7.3.5 Branching Strategy

In case the optimal solution of the master problem is fractional and no patterns with negative reduced costs exist, the algorithm branches to obtain an integer solution. Instead of branching on one of the fractional x values from the master

problem, we use higher order branching. We choose a team $\hat{i}$ and a slot $\hat{s}$ and create two new nodes by letting team $\hat{i}$ play home in slot $\hat{s}$ in one of the nodes and away in the other.

In order to find $\hat{i}$ and $\hat{s}$, we have implemented two strategies. The first strategy starts by finding the team $\hat{i}$ and pattern $\hat{j}_1$ which result in the highest fractional $\bar{x}_{\hat{i}\hat{j}_1}$ value such that

$$\bar{x}_{\hat{i}\hat{j}_1} = \max\{\bar{x}_{ij} | i \in T, j \in \bar{P}_i : 0 < \bar{x}_{ij} < 1\}.$$

Then it finds the pattern $\hat{j}_2$ which results in the second highest fractional value for team $\hat{i}$ such that

$$\bar{x}_{\hat{i}\hat{j}_2} = \max\{\bar{x}_{\hat{i}j} | j \in \bar{P}_{\hat{i}} : 0 < \bar{x}_{\hat{i}j} < 1 \wedge j \neq \hat{j}_1\}.$$

Finally, it finds a slot $\hat{s}$ where the two patterns $\hat{j}_1$ and $\hat{j}_2$ have a difference such that $h_{\hat{j}_1\hat{s}} \neq h_{\hat{j}_2\hat{s}}$.

The second strategy starts by finding the team $\hat{i}$ and pattern $\hat{j}_1$ such that the variable $\bar{x}_{\hat{i}\hat{j}_1}$ is as close to 0.5 as possible. It then finds a second pattern $\hat{j}_2$ different from $\hat{j}_1$ such that $\bar{x}_{\hat{i}\hat{j}_2}$ is as close to 0.5 and finally it finds a slot $\hat{s}$ such that $h_{\hat{j}_1\hat{s}} \neq h_{\hat{j}_2\hat{s}}$.

When we have found the branching team $\hat{i}$ and the branching slot $\hat{s}$ we can formulate the two branching cuts

$$\sum_{j \in P_{i_b}} h_{js_b} x_{i_b j} = 0 \qquad \sum_{j \in P_{i_b}} h_{js_b} x_{i_b j} = 1$$

where $i_b = \hat{i}$ and $s_b = \hat{s}$.

Now, we are ready to add two nodes η_1 and η_2 to the search tree. Assuming that the current node is node η, we let B^{η_1} and B^{η_2} be equal to B^{η} and add the first of the two branching constraints to B^{η_1} and the second to B^{η_2}.

7.4 Computational Results for the TCDMP

In order to explore the computational complexity of the TCDMP and to compare the proposed solution methods, we have tested all 4 methods on 60 instances ranging from 6 to 16 teams.

At the homepage http://mat.gsia.cmu.edu/TOURN/, Michael Trick's benchmark problems for the TTP can be found. These problems have been studied intensively and a number of solutions are presented at the web page. By letting $UB = 3$, using the presented distance matrices and permuting the slots of the presented TTP solutions, we have generated instances of the TCDMP.

For each even number of teams from 6 to 16 we have generated 10 instances by permuting slots of the following solutions. For 6 teams we have used the solution of Easton May 7, 1999; for 8 teams, the solution of Easton January 27, 2000; for 10 teams, the solution of Langford, June 13, 2005 and for 12, 14 and 16 teams, the solution of Zhang Xingwen August 28, 2002. All the tests have been performed on an Intel Xeon 2.67 GHz processor with 6 GB RAM. The IP and CP models have been solved by using OPL Studio [52] with the callable libraries CPLEX and Solver. The hybrid IP/CP approach and the branch and price algorithm have been implemented in OPL script.

The computational results are presented in Table 7.1. For each number of teams and each solution method, the table shows the number of instances solved and the minimum, average and maximum time used on the solved instances. We have used a time limit of 1800 seconds and instances which have not been solved within this time limit are not included in the average. Since we have 2 node selection strategies and two branching strategies for the branch and price algorithm, we present four versions of this solution method: BP-df-1, BP-df-2, BP-bv-1 and BP-bv-2. The terms df and bv refer to depth first and best value node selection, respectively, while 1 and 2 refer to the first and the second branching strategy.

Table 7.1 shows that, even though CP is better than IP at solving instances with 6 teams, it is not able to solve any of the instances with 8 teams. IP is doing a little better and is able to solve all the instances with less than 10 teams and a single instance with 10 teams. The fact that IP is able to handle larger instances than CP is no surprise, since IP models often excel compared to CP models when optimization problems are considered.

Both the hybrid IP/CP approach and the branch and price algorithm clearly outperform the IP and CP models. We see that the hybrid IP/CP approach shows the best results and, in addition to being the fastest on average, it is also the most stable of the solution methods. The only drawback of this method is a rather large memory consumption since all patterns are generated initially. For instances with 16 teams, it generates up to 85000 patterns and uses approximately 200 MB of memory.

The computation times of the branch and price algorithm are highly dependent on the size of the branching tree and we see that there is an order of magnitude in difference between the minimum and maximum time. This means that the algorithm is only competitive with the hybrid IP/CP approach when an integer solution is found relatively fast in the branching tree.

In addition to the tests presented here, we have tested the solution methods on instances with 18 teams but none of the methods were able to solve these instances within the given time limit. In this case the hybrid IP/CP approach generated up to 246000 patterns. Still, the tests have shown that the hybrid IP/CP approach is capable of solving the problem for practical applications like the National League Baseball which consists of 14 teams.

Table 7.1: Computational Results for the TCDMP.

n	Solution method	Number solved	Time (s) Min.	Avg.	Max.
6	IP	10	0.42	22.51	217.18
6	CP	10	8.03	17.00	24.94
6	IP/CP	10	0.06	0.07	0.08
6	BP-df-1	10	0.83	1.59	3.34
6	BP-bv-1	10	0.82	1.27	2.29
6	BP-df-2	10	0.84	1.74	3.43
6	BP-bv-2	10	0.83	1.25	2.19
8	IP	10	60.72	397.35	954.56
8	CP	0	–	–	–
8	IP/CP	10	0.41	0.44	0.47
8	BP-df-1	10	2.87	11.74	21.65
8	BP-bv-1	10	2.88	6.99	11.90
8	BP-df-2	10	2.83	10.05	22.94
8	BP-bv-2	10	2.82	6.15	8.19
10	IP	1	1273.74	1273.74	1273.74
10	CP	0	–	–	–
10	IP/CP	10	1.79	2.02	2.34
10	BP-df-1	10	7.23	27.01	157.56
10	BP-bv-1	10	7.05	16.55	46.50
10	BP-df-2	10	6.98	24.66	64.49
10	BP-bv-2	10	7.03	15.51	44.34
12	IP	0	–	–	–
12	CP	0	–	–	–
12	IP/CP	10	10.16	12.19	18.19
12	BP-df-1	10	24.29	281.60	1386.09
12	BP-bv-1	10	23.77	130.31	434.97
12	BP-df-2	10	23.80	243.23	1038.98
12	BP-bv-2	10	23.56	151.74	650.79
14	IP	0	–	–	–
14	CP	0	–	–	–
14	IP/CP	10	35.52	38.07	42.83
14	BP-df-1	10	49.62	95.00	248.32
14	BP-bv-1	10	48.61	91.71	240.47
14	BP-df-2	10	48.69	165.40	750.18
14	BP-bv-2	10	48.74	72.78	164.82
16	IP	0	–	–	–
16	CP	0	–	–	–
16	IP/CP	10	153.80	197.16	260.47
16	BP-df-1	9	122.47	866.13	1770.94
16	BP-bv-1	9	119.37	827.90	1645.91
16	BP-df-2	8	121.42	662.81	1377.21
16	BP-bv-2	9	119.38	631.30	1382.13

7.5 The Circular Traveling Salesman Approach

In this section we present a new heuristic solution method for solving the TTP. This method, called the *circular traveling salesman approach* (CTSA), obtains solutions by combining solutions for the traveling salesman problem with solutions for the circular distance TTP. Furthermore, we show how the CTSA can be extended by solving the TCDMP afterwards in which case we will refer to it as the *extended circular traveling salesman approach* (ECTSA).

One of the main problems when solving the TTP is the fact that the travel distance is heavily affected by both the timetable and the pattern set. This makes it very hard to find a good decomposition of the problem and without a decomposition approach, the problem is too hard to solve using exact solution methods even for instances with only eight teams. This has led to a number of highly specialized metaheuristic solution methods capable of finding very good solutions for all the instance classes of the TTP outlined at [88]. However, such algorithms are often cumbersome to implement and it may take a lot of fine tuning to obtain the best schedules.

The strength of the CTSA presented here is the fact that it is very simple to implement, solutions are found almost instantaneously and the solutions are not far from the best solutions currently found for the benchmark instances of the TTP. The basic idea is to use a two step approach. Step 1 approximates an instance of the TTP by an instance of the circular distance TTP, while Step 2 uses a solution for the circular distance TTP to obtain a solution for the TTP. The approximation is performed by finding the optimal traveling salesman tour through all the venues and then distributing the teams evenly on a circle with circumference n according to the order of the traveling salesman tour. This forms an instance of the circular distance TTP and by using a solution for this instance we have a schedule for the original TTP. The approach can be outlined as follows.

The circular traveling salesman approach.

Step 1 Solve the traveling salesman problem for the teams in the tournament.

Step 2 Solve the circular distance TTP with teams ordered according to the solution from Step 1 and use the resulting schedule to calculate the travel distance when the real distance matrix is used.

The TSP from Step 1 is solved using a standard solution method. The problem is solved without the subtour elimination constraints and, in case the solution contains subtours, cuts are added and the problem is solved again. This process continues until a feasible solution has been obtained.

In Step 2, we do not actually solve the circular distance TTP since the currently best solutions for the problem, presented at [88], can be used. In this step, the

team ordering 1,2,. . . ,n and the ordering n,1,2,. . . ,n-1 leads to the same objective value of the circular distance TTP - but they result in different schedules and hence different travel distances for the original TTP. This means that we are able to obtain $2n$ solutions from the TSP solution by rotating the original ordering and reversing the ordering. For example, the ordering 1,2,3, also leads to the orderings: 2,3,1; 3,1,2 and the reverse orderings 3,2,1; 2,1,3 and 1,3,2.

The CTSA has been tested on the benchmark instances: NL6, NL8,. . . , NL16 which can be found at [88] together with corresponding solutions. In Step 2 we need solutions for the circular distance TTP with 6 to 16 teams. For the instance with 10 teams, we use the solution by Langford found at [88] and for all other instances, we use the solutions obtained by Lim et al. [54]. In order to examine the effect of the $2n$ orderings, we have solved the problem for each ordering. The results are displayed in Table 7.2 which gives the best known upper and lower bound on the optimal travel distance, together with the maximum, minimum and average travel distance found by the CTSA for the $2n$ orderings. The solution method has been implemented in OPL script and it took less than 0.5 second to solve the TSP in all instances.

Table 7.2: Computational results of the CTSA

			Travel distance		
Instance	LB	UB	Minimum	Maximum	Average
NL6	23916	23916	24467	26472	25559.8
NL8	39479	39721	41754	44028	42822.5
NL10	57500	59436	63844	67943	66106.7
NL12	107483	111248	116598	124975	121339.8
NL14	182797	189759	216659	230463	223925.1
NL16	248852	267194	288674	307925	295858.3

Keeping the simpleness of the CTSA in mind, the solutions are surprisingly close to the current upper bound. In addition, to be used as final schedules, the solutions can also be used as initial starting points in some of the metaheuristic approaches for the TTP. Lim et al. [54] use a beam search algorithm for obtaining initial solutions in their simulated annealing algorithm. But although the solutions obtained by the CTSA can be found almost instantaneously, the quality is comparable to the beam search. For comparison, Table 7.3 displays the computational results of the beam search algorithm reported in [54] found on a 2.53 GHz Pentium 4 PC with 512 MB of RAM.

In order to improve the solutions of the CTSA, it is possible to add a Step 3 in which the pattern set is improved if possible. This is done by solving the

Table 7.3: Computational results of the beam search algorithm reported in [54].

	Travel distance			Time (s)		
Instance	Minimum	Maximum	Average	Minimum	Maximum	Average
NL6	24579	25146	24802.8	576	669	628.5
NL8	41265	42334	41852.6	3211	3443	3345.0
NL10	63337	65856	64544.2	5801	6299	6179.6
NL12	118047	123770	120528.8	8009	8730	8594.0
NL14	202916	208797	205929.3	10332	10947	10806.4
NL16	283795	295864	289235.2	12855	13889	13631.2

TCDMP, given the original distance matrix and the timetable found in Step 2, to obtain the optimal pattern set given the timetable. The results for this extended version ECTSA are reported in Table 7.4 together with the computation times. Notice that the computation times are mainly due to the time needed to solve the TCDMP.

Table 7.4: Computational results of the ECTSA.

	Travel distance			Time (s)		
Instance	Minimum	Maximum	Average	Minimum	Maximum	Average
NL6	24467	26255	25401	0.05	0.19	0.07
NL8	41754	44028	42822	0.36	0.39	0.37
NL10	63277	66080	64635	2.47	2.74	2.55
NL12	116421	124588	120838	11.39	20.66	12.60
NL14	215665	230463	223586	48.71	58.07	50.43
NL16	288674	307925	295628	239.90	294.30	246.44

Since the pattern set and the timetable are already optimized in the solution of the circular distance, only marginal reductions can be obtained by solving the TCDMP. However, in case the obtained solutions will be used as final schedules, it might be worth spending the additional computation time required by the ECTSA. Although the ECTSA uses much more computation time than the CTSA it is still substantially faster than the beam search.

Chapter 8

Job Scheduling

In this chapter we move from sports scheduling to a job scheduling problem provided by the Danish telecommunications net operator, Sonofon. By the end of each day a large number of jobs (End-of-Day jobs) have to be processed on three available servers. Each job is preassigned to one of the three servers and the objective is to schedule the jobs on the machines in order to minimize makespan. This task is complicated by the facts that a large number of precedence constraints among the jobs must be fulfilled, time windows must be obeyed and capacity limitations must be respected. In addition, the jobs are *elastic* which means that the duration of a particular job depends on the capacity assigned to the job. Elasticity of jobs complicates the problem considerably and we believe that this is the first work considering elastic jobs in large-scale scheduling.

The applications of scheduling problems are wide spread, and hence a considerable amount of promising research has been devoted to such problems both within the operations research literature and the computer science literature. As already mentioned in Section 2.3.1, Jain and Grossmann [53] and Hooker [47] have applied logic-based Benders decomposition for solving job scheduling problems. In addition we also want to mention the work of Baptiste and Le Pape [5] and Baptiste, Le Pape, and Nuijten [6]. These methods work very well for small and medium-scale scheduling problems but the complexity often prevents the use of exact solution methods when large-scale problems are considered. Furthermore, the precedence constraints in this application also constitute an obstacle to using logic-based Benders decomposition since the logic-based Benders cuts applied in the previous work become infeasible when precedence constraints are present. As a consequence we have decided to use a heuristic solution method instead of an exact method in order to solve the problem.

Metaheuristic solution methods are a good alternative to the exact solution methods and they have proved to be very well suited for solving scheduling prob-

lems. Often they are able to find good solutions within a reasonable amount of time even for hard instances. We have chosen to use tabu search, outlined in Section 2.4.1 since it has previously been successfully applied to scheduling problems. The papers on tabu search applied to scheduling problems are numerous, but let us for brevity only mention a few which all appeared recently. Grabowski and Wodecki [39] consider large-scale flow shop problems with makespan criterion and develop a very fast tabu search heuristic focusing on a lower bound for the makespan instead of the exact makespan value. Ferland, Ichoua, Lavoie, and Gagné [31] consider a practical problem of scheduling internships for physician students and propose several variants of tabu search procedures. The last three papers all consider the problem of scheduling a number of jobs to a set of heterogeneous machines under precedence constraints with the objective of minimizing makespan. In Porto, Kitajima, and Ribeiro [68] a parallel Tabu Search heuristic is developed and proved superior to a widely used greedy heuristic for the problem. In Chekuri and Bender [20] a new approximation algorithm is presented but, unfortunately, no computational results are reported. Finally, in Mansini, Speranza, and Tuza [56] jobs with up to three predecessors each are considered among groups of jobs requiring the same set of machines. The problem is formulated as a graph-theoretical problem. In the paper a number of approximation results are provided, but no computational experience is reported.

8.1 Problem Formulation

The Danish telecommunications net operator, Sonofon, faces a three-machine scheduling problem, with 346 End-of-Day jobs (EOD). Each job is dedicated to a particular server in advance and all the jobs must be processed without *preemption*. Preemption means that jobs can be interrupted during processing.

The scheduling time horizon runs from 7.00 pm to 8.00 am, and each job receives a time window in which it should be processed. The time windows are wide, leaving numerous feasible starting times for each job. Since most scheduling tools applying CP rely heavily on propagation techniques, the wide time windows have a negative influence on the performance of such scheduling packages. The time windows will be explored further in Section 8.2.1. Since the servers immediately after completing the EOD-jobs are assigned to other operations, the objective will be to minimize makespan.

Because of interrelations between jobs, a number of precedence constraints must be fulfilled. It might occur that a job needs information from a database to which another job (a predecessor) has written earlier.

In a real-world application each job can execute with varying capacity consumption during its runtime, as illustrated by job 1 in Figure 8.1(b). However, due to limitations of server exploitation, we can assume that each job has an upper

bound of capacity consumption. In Figure 8.1(a) we have illustrated a situation in which three jobs are placed at a machine to start processing at time 0. Each of the three jobs is assumed to have a maximal capacity consumption of 15 units, and the machine has capacity 30. Since all the jobs are scheduled to start at time 0, they must share the available capacity, as shown in Figure 8.1(b). Observe that in Figures 8.1(a) and (b) the two corresponding boxes for a job have the same area. This would be an incorrect representation of a real-world application, since in general (duration × capacity) increases with decreasing capacity, due to lost server efficiency from *swapping*. Swapping means time being spent for reading jobs into and out of the temporary memory, not processing any jobs. This fact is represented by the inclusion of the shaded area in Figure 8.1(c).

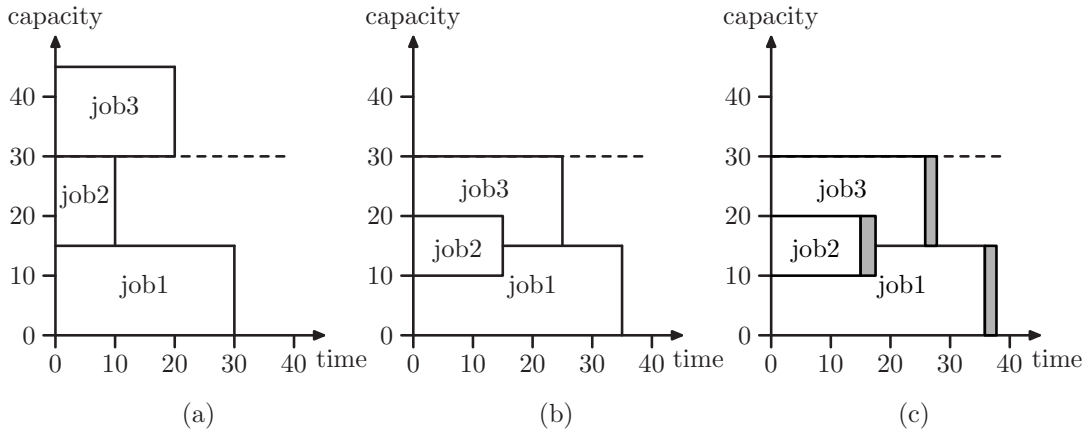

Figure 8.1: (duration × capacity) increases with decreasing capacity.

In our setup we shall assume that the capacity consumption for a job remains constant during its runtime but we do not restrict time and capacity consumption to be given beforehand. Instead we assume that jobs are elastic, and hence allow the time and capacity consumption to be found during the optimization process. This problem has a number of similarities with the problem of scheduling malleable tasks on parallel processors in which a number of processors can be assigned to each job, see Blażewicz, Machowiak, Węglarz, Kovalyov, and Trystram [14]. We deal with the non-linear functionality between time and capacity consumption by a rough approximation representing each job as a choice between three boxes, (see Figure 8.2).

The dimensions of the boxes for a given job j are explained in Table 8.1, where cap_j ($time_j$) corresponds to the capacity (duration) for the job-box having the least capacity consumption and hence the longest duration. The times we use constitute an average longest runtime provided by Sonofon from historical data. The second and third column gives the capacity and time consumption for a given

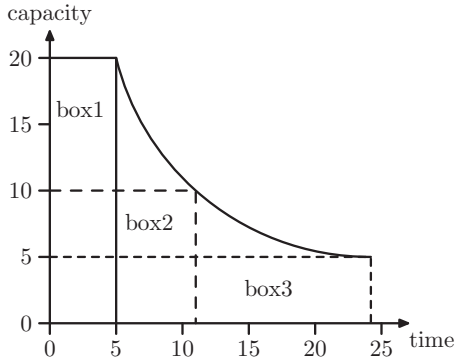

Figure 8.2: Three representations of a job.

box, and the last column gives (capacity consumption × duration). Notice that, by a 50% decrease in capacity, (capacity consumption × duration) increases by 10%. This trade-off between capacity assigned to a particular job and its duration was determined in correspondence with Sonofon and reflects the specific problem rather closely.

Table 8.1: Dimensions of boxes representing job j.

l	cap_{jl}	$time_{jl}$	$(cap_{jl} \times time_{jl})$
1	$4 \cdot cap_j$	$25/121 \cdot time_j$	$100/121 \cdot cap_j \cdot time_j$
2	$2 \cdot cap_j$	$5/11 \cdot time_j$	$10/11 \cdot cap_j \cdot time_j$
3	cap_j	$time_j$	$cap_j \cdot time_j$

Since the representation of scheduling problems is greatly simplified using the terminology from CP, we adapt this notation and present the problem as a CP model. However, to do that we need the following notation.

M = $\{1, 2, 3\}$ - Machines.
J = $\{1, \ldots, n\}$ - Jobs.
P = $\{(j, k)|$ job j shall precede job $k\}$ - Precedence constraints.
J_m = $\{j|$ job j shall be processed on machine $m\}$ - Job-machine constraints.
L = $\{1, 2, 3\}$ - Boxes for each job.

Notice $\cup_{m \in M} J_m = J$ since all jobs are allocated to a particular server in advance.

For each job j, we introduce the four variables

$j.start$ - Starting time,
$j.duration$ - Duration,

$j.end$ - Completion time,
$j.capacity$ - Capacity consumption,

connected by the implicit constraint $j.start + j.duration = j.end$. In addition, we have the parameters

R_m - Capacity available on machine m, $\forall m \in M$,
$[a_j, b_j]$ - Time window for job j, $\forall j \in J$,
$time_{jl}$ - Duration of the l'th box for job j, $\forall j \in J,\ \forall l \in L$,
cap_{jl} - Capacity consumption of the l'th box for job j, $\forall j \in J,\ \forall l \in L$.

By introducing the artificial job *makespan* with zero duration, and letting x_{jl} denote a binary variable which is 1 if box l is chosen for job j and 0 otherwise, we can now formulate the problem as the following CP model.

$$\begin{aligned}
\min\quad & makespan.end \\
\text{s.t.}\quad & \sum_{l \in L} x_{jl} = 1 && \forall j \in J && (8.1.1)\\
& j.duration = \sum_{l \in L} (x_{jl} \cdot time_{jl}) && \forall j \in J && (8.1.2)\\
& j.capacity = \sum_{l \in L} (x_{jl} \cdot cap_{jl}) && \forall j \in J && (8.1.3)\\
& a_j \le j.start && \forall j \in J && (8.1.4)\\
& j.end \le b_j && \forall j \in J && (8.1.5)\\
& j\ \ precedes\ \ makespan && \forall j \in J && (8.1.6)\\
& j\ \ precedes\ \ k && \forall (j,k) \in P && (8.1.7)\\
& cumulative \begin{pmatrix} \{j.start\}_{j \in J_m} \\ \{j.duration\}_{j \in J_m} \\ \{j.capacity\}_{j \in J_m} \\ R_m \end{pmatrix} && \forall m \in M && (8.1.8)\\
& x_{jl} \in \{0,1\} && \forall j \in J,\ \forall l \in L && (8.1.9)
\end{aligned}$$

where *cumulative* is a global constraint in CP, stating that, at all times, the total capacity is not exceeded by the capacity consumption of running jobs. The constraint can be rewritten as

$$\begin{aligned}
& cumulative\left((t_1,\ldots,t_n),(d_1,\ldots,d_n),(r_1,\ldots,r_n),R\right) \\
\Updownarrow\quad & \\
& \sum_{\{j \mid t_j \le t \le t_j + d_j\}} r_j \le R \quad \forall t
\end{aligned}$$

where the vector $(t_1, \ldots, t_n)$ represents starting times of jobs $1, \ldots, n$, with duration $(d_1, \ldots, d_n)$ and capacity consumption $(r_1, \ldots, r_n)$. Available capacity is R.

The above constraints (8.1.1) choose a box for each job, yielding a specific time and capacity consumption in cooperation with (8.1.2) and (8.1.3). Constraints (8.1.4) and (8.1.5) consider time windows. Constraints (8.1.6) together with the objective function minimize the completion time of the last job. Constraints (8.1.7) handle precedence constraints (j precedes k means $j.end \leq k.start$), whereas constraints (8.1.8) handle resource consumption for each machine.

8.2 Tabu Search

We use tabu search for solving the problem since the problem size prevent us from using an exact solution method like IP or CP. As described in Section 2.4.1 tabu search is a trajectory metaheuristic exploring the search space in order to find good solutions. In this problem a solution consist of a starting time and a box choice for each job since then the completion times, the durations and the capacity consumptions are implicitly determined. However, due to wide time windows numerous possible starting times exist for each job. This results in a huge search space and the tabu search face the risk of moving a few jobs without changing the job sequence. Instead we define a solution to be a box size for each job and a *job sequence* which specifies the order of the starting times. Given a sequence we let j_p denote the number of the job at position p and we impose constraints saying that job j_{p_1} must start no later than job j_{p_2} whenever $p_1 < p_2$. A solution to a problem with 9 jobs is shown in Figure 8.3 where the sequence is defined by $j_1 \ldots j_9$ and the box choices by the box numbers l_{j_p} stated below.

j_p	4	2	6	5	9	3	7	1	8
l_{j_p}	2	1	1	3	1	2	2	3	1
p	1	2	3	4	5	6	7	8	9

Figure 8.3: Sequence and box choices for example with 9 jobs.

The tabu search moves from one solution to the next by either changing the sequence or changing one of the box sizes. This is discussed in Section 8.2.4 which outlines the neighbourhood structure. Since the solutions considered in the tabu search does not include the starting times explicitly we need to be able to extend such a partial solution to a solution which explicitly states all the starting times. In Section 8.2.3 we show how to find to optimally extend a partial solution.

A solution is feasible, if it is possible to schedule all jobs according to the sequence and the box sizes and still satisfy all time windows, capacity constraints

and precedence constraints. It turns out that the problem of finding an initial solution is very hard, but a heuristic method for solving this problem is presented in Section 8.2.2.

Elements and features of the tabu search such as the neighbourhood, tabu lists, intensification strategies and diversification strategies are discussed in Sections 8.2.4, 8.2.5, 8.2.6 and 8.2.7, respectively. Part of the notation is inhereted from Chiang and Russell [21].

8.2.1 Preprocessing

In order to detect infeasible solutions quickly we tighten the time windows by considering precedence constraints. If a job j, has a time window $(0, t)$, but at the same time is a successor of another job $\hat{j}$, then the time window can be adjusted to start at the earliest completion time for job $\hat{j}$. To do this, a *precedence graph* G is constructed where all jobs are represented by a node, and all precedence constraints by a directed arc between the two nodes involved, pointing away from the predecessor.

For all connected components in the graph the following procedure adjusts the beginning of the time windows. Let $C \subseteq G$ be a connected component, and let $j \in C$ be a job in C. Then a_j denotes the earliest starting time, and $time_{j1}$ denotes the minimal duration for job j. We let P_j denote all predecessors of job j, note $P_j \subset C$. The earliest starting times for the jobs in C are now adjusted by setting $a_j = \max\{a_j, a_i + time_{i1} \ \forall i \in P_j\}$ for all $j \in C$, but in an order such that all predecessors of j have been adjusted before j. Such an order exists, since otherwise a directed cycle would exist, and the jobs would be impossible to schedule. The latest completion times can be adjusted in a similar manner by starting with the jobs in C having no successors.

8.2.2 Initial Solution

Garey and Johnson [33] have shown that, for a similar setup, the decision problem on determining the existence of a feasible schedule with a makespan less than a given deadline (in our case 8.00 am) is $\mathcal{NP}$-complete. In this section we shall describe a heuristical procedure to generate an initial feasible solution for this particular instance. The procedure is divided into five steps where Steps 1,2 and 3 use the precedence graph to generate a sequence. In Step 4, box sizes are chosen and in case the resulting solution is feasible the procedure stops. Otherwise, Step 5 relaxes the problem and uses the tabu search to find a feasible solution.

Procedure for finding initial solution.

Step 1:
Notice, to obtain a feasible solution, three groups of constraints must be fulfilled

simultaneously, namely precedence constraints, time window constraints and capacity constraints. To ensure fulfilment of the precedence constraints, we use the precedence graph described in Section 8.2.1 to divide the jobs into *layers*. The successor of a job will always be in a higher layer than the job itself, and the jobs in one layer cannot start before all jobs in preceding layers have started.

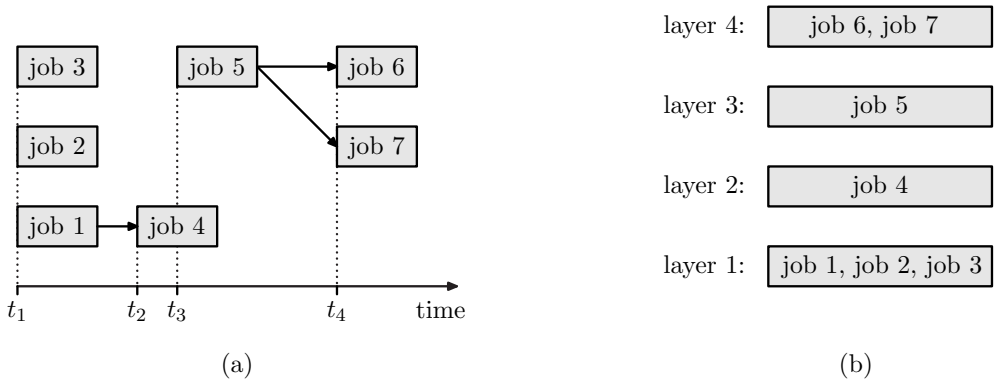

Figure 8.4: How jobs are divided into layers.

Figure 8.4(a) illustrates an example with seven jobs which must be divided into layers. In the figure the jobs are placed at their earliest starting time indicated by dotted lines and precedence constraints between jobs are indicated by arrows. In this situation job 4 must be in a higher layer than job 1 since it is a successor of job 1. However, if jobs are only divided according to the precedence constraints the process faces the risk of assigning jobs with late time windows to an early processing layer. This could happen if an entire component of the precedence graph has to be processed after a certain time, but the first job is assigned to layer 1. Jobs from other components, which could be processed early, would then be stalled if they were in layer 2, and the entire schedule would be delayed. This corresponds to assigning job 5 from Figure 8.4(a) to layer 1. If that happens job 4 must be scheduled after time t_3 since it would be allocated to a higher layer than job 5. Hence in our derivation of layers, we introduce a variable *start* and initialize it to 0. Then we assign jobs which are without predecessors and capable of starting before or at time *start* to layer 1. In Figure 8.4(a) job 1, job 2 and job 3 would hence be assigned to layer 1 as seen in Figure 8.4(b). All their successors, having no other predecessors and being able to start before or at time *start*, are then scheduled in the next layer etc. When no more jobs can be assigned due to either time window constraints or precedence constraints, the variable *start* is increased by one time unit, and a new level of layers can be derived with jobs being able to start before the new limit.

The jobs are numbered consecutively, starting with the jobs at the lowest layer. The difference between the width of the time window and the duration of

the smallest box expresses a degree of freedom for a given job. The higher this difference is, the higher degree of freedom the job possesses. Within each layer the jobs are numbered in an increasing order of this degree of freedom. This continues in an iterative fashion, until all jobs have been numbered, and we have a sequence containing all jobs. After the first three jobs from Figure 8.4(a) has been assigned to layer 1, the variable *start* is increased until it is equal to t_2 at which point job 4 can be assigned to layer 2. As the algorithm continues the remaining jobs will be assigned as shown in Figure 8.4(b).

Step 2:
This step is similar to Step 1 except the layers are generated backwards. This means that the layer containing the last jobs are generated first, and then the preceding layers are generated one by one. Again the successor of a job will always be in a higher layer than the job itself, and the job in one layer cannot start before all jobs in the preceding layer have started.

Step 3:
The sequence from Step 1 has the disadvantage that all jobs without precedence constraints and time windows are scheduled in the first layer, e.g. job 2 and job 3 in Figure 8.4(a). This means that jobs which could have been scheduled later might delay some of the large components of the dependency graph. The sequence developed in Step 2 has the opposite problem since in this case the jobs with few constraints are scheduled in the last layer and might cause jobs to break their time windows. Hence in this step we obtain a new sequence by taking a convex combination of the two sequences from Steps 1 and 2. This is done by calculating the convex combination of the positions in the two sequences for each job and then generating a sequence according to these numbers. Ties are broken arbitrarily. Notice that the new sequence still satisfies all precedence constraints.

Step 4:
First we choose a box size for each job j on machine m according to the following scheme:

$$\begin{aligned} x_{j1} &= 1 \quad \text{if } 0 < cap_{j3} \leq \tfrac{R_m}{10} \\ x_{j2} &= 1 \quad \text{if } \tfrac{R_m}{10} < cap_{j3} \leq \tfrac{R_m}{4} \\ x_{j3} &= 1 \quad \text{if } \tfrac{R_m}{4} < cap_{j3} \end{aligned}$$

These choices have proven efficient in the particular problem. After the boxes have been chosen a check is made to see if the sequence obtained in Step 3 together with the box choices constitute a feasible solution. This check is performed during the process of completing the solution as described in Section 8.2.3.

Step 5:
If the solution from Step 4 is infeasible we use the tabu search to find a feasible

solution. The problem is relaxed by setting $b_j = \infty$ for all j, i.e. the time windows have no upper limit. Notice that this problem always has a feasible solution when the capacity requirement for each job is less than the capacity on the corresponding machine. The objective in this part of the tabu search is to minimize the number of jobs which violate their original time windows, and the search stops when a solution with value 0 has been found.

The implemented idea corresponds to running a tabu search procedure in two phases – one phase ensuring feasibility and one phase minimizing makespan. These phases could alternatively be merged by using weights to yield an objective function combined of the makespan criterion and a penalty for the broken time windows, see Chiang and Russell [21]. However, since an implementation of this idea showed poor performance compared to the two phase tabu search, we chose to focus on the two phase approach.

8.2.3 Completing a Solution

As mentioned the solutions used in the tabu search only consist of box choices and a job sequence in order to reduce the search space. However, to be able to check feasibility of a solution we need to check whether the time windows and the capacity constraints are satisfied. Hence, we need to be able to extend a solution from the tabu search to a complete solution including starting times, completion times and capacity consumption for all jobs. In this section we outline a procedure capable of completing a partial solution in such a way that the makespan is minimized. This procedure is used for all considered moves in the tabu search and hence the efficiency of the procedure has great influence on the overall performance of the tabu search.

The procedure exploits the fact that an optimal schedule with respect to the given sequence and box choices can be generated by scheduling one job at a time in the order of the job sequence without backtracking. Notice that, the job sequence always satisfy the precedence constraints. Since j_p is the job at position p in the sequence we know that when j_p is about to be scheduled all jobs $j_{\bar{p}}$ with $\bar{p} < p$ have been scheduled and $j_{p-1}.start \leq j_p.start$ due to the sequence. Furthermore, all the jobs that have been scheduled so far, start before or at $j_{p-1}.start$ and therefore the capacity consumption on each machine must be decreasing in time after $j_{p-1}.start$. The optimal starting time for j_p will hence be the first time after $\max\{a_{j_p},\ j_{p-1}.start\}$ and $\max\{j_{\bar{p}}.end | (j_{\bar{p}}, j_p) \in P\}$ for which the capacity consumption, on the machine m used to process j_p, is less than or equal to $R_m - j_p.cap$. This means that a job is started the first time the four conditions shown in Figure 8.5 are fulfilled.

When the starting time of j_p has been determined the procedure checks if $j_p.end \leq b_{j_p}$ to see if the time window constraint is satisfied. If so, j_{p+1} is scheduled and otherwise the solution is infeasible and the procedure stops. If all jobs are

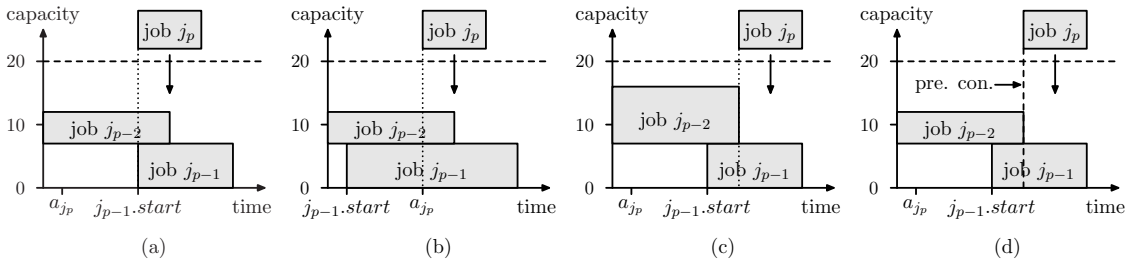

Figure 8.5: Scheduling the job j_p. The limiting constraints are: (a) The sequence, (b) time window, (c) capacity, (d) precedence constraint.

scheduled we have a feasible solution since all constraints are satisfied, and the makespan is equal to $\max\{j.end|j \in J\}$.

8.2.4 Neighbourhood

To characterize the neighbourhood of a given solution $\bar{x}$ we define two kinds of moves. A *position move* moves a job to a new position while the box sizes are kept constant, whereas a *box move* maintains the job sequence of $\bar{x}$ but changes the box choice for a single job. Figure 8.6 illustrates both kinds of moves. Notice in Figure 8.6(a) that, when job 9 at position 5 in the job sequence is moved to position 2, not only does job 9 get a new position, but the jobs at position 2, 3 and 4 are moved to the subsequent position.

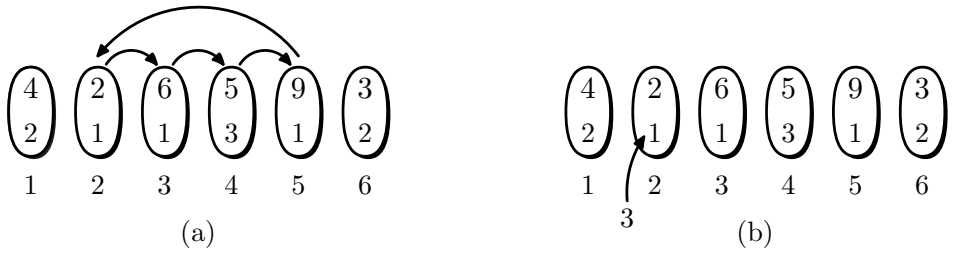

Figure 8.6: (a) Position move, (b) Box move.

The position move described above has been chosen instead of alternatives, such as exchanging two jobs, since the precedence constraints does not limit the flexibility of this move. Consider the move from Figure 8.6(a) and imagine that job 2 must precede job 6 and job 6 must precede job 5. In that case three "exchanges" of jobs are needed to perform the single position move shown in the figure.

The neighbourhood for solution $\bar{x}$ can be characterized as the union of solutions obtained by a single box move and solutions obtained by a single position move which fulfils the precedence constraints. The cardinality of the neighbourhood is

$O(n^2)$ due to the large number of position moves, and in the present implementation we must consider approximately 120,000 moves (some are ignored due to violation of the precedence constraints) for each solution. The ability to select only part of the neighbourhood for examination is therefore crucial. We use two methods for limiting the number of possible moves.

Restricting Position Moves

By introducing a limit *movelimit* on how far a job can move, the number of considered position moves are reduced. This leads to faster iterations but might restrict the search from choosing some very good solutions. To avoid the search from stalling due to the restriction, the entire neighbourhood is examined every time the algorithm has performed non-improving moves for a predefined number of iterations. This makes the search capable of performing a single time consuming move and then a number of fast iterations to exploit the new conditions.

Candidate Lists

The *Elite Candidate List* approach (see Glover and Laguna [36]), is used to limit the number of position moves by only evaluating moves belonging to candidate lists. In this setup two lists are used, and they are constructed by evaluating the neighbourhood of the initial solution. All moves which lead to an improving makespan are stored in list 1, and all moves leading to the same makespan are stored in list 2. In the following iterations only moves from the two candidate lists are considered. First the moves in list 1 are evaluated, and if one of these moves leads to an improving makespan the best move is chosen. If list 1 does not contain an improving move the moves in list 2 are evaluated, and the best move considering both list 1 and list 2 is chosen.

When a move has been chosen from one of the candidate lists both lists are updated by deleting all moves conflicting with the chosen one. This means that, if a position move for job j is chosen, then all other position moves for job j are deleted from the candidate lists and correspondingly for box moves. The candidate lists are used until no improving move has been found in the lists. When this happens both lists are deleted, and two new lists are generated by examining the possible moves of the current solution. Notice that, this might not be an evaluation of all possible moves, since the position moves might be restricted as explained in Section 8.2.4. The underlying assumption of the strategy is that a move which performs well in the current solution will probably also lead to improvements in the following iterations.

8.2.5 Tabu List

The corner stone in tabu search is the use of short-term memory by generating a tabu list. The tabu list stores the move from an iteration and keeps it for $TimeInTL$ iterations. This is done by keeping the iteration number $\hat{i}$ from the iteration in which the move is made tabu and deleting the move from the tabu list when the iteration number exceeds $\hat{i} + TimeInTL$. The tabu list differentiates between the two kinds of moves, but the number of the job involved is always stored. If a box move is performed for job j the tabu list restricts job j from performing a new box move in the following $TimeInTL$ iterations, unless the *aspiration criterion* is satisfied. If a position move is moving job j from position p the tabu list restricts the search from performing a new position move taking job j to a position $\bar{p}$ where $|p - \bar{p}| \leq tabuPosLimit$ in the following $TimeInTL$ iterations, unless the aspiration criterion is satisfied.

The aspiration criterion implemented checks if an improved makespan can be obtained by performing a forbidden move. If this is the case the tabu restriction is suspended, and the search is allowed to perform the move.

The tabu search implemented here has the ability to dynamically adjust the variable $TimeInTL$ which determines the number of iterations for which a move is tabu. $TimeInTL$ is decreased by the parameter $zdecrease = 0.9$ every time the search is trapped in a solution without a non-tabu or feasible neighbour and increased by $zincrease = 1.1$, when the same makespan has been found in many successive iterations.

In addition a variable *steps* is counting the number of moves without a change in $TimeInTL$, and $TimeInTL$ is decreased by $zdecrease$ if *steps* exceeds a fixed threshold *movingaverage*. This adjustment helps the search to avoid a lot of bad moves which could be the result of a long tabu list.

8.2.6 Intensification Strategy

A list $IntenArray$ holds moves which have led to improvements of the makespan. The moves are kept for $Intensize$ iterations, and corresponding moves for the same job are not allowed while the move is in the $IntenArray$. For example, if a position move is performed for job j in iteration $\hat{i}$, a new position move cannot be performed for j before iteration $\hat{i} + Intensize$. However, the intensification status is not considered if a job satisfies the aspiration criterion. In this case the job can be chosen even though the move is in the intensification array.

8.2.7 Diversification Strategies

The algorithm contains two kinds of diversification strategies. The first strategy is active throughout the search and helps the algorithm to perform a thorough

search in the current region of the solution space, while the other strategy forces the search to change the region.

Penalized Move Value

The quality of a move is measured by *movevalue*, which gives the difference between the current makespan and the makespan obtained by performing the move, $movevalue = newTime - curTime$. This *movevalue* could be used to guide the search, but in order to implement the first diversification strategy a *penalized move value pmv* is introduced. The *pmv* takes into account how many times the job has been moved before:

$$pmv = \begin{cases} movevalue + \alpha \cdot Move[j], & \text{if } movevalue \geq 0 \\ movevalue, & \text{if } movevalue < 0 \end{cases}$$

where $Move[j]$ counts the number of moves performed by job j and α is a parameter to adjust the penalty. By choosing moves according to lowest *pmv*, the algorithm automatically follows the diversification strategy.

Escape Procedure

In order to move the search from one region of the solution space to another, an escape procedure is invoked when too many successive iterations have resulted in the same makespan. The procedure makes a number of random moves which lead the algorithm away from the current region. During the escape procedure only feasible moves are allowed, since a feasible solution must be available when all the moves are performed. The general tabu search procedure adjusted according to the strategies above can be seen in Figure 8.7.

8.3 Computational Results

In this section we present the computational results of the tabu search. In addition to solving the problem faced by Sonofon we have performed extensive testings on random large-scale scheduling instances. The results for the practical application show that significant improvements can be gained within a short amount of time while the additional tests show the robustness and speed of the algorithm to instances with varying structure.

The results obtained by the tabu search are compared to a lower bound which is found by disregarding the precedence constraints. This allows us to schedule the three machines independently. Then for each job we let *total capacity consumption* be (capacity consumption $\times$ duration) for the smallest possible box (box 1). Now, for a particular machine we are able to construct a sequence by ordering the jobs according to the starting time of their time windows, ties are broken arbitrarily.

```
1  procedure tabu search
2      time = 0
3      adjust time windows (8.2.1)
4      find initial solution (8.2.2)
5      iteration = 0
6      while ((iteration < maxiteration) ∧ (time < timelimit)) do
7          curmove = ∅
8          update TabuList
9          update IntenArray
10         if (candlist1 ∪ candlist2 = ∅) then
11             create new candlist with respect to restrictions (8.2.4)
12         end if
13         for all (moves in candlist1) do
14             complete the resulting solution (8.2.3)
15             check the TabuList and IntenArray (8.2.5 & 8.2.6)
16         end for all
17         if no improving move has been found check candlist2
18         choose curmove according to pmv (8.2.7)
19         if (curmove = ∅) then
20             decrease TimeInTL and let steps = 0 (8.2.5)
21         end if
22         else
23             update curSol by performing curmove
24             add the move to tabulist (8.2.5)
25             add the move to IntenArray if it leads to an improvement (8.2.6)
26         end else
27         update candlist (8.2.4)
28         if (iterations with same makespan = escaperepetion) then
29             use escapeprocedure (8.2.7)
30         end if
31         iteration++
32     end while
33 end procedure
```

Figure 8.7: Pseudo code for the tabu search algorithm. Numbers in parentheses refer to the corresponding sections.

When the jobs are scheduled according to this sequence and treated as totally elastic without variation of the total capacity we obtain a lower bound on the makespan.

All computational results reported in this section have been found using an Intel Xeon 2.67 GHz processor with 4 GB RAM.

8.3.1 The Practical Application

The problem faced by Sonofon consists of 346 jobs and 587 precedence constraints. The average makespan reported by Sonofon is 821 minutes and the lower bound on the makespan is 591 minutes.

The algorithm presented in this paper yields a makespan of 615 minutes, which is only 4.06 percent above the lower bound while the makespan obtained by Sonofon is 38.92 percent above the lower bound. By a direct comparison of the two makespans it can be seen that our schedule saves 25.09 percent of scheduling time compared to the strategy implemented by Sonofon.

The best solution was found in 56 min, 31 sec, and hence the algorithm can be used on a daily basis to schedule the jobs which have to be processed during the night. Furthermore, Figure 8.8 shows that the significant improvements are obtained in a rather short amount of computation time, and afterwards only small improvements are made. This means that the algorithm is still applicable even though the job specifications are unknown until just prior to the actual scheduling process. The jumps for the current solution reported in Figure 8.8 are due to the escape procedure used in the diversification strategy.

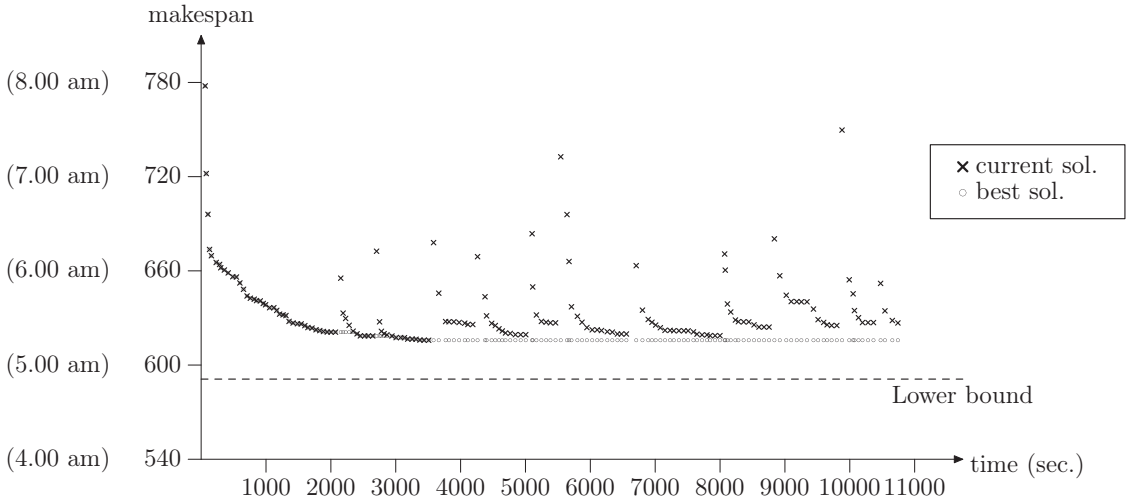

Figure 8.8: Makespan obtained by the tabu search for every 50 iterations.

In addition, the solution of the algorithm can be used to examine how the available capacity is used. Figure 8.9 shows a very uneven server exploitage during the night, and in particular if jobs were moved from machines 1 and 3 to machine 2 the makespan could be reduced.

For additional testing we have implemented the problem in OPL Studio (by ILOG [52]) and provided it with the search strategy to start with box choices according to the scheme in Step 4 of Section 8.2.2. OPL Studio with default

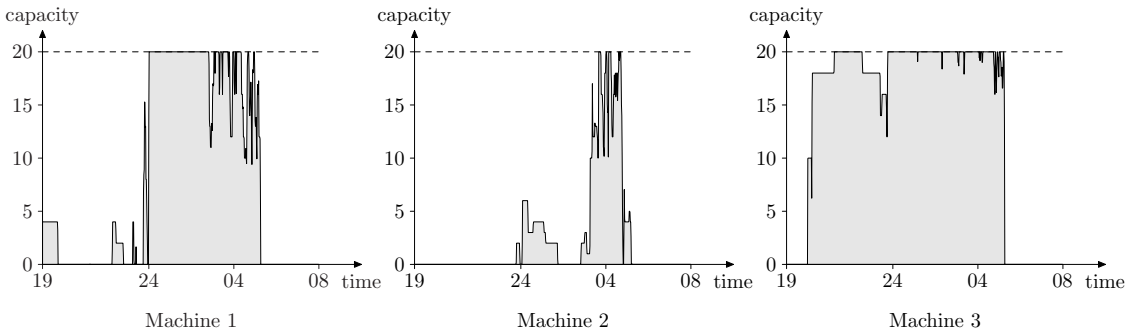

Figure 8.9: Capacity consumption on the three servers.

setting was unable to solve the problem in 24 hours. In fact, within 24 hours OPL Studio was unable even to find a feasible solution to the problem, whereas our algorithm provided a feasible solution in 1 min, 1 sec. The efficiency of the tabu search is in particular due to the speed of the procedure explained in Section 8.2.3 which schedules the jobs when a job sequence and box sizes are given. The order of magnitude for the average time used to run this procedure is 10^{-4} seconds.

We have also tested the benefits of scheduling all jobs on one large server instead of three separate ones. Within 3 hours of computation time our algorithm yields a makespan of 526 minutes and therefore supports such an implementation. This scenario has been considered by Sonofon but is not implementable with their current hardware.

8.3.2 General Large-scale Scheduling Instances

In order to test the robustness of the tabu search we have generated numerous random instances. The first half of these instances has a structure which resembles the structure seen in the data from Sonofon while the structure in the rest of the instances are random.

We have tested the tabu search on instances with 150, 300 and 400 jobs. For each number of jobs we have instances where the number of precedence constraints is equal to half the number of jobs, the number of jobs and twice the number of jobs, respectively. Furthermore, since the jobs are randomly generated, 10 instances have been solved for each specific number of jobs and precedence constraints to give a general idea of the performance of the tabu search. The tabu search have been running in 30 minutes on all instances.

We report the CPU time in seconds used to find an initial feasible solution and the time used to find the best solution within the 30 minutes time limit. To give an idea of the size of the makespan we give the minimum and maximum makespan

Table 8.2: Results for data resembling Sonofon data.

		CPU (avg.)		Makespan		Deviation (%)		
$\|J\|$	$\|P\|$	Initial	Best	min	max	min	avg.	max
150	75	0.00	68.26	335	355	0.00	0.06	0.30
150	150	0.00	17.64	334	433	0.00	0.11	0.82
150	300	0.00	3.04	480	630	0.00	0.16	0.80
300	150	0.00	1170.54	342	386	0.00	1.27	3.40
300	300	0.00	599.25	369	486	0.00	0.60	2.41
300	600	0.50	86.99	623	767	0.00	0.20	1.14
400	200	0.00	1419.62	382	430	0.23	5.60	9.42
400	400	0.00	1250.09	399	458	0.00	2.20	5.00
400	800	51.51	482.98	618	777	0.00	0.15	1.07

for the 10 instances and finally we report both minimum, maximum and average deviation from the lower bound in percent.

The servers at Sonofon face two peaks during the night. One at the beginning of the process where a large number of jobs are allowed to start and one at 24:00 where a second group of jobs are allowed to start due to the shift in date. The instances which resemble data from Sonofon adopt this structure in the sense that one quarter of the jobs must be finished before 24:00, one quarter of the jobs must start after 24:00 and the rest are free. Furthermore, three quarters of the jobs are small with duration randomly chosen between 1 and 5 minutes while the rest are large jobs with duration randomly chosen between 5 and 40 minutes. The capacity consumption is random and so are the precedence constraints.

The results for the instances resembling data from Sonofon are presented in Table 8.2. We see that the average time for finding the best solution increases with the number of jobs while it decreases with the number of precedence constraints. The latter observation shows that the precedence constraints actually constrain the problem in a way that makes it easier to solve.

The tabu search was able to solve all instances within the time limit and we see that the best solution is very close to the lower bound in almost all instances and in the worse case it only exceeds the lower bound with 9.42 percent.

For the instances with random data structure all jobs have a random time window, a random capacity consumption, a random duration and the precedence constraints are random. The results for these instances are reported in Table 8.3 and again we see that the computation time to obtain the best solution increases with the number of jobs and decreases with the number of precedence constraints. On the other hand this table shows that additional precedence constraints make it harder to find a feasible solution and in the case with 400 jobs and 800 precedence

constraints the tabu search is not able to find a feasible solution for 2 of the 10 instances within the time limit.

Table 8.3: Results for random data.

		CPU (avg.)		Makespan		Deviation (%)		
$\|J\|$	$\|P\|$	Initial	Best	min	max	min	avg.	max
150	75	0.00	107.48	514	547	0.00	0.30	0.96
150	150	0.00	43.76	521	568	0.00	0.19	0.75
150	300	0.00	3.80	568	668	0.00	0.10	0.32
300	150	0.00	1296.62	520	542	0.00	1.47	4.13
300	300	3.84	574.26	538	617	0.00	0.76	4.00
300	600	8.16	75.94	610	705	0.00	0.13	0.33
400	200	1.80	1607.16	536	563	1.90	5.21	8.35
400	400	135.35	1357.75	549	630	0.32	2.01	5.82
400	800*	212.85	553.53	609	690	0.00	0.26	1.00

*Only 8 of the 10 instances have been solved.

Bibliography

[1] A. Anagnostopoulos, L. Michel, P. Van Hentenryck, and Y. Vergados. A simulated annealing approach to the traveling tournament problem. In *Proceedings CPAIOR'03, Montreal*, 2003.

[2] Egon Balas, S. Ceria, and G. Cornuéjols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42:1229–1246, 1996.

[3] Egon Balas, S. Ceria, G. Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1–9, 1996.

[4] B.C. Ball and D.B. Webster. Optimal scheduling for even-numbered team athletic conferences. *AIIE Transactions*, 9:161–169, 1977.

[5] P. Baptiste and C. Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5(1-2):119–139, 2000.

[6] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-based scheduling: Applying constraint programming to scheduling problems*. Norwell, Ma: Kluwer Academic Publishers, 2001.

[7] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36:493–513, 1988.

[8] C. Barnhart, E.L. Johnson, G.L Nemhauser, M.W.P Savelsbergh, and P.H. Vance. Branch-and-price: Column generation for huge integer programs. *Operations Research*, 46:316–329, 1998.

[9] R. Barták. Constraint programming: In pursuit of the holy grail. In *Proceedings of the Week of Doctoral Students (WDS99) Part IV*, pages 555–564. MatFyzPress, Prague, 1999.

[10] T. Bartsch, A. Drexl, and S. Kröger. Scheduling the professional soccer leagues of Austria and Germany. *Computers and Operations Research*, 33 (7):1907–1937, 2006.

[11] J.C. Bean and J.R. Birge. Reducing travelling costs and player fatigue in the national basketball association. *Interfaces*, 10(3):98–102, 1980.

[12] J.F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[13] T. Benoist, F. Laburthe, and B. Rottembourg. Lagrange relaxation and constraint programming collaborative schemes for traveling tournament problems. In *Proceedings CPAIOR'01, Wye College (Imperial College), Ashford, Kent UK*, 2001.

[14] J. Blażewicz, M. Machowiak, J. Węglarz, M.Y. Kovalyov, and D. Trystram. Scheduling malleable tasks on parallel processors to minimize the makespan. *Annals of Operations Research*, 129:65–80, 2004.

[15] D.C. Blest and D.G. Fitzgerald. Scheduling sports competitions with a given distribution of times. *Discrete Applied Mathematics*, 22:9–19, 1988/89.

[16] C Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[17] E.K. Burke, D. de Werra, J.D. Landa Silva, and C. Raess. Applying heuristic methods to schedule sports competitions on multiple venues. In E.K. Burke and M.A. Trick, editors, *Proceedings PATAT 2004*, pages 451–456, 2004.

[18] W.O. Cain, Jr. The computer-assisted heuristic approach used to schedule the major league baseball clubs. In S.P. Ladany and R.E. Machol, editors, *Optimal Strategies in Sports*, volume 5 of *Studies in Management Science and Systems*, pages 32–41. North-Holland/American Elsevier, 1977.

[19] R.T. Campbell and D.S. Chen. A minimum distance basketball scheduling problem. In R.E. Machol, S.P. Ladany, and D.G. Morrison, editors, *Management Science in Sports*, volume 4 of *Studies in the Management Sciences*, pages 15–26. North-Holland Publishing Company, 1976.

[20] C. Chekuri and M. Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. *Journal of Algorithms*, 41: 212–224, 2001.

[21] W.-C. Chiang and R.A. Russell. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 9(4):417–430, 1997.

[22] D. Costa. An evolutionary tabu search algorithm and the NHL scheduling problem. *INFOR*, 33:161–178, 1995.

[23] F.D. Croce and D. Oliveri. Scheduling the Italian football league: an ILP-based approach. *Computers and Operations Research*, 33(7):1963–1974, 2006.

[24] H.P. Crowder, E.L. Johnson, and M. Padberg. Solving large scale zero-one linear programming problems. *Operations Research*, 31:803–834, 1983.

[25] A. Drexl and S. Knust. Sports league scheduling: Graph - and resource - based models. *Omega (to appear)*.

[26] K. Easton, G. Nemhauser, and M. Trick. The traveling tournament problem: Description and benchmarks. In T. Walsh, editor, *Principles and Practice of Constraint Programming - CP 2001*, volume 2239 of *Lecture Notes in Computer Science*, pages 580–585. Springer Berlin / Heidelberg, 2001.

[27] K. Easton, G. Nemhauser, and M. Trick. Solving the traveling tournament problem: a combined integer programming and constraint programming approach. In E. Burke and P. De Causmaecker, editors, *Practice and Theory of AutomatedTimetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 100–109. Springer Berlin / Heidelberg, 2003.

[28] M. Elf, M. Jünger, and G. Rinaldi. Minimizing breaks by maximizing cuts. *Operations Research Letters*, 31:343–349, 2003.

[29] T. Fahle, U. Junker, S.E. Karisch, N. Kohl, M. Sellmann, and B. Vaaben. Contraint programming based column generation for crew assignment. *Journal of Heuristics*, 8:59–81, 2002.

[30] J.A. Ferland and C. Fleurent. Computer aided scheduling for a sport league. *INFOR*, 29:14–25, 1991.

[31] J.A. Ferland, S. Ichoua, A. Lavoie, and E. Gagné. Scheduling using tabu search methods with intensification and diversification. *Computers & Operations Research*, 28(11):1075–1092, 2001.

[32] C. Fleurent and J.A. Ferland. Allocating games for the NHL using integer programming. *Operations Research*, 41(4):649–654, 1993.

[33] M.R. Garey and D.S. Johnson. *Computers and intractability, A guide to the theory of NP-completeness*. New York: W.H. Freeman and Company, 1979.

[34] F. Glover. Tabu search - part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[35] F. Glover. Tabu search - part ii. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[36] F. Glover and M. Laguna. *Tabu search*. Ma: Kluwer Academic Publishers, 1997.

[37] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.

[38] R.E. Gomory. An algorithm for integer solutions to linear programs. In R. L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, New York, 1963.

[39] J. Grabowski and M. Wodecki. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Computers & Operations Research*, 31(11):1891–1909, 2004.

[40] J.-P. Hamiez and J.-K. Hao. Solving the sports league scheduling problem with tabu search. In A. Nareyek, editor, *Local Search for Planning and Scheduling*, volume 2148 of *Lecture Notes in Computer Science*, pages 24–36. Springer Berlin / Heidelberg, 2001.

[41] J.-P. Hamiez and J.-K. Hao. A linear-time algorithm to solve the sports league scheduling problem (prob026 of CSPLib). *Discrete Applied Mathematics*, 143:252–265, 2004.

[42] P.V. Hentenryck and Y. Vergados. Traveling tournament scheduling: A systematic evaluation of simulated annealling. In J.C. Beck and B.M. Smith, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3990 of *Lecture Notes in Computer Science*, pages 228–243. Springer Berlin / Heidelberg, 2006.

[43] M. Henz. Scheduling a major college basketball conference - revisited. *Operations Research*, 49:163–168, 2001.

[44] M. Henz. Playing with constraint programming and large neighborhood search for traveling tournaments. In E. Burke and M. Trick, editors, *Proceedings PATAT 2004*, pages 23–32, 2004.

[45] M. Henz. Constraint-based round robin tournament planning. In D. De Schreye, editor, *Proceedings of the International Conference on Logic Programming, Las Cruces, New Mexico*, pages 545–557. MIT Press, 1999.

[46] M. Henz, T. Müller, and S. Thiel. Global constraints for round robin tournament scheduling. *European Journal of Operational Research*, 153:92–101, 2004.

[47] J. N. Hooker. A hybrid method for planning and scheduling. *Constraints*, 10:385–401, 2005.

[48] J.N. Hooker. *Logic-based methods for optimization combining optimization and constraint satisfaction.* Wiley Interscience, 2000.

[49] J.N. Hooker. Logic, optimization, and constraint programming. *INFORMS Journal on Computing*, 14(4):295–321, 2002.

[50] J.N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

[51] J.N. Hooker and H. Yan. Logic circuit verification by Benders decomposition. In V. Saraswat and P. Van Hentenryck, editors, *Principles and Practice of Constraint Programming: The Newport Papers*, pages 267–288. MIT Press (Cambridge, MA), 1995.

[52] *ILOG OPL Studio 3.7, Language Manual.* ILOG, 2003.

[53] V. Jain and I.E. Grossmann. Algorithms for hybrid MILP, CP models for a class of optimization problems. *INFORMS Journal on Computing*, 13(4): 258–276, 2001.

[54] A. Lim, B. Rodrigues, and X. Zhang. A simulated annealing and hill-climbing algorithm for the traveling tournament problem. *European Journal of Operational Research (In press).*

[55] L. Lovász and M.D. Plummer. *Matching Theory.* Akadémiai Kiadó, Budapest, 1986.

[56] R. Mansini, M.G. Speranza, and Z. Tuza. Scheduling groups of tasks with precedence constraints on three dedicated processors. *Discrete Applied Mathematics*, 134:141–168, 2004.

[57] K. Marriott and P. Stuckey. *Programming with constraints an introduction.* MIT Press, Cambridge, MA, 1998.

[58] K. McAloon, C. Tretkoff, and G. Wetzel. Sports league scheduling. In *Proceeding of the 3rd ILOG Optimization Suite International Users Conference, Paris*, 1997.

[59] E. Mendelsohn and A. Rosa. One-factorizations of the complete graph - a survey. *Journal of Graph Theory*, 9:43–65, 1985.

[60] John E. Mitchell. Branch-and-cut algorithms for combinatiorial optimization problems. In Panos M. Pardalos and Mauricio G. C. Resende, editors, *Handbook of applied of Applied Optimization*, pages 65–77. Oxford University Press, 2002.

[61] R. Miyashiro and T. Matsui. Round-robin tournaments with a small number of breaks, 2003. URL `citeseer.ist.psu.edu/miyashiro03roundrobin.html`.

[62] R. Miyashiro and T. Matsui. A polynomial-time algorithm to find an equitable home-away assignment. *Operations Research Letters*, 33:235–241, 2005.

[63] R. Miyashiro and T. Matsui. Semidefinite programming based approaches to the break minimization problem. *Computers & Operations Research*, 33 (7):1975–1982, 2006.

[64] R. Miyashiro, H. Iwasaki, and T. Matsui. Characterizing feasible pattern sets with a minimum number of breaks. In E. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 78–99. Springer Berlin / Heidelberg, 2003.

[65] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley Interscience, 1988.

[66] G.L. Nemhauser and M.A. Trick. Scheduling a major college basketball conference. *Operations Research*, 46(1):1–8, 1998.

[67] C.R. Pedersen, R.V. Rasmussen, and K.A. Andersen. Solving a large-scale precedence constrained scheduling problem with elastic jobs using tabu search. *Computers & Operations Research (Article in press)*.

[68] S.C.S. Porto, J.P.F.W. Kitajima, and C.C. Ribeiro. Performance evaluation of a parallel tabu search task scheduling problem. *Parallel Computing*, 26: 73–90, 2000.

[69] G. Post and G.J. Woeginger. Sports tournaments, home-away assignments, and the break minimization problem. *Discrete Optimization (To appear)*.

[70] R.V. Rasmussen. Scheduling a triple round robin tournament for the best Danish soccer league. *Working Paper no. 2006/1, Department of Operations Research, University of Aarhus*, 2006. URL `http://www.imf.au.dk/publs?id=596`. (Submitted).

[71] R.V. Rasmussen. Benchmark place constraints. URL `http://home.imf.au.dk/vinther/benchmarks`.

[72] R.V. Rasmussen and M.A. Trick. A Benders approach for constrained minimum break problem. *European Journal of Operational Research (In Press)*.

[73] R.V. Rasmussen and M.A. Trick. The timetable constrained distance minimization problem. In J.C. Beck and B.M. Smith, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3990 of *Lecture Notes in Computer Science*, pages 167–181. Springer Berlin / Heidelberg, 2006.

[74] R.V. Rasmussen and M.A. Trick. Round robin scheduling - a survey. *Working Paper no. 2006/2, Department of Operations Research, University of Aarhus*, 2006. URL `http://www.imf.au.dk/publs?id=610`.

[75] J.-C. Régin. Minimization of the number of breaks in sports scheduling problems using constraint programming. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 57:115–130, 2001.

[76] C.C. Ribeiro and S. Urrutia. Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research (In press)*.

[77] A. Rosa and W.D. Wallis. Premature sets of 1-factors or how not to schedule round-robin tournaments. *Discrete Applied Mathematics*, 4:291–297, 1982.

[78] R.A. Russell and J.M.Y. Leung. Devising a cost effective schedule for a baseball league. *Operations Research*, 42(4):614–625, 1994.

[79] R.A. Russell and T.L. Urban. A constraint programming approach to the multiple-venue, sport-scheduling problem. *Computers & Operations Research*, 33:1895–1906, 2006.

[80] M.W.P. Savelsbergh. Branch-and-price: integer programming with column generation. In C. Floudes and P. Pardalos, editors, *Encyclopedia of Optimization*, 2001.

[81] A. Schaerf. Scheduling sport tournaments using constraint logic programming. *Constraints*, 4:43–65, 1999.

[82] J.A.M. Schreuder. Constructing timetables for sport competitions. *Mathematical Programming Study*, 13:58–67, 1980.

[83] J.A.M. Schreuder. Combinatorial aspects of construction of competition dutch professional football leagues. *Discrete Applied Mathematics*, 35:301–312, 1992.

[84] Alexander Schrijver. *Theory of linear and integer programming.* Wiley Interscience, 1986.

[85] M. Sellmann, K. Zervoudakis, P. Stamatopoulos, and T. Fahle. Crew assignment via constraint programming: Integrating column generation and heuristic tree search. *Annals of Operations Research*, 115:207–225, 2002.

[86] M. Trick. A schedule-then-break approach to sports timetabling. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 242–252. Springer Berlin / Heidelberg, 2001.

[87] M. Trick. Integer and constraint programming approaches for round robin tournament scheduling. In E. Burke and P. De Causmaecker, editors, *Practice and Theory of AutomatedTimetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 63–77. Springer Berlin / Heidelberg, 2003.

[88] M.A. Trick. Michael trick's guide to sports scheduling. URL `http://mat.tepper.cmu.edu/sports/Instances/`.

[89] T.L. Urban and R.A. Russell. Scheduling sports competitions on multiple venues. *European Journal of Operational Research*, 148:302–311, 2003.

[90] S. Urrutia and C.C. Ribeiro. Minimizing travels by maximizing breaks in round robin tournament schedules. *Electronic Notes in Discrete Mathematics*, 18:227–233, 2004.

[91] T.V. Voorhis. College basketball scheduling with travel swings. *Computers & Industrial Engineering*, 48:163–172, 2005.

[92] M. Wallace. Practical applications of constraint programming. *Constraints*, 1(1&2):139–168, 1996.

[93] D. de Werra. Geography, games and graphs. *Discrete Applied Mathematics*, 2:327–337, 1980.

[94] D. de Werra. Scheduling in sports. In P. Hansen, editor, *Studies on Graphs and Discrete Programming*, pages 381–395. North-Holland, Amsterdam, 1981.

[95] D. de Werra. Minimizing irregularities in sports schedules using graph theory. *Discrete Applied Mathematics*, 4:217–226, 1982.

[96] D. de Werra. Some models of graphs for scheduling sports competitions. *Discrete Applied Mathematics*, 21:47–65, 1988.

[97] D. de Werra, L. Jacot-Descombes, and P. Masson. A constrained sports scheduling problem. *Discrete Applied Mathematics*, 26:41–49, 1990.

[98] D. de Werra, T. Ekim, and C. Raess. Construction of sports schedules with multiple venues. *Discrete Applied Mathematics*, 154:47–58, 2006.

[99] R.J. Willis and B.J. Terrill. Scheduling the Australian state cricket season using simulated annealing. *Journal of the Operational Research Society*, 45 (3):276–280, 1994.

[100] M.B. Wright. Scheduling fixtures for basketball New Zealand. *Computers & Operations Research*, 33:1875–1893, 2006.

[101] M.B. Wright. Timetabling county cricket fixtures using a form of tabu search. *Journal of the Operational Research Society*, 45(7):758–770, 1994.

Index