A Benders approach for the constrained minimum break problem

Rasmus V. Rasmussen[1] and Michael A. Trick[2]

[1] Department of Operations Research, University of Aarhus, Ny Munkegade, Building 530, 8000 Aarhus C, Denmark

[2] Tepper School of Business, Carnegie Mellon University, Pittsburgh PA 15213, USA

Abstract. This paper presents a hybrid IP/CP algorithm for designing a double round robin schedule with a minimal number of breaks. Both mirrored and non-mirrored schedules with and without place constraints are considered. The algorithm uses Benders cuts to obtain feasible home-away pattern sets in few iterations and this approach leads to significant reductions in computation time for hard instances. Furthermore, the algorithm is capable of solving a number of previously unsolved benchmark problems for the Traveling Tournament Problem with constant distances.

Keywords: Timetabling; Hybrid integer programming/constraint programming; Sports scheduling; Logic based Benders cuts.

1 Introduction

During the last 30 years, the field of sports scheduling has attracted an increasing interest due to very hard problems with obvious practical applications. The area has provided unsolved theoretical problems concerning the characterization of feasible structures, as well as challenging benchmark problems still unsolved by state of the art algorithms, like the Traveling Tournament Problem of Easton, Nemhauser and Trick [5]. In addition, the combination of optimization and feasibility issues seen in sports scheduling problems is well suited for the grow-

ing interest of combining integer programming (IP) and constraint programming (CP).

In the 1980s, de Werra [2,3,4] and Schreuder [22] obtained a number of theoretical results for round robin tournaments. Among these results were a constructive method for generating a round robin tournament with a minimal number of breaks called the *canonical schedule*. This method can be used directly when no additional constraints are present. However, real sports league schedules are often constrained by special requests from the teams and these requests exclude the use of the canonical schedule. Nemhauser and Trick [18] outline examples of such constraints for the Atlantic Coast Conference.

Our work examines computational approaches to the constrained minimum break problem. This problem has been studied in several papers and both practical applications [1,8,18,23] as well as general solution methods [6,7,9,15,20,21,24,25] have been examined. A three phase approach has proven to be very efficient and has been widely used in the literature, though the order of the following three phases varies. Phase one generates *home away patterns* (patterns) and *pattern sets*. Phase two finds *timetables* which are assignments of games to time slots and phase three allocates teams to patterns.

The hybrid IP/CP approach presented in this paper also adopts the three phase approach 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 phase one by using Benders cuts.

The approach is inspired by the work of Hooker and Ottosson [10] in which the concept of logic based Benders decomposition is developed. This generalization makes it possible to apply the Benders decomposition strategy on problems without a linear programming subproblem. In particular, the method can be used when merging IP and CP since it allows a CP subproblem as in [11] and [13] in which a machine scheduling application is considered.

Our approach for the constrained minimum break problem treats phase one as a master problem while phase two and three are subproblems. The master problem finds a pattern set with a minimal number of breaks and the subproblems check feasibility of the pattern set. If the subproblems are feasible, a schedule has been found; otherwise a *Benders cut* is added to the master problem to cut off infeasible solutions in future iterations. The approach will be referred to as the *pattern generating Benders approach* (PGBA).

Computational tests show that the PGBA leads to substantial savings in computation time for most instances when compared to the three phase approach. The savings are small for small instances but increase with the number of teams and for very difficult instances. These speedups make it possible to consider non-mirrored tournaments of realistic size.

We extend this work to a version of the Traveling Tournament Problem. In Urrutia and Ribeiro [27], it is shown that for the *Constant Distance Traveling Tournament Problem* (CTTP), the problem of minimizing the total travel distance is equivalent to maximizing the number of breaks. With some modifications of the algorithm presented here, it is possible to solve a number of previously unsolved benchmark problems for CTTP to optimality.

The rest of the paper is organized as follows. The problem formulation is stated in Section 2 and the solution method is outlined in Section 3. Sections 4 and 5 present the models used to generate patterns and find pattern sets respectively. Section 6 introduces models for checking feasibility and presents Benders cuts while Section 6.4 shows how timetables and team allocations are found. The algorithm is formally stated in Section 7, computational results are shown in Section 8 and concluding remarks are given in Section 9.

2 Problem formulation

Before stating the problem some basic terminology is needed. We consider *double round robin tournaments* which are tournaments where all teams meet twice. Each team has a venue and it plays all other teams once at the home venue

and once at the venue of the opponent. The tournament is partitioned into *time slots* and each team cannot play more than one game in each slot. Since only tournaments with an even number of teams are considered we require that each team plays in each slot.

Each team has a *home-away pattern* (pattern) which is an array of zeros and ones with an entrance for each slot. An entrance with a 0 (1) means that the team plays away (home) in the corresponding slot. A set of patterns for all the teams in a tournament is called a *home-away pattern set* (pattern set) and it is feasible when a corresponding *timetable* exists. A timetable is a matrix with a row for each team and a column for each slot where entrance (i, s) gives the opponent of team i in slot s. Figure 2.1 shows a feasible pattern set for a tournament with 6 teams and a corresponding timetable.

Slot	1	2	3	4	5	6	7	8	9	10
p_1	0	1	0	1	0	1	0	1	0	1
p_2	0	1	1	0	1	1	0	0	1	0
p_3	1	0	1	1	0	0	1	0	0	1
p_4	0	1	0	0	1	1	0	1	1	0
p_5	1	0	1	0	1	0	1	0	1	0
p_6	1	0	0	1	0	0	1	1	0	1

(a)

Slot	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

(b)

Fig. 2.1. (a) Home away pattern set, (b) Corresponding timetable

In sports leagues where teams return home after each game, alternating patterns of home and away games are normally preferred. Patterns which does not alternate but have two consecutive home games or two consecutive away games are said to have a *break* in the last of the two slots. The breaks in the pattern set of Figure 2.1 is underlined. In order to create a good schedule the number of breaks is minimized and since patterns with consecutive breaks are considered highly undesirable such patterns are discarded. Notice that a feasible pattern set cannot contain more than two patterns without breaks.

Apart from providing good patterns to the teams, the schedule must also comply with whatever needs the teams may have for playing home or away at certain dates. We will define a *place constraint* to be a requirement saying that a team must play home or must play away in a given time slot. These place constraints typically arise from venue limitations and they are generally not satisfied by the canonical schedule.

When feasible, a *mirrored* tournament is often a good choice since games between opponents are well separated. A tournament is mirrored when each game played in the first half is repeated in the corresponding time slot of the second half with venues reversed, see Figure 2.1. However, when mirrored tournaments are used, some flexibility is lost and it might not be possible to satisfy the place constraints. Instead a non-mirrored tournament can be used but then an extra constraint is needed to separate games with the same opponents.

The problem addressed in this paper is to find a schedule which accommodates the above requirements. More specifically, it is to find a double round robin tournament for $2n$ teams such that the games are partitioned in $4n-2$ time slots and each team plays one game in each slot. All place constraints must be satisfied, consecutive breaks are not allowed and the total number of breaks must be minimized. Both the mirrored and the non-mirrored cases will be considered and, in the non-mirrored case, 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. We introduce the following variables:

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

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

$x_{ij} \in S$ gives the slot in which team j visits team i.

The problem can be formally stated as a CP problem:

$$\min \sum_{i \in T} \sum_{s \in S} b_{is} \tag{2.1}$$

$$\text{s.t.} \quad \textit{sequence}(1, 2, 3, \textit{all}(s \in S)\ \ h_{is},$$

$$1, 2n-1) \qquad i \in T \tag{2.2}$$

$$(b_{is} = 1) \Leftrightarrow (h_{is-1} = h_{is}) \qquad i \in T,\ \ s \in S \setminus \{1\} \tag{2.3}$$

$$b_{i1} = 0 \qquad i \in T \tag{2.4}$$

$$\sum_{i \in T} h_{is} = n \qquad s \in S \tag{2.5}$$

(CPP)

$$h_{is} = 1 \qquad i \in T,\ \ s \in I_i^1 \tag{2.6}$$

$$h_{is} = 0 \qquad i \in T,\ \ s \in I_i^0 \tag{2.7}$$

$$(h_{is} = 0) \vee (h_{js} = 1) \Rightarrow (x_{ij} \neq s) \qquad i, j \in T,\ \ i \neq j \tag{2.8}$$

$$\textit{alldifferent}(\textit{all}(j \in T \setminus i)\ \ x_{ij},$$

$$\textit{all}(j \in T \setminus i)\ \ x_{ji}) \qquad i \in T \tag{2.9}$$

$$(x_{ij} - x_{ji} < -k) \vee (x_{ij} - x_{ji} > k) \qquad i, j \in T,\ \ i < j \tag{2.10}$$

$$h_{is},\ \ b_{is} \in \mathbb{B} \qquad i \in T,\ \ s \in S \tag{2.11}$$

$$x_{ij} \in S \qquad i, j \in T,\ \ i \neq j \tag{2.12}$$

Constraint (2.2) is a global constraint which takes six arguments, three integers: `nbMin`, `nbMax` and `width`, and three one dimensional arrays: `vars`, `values` and `card`.

$$\textit{sequence}(\texttt{nbMin}, \texttt{nbMax}, \texttt{width}, \texttt{vars}, \texttt{values}, \texttt{card})$$

`values` and `card` must have the same index set I and in the given application the two arrays are of size 1. `vars` is an array of variables and the constraint requires that for all $i \in I$ the number of variables in `vars`, which is equal to `values`[i], must be `card`[i]. Furthermore, any subsequence of size `width` must contain at least `nbMin` and at most `nbMax` values from `values` [12]. The constraint requires that each team plays exactly $2n - 1$ home games and has a pattern

without consecutive breaks since three consecutive slots must contain at least one and at most 2 home games. Constraint (2.3) defines a break and since breaks are impossible in the first slot, (2.4) sets all break variables for slot 1 to zero. Constraint (2.5) states that in each time slot, exactly n teams play home and (2.6) and (2.7) make sure that the place requirements are satisfied. Constraint (2.8) makes sure that team j does not visit team i in a slot where team i plays away or team j plays home. The *alldifferent* constraint (2.9) takes an array of variables `vars` with index set I as argument and it requires that all variables in `vars` must be different [12]. Since `vars` is the array of all games played by a specific team the constraint makes sure that a team does not play more than a single game in each time slot. Finally (2.10) separates two games played by the same teams with at least k time slots.

Notice, that the constraints (2.2)-(2.4) are all constraints for individual patterns, (2.5) is a constraint for the set of patterns and (2.8)-(2.10) consider assignments of games to the pattern set. This partitioning will be used in the solution method.

3 Methodology

To solve the problem defined in Section 2, 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 team allocation exist. The check for optimality is discussed in Section 7.

Furthermore, the PGBA introduces the third component for checking feasibility of the pattern sets. The component heuristically determines infeasibility and, when successful, adds Benders cuts to the master problem. A number of models are used to perform this check and they are presented in Section 6.

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 3.1 gives an overview of how the algorithm works.

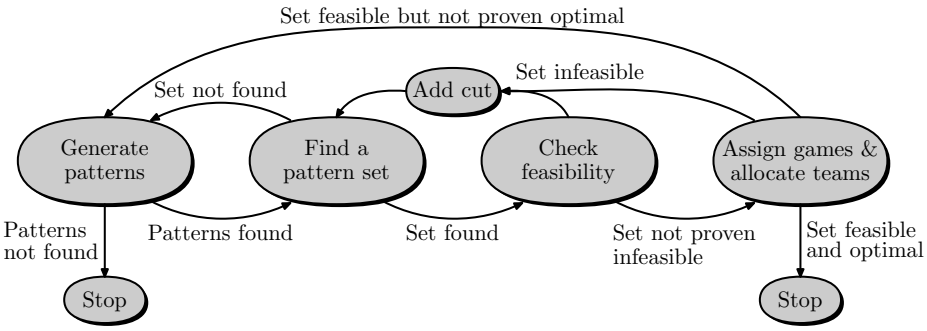

Fig. 3.1. Overview of the algorithm.

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 3.1 is given in Section 7.

4 Generating patterns

A CP model is used for generating patterns. The model generates all patterns which begin with an away game, satisfy the constraint (2.2) 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 is one (zero) if the pattern has a home (away) game in slot s, the CP model looks as follows:

(PM)

$$sequence(1, 2, 3, all(s \in S)\ \ h_s, 1, 2n-1) \tag{4.1}$$

$$h_1 = 0 \tag{4.2}$$

$$\sum_{s=1}^{|S|-1} (h_s = h_{s+1}) = c \tag{4.3}$$

$$h_s + h_{s+2n-1} = 1 \qquad s = 1, \ldots, 2n-1 \tag{4.4}$$

$$h_s \in \mathbb{B} \qquad s \in S \tag{4.5}$$

Constraint (4.1) corresponds to (2.2) from (CPP) presented in Section 2. (4.2) restricts the search to patterns which begin with an away game and (4.3) limits the search to patterns with c breaks. (4.4) requires that the pattern is mirrored and is only used when a mirrored tournament is considered.

5 Pattern sets

Given a set of patterns found by (PM) we want to find a subset of $2n$ patterns with a minimal number of breaks. Since all the patterns satisfy the constraint (2.2), and the game assignment and team allocation are postponed, the only constraint from (CPP) which should be considered is (2.5). However, restricting the model to find pattern sets for which all teams can be allocated to at least

one pattern based on 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 7.

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 is used.

(PSM)

$$\min \sum_{j \in P} c_j p_j \tag{5.1}$$

$$\text{s.t.} \sum_{j \in P} p_j = 2n \tag{5.2}$$

$$\sum_{j \in P} h_{js} p_j = n \qquad s \in S \tag{5.3}$$

$$\sum_{j \in P_i} p_j \geq 1 \qquad i \in T \tag{5.4}$$

$$\sum_{j \in P} c_j p_j \geq LB \tag{5.5}$$

$$\sum_{j \in P} c_j p_j \leq UB \tag{5.6}$$

$$p_j \in \mathbb{B} \qquad j \in P \tag{5.7}$$

Constraint (5.2) makes sure that exactly $2n$ patterns are chosen, (5.3) corresponds to (2.5), (5.4) requires that all teams can be allocated to at least one pattern and (5.5) and (5.6) enforce a lower and an upper bound on the number of breaks.

6 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 [10], 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 Benders cut for each of the conditions. The pattern set which is currently checked is denoted P^C.

6.1 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 6.1 give an example of such a bipartite graph.

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

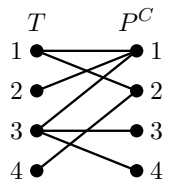

Fig. 6.1. Example of bipartite graph used to find a team allocation.

This means that Hall's theorem [14] gives a necessary and sufficient condition for a team allocation to exist.

Hall's theorem *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 $X \subseteq A$.*

In our case $\Gamma(X) = \cup_{i \in X}(P_i \cap P^C)$. We can use the Hungarian method to find a 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.

$$\sum_{j \in \cup_{i \in X} P_i} p_j \geq |X| \tag{6.1}$$

6.2 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. Miyashiro et. al. [17] give the following necessary condition which must be satisfied by all subsets $\bar{P}$ for a pattern set to be feasible.

$$\sum_{s \in S} \left(\min \left\{ \sum_{j \in \bar{P}} h_{js}, \sum_{j \in \bar{P}} (1 - h_{js}) \right\} \right) \geq |\bar{P}|(|\bar{P}| - 1) \tag{6.2}$$

The parameter h_{js} tells whether pattern j plays home or away in slot s.

The condition requires that the sum of the maximum number of mutual games, which can be played by patterns from $\bar{P}$ in each slot, must be no less than the required number of mutual games. For each slot the minimum of the number of home games and the number of away games played by patterns from $\bar{P}$ is used as an upper bound on the number of mutual games. Notice that the condition is always satisfied for subsets of size two since all patterns are distinct.

Instead of checking each subset, we formulate an IP model which, for a given subset size Z, finds the subset of teams with the smallest left hand side in 6.2. 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.

$$\min \sum_{s \in S} \beta_s \tag{6.3}$$

$$\text{s.t.} \quad \sum_{j \in P^C} \alpha_j = Z \tag{6.4}$$

(UBM)

$$\beta_s - \sum_{j \in P^C} h_{js}\alpha_j + Z(1 - \delta_s) \geq 0 \qquad s \in S \tag{6.5}$$

$$\beta_s - \sum_{j \in P^C} (1 - h_{js})\alpha_j + Z\delta_s \geq 0 \qquad s \in S \tag{6.6}$$

$$\alpha_j, \ \ \delta_s \in \mathbb{B} \qquad j \in P^C, \ \ s \in S \tag{6.7}$$

$$\beta_s \in \mathbb{R}_+ \qquad s \in S \tag{6.8}$$

In this problem (6.4) ensures that exactly Z patterns are chosen and (6.5) and (6.6) make sure that the number of home games are counted in slot s if $\delta_s = 1$ and away games are counted if $\delta_s = 0$.

This problem can be used to formulate a necessary condition for P^C to be feasible since (6.2) 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 not satisfied the Benders cut

$$\sum_{\substack{j \in P: \\ \alpha_j = 1}} p_j \leq Z - 1 \tag{6.9}$$

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 (6.9) can be added to (PSM) if the objective of (UBM) is strictly less than $\frac{Z(Z-1)}{2}$.

6.3 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 6.2 for an example when $k \geq 2$.

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

Fig. 6.2. Example of two patterns which cannot meet when $k \geq 2$.

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

Lemma 1. *The two games between patterns i and j can be assigned to two slots in S separated by k slots if and only if*

$$(s^l_{ij} - s^f_{ji} > k) \vee (s^l_{ji} - s^f_{ij} > k). \tag{6.10}$$

In Figure 6.2 we see that $s^f_{12} = 2$, $s^f_{21} = 3$, $s^l_{12} = 5$ and $s^l_{21} = 4$. This means that (6.10) is violated when $k \geq 2$.

Lemma 1 implies the necessary condition.

The pair separation condition: *Given a pattern set P^C and two patterns i, j from this set, then the pattern set is feasible only if i, j satisfy 6.10.*

All pairs i, j violating this condition lead to the Benders cut:

$$p_i + p_j \leq 1 \tag{6.11}$$

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 (2.8) - (2.10) but they only consider the patterns from $\bar{P}$. This gives the following CP model.

(GAM)

$$(h_{is} = 0) \vee (h_{js} = 1) \Rightarrow (x_{ij} \neq s) \quad i, j \in \bar{P},\ i \neq j,\ s \in S \tag{6.12}$$

$$alldifferent(all(j \in \bar{P} \setminus i)\ x_{ij},$$

$$all(j \in \bar{P} \setminus i)\ x_{ji}) \qquad i \in \bar{P} \tag{6.13}$$

$$(x_{ij} - x_{ji} < -k) \vee (x_{ij} - x_{ji} > k) \quad i, j \in \bar{P},\ i < j \tag{6.14}$$

$$x_{ij} \in S \qquad i, j \in \bar{P},\ i \neq j \tag{6.15}$$

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)

$$\sum_{j \in \bar{P}} p_j \leq |\bar{P}| - 1 \tag{6.16}$$

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

6.4 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 6.3 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 (6.14) is redundant and can be removed.

In case (GAM) is feasible a feasible game assignment is found and otherwise the Benders cut (6.16) is added to (PSM).

7 The algorithm

This section presents pseudo code for each of the four components illustrated in Figure 3.1 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 7.1. 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 7.2 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      else
5          Find all solutions to (PM)
6          Update nbPatterns
7          Let c = c + 1
8          if (nbPatterns < 2n) then
9              Generate Patterns
10         else
11             Find Pattern Set
12 end procedure
```

Fig. 7.1. 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      else
8          Generate Patterns
9  end procedure
```

Fig. 7.2. Procedure for finding pattern sets.

Feasibility. The procedure *Check Feasibility*, is used to check if any of the necessary conditions from Section 6 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

```
1  procedure Check Feasibility
2      cutAdded = 0, card = 2
3      Use the Hungarian method
4      if (∃X ⊆ T :  Γ(X) < X) then
5          Add (6.1) to (PSM), cutAdded = 1
6      if (cutAdded = 0) then
7          for all (i, j ∈ P^C :  i < j) do
8              if ((6.10) is violated) then
9                  Add (6.11) to (PSM), cutAdded = 1
10     while ((card < maxCard_GAM) ∧ (cutAdded = 0)) do
11         card = card + 1
12         for all (P̄ ⊆ P^C :  |P̄| = card) do
13             if (cutAdded = 0) then
14                 Solve (GAM) for P̄
15                 if ((GAM) is infeasible) then
16                     Add (6.16) to (PSM), cutAdded = 1
17     if (cutAdded = 0) then
18         Find Timetable
19     else
20         Find Pattern Set
21 end procedure
```

Fig. 7.3. Procedure for checking feasibility.

multiple pattern separation condition is omitted in case of mirrored tournaments. Figure 7.3 displays the procedure for the non-mirrored case.

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 (6.1) is added to (PSM). If a team allocation is found the algorithm adds the cut (6.11) to (PSM) for all pairs of patterns which violate (6.10). In case

all pairs satisfy (6.10) 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 (6.16) 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 (6.16) to (PSM)
5          Find Pattern Set
6      else
7          Let UB = LB − 2
8          Let LB = 2n − 3 + c
9          if (c ≤ UB − 2n + 3) then
10             for all (i ∈ {c, . . . , max{2n − 1, UB − 2n + 3}}) do
11                 Generate Patterns
12             Let c = 2n
13             Find Pattern Set
14         else
15             Stop, optimal solution found
16 end procedure
```

Fig. 7.4. Procedure for finding a timetable.

Timetable. The procedure for finding a timetable is shown in Figure 7.4. In this procedure (GAM) is solved for P^C and in case of infeasibility (6.16) 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.

8 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 [18] and Henz [8]. 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 [12] 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 8.1 shows the computation times for mirrored instances without place constraints. The instance with 4 teams is infeasible since consecutive breaks are forbidden.

Table 8.1. 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

In Table 8.2 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.

Since different place constraints affect the complexity of an instance differently we have tested the algorithms on 10 sets of randomly generated place constraints 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 [19].

Tables 8.3 and 8.4 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.

Table 8.2. Non-mirrored instances without place constraints.

Instance	Breaks	TPA	PGBA	Instance	Breaks	TPA	PGBA
np-nm-k0-n4	2	0.02	0.02	np-nm-k1-n18	18	-	5.00
np-nm-k0-n6	4	0.03	0.09	np-nm-k1-n20	20	-	14.70
np-nm-k0-n8	6	0.28	0.17	np-nm-k1-n22	?	-	-
np-nm-k0-n10	8	1.45	0.36	np-nm-k2-n4 *	-	0.16	0.14
np-nm-k0-n12	10	29.78	1.41	np-nm-k2-n6	10	555.11	15.14
np-nm-k0-n14	12	885.80	3.95	np-nm-k2-n8	8	19.61	0.55
np-nm-k0-n16	14	-	1.70	np-nm-k2-n10	10	91.03	0.89
np-nm-k0-n18	16	-	4.36	np-nm-k2-n12	12	-	0.92
np-nm-k0-n20	18	-	7.16	np-nm-k2-n14	14	-	2.14
np-nm-k0-n22	20	-	6.17	np-nm-k2-n16	16	-	3.48
np-nm-k0-n24	22	-	9.31	np-nm-k2-n18	18	-	4.97
np-nm-k0-n26	24	-	17.27	np-nm-k2-n20	20	-	14.75
np-nm-k0-n28	26	-	129.00	np-nm-k2-n22	?	-	-
np-nm-k0-n30	?	-	-	np-nm-k3-n4 *	-	0.16	0.13
np-nm-k1-n4	6	0.06	0.09	np-nm-k3-n6	12	-	70.02
np-nm-k1-n6	10	546.88	20.70	np-nm-k3-n8	12	-	137.00
np-nm-k1-n8	8	19.48	0.56	np-nm-k3-n10	?	-	-
np-nm-k1-n10	10	90.03	0.94	np-nm-k3-n12	16	-	8.55
np-nm-k1-n12	12	-	0.91	np-nm-k3-n14	18	-	8.86
np-nm-k1-n14	14	-	1.92	np-nm-k3-n16	20	-	17.55
np-nm-k1-n16	16	-	3.99	np-nm-k3-n18	?	-	-

* Problem is infeasible.

Table 8.3. Mirrored instances with place constraints.

Instance	# Feasible	# 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

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

Table 8.4. Non-mirrored instances with place constraints.

Instance	# Feasible	# 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

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 stabile when random place constraints are considered. In Table 8.4 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.

8.1 The constant distance Traveling Tournament Problem

The Traveling Tournament Problem (TTP) was stated by Easton, Nemhauser and Trick [5] and originates from the scheduling of Major League Baseball in the United States. The problem is to minimize the total travel distance of all teams while the basic constraints of a tournament must be satisfied.

Input: The number of teams in the tournament n, an $n \times n$ distance matrix D, where D_{ij} is the distance between the venues of team i and team j, and two integer parameters L and U.

Output: A double round robin tournament for the n teams satisfying that the number of consecutive home games and the number of consecutive away games are between L and U and the total distance traveled by the teams is minimized.

A number of Benchmark problems can be seen at [26]. In all of these problems $L = 1$, $U = 3$ and the problem is either mirrored or a non-repeater constraint is added.

Urrutia and Ribeiro [27] 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 CTTP. Miyashiro and Matsui [16] 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 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 [26].

Table 8.5 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 [27] 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 8.6 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 [26].

Table 8.5. Solution values to 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	-	-	-

Table 8.6. 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	-	-	-	-

9 Conclusion

We have presented the pattern generating Benders approach which is a new solution method for solving the place constrained break minimization problem. The approach excels compared to previous methods by limiting the number of patterns which are generated and using Benders cuts to avoid infeasible pattern sets.

Extensive computational testing shows that the approach is extremely stabile to varying place constraints which is a valuable property when real sports leagues are considered. Furthermore, the overall runtime of the algorithm is comparable with that of previous approaches for easy instances while savings of several orders of magnitude are obtained for some of the hard problems considered.

The algorithm also proves its speed by solving a number of previously unsolved Traveling Tournament Problems. These problems are solved by modifying the algorithm to maximizing the number of breaks instead of minimizing it.

Though the algorithm in general performs well some questions remain open. In tables 8.1 and 8.2 the computation time of PGBA increases quite slowly with the number of teams until it suddenly exceeds 30 minutes. It is unknown what triggers this sudden increase but it would be interesting to examine this problem and perhaps expand the set of problems for which the algorithm can be used.

In this work we have considered place constraints but in the sports scheduling literature several other types of constraints are discussed. These include complementary constraints when two teams share the same stadium, fixed game constraints when two teams must meet on a certain date, top team constraints when special rules apply for the best teams and fairness constraints to distribute the breaks evenly. A direction for future research would be to analyse how these constraints can be implemented in the algorithm.

References

1. F.D. Croce, D. Oliveri, Scheduling the Italian football league: An ILP-based approach, Computers & Operations Research (to appear).
2. D. de Werra, Scheduling in sports, Annals of Discrete Mathematics 11 (1981)381-395.
3. D. de Werra, Some Models of graphs for scheduling sports competitions, Discrete Applied Mathematics 21 (1988)47-65.
4. D. de Werra, L. Jacot-Descombes, P. Masson, A Constrained Sports Scheduling Problem, Discrete Applied Mathematics 26 (1990)41-49.
5. K. Easton, G. Nemhauser, M. Trick, The traveling tournament problem: Description and benchmarks, in: Proceedings CP'01, Lecture Notes in Computer Science 2239 (2001)580-584.
6. M. Elf, M. Jünger, G. Rinaldi, Minimizing breaks by maximizing cuts, Operations Research Letters, 31 (2003)343-349

7. M. Henz, Constraint-based round robin tournament planning, in: D. De Schreye (Eds.), Proceedings of the International Conference on Logic Programming, Las Cruces, New Mexico, MIT Press, 1999, pp. 545-557.
8. M. Henz, Scheduling a major college basketball conference - revisited, Operations Research 49 (2001)163-168.
9. M. Henz, T. Müller, S. Thiel, Global constraints for round robin tournament scheduling, European Journal of Operational Research 153 (1)(2004)92-101.
10. J.N. Hooker, G. Ottosson, Logic-based Benders decomposition, Mathematical Programming 96 (2003)33-60.
11. J.N. Hooker, A hybrid method for planning and scheduling, in: M. Wallace, (Eds.), Principles and Practice of Constraint Programming (CP 2004), Lecture Notes in Computer Science 3258, Springer, 2004, pp. 305-316.
12. ILOG, ILOG OPL Studio 3.7, Language Manual, 2003.
13. V. Jain, I.E. Grossmann, Algorithms for hybrid MILP-CP models for a class of optimization problems, INFORMS Journal on Computing 13 (4)(2001)258-276.
14. L. Lovsz, M.D. Plummer, Matching Theory, Akadémiai Kiadó, Budapest 1986.
15. R. Miyashiro, T. Matsui, Semidefinite programming based approaches to the break minimization problem, Computers & Operations Research (to appear)
16. R. Miyashiro, T. Matsui, A polynomial-time algorithm to find an equitable home-away assignment, Operations Research Letters 33 (2005)235-241
17. R. Miyashiro, H. Iwasaki, T. Matsui, Characterizing feasible pattern sets with a minimum number of breaks" in: E. Burke, P. De Causmaecker (Eds.), (PATAT 2002), Lecture Notes in Computer Science 2740, Springer, 2003, pp. 78-99.
18. G.L. Nemhauser, M.A. Trick, Scheduling a Major College Basketball Conference, Operations Research 46 (1)(1998)1-8.
19. R.V. Rasmussen, Benchmark Place Constraints, http://home.imf.au.dk/vinther/benchmarks.
20. J.C. Régin, Minimization of the number of breaks in sports scheduling problems using constraint programming, Constraint programming and large scale discrete optimization, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 57 (2001)115-130.
21. A. Schaerf, Scheduling Sport Tournaments using Constraint Logic Programming, Constraints 4 (1999)43-65.

22. J.A.M. Schreuder, Constructing Timetables for Sport Competitions, Mathematical Programming Study 13 (1980)58-67.
23. J.A.M. Schreuder, Combinatorial aspects of construction of competition Dutch Professional Football Leagues, Discrete Applied Mathematics 35 (1992)301-312.
24. M.A. Trick, Integer and Constraint Programming Approaches for Round Robin Tournament Scheduling, in: E. Burke, P. de Causmaecker (Eds.), (PATAT 2002), Lecture Notes in Computer Science 2740, Springer, 2003, pp. 63-77.
25. M.A. Trick, A Schedule-Then-Break Approach to Sports Timetabling, in: Burke, E.K., Erben, W. (Eds.), (Practice and Theory of Automated Timetabling III) Lecture Notes in Computer Science 2079, Springer-Verlag, Berlin Heidelberg New York, 2001, pp. 242-253.
26. M.A. Trick, Michael Trick's Guide to Sports Scheduling, http://mat.tepper.cmu.edu/sports/Instances/
27. S. Urrutia, C.C. Ribeiro, Maximizing Breaks and Bounding Solutions to the Mirrored Traveling Tournament Problem 2005 (Downloadable from website http://www.inf.puc-rio.br/ useba/publicacoes.htm)