Horbach, Andrei; Bartsch, Thomas; Briskorn, Dirk

Working Paper

Optimally scheduling real world sports leagues by reduction to SAT

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 646

Provided in Cooperation with:
Christian-Albrechts-University of Kiel, Institute of Business Administration

Suggested Citation: Horbach, Andrei; Bartsch, Thomas; Briskorn, Dirk (2009) : Optimally scheduling real world sports leagues by reduction to SAT, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 646, Universität Kiel, Institut für Betriebswirtschaftslehre, Kiel

This Version is available at:
https://hdl.handle.net/10419/147564

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 646

Optimally Scheduling Real World Sports Leagues by Reduction to SAT

Andrei Horbach[1], Thomas Bartsch[2,3], Dirk Briskorn[1]

Mai 2009

[1]: Andrei Horbach, Dirk Briskorn
Christian-Albrechts-Universität zu Kiel,
Institut für Betriebswirtschaftslehre,
Olshausenstr. 40, 24098 Kiel, Germany
http://www.bwl.uni-kiel.de/Prod
horbach@bwl.uni-kiel.de, briskorn@bwl.uni-kiel.de

[2]: SAP AG
Neurottstraße 16, 69190 Walldorf, Germany
thomas.bartsch@sap.com

[3]: TB - Sports League Scheduling
Mosbacher Str. 178, 68259 Mannheim, Germany
thomas.bartsch@conbus.de

Abstract

Several types of constraints must be satisfied by schedules of real world sports leagues, e. g. stadium unavailability, fixed matches, forbidden matches, minimum number of breaks. Taking into account such constraints, determining a feasible schedule becomes a challenging task. Usually, there is no schedule satisfying all given constraints and, hence, some of the constraints are considered as 'soft' ones. There are various models appropriately describing the environment of real sport leagues. Only heuristic methods are known from the literature for solving instances of such models for real dimensions. We consider here a model which satisfies the demands of many sports leagues. We solve our model by a method which consists in formulating the problem as series of instances of the propositional satisfiability problem and adaption of a satisfiability solver for these specific instances. We test our method on two real world sports leagues and solve the problem optimally in each case. Our solver shows good computational results also on generated test instances, which are motivated by real life requirements of sports league scheduling. Our solver can be easily extended to meet the demands of other sports leagues.

Keywords: Timetabling, sports, sports league scheduling, round robin tournaments, soft constraints, propositional satisfiability

1 Introduction and Motivation

Many real world sports leagues are carried out according to a round robin tournament (RRT). A RRT is a structure for competitions between the teams of a league. A RRT is divided into rounds. Each round consists of periods. In each round teams T, $|T| = n$, n is even, compete such that each team plays exactly once against each other team, either at home or away. Furthermore, each team $i, i \in T$ has to play exactly once in each period and, hence, a round consists of exactly $n - 1$ periods. Depending on the number of rounds r, RRTs are called r-RRT. RRTs with $r = 1$ and $r = 2$ are called single RRTs and double RRTs respectively.

Two periods p and p' are called *mirrored*, if i plays at home against j in period p if and only if j plays at home against i in period p' for each team pair i, j. A double RRT is a *mirrored double RRT* if periods p and $n-1+p$ are mirrored for each $p = 1, \ldots, n-1$. A double RRT is *mirrored according to English system*, if the last period of the first round (period $n-1$) is mirrored to the first period of the second round (n), and periods 1 to $n-2$ are mirrored to $n+1$ to $2n-2$ correspondingly.

Note that a mirrored double RRT as well as a double RRT mirrored according to English system can be represented by a single RRT by mirroring matches of the first round to the corresponding periods of the second round. An illustrative example for a single RRT where $n = 6$ is given in Table 1. Here i-j denotes that team i plays at home against team j.

We say team i has a break in period p if and only if i plays twice at home or twice away in periods $p-1$ and p. Since each match has to be carried out at one of the venues of both

period	1	2	3	4	5
match 1	3-4	4-5	2-4	4-6	6-5
match 2	5-2	1-3	5-1	3-5	4-1
match 3	6-1	2-6	6-3	1-2	2-3

Table 1: Single RRT for $n = 6$

opponents, breaks can not be completely avoided. The most common goal concerning breaks is to minimize the number of their occurrences. De Werra [10] shows that the number of breaks cannot be less than $n-2$ in a single RRT. Then, it is easy to see that the number of breaks cannot be less than $2(n-2)$ and $3(n-2)$ in a double RRT and a mirrored double RRT, respectively. Practically, this condition is usually replaced by the requirement that no team has more than one break within a round and one break in the first period of each but the first round. Moreover, breaks must not occur in a given subset of periods. These requirements are imposed to guarantee more fairness for all teams in all phases of the tournament. The single RRT provided in Table 1 has 4 breaks for 6 teams and, therefore, has the minimum number of breaks.

The constraints mentioned above are due to the type of tournament. Beside these, there are further side constraints motivated by real world environments. First, due to various reasons, e. g. other events and construction works, the predefined home venue of a team might be not available in some periods. Thus, in such periods the schedule must prescribe a match of team i at its opponents venue. On the other hand, a team may have a strong preference to play at home in a specific period. Second, the number of simultaneous home matches of specific subsets of teams is limited. For a specific subset of teams the number of matches played at home per period may be restricted to be no more than a given threshold. For example, considering metropolitan areas being home of several teams it can be desired to limit the number of parallel matches for security reasons or in order to limit the burden to the infrastructure. Third, the competition between two teams may be required to be carried out in a certain subset of periods. This can be due to, for example, arranging matches between top level teams at the end of the season in order to provide a thrilling final phase. Fourth, if there is a subset of competitions between teams that are highly attractive, then these should be distributed as equally as possible over the whole season.

In real world leagues teams communicate their requirements to the organizer and the organizer is responsible for the schedule. No team is willing or able to consider the requirements of the other teams when providing its own. Typically, it is impossible to satisfy the requirements of all teams. This is why we consider such requirements as *soft* and try to satisfy as many of them as possible. As scheduling is a multi-stage process, this simple objective usually suffices. Having obtained an optimal schedule, the organizer analyzes it. If she finds out that a team is

discriminated, she can moderate the requirements of the others for the next run of the solver.

Models for sports league scheduling have been the topic of extensive research. A whole stream of papers is based on the analogy between sports league scheduling and edge coloring of complete graphs. Examples are de Werra [10, 11, 12, 13, 14], de Werra et al. [15], and Drexl and Knust [17]. Brucker and Knust [6] and Drexl and Knust [17] analyze the relationship between sports league scheduling and multi-mode resource-constrained project scheduling. Briskorn et al. [5] line out the similarity of structures of single RRTs and planar three index assignments. Real world problems have been considered in Bartsch et al. [1], Della Croce and Oliveri [16], Durán et al. [18], Kendall [22], Rasmussen [26], Ribeiro and Urrutia [28] and Schreuder [29] examine particular formulations.

Extensive overviews of literature on sports leagues scheduling in the context of operations research are provided by Knust [23] and Rasmussen and Trick [27]. Linear integer programming is one possible approach to the problem. Computational experiments show, however, only limited capability to schedule round robin tournaments. Briskorn and Drexl [3] state in their experiments that the industrial solver Ilog CPLEX is not able to find any solution with the minimum number of breaks even after 6 days of computation for instances of single round robin tournament problem with 10 teams. Using their advanced branching schema they solve optimally (regarding the minimization of overall cost assuming that each match imposes individual costs) single RRT instances with at most 10 teams within the runtime of approximately 2 hours. Generalizing the mentioned branching scheme, Briskorn [2] obtains similar results taking into account stadium unavailability and forbidden matches additionally. However, this is not enough for real world scheduling problems.

Our model is very similar to the model considered and solved by Bartsch et al. [1]. The exact methods suggested by Bartsch et al. [1] are only able to solve instances with at most 8 teams. To solve real world instances the authors use a rather elaborated 'semi greedy' heuristic which determines good but not optimal solutions.

The method we suggest in this paper solves the problem without compromising the optimality. The key element of our method is a reduction to the conjunctive normal form satisfiability problem (SAT). To the best of our knowledge there is no approach to sports leagues scheduling employing a SAT solver described in the literature so far.

The decision version of SAT has been historically the first problem proven to be NP-complete (Cook [7] and Levin [24]).

Recently, the SAT techniques have been developed to such an extent that for several combinatorial problems a reduction to SAT and the use of an adapted SAT solver can be a good alternative to a problem specific solver running on the original problem formulation.

One of the simplest but fast and robust implementations of the SAT solver is MiniSat developed by Eén and Sörensson [19]. MiniSat is an extended C++ implementation of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm. In this paper we adapt MiniSat to achieve efficiency

in the specific domain of sports league scheduling. Our method can be used in combination with any SAT solver based on the DPLL algorithm.

The remainder of this paper is organized as follows. In Section 2 we formulate the sports league scheduling problem formally and represent it by means of a SAT model in Section 3. Section 4 provides details about the used SAT solver and its adaption for sports scheduling. In Section 5 we line out test instances and computational results. Section 6 concludes.

2 Round Robin Tournament Problem with Side Constraints

2.1 General Problem Formulation

Exactly n teams play in r rounds. Both n and r are even. Each round P_l, $l \in \{1,\ldots,r\}$, consists of $(n-1)$ consecutive periods $\{(l-1)\times(n-1)+1,\ldots,l\times(n-1)\}$. The season P is the union of all rounds: $P := \bigcup_{l=1,\ldots,r} P_l$.

Let S be a subset of $T \times T \times P$. We say that team i plays against team j at home (away) in period p according to S if $(i,j,p) \in S$ ($(j,i,p) \in S$). Set S is a *schedule* if according to S

1) each team plays in each period,

2) for each pair of teams, there is exactly one period in each round where they play against each other,

3) for an odd round l and each pair of teams $\{i,j\}$ team i plays at home against j at least once and j plays at home against i at least once within rounds l and $l+1$.

Given a schedule S, we define a period p, $p>1$, to be a *break period* of team i, if i plays at home in both periods $p-1$ and p, or i plays away in both periods $p-1$ and p. Given a subset P^{bf} of P as a set of break-free periods, schedule S is *break consistent* with respect to P^{bf} if

4) no team has more than one break period per round,

5) no period from P^{bf} is a break period for any team.

Set P^{bf} usually consists of periods in the beginning and the end of each round.

Usually, the schedule must be *mirrored* according to some system, i.e. the matches of each period in any even round l are exactly the matches of some predefined period in round $l-1$ with inverse home rights.

Next, we define several additional restriction that can be imposed on S. Various real world sports scheduling problems use different combinations of these constraints.

For a subset TM of $T \times T$, constraint `top_games`(TM) is defined as:

6) no two pairs of TM play at the same period according to S.

For a pair of teams $\{i, j\}$ and a period p, constraint `region`$(\{i, j\},\ p)$ is defined as:

7) at least one of teams i or j plays away in period p according to S.

This constraint is usually imposed on the schedule for two teams sharing the same stadium.

A mirrored *break consistent* schedule is *feasible* if constraint `top_games`(TM) and all constraints `region`$(\{i, j\},\ p)$, $\{i, j\} \in T \times T$, $p \in P$, are satisfied.

For a teams i and a period p soft constraint `unavailable_stadium`$(i,\ p)$ is defined as:

8) team i plays away in period p.

We call a feasible schedule *efficient* for a given set US of `unavailable_stadium()` constraints, if it minimizes the number of violated constraints from US.

For a pair of teams (i, j) and a subset of periods P' soft constraint `set_pair`$(\{i, j\},\ P')$ is defined as:

9) there is at least one home match of i against j in one of the periods from P'.

We impose a lexicographic order on the two objectives, that is, first, the number of violated constraints of type `unavailable_stadium` and, then, the number of violated constraints of type `set_pair` is to be minimized.

In its most general form our problem can be formulated as follows. Given an even integer n as number of teams, an even integer r as number of rounds, a mirroring system (i.e. the periods must be mirrored 'classically' or according to English system), a `top_games()` constraint, a set of `region` constraints R, a set of `unavailable_stadium` constraints US, and a set of `set_pair()` constraints SP. Find an efficient schedule minimizing the number of violated constraints from SP.

To simplify our following notations we will consider sets US and SP as ordered sets and denote by US_s and SP_s constraint number s in the corresponding set.

2.2 Real World Problems

2.2.1 First Austrian Soccer League

In the first Austrian soccer league $n = 10$ teams play in $r = 4$ rounds. Hence, $|P| = 36$ periods have to be played and each round consists of 9 periods, see also Bartsch et al. [1]. Rounds 1 and 2 as well as rounds 3 and 4 are mirrored as a double RRT according to English system given by the following restriction that must be satisfied for each feasible schedule S:

10a) if $(i, j, p) \in S$, then $(j, i, p') \in S$ where $p' = ((p + 1) \bmod (n - 1)) + l(n - 1) + 1$, for each $p \in P_l$, $l \in \{1, 3\}$,

i.e. the first period of the second round is the mirrored last period of the first round, the second period of the second round is the first period of the first round, the third is the second and so on.

Table 2 summarized the information on involvement of teams in constraints introduced in Section 2.1 based on data for season 2004/2005.

Team	top_games	region	set_pair	unavailable_stadium
Grazer AK	✓	✓	✓	
Austria Wien	✓	✓	✓	
Pasching				
Rapid Wien	✓	✓	✓	✓
Bregenz				
Admira				
Salzburg				✓
Matttersburg				✓
Sturm Graz	✓	✓	✓	
Kärnten				✓

Table 2: Teams of the First Austrian Soccer League and constraints they are involved in.

Furthermore, both double RRTs are connected by the following constraint:

11a) if $(i, j, p) \in S$, then $(i, j, p + 1) \notin S$ and $(j, i, p + 2) \notin S$, for $p \in \{17, 18\}$.

The schedule must be break consistent for $P^{bf} = \{2, 18, 19, 20, 36\}$. These are the very first and the very last periods of each half season, were breaks are considered as very undesirable.

Constraint `top_games` is instantiated by

$$TM = \{(\textit{Austria Wien}, \textit{Rapid Wien}), (\textit{Grazer AK}, \textit{Sturm Graz})\}.$$

These matches are considered as especially interesting because of the historical local derbies (Vienna Derby and Graz Derby) between the teams of each pair.

Constraints `region` are instantiated by pairs {*Austria Wien*, *Rapid Wien*} and {*Grazer AK*, *Stuŕm Graz*}. The teams of each pair share one main stadium and therefore can not play home simultaneously.

Potentially, each teams can be involved in constraints of type `unavailable_stadium` and `set_pair`. As for season 2004/2005, the most restricted in its ability to use the stadium is *Kärnten* and *Salzburg* with their 5 periods of closed stadium each. Some other teams don't have any restrictions of this type.

A further requirement is a special one and is not covered in the general problem description. It is introduced to satisfy the supporters of teams *Rapid Wien* and *Mattersburg*. It is well known, that many people support both these teams and desire to visit the home matches of both of them. To do them a favor, the league asks for a schedule where these teams do not have more than four parallel home matches. The same is true for teams *Austria Wien* and *Admira*.

2.2.2 First German Handball League

In the first German handball league $n = 18$ teams play in $r = 2$ rounds. Hence, there must be $|P| = 34$ periods in the season and 17 periods in each round, see also Bartsch et al. [1]. The season is played as a mirrored double RRT, that is, a feasible schedule S must satisfy:

10g) if $(i, j, p) \in S$ with $p \in P_1$, then $(j, i, p + n - 1) \in S$.

Table 3 lists the teams' names, strength group they belong to and involvement in constraints introduced in Section 2.1 based on data for season 2006/2007.

The set of break free periods are instantiated by

$$P^{bf} = \{2, 4, 6, 13, 15, 17, 19, 21, 23, 30, 32, 34\}.$$

The set of top matches is instantiated by $TM = \{(i, j) \mid i < j \leq 6\}$, that is all matches between teams in strength group 1 are rated as top matches. Potentially, all teams are involved in constraints of type `set_pair` and `unavailable_stadium`.

3 SAT formulation

In this section we reduce the sports league scheduling problem described in Section 2 to series of instances of the conjunctive normal form satisfiability problem.

Team	group	top_games	set_pair	unavailable_stadium
Vfl Gummersbach	1	✓	✓	
SG Flensburg-Handewitt	1	✓		✓
SC Magdeburg	1	✓	✓	
TBV Lemgo	1	✓	✓	
THW Kiel	1	✓	✓	✓
HSG Nordhorn	1	✓		✓
TV Growallstatt	2			✓
SG Kronau-Östringen	2			✓
HSV Hamburg	2			✓
FA Göppingen	2			
MT Melsungen	2			
Tus-N-Lübbecke	2			
HSG Wetzlar	3			✓
Vfl Pfullingen-Stuttgart	3			✓
SC Concordia Delitzsch	3			
HSG Düsseldorf	3			
TSV GWD Minden	3			
Wilhelmshavener HV	3			

Table 3: Teams of the First German Handball League and constraints they are involved in.

A propositional formula is a formula that is defined over variables that take values in the set $\{true, false\}$. A propositional formula is in conjunctive normal form (CNF) if it is a conjunction (AND, $\wedge$) of *clauses*, where each clause is a disjunction (OR, $\vee$) of *literals*. A literal is either a variable, then it is called a positive literal, or its negation (NOT, $\neg$), then it is called a negative literal.

The conjunctive normal form satisfiability problem (CNF SAT or just SAT) is defined as follows. Given a CNF, does there exist an assignment of the variables, such that the CNF evaluates to $true$ under such assignment? If such a satisfying assignment exists, we say that the formula is satisfiable and the assignment is called a *model*. Otherwise the formula is unsatisfiable.

In the following we first formulate a CNF whose satisfying assignments encode *feasible* schedules. Then we define a variable for each soft constraint `set_pair()` and `unavailable_stadium()` that indicates whether the constraint is violated. Next, we state clauses preventing more than a certain number of soft constraints of each type from being violated.

First, we introduce necessary decision variables. For each team pair (i, j) and each period p we define a *match variable* x_{ijp} as

$$x_{ijp} = \begin{cases} true & \text{if team } i \text{ plays at home against } j \text{ in period } p, \\ false & \text{otherwise.} \end{cases}$$

For each team i and each period p we define a *home break variable* hb_{ip} and an *away break variable* ab_{ip} as

$$hb_{ip} = \begin{cases} true & \text{if team } i \text{ has a home break in period } p, \\ false & \text{otherwise,} \end{cases}$$

$$ab_{ip} = \begin{cases} true & \text{if team } i \text{ has an away break in period } p, \\ false & \text{otherwise.} \end{cases}$$

For each team i and each period p we define a *home match variable* h_{ip} as

$$h_{ip} = \begin{cases} true & \text{if team } i \text{ plays at home in period } p, \\ false & \text{otherwise.} \end{cases}$$

Then, constraints 1) to 3) are transformed into clauses (1) to (5). Here, clauses (1) to (3) force each team to play exactly once per period. Clause (1) ensures that each team has a match in each period. Clauses (1) to (3) guarantee that each team plays exactly once per period. Clause (4) and (5) ensures that each pair of teams has a match in each round and matches of the same pair of teams are carried out in different venues in consecutive rounds, respectively. Then, clauses (6) and (7) force the variable h_{ip} to indicate if team i plays at home or away in period p.

$$\bigvee_{j \in T \setminus \{i\}} (x_{ijp} \vee x_{jip}) \qquad \forall i \in T, p \in P \tag{1}$$

$$\neg x_{ijp} \vee \neg x_{ikp} \qquad \forall i, j, k \in T, i \neq j, j \neq k, i \neq k, p \in P_l, l \in \{1, \ldots, r-1\} \tag{2}$$

$$\neg x_{jip} \vee \neg x_{kip} \qquad \forall i, j, k \in T, i \neq j, j \neq k, i \neq k, p \in P_l, l \in \{1, \ldots, r-1\} \tag{3}$$

$$\bigvee_{p \in P_l} (x_{ijp} \vee x_{jip}) \qquad \forall i, j \in T, i \neq j, l = \{1, \ldots, r\} \tag{4}$$

$$\bigvee_{p \in P_l \bigcup P_{l+1}} x_{ijp} \qquad \forall i, j \in T, i \neq j,\ l \in \{1, \ldots, r\}, l \text{ is odd} \tag{5}$$

$$\neg x_{jip} \vee \neg h_{ip} \qquad \forall i, j \in T, j \neq i, p \in P \tag{6}$$

$$\neg x_{ijp} \vee h_{ip} \qquad \forall i, j \in T, j \neq i, p \in P \tag{7}$$

The following clauses connect *home variables* with *home break variables* and *away break variables*, respectively. Clause (8) causes $hb_{ip} = true$ if i plays at home in both, $p-1$ and p, according to the assignments of the *home variables*. Clauses (9) and (10) set $h_{i(p-1)} = h_{ip} = true$ if i has a home break in p. Clauses (11) to (13) are strictly analogue for *away break variables*.

$$hb_{ip} \vee \neg h_{ip} \vee \neg h_{i(p-1)} \qquad \forall i \in T, p \in P, p > 1 \tag{8}$$

$$\neg hb_{ip} \vee h_{ip} \qquad \forall i \in T, p \in P, p > 1 \tag{9}$$

$$\neg hb_{ip} \vee h_{i(p-1)} \qquad \forall i \in T, p \in P, p > 1 \tag{10}$$

$$ab_{ip} \vee h_{ip} \vee h_{i(p-1)} \qquad \forall i \in T, p \in P \tag{11}$$

$$\neg ab_{ip} \vee \neg h_{ip} \qquad \forall i \in T, p \in P, p > 1 \tag{12}$$

$$\neg ab_{ip} \vee \neg h_{i(p-1)} \qquad \forall i \in T, p \in P, p > 1 \tag{13}$$

Constraints 4) to 5) are transformed into clauses (14) to (18). Clauses (14) to (16) forbid two breaks for a team in a round. Clauses (17) and (18) trivially prevent breaks in break free periods.

$$\neg hb_{ip} \vee \neg hb_{ip'} \quad \forall i \in T, p, p' \in P_l, p' > p, l \in \{1, \ldots, r\} \tag{14}$$

$$\neg ab_{ip} \vee \neg ab_{ip'} \quad \forall i \in T, p, p' \in P_l, p' > p, l \in \{1, \ldots, r\} \tag{15}$$

$$\neg hb_{ip} \vee \neg ab_{ip'} \quad \forall i \in T, p, p' \in P_l, p' \neq p, l \in \{1, \ldots, r\} \tag{16}$$

$$\neg hb_{ip} \quad \forall i \in T, p \in P^{bf} \tag{17}$$

$$\neg ab_{ip} \quad \forall i \in T, p \in P^{bf} \tag{18}$$

Constraints 6) and 7) are transformed into clauses (19) to (20) straightforwardly.

$$\neg x_{ijp} \vee \neg x_{i'j'p} \quad \forall (i,j) \in TM, (i',j') \in TM, p \in P \tag{19}$$

$$\neg h_{ip} \vee \neg h_{jp} \quad \forall \texttt{region}(\{i,j\},\ p) \tag{20}$$

The clauses specified so far suffice to represent constraints 1) to 7). However, we add several clauses being non-trivially implied by those introduced above in order to improve performance of the solver.

The following clauses state that there is a home-break in period p if and only if there is an away-break in p. This clause is due to a well-known property of RRTs having the minimum number of breaks: in each period there is either a home-break as well as an away-break or no break at all, see Briskorn and Drexl [4] for example. Note that we do not have the minimum number of breaks here. However, this property holds for RRTs with exactly one break per team also. Clause (21) states that there is either no home-break of i in p or there must be an away-break in p as well. Clause (22) is analogous.

$$\bigvee_{j \in T \setminus \{i\}} ab_{jp} \vee \neg hb_{ip} \quad \forall i \in T, p \in P \tag{21}$$

$$\bigvee_{j \in T \setminus \{i\}} hb_{jp} \vee \neg ab_{ip} \quad \forall i \in T, p \in P \tag{22}$$

Clauses (23) and (24) set *break period variable* $b_p = true$ if and there is a home-break in period p and $b_p = false$ otherwise. Clause (25) states that there cannot be three consecutive break periods. This is known for RRTs having the minimum number of breaks and for RRTs with exactly one break per team and round, see Briskorn and Drexl [4] for example. Furthermore, there cannot be 6 breaks periods in 8 consecutive periods which can be represented by a set of clauses as well.

$$\neg hb_{ip} \vee b_p \quad \forall i \in T, p \in P \tag{23}$$

$$\bigvee_{i \in T} hb_{ip} \vee \neg b_p \quad \forall p \in P \tag{24}$$

$$\neg b_{ip} \vee \neg b_{i(p+1)} \vee \neg b_{i(p+2)} \quad \forall l \in \{1, \ldots, r\}, p \in P_l, p < l(n-1) - 2 \tag{25}$$

While clauses (1) to (25) ensure a *feasible* schedule we neither determine no restrict the number of violated soft constraints.

For each constraint $SP_s \in SP$ we define a *set pair violation variable* vsp^s as

$$vsp^s = \begin{cases} true & \text{if constraint } SP_s \text{ is violated,} \\ false & \text{otherwise.} \end{cases}$$

For each constraint $US_s \in US$ we define a *unavailable stadium violation variable* vus^s as

$$vus^s = \begin{cases} true & \text{if constraint } US_s \text{ is violated,} \\ false & \text{otherwise.} \end{cases}$$

Clauses (26) and (27) ensure that $vsp^s = true$ and $vus^s = true$, respectively, if the corresponding soft constraint is violated.

$$\neg h_{ip} \vee vus^s \qquad \forall \texttt{unavailable\_stadium}(i,\ p) = US_s,\ US_s \in US \tag{26}$$

$$\bigvee_{p \in P'} x_{ijp} \vee vsp^s \qquad \forall \texttt{set\_pair}(\{i,j\},\ P') = SP_s,\ SP_s \in SP \tag{27}$$

Now, we introduce clauses that restrict the number of violated soft constraints to be at most $UB_{stadiums}$ and UB_{pairs}, respectively.

$$\bigvee_{US_s \in C} \neg vus^s \qquad \forall C \subseteq US, |C| = UB_{stadiums} + 1 \tag{28}$$

$$\bigvee_{SP_s \in C} \neg vsp^s \qquad \forall C \subseteq SP, |C| = UB_{pairs} + 1 \tag{29}$$

Hence, the CNF which is the conjunction of (1) to (29) asks for a *feasible* schedule having no more than $UB_{stadiums}$ violations of constraints `unavailable_stadium()` and no more than UB_{pairs} violations of constraints $\texttt{set\_pair}(\{i,j\},\ P')$, respectively.

Note that the number of clauses (28) and (29) is exponential in the number of constraints of the corresponding type. Section 4 explains how the SAT solver can be adapted to dynamically handle a huge number of such clauses.

Next, we introduce clauses representing constraints for the Austrian soccer league, first, and for the German handball league, afterwards. Clauses (30) and (31) represent constraint 10a) for the Austrian soccer league.

$$\neg x_{ijp} \vee x_{ji(((p+1) \bmod (n-1))+l(n-1)+1)} \qquad \forall i,j \in T, j \neq i, p \in P_l, l \in \{1,3\} \tag{30}$$

$$x_{jip} \vee \neg x_{ji(((p+1) \bmod (n-1))+l(n-1)+1)} \qquad \forall i,j \in T, j \neq i, p \in P_l, l \in \{1,3\} \tag{31}$$

Clause (32) represents constraint 11a) for the Austrian soccer league.

$$\bigvee_{p \leq p' \leq p+2} (\neg x_{ijp'} \vee \neg x_{jip'}) \qquad \forall i,j \in T, j \neq i, p \in \{2n-3, 2n-2\} \tag{32}$$

Clauses (33) and (34) represent constraint 10g) for the German handball league.

$$\neg x_{ijp} \vee x_{ji(p+n-1)} \qquad \forall i,j \in T, j \neq i, p \in P_1 \tag{33}$$

$$x_{jip} \vee \neg x_{ji(p+n-1)} \qquad \forall i,j \in T, j \neq i, p \in P_1 \tag{34}$$

4 The Solver

The basic algorithm for SAT has been proposed by Davis and Putnam [8] and Davis et al. [9], see Algorithm 1. This algorithm not only answers the question of satisfiability but also finds a satisfying assignment if it exists. This algorithm is the basis of the most modern complete SAT solvers. A version of this algorithm is implemented by Eén and Sörensson [19] in their solver MiniSat.

Algorithm 1 DPLL-Algorithm

```
decision_level = 0
while true do
  propagate()
  if not conflict then
    if all variables assigned then
      return Satisfiable
    else
      ++decision_level, decide()
    end if
  else
    analyze()
    if top-level conflict found then
      return Unsatisfiable
    else
      backtrack()
    end if
  end if
end while
```

The algorithm starts at decision level zero, chooses a literal (a variable and the $true$ or the $false$ direction) to branch on, makes the $true$ assignment of the literal and completes it by *unit propagation* which is done in `propagate()`. If a variable is chosen for branching it is marked as *decision* variable. If the value is assigned to the variable at unit propagation, it is a *propagated* variable. If `propagate()` returns no conflict, the algorithm steps to the next decision level and chooses a next literal to branch on. If under the current partial assignment `propagate()` determines a conflict (a clause can not be satisfied under this partial assignment), the conflict is returned. The conflict is analyzed. The result of `analyze()` is the backtracking level, the largest level up to that all assignments have to be canceled such that the last conflict is not

$false$ under the current partial assignment. In the state-of-the-art solvers `analyze()` returns also a *learnt clause*, which is a consequence of the other clauses in the database and, hence, must be satisfied by each satisfying assignment and can be considered as a *reason* of the last conflict. The learnt clause is added to the database and propagation goes on.

The components of DPLL are:

- branching procedure `decide()`, which makes choice of the next literal to branch on;
- procedure `propagate()`, which makes unit propagation and determines conflicts;
- procedure `analyze()`, which analyzes conflicts, decides about the level of backtracking, and generates learnt clauses;
- procedure `backtrack()`, which unwinds all made assignments up to the backtracking level returned by `analyze()`.

Two techniques should be noted as making special contribution to the efficiency of modern SAT solvers. *Clause learning*, being involved in unit propagation, speeds up the recognition of future conflicts and may lead to an impressive rise of performance. *The watched literals scheme* proposed by Moskewicz et al. [25] and first implemented in their solver zChaff is now a standard method for efficient constraint propagation. The key idea behind this scheme is to maintain two special literals for each not yet satisfied clause that are not $false$ under the current partial assignment. As long as the clause has two such literals, it cannot be involved in unit propagation. Only if $false$ is assigned to one of these two literals, `propagate()` assigns $true$ to the other literal, if the clause is not satisfied yet. More details on state-of-the-art SAT solvers can be found in Gomes et al. [20].

Because of the huge amount of clauses guaranteeing upper bounds for the number of violated constraints we add them to the solver only if they are violated or can be used for unit propagation. We adapt the method `propagate()` so that each time a soft constraint violation variable is fixed to *true*, it tests if the number of violated soft constraints of the same type is greater than the current upper bound. If it is the case, a conflict clause of type (28) or (29) is generated and returned to the overall search procedure. If the number of violated soft constraints is exactly the corresponding upper bound, a set of all relevant clauses of type (28) or (29) is generated, added to the database and immediately used for unit propagation. As result of the unit propagation all other soft constraint violation variables of the same type are fixed to *false*.

It is well-known, that the branching order significantly influence the performance of a SAT solver. Briskorn [2] and Briskorn and Drexl [3] show that the computational effort in order to find feasible and optimal solutions, respectively, to optimization problems concerning RRTs can be significantly reduced by branching on *home break variables* and *away break variables*

first. The advantage of this branching scheme can be explained by the fact that fixing the break of a team implies the venue (home/away) of this team in each period if the number of breaks per team in a round is limited to be no more than one. Therefore, in our case it was critically to make the solver to branch first on the break variables hb_{ip} and ab_{ip}.

5 Computational Study

We tested our solver on real world instances as well as on instances designed for evaluation purposes and outline results in Sections 5.1 and 5.2, respectively.

Our results were obtained on a PC with Intel Core 2 Quad CPU Q9550 and 4 GB of RAM running under Windows 2003 Server. Our solver uses only one of four available processor kernels. The code is written in C++ and compiled with Microsoft Visual C++ 2005 compiler.

5.1 Solving Real World Instances

We tested the solver on an instance of the First German Handball League (season 2006/2007) and on an instance of the First Austrian Soccer League (season 2004/2005). It should be noted, that the solver at hand was not used for scheduling of these two leagues at that time. Instead, an elaborated heuristic from Bartsch et al. [1] was used. It required quite a lot of problem specific analysis and provided good, but not optimal solutions.

Our solver needed 194 seconds to optimally solve the instance of the First German Handball League. The optimal solution satisfies 33 out of 46 `unavailable_stadion` constraints and all 16 `set_pair` constraints.

The satisfaction of all `set_pair` constraints is explained by the fact that some evidently unsatisfiable constraints were sorted out already on the stage of data mining. It would be not necessarily for our solver.

The instance of the First Austrian Soccer League was solved optimally within 11 seconds. 16 out of 17 `unavailable_stadion` constraints and 12 out of 13 `set_pair` constraints were satisfied.

5.2 Solving Designed Instances

5.2.1 Instance Generator

To test the solver we generated various test instances of different dimensions. We simulate factors that influence constraints appearing in real life problems of the First German Handball League. We generate instances with $n = 10, \ldots, 20$ teams, n taking even values.

Usually, the league needs a schedule satisfying a restriction of a type: each team should have in the first $2k$ and in the last $2k$ periods of each round exactly the same number of home and away matches. This restriction can be expressed by requiring the periods $2i, i = 1, \ldots, k$ and $n - 1 - 2k + 2i, i = 1, \ldots, k$ to be break free. We set $k = 1$ for instances with 10 teams and require hereby that in the first 2 and in the last 2 periods of each round each team plays the same number of home and away matches. For $n = 12$ and $n = 14$ we set $k = 2$, and for $n = 16, \ldots, 20$ we set $k = 3$.

In the reality, most of the teams have at their disposal more than one stadium. Usually, one of the stadiums is more preferable for all parties involved. This stadium can be closed due to reconstruction or, more often, due to other events (e.g. rock concerts) planned for the same time. The probability that a stadium is unavailable in a period is different for different teams. As in our real life instance, the best stadium of *HSH Hamburg* is closed in 9 out of 34 periods, the stadium of *THW Kiel* is closed in 5 periods. For *TBV Lemgo* there are no restrictions to use the stadium at all.

Without loss of generality, we assume that the stadium of team 1 is the least and the stadium of team n is the most probable to be closed for the matches in each particular period. So we set the probability that the stadium of team i is closed in a particular period to be $\frac{i \times \texttt{pr\_forbidden\_stad}}{n \times |P|}$. Where `pr_forbidden_stad` is a parameter of the instance generator.

Then, the *high quality group* is generated by drawing k teams on random. We set $k = 4$ for instances with up to 12 teams, $k = 5$ for instances with 14 teams, and $k = 6$ for instances with 16 and more teams. All n teams have equal probability to be chosen in the high quality group and are chosen independently of each other.

The `set_pair` constraints arise mostly from the fact, that some periods are more attractive for a team and desired to be used for 'important' matches. The attractiveness of a period can be influenced by seasonal issues or, more frequently, by availability or unavailability of a better stadium.

We instantiated three parameters to generate `set_pair` constraints. These are: `prob_be_chosen`, `max_prob_be_chosen_opp` and `plus_factor`. The generation is then organized as follows. We pick a team i with probability `prob_be_chosen` as the host team for a `set_pair` constraint. Then we generate a probability pb, which is drown as uniformly distributed random value from the interval `[0, max_prob_be_chosen_opp]`. For each team $\{1, \ldots n\} \setminus \{i\}$ we decide with probability pb if it is the visitor team of i in a `set_pair` constraint. Having chosen l different visitor teams we generate l `set_pair` constraints with i as the host team. Each such a `set_pair` constraint we instantiate with the same set of periods. l till $l +$ `plus_factor` periods are drawn independently on random for this purpose.

In total, we instantiate 4 parameters to generate an instance of each dimension. This results in 16 different combinations of parameter values. For each such a combination five instances for different *seed values* are generated. Consequently, we generate 80 instances of each dimension.

n	type of solution	average time (sec.)	solved within time limit (sec.)							
			1	5	10	30	60	120	300	600
10	feasible	0.07	80							
10	efficient	0.15	79	80						
10	optimal	0.30	76	79	80					
12	feasible	0.10	80							
12	efficient	0.21	80							
12	optimal	1.61	74	78	78	79	79	80		
14	feasible	0.28	80							
14	efficient	0.76	68	79	80					
14	optimal	3.80	44	69	71	74	75	76	76	76
16	feasible	1.12	50	79	79	80				
16	efficient	17.02	8	61	75	76	78	78	78	79
16	optimal	31.66	0	37	54	61	66	68	69	72
18	feasible	7.32	3	44	63	80				
18	efficient	29.02	0	5	16	52	74	78	78	79
18	optimal	31.99	0	2	3	29	50	61	62	62
20	feasible	106.24	0	6	10	21	34	48	68	75
20	efficient	279.31	0	0	0	3	5	14	37	63
20	optimal	257.93	0	0	0	1	1	6	20	30

Table 4: Results: the average time over solved instances and the number of solved instances (out of 80) of each set within each time limit.

5.2.2 Computational Results

We tested our solver on 6 instance sets with the dimension of 10 to 20 teams limiting the run time to 600 sec. for each instance. The results for each of the instance sets are summarized in Tables 4 and 5.

Table 4 provides the number of instances (out of 80 in each set) with determined feasible, efficient and optimal solutions for each instance set within the given timelimit. Additionally, the average time (calculated only over instances solved within 600 sec.) to find and prove each type of solution is given. We could solve optimally all instances with up to 12 teams. For almost all instances with up to 18 teams an efficient solution was found and proved. Most of the instances with up to 18 teams were solved to optimality.

Table 5 provides the fraction (calculated for all instances and for instances solved efficiently/optimally) of satisfied `unavailable_stadions` and `set_pairs` constraints for each problem size.

n	#solved			percent of satisfied constraints			
				all instances		efficient/optimally solved	
	feasible	efficient	optimal	unavailable_stadions	set_pairs	unavailable_stadions	set_pairs
10	80	80	80	85.81	73.53	85.81	73.53
12	80	80	80	87.43	69.81	87.43	69.81
14	80	80	76	87.43	66.61	87.43	68.60
16	80	79	72	88.34	69.51	88.47	73.28
18	80	79	62	88.60	60.92	88.70	69.26
20	75	63	30	82.75	34.90	88.94	71.88

Table 5: Summarized results.

6 Conclusions and Outlook

We show the efficiency of our approach for constructing schedules for round robin tournaments with up to 20 teams. The performance decrease rapidly with the growth of the number of teams. The dimension of 22 teams seems to be the limit for out method in its current form. This suffices however for scheduling most of real world sport leagues playing round-robin tournaments.

For solving instances with 22 and more teams we see a possibility for improvements. It is well known, that a SAT solver based on DPLL algorithm can not efficiently deal with some kinds of infeasibility, as e.g. the infeasibility of the well known pigeon-hole problem (Haken [21]). We believe, that a similar situation occurs frequently in our problem. We think, that this can be improved by hybridizing a SAT solver with a linear solver or with a bipartite feasible matching solver.

Furthermore, our method provides the capability to efficiently deal with even more types of (soft) constraints, so that even more attractive schedules can be constructed.

References

[1] T. Bartsch, A. Drexl, and S. Kröger. Scheduling the Professional Soccer Leagues of Austria and Germany. *Computers & Operations Research*, 33:1907–1937, 2006.

[2] D. Briskorn. A Branching Scheme for Minimum Cost Tournaments with regard to Real World Constraints. *Working Paper*, 2009.

[3] D. Briskorn and A. Drexl. Branching Based on Home–Away–Pattern Sets. In K.-H. Waldmann and U. M. Stocker, editors, *Operations Research Proceedings 2006 - Selected*

Papers of the Annual International Conference of the German Operations Research Society (GOR), Karlsruhe, September 6th - 8th 2006, pages 523–528. Springer, Berlin, Germany, 2007.

[4] D. Briskorn and A. Drexl. A branching scheme for finding cost-minimal round robin tournaments. *European Journal of Operational Research*, 197(1):68–76, 2009.

[5] D. Briskorn, A. Drexl, and F. C. R. Spieksma. Round Robin Tournaments and Three Index Assignment. *Working Paper*, 2006.

[6] P. Brucker and S. Knust. *Complex Scheduling*. Springer, Berlin, 2006.

[7] S. A. Cook. *Conf. Record of 3rd STOC*, chapter The complexity of theorem proving procedures, pages 151–158. Shaker Height, OH, May 1971.

[8] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960. ISSN 0004-5411.

[9] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[10] D. de Werra. Geography, Games and Graphs. *Discrete Applied Mathematics*, 2:327–337, 1980.

[11] D. de Werra. Scheduling in Sports. In P. Hansen, editor, *Studies on Graphs and Discrete Programming*, pages 381–395. North-Holland, Amsterdam, The Netherlands, 1981.

[12] D. de Werra. Minimizing Irregularities in Sports Schedules Using Graph Theory. *Discrete Applied Mathematics*, 4:217–226, 1982.

[13] D. de Werra. On the Multiplication of Divisions: The Use of Graphs for Sports Scheduling. *Networks*, 15:125–136, 1985.

[14] D. de Werra. Some Models of Graphs for Scheduling Sports Competitions. *Discrete Applied Mathematics*, 21:47–65, 1988.

[15] D. de Werra, T. Ekim, and C. Raess. Construction of Sports Schedules with Multiple Venues. *Discrete Applied Mathematics*, 154:47–58, 2006.

[16] F. Della Croce and D. Oliveri. Scheduling the Italian Football League: An ILP–Approach. *Computers & Operations Research*, 33:1963–1974, 2006.

[17] A. Drexl and S. Knust. Sports League Scheduling: Graph- and Resource–Based Models. *Omega*, 35:465–471, 2007.

[18] G. Durán, M. Guajardo, J. Miranda, D. Sauré, and A. Weintraub. Scheduling the Chilean Soccer League by Integer Programming. *Interfaces*, 37:539–552, 2007.

[19] N. Eén and N. Sörensson. *Theory and Applications of Satisfiability Testing*, chapter An extensible SAT-solver, pages 502–518. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004.

[20] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman. *Handbook of Knowledge Representations*, chapter Satisfiability Solvers, pages 89–132. Elsevier, 2008.

[21] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

[22] G. Kendall. Scheduling English Football Fixtures Over Holiday Periods. *Journal of the Operational Research Society*, 59:743–755, 2008.

[23] S. Knust. Classification of literature on sports scheduling, 2009. URL `http://www.inf.uos.de/knust/sportlit_class/`. (January 29th, 2009).

[24] L. Levin. Universal sequential search problem. *Problems of Information Transmission*, 9: 265–266, 1973.

[25] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. *DAC*, pages 530–535, 2001.

[26] R. V. Rasmussen. Scheduling a Triple Round Robin Tournament for the Best Danish Soccer League. *European Journal of Operational Research*, 185:795–810, 2008.

[27] R. V. Rasmussen and M. A. Trick. Round Robin Scheduling – a survey. *European Journal of Operational Research*, 188:617–636, 2008.

[28] C. C. Ribeiro and S. Urrutia. Scheduling the Brazilian soccer tournament with fairness and broadcast objectives. Lecture Notes in Computer Science 3867, pages 149–159, 2007.

[29] J. A. M. Schreuder. Combinatorial Aspects of Construction of Competition Dutch Professional Football Leagues. *Discrete Applied Mathematics*, 35:301–312, 1992.