# Minimizing Breaks in Sport Scheduling with Local Search

**P. Van Hentenryck** and **Y. Vergados**
Brown University, Box 1910
Providence, RI 02912

### Abstract

Sport scheduling has received significant attention in the recent years and has contributed many challenging applications for optimization technologies. This paper reconsiders the problem of minimizing the number of breaks in round-robin tournaments, an application which has been recently tackled by sophisticated constraint and mathematical programming approaches. It proposes a simulated annealing algorithm that finds optimal solutions very quickly for the largest instances available. On the larger instances (e.g., 28 teams), the algorithm is several orders of magnitude faster than earlier approaches. In addition, the experimental results indicate that near-optimal solutions can be obtained within 10 seconds for large instances. Of particular interest is the simplicity of the implementation, its automatic tuning of the parameters, and the fact that simulated annealing is an effective technique for another sport-scheduling application.

## Introduction

The scheduling of sport leagues has become an important class of combinatorial optimization applications in recent years for two main reasons. On the one hand, sport leagues represent significant sources of revenue for radio and television networks around the world. On the other hand, sport leagues generate extremely challenging optimization problems. See (Easton, Nemhauser, & Trick 2001) for an excellent review of these problems, recent research activities, and several solution techniques.

This paper considers the constrained break minimization problem, a classical application in sport scheduling that has been widely studied (e.g., (Schreuder 1992; Régin 1998; Trick 2000; Elf, Jünger, & Rinaldi 2003)). Given a schedule for a round-robin tournament without specifications of the home/away roles, it consists of finding out which team plays home (resp. away) in each of the games in order to minimize the number of breaks in the schedule. Breaks are a common quality measure for schedules and arise when a team plays two consecutive game at home or two consecutive games away. The break minimization problems arise in many sport-scheduling applications, including the scheduling of soccer leagues in Europe.

Sophisticated optimization approaches have been applied to the break minimization problem, including constraint programming (Régin 1998), integer programming (Trick 2000) and, very recently, an elegant reduction to a cut maximization problem (Elf, Jünger, & Rinaldi 2003). The recent results indicate that problems with 28 teams can be solved optimally within reasonable times (e.g., about 8 minutes), although some instances may take significantly longer (e.g., about 30 minutes). However, these solutions are rather involved conceptually and employ state-of-the-art constraint or mathematical programming tools.

This paper proposes a simulated annealing algorithm (BMSA) for break minimization. Its neighborhood is very simple and consists of swapping the home/away teams of a particular game, thus maintaining feasibility at all times. Its meta-heuristic fully automates the cooling schedule and the termination criteria by collecting statistics online during the search process. BMSA was applied to the instances proposed in (Elf, Jünger, & Rinaldi 2003). It finds optimal solutions to all instances regardless of its (random) starting schedules. On the larger instances involving more than 20 teams, it produces significant performance improvements. In particular, it solves instances with 28 teams in less than 20 seconds in average and never takes more than 2 minutes. Moreover, the experimental results indicate that BMSA finds optimal or near-optimal solutions to these large instances within 10 seconds on a 1.66GHz PC. This is an important functionality, since sport scheduling often requires a close interaction between the modeler and the optimization software.

The results are interesting for a number of reasons. First, they show that there exists a local search algorithm for break minimization that is both conceptually simple and easy to implement. Second, they indicate that the performance and the quality of the algorithm are excellent, even when CPU time is severely limited. Third, and most intriguingly, the paper identifies another sport-scheduling application, after the travelling tournament problem (Anagnostopoulos *et al.* 2003), where simulated annealing is a very effective solution technique.

The rest of the paper first reviews the problem and related work. It then presents the neighborhod, the heuristic, and

the meta-heuristic. The experimental results are presented next before the conclusion.

## The Problem

The sport-scheduling application considered in this paper consists of minimizing breaks in single round-robin tournaments. Recall that, in a round-robin tournament with $n$ teams ($n$ even), each team plays against every other team over the course of $n-1$ consecutive rounds. Every round consists of $n/2$ games, each team plays exactly once in a round and each game is played in one of the two opponents' home. In other words, a feasible schedule satisfies the following constraints:

- Every team plays exactly once against every other team.
- Every team plays exactly once in every round.
- If team $i$ plays at home against team $j$ in round $k$, then team $j$ plays against $i$ away, i.e., at $i$'s home.

In this paper, feasible schedules are represented by a $n \times (n-1)$ matrix $S$, such that $S[i,k] = j$ if team $i$ is playing against team $j$ at home in round $k$, and $S[i,k] = -j$ if team $i$ is playing against team $j$ away in round $k$. The following matrix

| T\R | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 6 | -2 | 4 | 3 | -5 |
| 2 | 5 | 1 | -3 | -6 | 4 |
| 3 | -4 | 5 | 2 | -1 | 6 |
| 4 | 3 | 6 | -1 | -5 | -2 |
| 5 | -2 | -3 | 6 | 4 | 1 |
| 6 | -1 | -4 | -5 | 2 | -3 |

is a feasible schedule with 6 teams. The schedule of team $i$ is depicted in row $i$. For instance, team 3 plays team 4 away, then teams 5 and 2 at home, against team 1 away, and finishes at home against team 6. The columns represent the various rounds.

One quality measure for schedules is the number of *breaks*. A break occurs when a team plays two consecutive games at home or two consecutive games away. For instance, in the above example, the games in rounds 2 and 3 of team 3 are both at home, contributing one break. The total number of breaks in the example is 12. More formally, the number of breaks in a schedule is equal to

$$\#\{(i,j) : S[i,j] \cdot S[i,j+1] > 0\,, 1 \leq i \leq n\,, 1 \leq j \leq n-2\}$$

and the goal is to minimize the number of breaks.

More precisely, this paper considers the minimization of breaks for a fixed schedule where only the home/away decisions have been omitted. For instance, the problem may receive, as input, the matrix

| T\R | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 6 | 2 | 4 | 3 | 5 |
| 2 | 5 | 1 | 3 | 6 | 4 |
| 3 | 4 | 5 | 2 | 1 | 6 |
| 4 | 3 | 6 | 1 | 5 | 2 |
| 5 | 2 | 3 | 6 | 4 | 1 |
| 6 | 1 | 4 | 5 | 2 | 3 |

and must produce an assignment of home/away teams minimizing the number of breaks. More formally, the input is a matrix $S$ and the break minimization problem amounts to finding a feasible schedule $S_o$ satisfying

$$\forall i \in \mathit{Teams}, k \in \mathit{Rounds} : |S_o[i,j]| = S[i,j]$$

and minimizing the number of breaks, where $\mathit{Teams} = 1..n$ and $\mathit{Rounds} = 1..n-1$ .

## Related Work

Early results on break minimization are due to Schreuder (Schreuder 1992) who studied the Dutch soccer league. In that paper, Schreuder proved that, for every $n$, there is a schedule with exactly $n-2$ breaks and that this number is optimal. Schreuder also gave a polynomial time algorithm for constructing such a schedule. This version of the problem is called the *Unconstrained Break Minimization Problem* since the schedule is not fixed. The problem studied in this paper, where the schedule is given but not the home/away decisions, is called the *Constrained Break Minimization Problem*.

Regin (Régin 1998) proposed a constraint programming (CP) approach for break minimization. This approach solves the unconstrained version efficiently, but was limited to 20 teams for the constrained version (at least at the time). Trick (Trick 2000) proposed an integer programming approach which provides solutions to instances with at most 22 teams in times comparable to the CP approach. In very recent work, Elf et al., (Elf, Jünger, & Rinaldi 2003) presented a reduction of the constrained version to a cut maximization problem. With the aid of a specialized MAX-CUT solver, they found optimal solutions in reasonable times (e.g., less than 10 minutes) for problems with up to 28 teams. Elf et. al. also conjecture that the constrained problem is $NP$-hard in general, but in $P$ for schedules which have an actual home/away assignment with $n-2$ breaks. This conjecture was (positively) closed in (Miyashiro & Matsui 2003) by a transformation to 2SAT. Miyashiro and Matsui (Miyashiro & Matsui 2005a) also proved that the problem with $n$ breaks are also in $P$. Moreover, more recently, Miyashiro and Matsui Miyashiro (Miyashiro & Matsui 2005b) modeled break minimization as a MAX RES CUT problem and applied Goemans and Williamson's approximation algorithm based on semi-definite programming (Goemans & Williamson 1995). The resulting solutions are not optimal and the distance with respect to the optimum increases with the size of the instances. In particular, they report solutions with an additional 3.5 breaks on 24 teams.

Observe that these approaches use rather "heavy machinery", involving specialized integer programming solvers and constraint programming tools. Moreover, the results were obtained by experts in the underlying technologies and/or in modeling sport-scheduling applications. In contrast, the simulated annealing approach proposed herein is conceptually simple and easy to implement. Yet it produces optimal solutions and is significantly faster for large instances.

## The Neighborhood

The neighborhood in the simulated annealing algorithm is remarkably simple: It consists of swapping the home and away teams of a game in a given week. Since a schedule consists of $n(n-1)/2$ matches, the neighborhood size is $O(n^2)$. More formally, the *SwapHomes* move is defined as follows.

***SwapHomes***$(S, T_i, T_j)$ The move swaps the home/away roles of teams $T_i$ and $T_j$. In other words, if team $T_i$ plays at home (resp. away) against team $T_j$ in round $r_k$, *SwapHomes*$(S, T_i, T_j)$ is the same schedule as $S$, except that team $T_i$ now plays away (resp. at home) against team $T_j$ at round $r_k$. Consider the schedule $S$:

| T\R | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 6 | -2 | 4 | 3 | -5 |
| 2 | 5 | 1 | -3 | -6 | 4 |
| **3** | -4 | 5 | 2 | -1 | 6 |
| 4 | 3 | 6 | -1 | -5 | -2 |
| **5** | -2 | -3 | 6 | 4 | 1 |
| 6 | -1 | -4 | -5 | 2 | -3 |

The move *SwapHomes*$(S, T_3, T_5)$ produces the schedule

| T\R | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 6 | -2 | 4 | 3 | -5 |
| 2 | 5 | 1 | -3 | -6 | 4 |
| **3** | -4 | -5 | 2 | -1 | 6 |
| 4 | 3 | 6 | -1 | -5 | -2 |
| **5** | -2 | 3 | 6 | 4 | 1 |
| 6 | -1 | -4 | -5 | 2 | -3 |

It is important to note that the neighborhood is connected and that the quality of a move can be evaluated in constant time, since there are at most 2 adjacent teams on each affected line. As a consequence, the simulated annealing algorithm can evaluate a large number of moves over the course of its computations.

Other, more complex, neighborhoods were considered, but they do not come close to the effectiveness of this simple neighborhood when it is combined with simulated annealing. These more complex neighborhoods swaps (partial) rounds and (partial) team schedules and were very effective for the travelling tournament problems (Anagnostopoulos *et al.* 2003).

## The Metropolis Heuristic

The core of the algorithm is a Metropolis heuristic which selects schedules randomly from the neighborhood (Metropolis *et al.* 1953). The algorithm moves to the neighboring schedule if it decreases the number of breaks or with probability $\exp(-\Delta/T)$ where $\Delta = cost(S') - cost(S)$, $S$ is the current schedule, and $S'$ is the selected neighbor. The algorithm is depicted in Figure 1. It receives, as input, a schedule and the parameter $T$ (the temperature). Note the

```
1.  Metropolis(S, T)
2.      bestCost ← cost(S);
3.      best ← S;
4.      c ← 0;
5.      while c ≤ phaseLength do
6.          select S' ∈ neighborhood(S);
7.          Δ ← cost(S') − cost(S);
8.          if Δ ≤ 0 then
9.              accept ← true;
10.         else
11.             accept ← true with probability exp(−Δ/T),
12.                      false otherwise;
13.         if accept then
14.             S ← S';
15.             if cost(S) < bestCost then
16.                 bestCost ← cost(S);
17.                 best ← S;
18.         c++;
```

Figure 1: The Metropolis Algorithm

termination condition (line 5) which specifies a maximum number of iterations and lines 11-12 which define when to accept moves increasing the number of breaks. Note that the meaning of parameter *phaseLength* will become clear in the next section.

## The Simulated Annealing Meta-Heuristic

To control the temperature $T$ in the Metropolis heuristic, the algorithm uses a simulated annealing meta-heuristic. More precisely, the simulated annealing algorithm iterates the Metropolis algorithm for different values of $T$. It starts with high temperatures, where many non-improving moves are accepted, and progressively decreases the temperature to focus the search.

The choice of the initial temperature and its updating rule is fully automated in the algorithm and does not require any user intervention. These operations, as well as various termination conditions, are derived from analogies with statistical physics. The rest of this section describes these elements in detail, starting from the analogy with statistical physics in order to provide the necessary intuition. (For an extensive treatment of the analogy, see (van Laarhoven & Aarts 1987; van Laarhoven 1988).)

**The Analogy with Statistical Physics** The analogy consists in viewing the search process (the Metropolis algorithm) as a physical system. Given a temperature $T$ and a starting state $S_T$, the physical system performs a number of transitions moving from states to states. The sequence of transitions is called a *phase* and the number of transitions is called the *phase length*. It can be shown that, under some assumptions, the physical system will reach an equilibrium. This means that, as the phase length goes to infinity, the probability distribution of the states (i.e., the probability that the system be in a given state) converges to a stationary

distribution.

As a consequence, for a given temperature $T$, it makes sense to talk about the statistical measures of the equilibrium, including its expected cost $\mu_T(C)$ and its variance $\sigma_T^2(C)$. Obviously, since the search algorithm is finite, it does not reach an equilibrium in general, which is why one generally talks about quasi-equilibria (which approximate the "real" equilibria).

**Estimating the Statistical Measures** To automate the choice of the temperatures and the termination conditions, the algorithm estimates the values $\mu_T(C)$ and $\sigma_T^2(C)$ for various pairs $(T, S_T)$. The estimation process is simple: It simply consists of performing a relatively large number of transitions. If $N$ denotes the number of transitions and $C_k$ denotes the cost after $k$ transitions, the estimations can be specified as

$$\mu_T(C) \simeq \overline{C}_T = \frac{1}{N}\sum_{k=1}^{N} C_k \tag{1}$$

and

$$\sigma_T^2(C) \simeq \overline{\sigma}_T^2 = \frac{1}{N}\sum_{k=1}^{N}(C_k - \overline{C}_T)^2 \tag{2}$$

The value $N$ is simply chosen as the phase length of the Metropolis algorithm.

**The Initial Temparature** Now that the values $\mu_T(C)$ and $\sigma_T^2(C)$ can be estimated, the temperature $T$ is simply initialized to $\alpha \times \overline{\sigma}_\infty$, i.e., a constant times the estimation of the standard deviation when all moves are accepted. This initialization is in fact a variation of the scheme proposed in (Huang, Romeo, & Sangiovanni-Vincentelli 1986).

**Updating the Temperature** When the quasi-equilibrium is reached for a temperature $T$, simulated annealing decreases $T$ before applying the Metropolis algorithm again. The stationary distribution of the schedules generated during a phase generally depends on the temperature $T$ and, whenever the temperature is decreased, the algorithm will move away from the previous equilibrium and reach a new one. Intuitively, one would expect that significant decreases in the temperature (and hence few phases) require longer phases to reach their quasi-equilibria. As consequence, there is a tradeoff between the temperature decreases and the length of the phases.

Experimental results on travelling tournament problems (TTPs) (Anagnostopoulos *et al.* 2003) have shown that slow cooling schedules, together with short phases, produce high-quality solutions in reasonable time. The algorithm presented here adopts, and automates, the same design decision. Informally speaking, the idea is to strive for a smooth evolution of the equilibria. More precisely, the probabilities $q_k(S)$ of having a schedule with cost $S$ in the stationary cost distribution in phase $k$ should be close to the same probabilities in phase $k+1$ or, in symbols,

$$\forall S:\ \frac{1}{1+\beta} < \frac{q_k(S)}{q_{k+1}(S)} < 1+\beta, k = 1, 2, \ldots, \tag{3}$$

for some constant $\beta$.

Aarts and van Laarhoven (Aarts & van Laarhoven 1985), show that, under some reasonable assumptions, equation (3) yields the following temperature update rule:

$$T_{k+1} = T_k \cdot \left(1 + \frac{\ln(1+\beta)\cdot T_k}{3\overline{\sigma}_{T_k}(C)}\right)^{-1} \tag{4}$$

where $T_k$ is the temperature of phase $k$.

**Early Phase Termination** A phase typically terminates when it has performed its maximum number of iterations. However, now that estimations of the statistics of the quasi-equilibria are available, it is possible to speed up the algorithm by detecting that the search process is close to a quasi-equilibrium (Huang, Romeo, & Sangiovanni-Vincentelli 1986). Assuming a normal distribution for the costs,[1] the probability of the cost being within the interval

$$[\mu_T(C) - \delta\cdot\sigma_T(C), \mu_T(C) + \delta\cdot\sigma_T(C)]$$

for a chosen $\delta$ is equal to

$$\texttt{erf}(\delta/\sqrt{2})$$

where

$$\texttt{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$$

is the error function of the normal distribution. Thus, when the number of the schedules with costs in the above interval divided by the number of transitions performed in the current phase gets close enough to $\texttt{erf}(\delta/\sqrt{2})$ (e.g., within 1%), the phase is terminated prematurely.

**Early Termination of the Cooling Process and Reheats** The cooling process itself can also be terminated early. This happens when the temperature or the variance estimation are too low, in which case the algorithm is typically stuck in a local optimum. When this happens and additional computing time is available, the simulated annealing algorithm performs a reheat, i.e., it resets the temperature to three times the temperature of the current best schedule.[2] The algorithm terminates when the computing time available is exhausted or when it has performed a fixed number of reheats without improvement to the best solution.

**The Simulated Annealing Algorithm** Figure 3 depicts the simulated annealing scheme and Figure 2 revisits the Metropolis to remove some of the initializations which are no longer necessary and to incorporate the early termination criterion (line 3). The algorithm can be viewed as an hybridization of the schemes proposed in (Huang, Romeo, & Sangiovanni-Vincentelli 1986; Aarts & van Laarhoven 1985), together with some extensions to improve performance. Line 1 initializes the schedule randomly, i.e., it

[1] Normal distribution assumptions are supported by numerical experiments for large random combinatorial optimization problems; see also Section (4.3) in (van Laarhoven & Aarts 1987).

[2] This value was chosen based on verylimited experimentations.

```
1.  Metropolis(S, T, μ_T, σ²_T, bestCost, best)
2.    c ← 0;
3.    while c ≤ phaseLength ∧ ¬equilibrium(μ_T, σ²_T) do
4.      select S' ∈ neighborhood(S);
5.      Δ ← cost(S') − cost(S);
6.      if Δ ≤ 0 then
7.        accept ← true;
8.      else
9.        accept ← true with probability exp(−Δ/T),
10.                 false otherwise;
        end if
11.     if accept then
12.       S ← S';
13.       if cost(S) < bestCost then
14.         bestCost ← cost(S);
15.         best ← S;
          end if
        end if
16.     c++;
      end while
17.   return S;
```

Figure 2: The Metropolis Algorithm Revisited

```
1.  S ← a random initial schedule;
2.  bestCost ← cost(S);
3.  best ← S;
4.  T ← α · σ̄_∞
5.  bestTemp ← T;
6.  reheat ← 0;
7.  while reheat ≤ maxReheats do
8.      k ← 0;
9.      do
10.         compute μ̄_T(C) and σ̄²_T(C);
11.         oldBestCost ← bestCost;
12.         S ← metropolis(S, T, μ̄_T(C), σ̄²_T(C));
13.         if bestCost < oldBestCost then
14.             reheat ← 0;
15.             k ← 0;
16.             bestTemperature ← T;
            end if
17.         T ← T · (1 + ln(1+β)·T / (3σ̄_T(C)))^−1;
18.         k++;
19.     while k ≤ maxPhases ∧ T > ε ∧ σ̄_T(C) > 0 do
20.     reheat++;
21.     T ← γ · bestTemperature
    end while
```

Figure 3: The Simulated Annealing Algorithm

assigns the home/away decisions randomly. Lines 2-5 performs a number of initializations, including the temperature in line 4. Line 7-21 iterates the core of the algorithm. Each iteration consists of a number of phases (lines 8-19) and a reheat (lines 20-21). Line 20 simply increases the reheat counters, while line 21 resets the temperature to a constant times the temperature of the best solution found so far. The phases are executed for *maxPhases* iterations or until the temperature or the standard deviation are too small (line 19). A phase estimates the mean and variance for the specific temperature (as indicated earlier in paragraph "Estimating the Statistical Measures"), applies the Metropolis algorithm, and decreases the temperature. If the cost has improved during the Metropolis algorithm, the reheat and the phase counters are reset.

## Experimental Results

This section presents the experimental results of the simulated annealing algorithm (BMSA), compares them to the state-of-the-art algorithm presented in (Elf, Jünger, & Rinaldi 2003; Miyashiro & Matsui 2005b) (MCMP), and indicates that BMSA clearly dominates earlier approaches.

### The Instances

The instances used in the experimental results were provided by the authors of (Elf, Jünger, & Rinaldi 2003). For some reasons, these instances are not exactly the same as those they used in (Elf, Jünger, & Rinaldi 2003), although they are generated in the same fashion. However, the authors also gave us their optimal solutions, as well as their computation times on a double processor PC running Linux with two Intel Pentium 4 CPU at 2.80GHz with 512 KB cache and 1GB RAM. Their code uses CPLEX 7.1 and runs only on one CPU. Note also that these authors ran their code without any parameter tuning, leaving some room for possible improvements (Elf, Jünger, & Rinaldi 2004).

The instances are generated as follows (Elf, Jünger, & Rinaldi 2003). For every value of $n$, an optimal schedule with $n-2$ breaks is computed using Schreuder's algorithm. Then, 10 different instances are created by random permutations of the columns. The resulting schedules typically feature many more breaks in their optimal solutions.

### Experimental Setting for BMSA

The experimental results for BMSA were obtained as follows. For each instance, BMSA was applied on 20 different random initial schedules with 100 phases of length

$$20 * |neighborhood| = 20 * n * (n-1)/2,$$

*maxReheats = 200*, $\alpha = 0.5$, $\beta = 0.1$, $\gamma = 3$, $\delta = 1.2$. No special effort has been spent tuning these parameters, which are rather natural. To increase performance, the algorithm also terminates early when BMSA has not improved the best solution for $2t$ seconds, where $t$ is the CPU time BMSA took to find the current best solution. For instance, if BMSA found the current best solution after $t = 15$ CPU seconds, it will terminate in the next $2t = 30$ seconds if no better solution is found. BMSA was run on an AMD Athlon MP 2000+ at 1.66GHz.

| | MCMP | | | BMSA | | |
|---|---|---|---|---|---|---|
| $n$ | Minimum | Average | Maximum | Minimum | Average | Maximum |
| 4 | 2 | 2.0 | 2 | 2 | 2.0 | 2 |
| 6 | 4 | 4.0 | 4 | 4 | 4.0 | 4 |
| 8 | 6 | 7.2 | 8 | 6 | 7.2 | 8 |
| 10 | 10 | 12.2 | 14 | 10 | 12.2 | 14 |
| 12 | 14 | 17.6 | 20 | 14 | 17.6 | 20 |
| 14 | 22 | 24.8 | 28 | 22 | 24.8 | 28 |
| 16 | 28 | 31.0 | 34 | 28 | 31.0 | 34 |
| 18 | 38 | 41.4 | 46 | 38 | 41.4 | 46 |
| 20 | 48 | 52.0 | 54 | 48 | 52.0 | 54 |
| 22 | 56 | 61.0 | 66 | 56 | 61.0 | 66 |
| 24 | 66 | 73.6 | 78 | 66 | 73.6 | 78 |
| 26 | 82 | 85.0 | 90 | 82 | 85.0 | 90 |
| 28* | 94 | 99.2 | 104 | 94 | 99.2 | 104 |
| 28 | - | - | - | 92 | 98.00 | 106 |
| 30 | - | - | - | 110 | 116.23 | 124 |

Table 1: Quality of the Schedules: Number of Breaks in MCMP and BMSA

| | MCMP | | | BMSA | | |
|---|---|---|---|---|---|---|
| $n$ | Minimum | Average | Maximum | Minimum | Average | Maximum |
| 4 | 0.0 | 0.00 | 0.0 | 0.0 | 0.00 | 0.0 |
| 6 | 0.0 | 0.00 | 0.0 | 0.0 | 0.00 | 0.0 |
| 8 | 0.0 | 0.00 | 0.0 | 0.0 | 0.00 | 0.1 |
| 10 | 0.0 | 0.01 | 0.0 | 0.0 | 0.10 | 0.2 |
| 12 | 0.0 | 0.01 | 0.0 | 0.1 | 0.29 | 0.5 |
| 14 | 0.0 | 0.04 | 0.1 | 0.3 | 0.57 | 1.3 |
| 16 | 0.0 | 0.08 | 0.1 | 0.7 | 1.02 | 3.1 |
| 18 | 0.1 | 0.43 | 1.6 | 1.2 | 2.36 | 16.0 |
| 20 | 0.3 | 0.68 | 2.1 | 1.8 | 3.06 | 24.8 |
| 22 | 0.4 | 3.05 | 18.0 | 2.6 | 4.42 | 25.3 |
| 24 | 0.8 | 24.00 | 179.9 | 3.6 | 9.52 | 71.4 |
| 26 | 2.2 | 53.04 | 435.6 | 4.9 | 13.30 | 92.2 |
| 28* | 7.0 | 465.60 | 1905.0 | 6.8 | 18.76 | 104.9 |
| 28 | - | - | - | 5.6 | 22.07 | 296.9 |
| 30 | - | - | - | 8.2 | 78.58 | 684.1 |

Table 2: Performance of the Algorithms: Computation Times in CPU seconds of MCMP and BMSA

### Quality of the Schedules

Table 1 depicts the quality results. It compares MCMP, a complete search method, and BMSA. A line in the table correponds to a number of teams $n$ and the results report the minimum, maximum, and average number of breaks for all instances with $n$ teams. Interestingly, for every number of teams $n$ and every instance with $n$ teams, BMSA finds the optimal number of breaks and this regardless of the starting point. In other words, for every number of teams, BMSA finds the optimal solution for all instances and all the starting points. For this reason, we isolate these last rows in the table for clarity. Note that there are two lines for the results for 28 teams. The line 28* reports only results on the five instances that MCMP can solve optimally. Note also that, for 30 teams, the BMSA solutions are 6 breaks below those reported in (Miyashiro & Matsui 2005b).

### Performance of the Algorithm

Table 2 depicts the computation times in seconds for both MCMP and BMSA. Recall that the results for MCMP are on a 2.8GHz machine, while the results for BMSA are for a 1.66GHz machine (about 40% slower). When $n \leq 20$, MCMP is extremely fast and BMSA does not bring any advantage besides its simplicity. For larger number of teams, BMSA brings significant benefits and it scales much better than MCMP. For $n = 28$, BMSA is about 25 times faster than MCMP without even accounting for the faster speed of their machine. For up to 28 teams, BMSA is also faster than the approximation algorithm and produces superior solutions. The effeciency of the algorithm seem comparable on 30 teams, although BMSA once again significantly reduces the number of breaks. As a consequence, the performance, and the scaling, of BMSA is quite remarkable, especially in light of the simplicity of the implementation, its generality, and the quality of the results.

### Restarting

It is also interesting to study the impact of restarting the algorithm from scratch periodically. Table 3 report the quality and performance results for BMSA-R where *maxReheats = 20* but the recomputation is restarted from scratch after that for a number of times. Note that the results are given for all instances with 28 and 30 teams. In the average, BMSA-R dominates slightly BSMA, although some instances are solved faster with BSMA. The quality of the solutions of BMSA and BMSA-R is the same on these instances.

### Quality under Strict Time Constraints

It is often the case in sport scheduling that users like very fast interactions with the scheduler to find and evaluate different schedules quickly (e.g., (Nemhauser & Trick 1998)). It is thus interesting to see how BMSA behaves when the CPU time is limited. Tables 4, 5, and 6 depict the results for large number teams ($n = 24, 26, 28$) under different time constraints, where the CPU time is limited to atmost $1, 2, 5, 10, \ldots$ seconds. Interestingly, for 24, 26, and 28 teams, BMSA finds near-optimal results in about 10 seconds. This clearly shows that BMSA outperforms the approximation algorithm in (Miyashiro & Matsui 2005b), both in quality and efficiency.

| | BMSA-R | | |
|---|---|---|---|
| $n$ | Minimum | Average | Maximum |
| 4 | 0.0 | 0.00 | 0.0 |
| 6 | 0.0 | 0.00 | 0.0 |
| 8 | 0.0 | 0.00 | 0.0 |
| 10 | 0.0 | 0.07 | 0.1 |
| 12 | 0.1 | 0.17 | 0.3 |
| 14 | 0.2 | 0.33 | 0.8 |
| 16 | 0.3 | 0.57 | 1.1 |
| 18 | 0.7 | 1.36 | 17.2 |
| 20 | 1.0 | 1.86 | 9.1 |
| 22 | 1.4 | 2.96 | 18.0 |
| 24 | 2.0 | 6.18 | 42.5 |
| 26 | 2.8 | 11.41 | 95.2 |
| 28 | 3.1 | 16.10 | 116.5 |
| 30 | 4.2 | 61.39 | 719.8 |

Table 3: Performance of BMSA-R: Computation Times in CPU seconds

| $n = 24$ | BMSA under Restricted CPU times | | |
|---|---|---|---|
| sec | Minimum | Average | Maximum |
| 1 | 138 | 160.38 | 170 |
| 2 | 102 | 125.31 | 140 |
| 5 | 66 | 75.02 | 82 |
| 10 | 66 | 74.02 | 78 |
| 20 | 66 | 73.80 | 78 |
| 50 | 66 | 73.63 | 78 |
| 100 | 66 | 73.60 | 78 |

Table 4: Quality of BMSA with Limited CPU Time: n=24

## Conclusion

This paper considered the constrained break minimization problem, which consists of finding an assignment of home/away roles in a round-robin schedule to minimize the number of breaks. It proposed a simulated annealing algorithm based on a simple connected neighborhood and a systematic scheme for cooling the temperature and deciding termination.

The resulting algorithm is conceptually simple and easy to implement. Yet it always finds optimal solutions on the instances used in evaluating the state-of-the-art algorithm of (Elf, Jünger, & Rinaldi 2003), regardless of its starting points. The simulated annealing algorithm exhibits excellent performance and significantly outperforms earlier approaches on instances with more than 20 teams. Moreover, it was shown to produce near-optimal solutions within 10 seconds on a 1.66 GHz PC for instances with to 28 teams.

It is particular intriguing to notice the excellent performance of simulated annealing on another sport-scheduling application. Indeed, simulated annealing was shown to be the method of choice for finding high-quality solutions to

| $n = 26$ | BMSA under Restricted CPU times | | |
|---|---|---|---|
| sec | Minimum | Average | Maximum |
| 1 | 192 | 206.28 | 216 |
| 2 | 142 | 169.35 | 184 |
| 5 | 82 | 95.54 | 110 |
| 10 | 82 | 85.74 | 92 |
| 20 | 82 | 85.32 | 92 |
| 50 | 82 | 85.08 | 90 |
| 100 | 82 | 85.00 | 90 |

Table 5: Quality of BMSA when Limited CPU Time: n=26

| $n = 28$ | BMSA under Restricted CPU times | | |
|---|---|---|---|
| sec | Minimum | Average | Maximum |
| 1 | 230 | 251.85 | 266 |
| 2 | 198 | 217.77 | 230 |
| 5 | 98 | 131.11 | 148 |
| 10 | 92 | 98.84 | 108 |
| 20 | 92 | 98.62 | 106 |
| 50 | 92 | 98.17 | 106 |
| 100 | 92 | 98.05 | 106 |
| 200 | 92 | 98.02 | 106 |

Table 6: Quality of BMSA when Limited CPU Time: n=28

the Traveling Tournament Problem proposed in (Easton, Nemhauser, & Trick 2001). The simulated annealing for the TTP (Anagnostopoulos *et al.* 2003) used a complex neighborhood allowing infeasible schedules, while the algorithm presented here uses a simple neighborhood and maintains feasibility at all times.

An interesting issue is whether tabu search can be engineered to be effective on the constrained break minimization problem. Our preliminary experimental results were not encouraging but they were far from being exhaustive. The greedy nature of tabu search does not seem beneficial in these applications, although it would be very interesting to understand this issue in more detail.

## Acknowledgements

Special thanks to the authors of (Elf, Jünger, & Rinaldi 2003) for providing us with their instances, their solutions, and their experimental results, to the reviewers for many useful comments, and to Irv Lustig for many interesting conversations on sport scheduling. This work was partially supported by NSF ITR Awards ACI-0121497.

## References

Aarts, E., and van Laarhoven, P. 1985. Statistical Cooling: A General Approach to Combinatorial Optimization Problems. *Philips Journal of Research* 40:193–226.

Anagnostopoulos; A. Michel, L.; Van Hentenryck, P.; and Vergados, Y. 2003. A Simulated Annealing Approach to the Traveling Tournament Problem. In *CP-AI-OR'03*.

Easton, K.; Nemhauser, G.; and Trick, M. 2001. The Traveling Tournament Problem Description and Benchmarks. In *Seventh International Conference on the Principles and Practice of Constraint Programming (CP'01)*, 580–589. Paphos, Cyprus: Springer-Verlag, LNCS 2239.

Elf, M.; Jünger, M.; and Rinaldi, G. 2003. Minimizing breaks by maximizing cuts. *Operations Research Letters* 31:343–349.

Elf, M.; Jünger, M.; and Rinaldi, G. 2004. Personal communication.

Goemans, M., and Williamson, D. 1995. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems using Semidefinite Programming. *Journal of the ACM* 42(6):1115–1145.

Huang, M.; Romeo, F.; and Sangiovanni-Vincentelli, A. 1986. An Efficient General Cooling Schedule for Simulated Annealing. In *Proc. IEEE International Conference on Computer-Aided Design*, 381–384.

Metropolis, N.; Rosenbluth, A.; Rosenbluth, M.; Teller, A.; and Teller, E. 1953. Equation of state calculation by fast computer machines. *Journal of Chemical Physics* 21(6):1087–1092.

Miyashiro, R., and Matsui, T. 2003. Round-robin tournaments with a small number of breaks. Technical Report Mathematical Engineering Technical Reports, METR 2003-29, Department of Mathematical Informatics, Graduate School of Information Science and Technology, the University of Tokyo.

Miyashiro, R., and Matsui, T. 2005a. A polynomial-time algorithm to find an equitable home-away assignment. *Operations Research Letters* 33(2):235–241.

Miyashiro, R., and Matsui, T. 2005b. Semidefinite Programming Based Approaches to the Break Minimization Problem. *Computers & Operations Research (to appear)*.

Nemhauser, G., and Trick, M. 1998. Scheduling a Major College Basketball Conference. *Operations Research* 46(1):1–8.

Régin, J. C. 1998. Minimization of the number of breaks in sports scheduling problems using constraint programming. In Freuder, E. C., and Wallace, R. J., eds., *Constraint Programming and Large Scale Discrete Optimization*, volume 57 of *DIMACS*, 115–130. American Mathematical Society Publications.

Schreuder, J. A. M. 1992. Combinatorial aspects of construction of competition dutch professional football leagues. *Discrete Applied Mathematics* 35:301–312.

Trick, M. A. 2000. A schedule-then-break approach to sports timetabling. In *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, 242–253. Springer-Verlag.

van Laarhoven, P., and Aarts, E. 1987. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, Dordrecht, Holland.

van Laarhoven, P. 1988. *Theoretical and Computational Aspects of Simulated Annealing*. Stichting Mathematisch Centrum, Amsterdam.