A Simulated Annealing Approach to the Traveling Tournament Problem*

A. Anagnostopoulos, L. Michel†, P. Van Hentenryck, and Y. Vergados
Brown University, Box 1910, Providence, RI 02912

Abstract

Automating the scheduling of sport leagues has received considerable attention in recent years, as these applications involve significant revenues and generate challenging combinatorial optimization problems. This paper considers the traveling tournament problem (TTP) which abstracts the salient features of major league baseball (MLB) in the United States. It proposes a simulated annealing algorithm (TTSA) for the TTP that explores both feasible and infeasible schedules, uses a large neighborhood with complex moves, and includes advanced techniques such as strategic oscillation and reheats to balance the exploration of the feasible and infeasible regions and to escape local minima at very low temperatures. TTSA matches the best known solutions on the small instances of the TTP and produces significant improvements over previous approaches on the larger instances. Moreover, TTSA is shown to be robust, since its worst solution quality over 50 runs is always smaller or equal to the best known solutions.

Keywords: Sport Scheduling, Travelling Tournament Problems, Local Search, Simulated Annealing.

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 [4] for an excellent review of these problems, recent research activities, and several solution techniques.

This paper considers the traveling tournament problem (TTP) proposed in [4] to abstract the salient features of Major League Baseball (MLB) in the United States. The key to the MLB schedule is a conflict between minimizing travel distances and feasibility constraints on the home/away patterns. Travel distances are a major issue in MLB because of the number of teams and the fact that teams go on "road trips" to visit several opponents before returning home. The feasibility constraints in the MLB restricts the number of successive games that can be played at home or away.

*A Preliminary version of this paper was presented at the CP'AI'OR'03 Workshop.

†Current Address: University Of Connecticut, Stoors, CT 06269

The TTP is an abstraction of the MLB intended to stimulate research in sport scheduling. A solution to the TTP is a double round-robin tournament which satisfies sophisticated feasibility constraints (e.g., no more than three away games in a row) and minimizes the total travel distances of the teams. While both minimizing the total distance traveled, and satisfying the feasibility constraints, are separately easy problems, and current software can solve big instances, it seems that the combination of the two (which is captured by the TTP) makes the problem very difficult; even instances with as few as 8 teams are hard to solve, requiring techniques used by both constraint programming and mathematical programming communities. [4] argues that, without an approach to the TTP, it is unlikely that suitable schedules can be obtained for the MLB. The TTP has raised significant interest in recent years since the challenge instances were proposed. [4] describes both constraint and integer programming approaches to the TTP which generate high-quality solutions. [1] explores a Lagrangian relaxation approach (together with constraint programming techniques) which improves some of the results. Other lower and upper bounds are given in [10], although the details of how they are obtained do not seem to be available.

This paper proposes a simulated annealing algorithm (TTSA) for the traveling tournament. TTSA contains a number of interesting features that are critical to obtain high-quality solutions:

1. TTSA separates the tournament constraints and the pattern constraints into hard and soft constraints and explores both feasible and infeasible schedules.

2. TTSA uses a large neighborhood of size $O(n^3)$, where n is the number of teams. Some of the moves defining the neighborhood are rather complex and affect the schedule in significant ways. Others can be regarded as a form of ejection chains [7, 9].

3. TTSA includes a strategic oscillation strategy to balance the time spent in the feasible and infeasible regions.

4. TTSA incorporates the concept of "reheats" to escape from local minima with very low temperatures.

TTSA was applied to the challenge instances of the TTP. It matches the best found solutions on the smaller instances (up to 8 teams) and produces significant improvements in travel distances on the larger instances. TTSA is also shown to be robust, since its average solution quality is, in general, smaller than the best known solutions. Note also that previous results have been obtained with relatively heavy machinery and state-of-the-art techniques (e.g., [4] mixes constraint and integer programming, [1] combines a CP solver with Language relaxation). TTSA demonstrates the ability of simulated annealing to attack these hard combinatorial optimization problems successfully with a much simpler machinery.

The rest of the paper is organized as follows. Section 1 describes the problem. Section 2 describes the new algorithm, including its neighborhood and the various advanced techniques that it uses. Section 3 presents the experimental results. Section 4 concludes the paper.

1 Problem Description

The problem was introduced by Easton, Nemhauser and Trick [10, 4], which contains many interesting discussions on sport scheduling. An input consists of n teams (n even) and an $n \times n$ symmetric matrix d, such that d_{ij} represents the distance between the homes of teams T_i and T_j. A solution is a schedule in which each team plays with each other twice, once in each team's

home. Such a schedule is called a *double round-robin tournament*. It should be clear that a double round-robin tournament has $2n-2$ rounds. It turns out that tournaments with $2n-2$ rounds can be constructed for each n and we only consider tournaments with this minimal number of rounds. In such tournaments, the number of games per round is always n.

For a given schedule S, the cost of a team as the total distance that it has to travel starting from its home, playing the scheduled games in S, and returning back home. The cost of a solution is defined as the sum of the cost of every team.

The goal is to find a schedule with minimum cost satisfying the following two constraints:

1. **Atmost Constraints:** No more than three consecutive home or away games are allowed for any team.

2. **Norepeat Constraints:** A game of T_i at T_j's home cannot be followed by a game of T_j at T_i's home.

As a consequence, a double round-robin schedule is feasible if it satisfies the atmost and norepeat constraints and is infeasible otherwise.

In this paper, a schedule is represented by a table indicating the opponents of the teams. Each line corresponds to a team and each column corresponds to a round. The opponent of team T_i at round r_k is given by the absolute value of element (i,k). If (i,k) is positive, the game takes place at T_i's home, otherwise at T_i's opponent home. Consider for instance the schedule S for 6 teams (and thus 10 rounds).

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

Schedule S specifies that team T_1 has the following schedule. It successively plays against teams T_6 at home, T_2 away, T_4 at home, T_3 at home, T_5 away, T_4 away, T_3 away, T_5 at home, T_2 at home, T_6 away. The travel cost of team T_1 is

$$d_{12} + d_{21} + d_{15} + d_{54} + d_{43} + d_{31} + d_{16} + d_{61}.$$

Observe that long stretches of games at home do not contribute to the travel cost but are limited by the atmost constraints. This kind of tension is precisely why this problem is hard to solve in practice.

2 The Local Search

This paper proposes an advanced simulated annealing algorithm (TTSA) for the TTP. As usual, the algorithm starts from an initial configuration. Its basic step moves from the current configuration c to a configuration in the neighborhood of c. TTSA is based on four main design decisions:

1. Constraints are separated into two groups: hard constraints, which are always satisfied by the configurations, and soft constraints, which may or may not be satisfied. The hard constraints are the round-robin constraints, while the soft constraints are the norepeat and atmost constraints. In other words, all configurations in the search represents a double round-robin tournament, which may or may not violate the norepeat and atmost constraints. Exploring the infeasible region seems to be particularly important for this problem. Obviously, to drive the search toward feasible solutions, TTSA modifies the original objective function to include a penalty term.

2. TTSA is based on a large neighborhood of size $O(n^3)$, where n is the number of teams. In addition, these moves may affect significant portions of the configurations. For instance, they may swap the schedule of two teams, which affects $4(n-2)$ entries in a configuration. In addition, some of these moves can be regarded as a form of ejection chains which is often used in tabu search [7, 9].

3. TTSA dynamically adjusts the objective function to balance the time spent in the feasible and infeasible regions. This adjustment resembles the strategic oscillation idea [5] successfully in tabu search to solve generalized assignment problems [3], although the details differ since simulated annealing is used as the meta-heuristics.

4. TTSA also uses reheats (e.g., [2]) to escape local minima at low temperatures. The "reheats" increase the temperature again and divide the search in several phases.

The rest of this section explore some of these aspects in more detail. Since they are double round-robin tournaments, configurations are called schedules in the following.

2.1 The Neighborhood

The neighborhood of a schedule S is the set of the (possibly infeasible) schedules which can be obtained by applying one of five types of moves. The first three types of moves have a simple intuitive meaning, while the last two generalize them.

***SwapHomes*(S, T_i, T_j)** This move swaps the home/away roles of teams T_i and T_j. In other words, if team T_i plays home against team T_j at round r_k, and away against T_j's home at round l, *SwapHomes*(S, T_i, T_j) is the same schedule as S, except that now team T_i plays away against team T_j at round r_k, and home against T_j at round r_l. There are $O(n^2)$ such moves. Consider the schedule S:

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

The move *SwapHomes*(S, T_2, T_4) produces the schedule:

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	-4	3	6	4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	2	1	5	-2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

***SwapRounds*(S, r_k, r_l)** The move simply swaps rounds r_k and r_l. There are also $O(n^2)$ such moves. Consider the schedule S:

T\R	1	2	**3**	4	**5**	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

The move *SwapRounds*(S, r_3, r_5) produces the schedule

T\R	1	2	**3**	4	**5**	6	7	8	9	10
1	6	-2	-5	3	4	-4	-3	5	2	-6
2	5	1	4	-6	-3	3	6	-4	-1	-5
3	-4	5	6	-1	2	-2	1	-6	-5	4
4	3	6	-2	-5	-1	1	5	2	-6	-3
5	-2	-3	1	4	6	-6	-4	-1	3	2
6	-1	-4	-3	2	-5	5	-2	3	4	1

***SwapTeams*(S, T_i, T_j)** This move swaps the schedule of Teams T_i and T_j (except, of course, when they play against each other). There are $O(n^2)$ such moves again. Consider the schedule S:

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

The move *SwapTeams*(S, T_2, T_5) produces the schedule

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-5	4	3	-2	-4	-3	2	5	-6
2	5	-3	6	4	1	-6	-4	-1	3	-5
3	-4	2	5	-1	6	-5	1	-6	-2	4
4	3	6	-1	-2	-5	1	2	5	-6	-3
5	-2	1	-3	-6	4	3	6	-4	-1	2
6	-1	-4	-2	5	-3	2	-5	3	4	1

Note that, in addition to the changes in lines 2 and 5, the corresponding lines of the opponents of T_i and T_j must be changed as well. As a consequence, there are four values per round (column) that are changed (except when T_i and T_j meet).

It turns out that these three moves are not sufficient for exploring the entire search space and, as a consequence, they lead to suboptimal solutions for large instances. To improve these results, it is important to consider two, more general, moves. Although these moves do not have the apparent interpretation of the first three, they are similar in structure and they significantly enlarge the neighborhood, resulting to a more connected search space. More precisely, these moves are partial swaps: they swap a subset of the schedule in rounds r_i and r_j or a subset of the schedule for teams T_i and T_j. The benefits from these moves come from the fact that they are not as global as the "macro"-moves *SwapTeams* and *SwapRounds*. As a consequence, they may achieve a better tradeoff between feasibility and optimality by improving feasibility in one part of the schedule, while not breaking feasibility in another one. They are also more "global" than the "micro"-moves *SwapHomes*.

***PartialSwapRounds*(S, T_i, r_k, r_l):** This move considers team T_i and swaps its games at rounds r_k and r_l. Then the rest of the schedule for rounds r_k and r_l is updated (in a deterministic way) to produce a double round-robin tournament. Consider the schedule S

T\R	1	**2**	3	4	5	6	7	8	**9**	10
1	6	-2	2	3	-5	-4	-3	5	4	-6
2	5	1	-1	-5	4	3	6	-4	-6	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	6	-3	-6	-2	1	5	2	-1	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	-4	-5	4	-3	5	-2	3	2	1

and the move *PartialSwapRounds*(S, T_2, r_2, r_9). Obviously swapping the game in rounds r_2 and r_9 would not lead to a round-robin tournament. It is also necessary to swap the games of team 1, 4, and 6 in order to obtain:

T\R	1	**2**	3	4	5	6	7	8	**9**	10
1	6	4	2	3	-5	-4	-3	5	-2	-6
2	5	-6	-1	-5	4	3	6	-4	1	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	-1	-3	-6	-2	1	5	2	6	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	2	-5	4	-3	5	-2	3	-4	1

This move, and the next one, can thus be regarded as a form of ejection chain [7, 9].

Finding which games to swap is not difficult: it suffices to find the connected component which contains the games of T_i in rounds r_k and r_l in the graph where the vertices are the teams and where an edge contains two teams if they play against each other in rounds r_k and r_l. All the teams in this component must have their games swapped. Note that there are $O(n^3)$ such moves.

***PartialSwapTeams*(S, T_i, T_j, r_k)** This move considers round r_k and swaps the games of teams T_i and T_j. Then, the rest of the schedule for teams T_i and T_j (and their opponents) is updated to produce a double round-robin tournament. Note that, as was the case with *SwapTeams*, four entries per considered round are affected. There are also $O(n^3)$ such moves. Consider the schedule S

T\R	1	2	3	4	5	6	7	8	**9**	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

The move *PartialSwapRounds*(S, T_2, T_4, r_9) produces the schedule

T\R	1	2	3	4	5	6	7	8	**9**	10
1	6	-2	2	3	-5	-4	-3	5	4	-6
2	5	1	-1	-5	4	3	6	-4	-6	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	6	-3	-6	-2	1	5	2	-1	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	-4	-5	4	-3	5	-2	3	2	1

2.2 Simulated Annealing

As mentioned, TTSA uses a simulating annealing meta-heuristics to explore the neighborhood graph [6]. TTSA starts from a random initial schedule which is obtained using a simple backtrack search. No special attention has been devoted to this algorithm and feasible schedules were easily obtained. TTSA then follows the traditional simulating algorithm schema. Given a temperature T, the algorithm randomly selects one of the moves in the neighborhood and computes the variation Δ in the objective function produced by the move. If $\Delta < 0$, TTSA applies the move. Otherwise, it applies the move with probability $\exp(-\Delta/T)$.

As typical in simulated annealing, the probability of accepting a non-improving move decreases over time. This behavior is obtained by decreasing the temperature as follows. TTSA uses a variable *counter* which is incremented for each non-improving move and reset to zero when the best solution found so far is improved. When *counter* reaches a particular upper limit, the temperature is updated to $T \cdot \beta$ (where β is a fixed constant smaller than 1) and *counter* is reset to zero.

Figure 1 depicts the simulated annealing algorithm in more detail. The algorithm keeps an implicit representation of the neighborhood as a set of pairs and triplets, since all moves can

```
1.   find random schedule S;
2.   bestSoFar ← cost(S);
3.   counter ← 0;
4.   while phase ≤ maxP do
5.       phase ← 0;
6.       counter ← 0;
7.       while counter ≤ maxC do
8.           select a random move m from neighborhood(S);
9.           let S' be the schedule obtained from S with m;
10.          if cost(S') < cost(S) then
11.              accept ←true;
12.          else
13.              accept ← true with probability exp(−Δ/T),
14.                       false otherwise;
15.          end if
16.          if accept then
17.              S ← S';
18.              if cost(S') < bestSoFar then
19.                  counter ← 0; phase ← 0;
20.                  bestSoFar ← cost(S');
21.              else
22.                  counter++;
23.              end if
24.          end if
25.          end while
26.      phase++;
27.      T ← T · β;
28.  end while
```

Figure 1: The Simulated Annealing Algorithm

be characterized this way. For instance, the ***partialSwapTeam(***S, T_i, T_j, r_k***)*** are characterized by triplets of the form $\langle T_i, T_j, r_k \rangle$. Note that a Metropolis algorithm can be obtained by removing line 27 which updates the temperature.

2.3 The Objective Function

As mentioned already, the configurations in algorithm TTSA are schedules which may or may not satisfy the norepeat and atmost constraints. In addition, the moves are not guaranteed to maintain feasibility even if they start with a feasible schedule. The ability to explore infeasible schedules appears critical for the success of simulated annealing on the TT.

To drive toward feasible solution, the standard objective function function $cost$ is replaced by a more complex objective function which combines travel distances and the number of violations.

find random schedule S;
$bestFeasible \leftarrow \infty$; $nbf \leftarrow \infty$;
$bestInfeasible \leftarrow \infty$; $nbi \leftarrow \infty$;
$reheat \leftarrow 0$; $counter \leftarrow 0$;
while $reheat \leq maxR$ **do**
  $phase \leftarrow 0$;
  while $phase \leq maxP$ **do**
    $counter \leftarrow 0$;
    while $counter \leq maxC$ **do**
      select a random move m from $neighborhood(S)$;
      let S' be the schedule obtained from S with m;
      if $C(S') < C(S)$ **or**
        $nbv(S') == 0$ **and** $C(S') < bestFeasible$ **or**
        $nbv(S') > 0$ **and** $C(S') < bestInfeasible$
      then
        $accept \leftarrow$ **true**;
      else
        $accept \leftarrow$ **true** with probability $\exp(-\Delta C/T)$,
                false otherwise;
      end if
      if $accept$ **then**
        $S \leftarrow S'$;
        if $nbv(S) == 0$ **then**
          $nbf \leftarrow \min(C(S), bestFeasible)$;
        else
          $nbi \leftarrow \min(C(S), bestInfeasible)$;
        end if
        if $nbf < bestFeasible$ **or** $nbi < bestInfeasible$ **then**
          $reheat \leftarrow 0$; $counter \leftarrow 0$; $phase \leftarrow 0$;
          $bestTemperature \leftarrow T$;
          $bestFeasible \leftarrow nbf$;
          $bestInfeasible \leftarrow nbi$;
          if $nbv(S) == 0$ **then** $w \leftarrow w/\theta$; **else** $w \leftarrow w \cdot \delta$; **end if**
        else
          $counter$++;
        end if
      end while
    $phase$++;
    $T \leftarrow T \cdot \beta$;
  end while
  $reheat$++;
  $T \leftarrow 2 \cdot bestTemperature$;
end while

Figure 2: The Simulated Annealing Algorithm TTSA

The new objective function C is defined as follows:

$$C(S) = \begin{cases} cost(S) \text{ if } S \text{ is feasible,} \\ \sqrt{cost(S)^2 + [w \cdot f(nbv(S))]^2} \text{otherwise,} \end{cases}$$

where $nbv(S)$ denotes the number of violations of the norepeat and atmost constraints, w is a weight, and $f : N \to N$ is a sublinear function such that $f(1) = 1$.

It is interesting to give the rationale behind the choice of f. The intuition is that the first violation costs more than subsequent ones, since adding 1 violation to a schedule with 6 existing ones does not make much difference. More precisely, crossing the feasible/infeasible boundary costs w, while v violations only cost $wf(v)$, where $f(v)$ is sublinear in v. In our experiments, we chose $f(v) = 1 + \sqrt{v} \ln v/2$. This choice makes sure that f does not grow too slowly to avoid solutions with very many violations.

Of course, it should be clear that TTSA applies the simulated annealing presented earlier, not on the travel distance function $cost$, but on the function C. TTSA must also keep track of the best feasible solution found so far.

2.4 Strategic Oscillation

TTSA also includes a strategic oscillation strategy which has been often used in tabu search when the local search explores both the feasible and infeasible region (e.g., [5, 3]). The key idea is to vary the weight parameter w during the search. In advanced tabu-search applications (e.g., [3]), the penalty is updated according to the frequencies of feasible and infeasible configurations in the last iterations. Such a strategy is meaningful in that context, but is not particularly appropriate for simulated annealing since very few moves may be selected. TTSA uses a very simple scheme. Each time it generates a new best solution (line 28 of the algorithm), TTSA multiplies w by some constant $\delta > 1$ if the new solution is infeasible or divide w by some constant $\theta > 1$ if the new solution is feasible.

The rationale here is to keep a balance between the time spent exploring the feasible region and the time spent exploring infeasible schedule. After having spent a long time in the infeasible region, the weight w, and thus the penalty for violations, will become large and it will drive the search toward feasible solutions. Similarly, after having spent a long time in the feasible region, the weight w, and thus the penalty for violations, will become small and it will drive the search toward infeasible solutions. In our experiments, we chose $\delta = \theta$ for simplicity.

2.5 Reheats

The last feature of TTSA is the use of reheating, a generalization of the standard simulated annealing cooling schemes has been proposed by several authors (see, for example, [8]). The basic idea is that, once simulated annealing reaches very low temperatures, it has difficulties to escape from local minima, because the probability of accepting non-decreasing moves is very low. Reheating is the idea of increasing the temperature again to escape the current local minimum.

TTSA uses a relatively simple reheating method. The idea is to reheat upon completion of the outermost loop by increasing the temperature to twice its value when the best solution was found. TTSA now terminates when the number of consecutive reheats without improving the best solution reaches a given limit. The algorithm including all these modifications is shown in Figure 2.

```
1.  RANDOMSCHEDULE() {
2.    Q ← {⟨t, w⟩ | t ∈ Teams & w ∈ Weeks};
3.    GENERATESCHEDULE(Q, S);
4.    return S;
5.  }
6.  bool GENERATESCHEDULE(Q, S) {
7.    if Q = ∅ then return true; end if
8.    select ⟨t, w⟩ ∈ Q such that ∀⟨t', w'⟩ ∈ Q : ⟨t', w'⟩ ≥ ⟨t, w⟩;
9.    Choices ← {1, −1, ..., t − 1, −(t − 1), t + 1, −(t + 1), ..., n, −n};
10.   forall o ∈ Choices in random order do
11.     if ⟨o, w⟩ ∉ Q then
12.       S[t, w] ← o;
13.       if o > 0 then
14.         S[o, w] ← −t;
15.       else
16.         S[−o, w] ← t;
17.       end if
18.       if GENERATESCHEDULE(Q \ {⟨t, w⟩, ⟨|o|, w⟩}, S) then
19.         return true;
20.       end if
21.     end if
22.   end forall
23.   return false;
24. }
```

Figure 3: The Generation of Initial Random Schedules

2.6 Initial Solutions

The algorithm to generate of random schedules satisfying the hard constraints is depicted in Figure 3. The algorithm uses a set Q containing all the position (Team,Week) to complete the schedule. The set Q is initialized in line 2, before calling the recursive procedure GENERATESCHEDULE. This procedure returns true (and the schedule S) whenever Q is empty (line 7). Otherwise, it selects the position $\langle t, w\rangle$, which is lexicographically smallest (line 8), and tries all its possible choices randomly (line 9). The choices are simply the other teams, either at home or away. If the selected opponent o is not already assigned in week w, then the schedule S is updated and the algorithm is called recursively with the set Q where $\langle t, w\rangle$ and $\langle |o|, w\rangle$ have been removed (line 18). This procedure is very simple and can be improved considerably, but it appears sufficient to find schedules satisfying the hard constraints reasonably fast.

3 Experiments

This section describes the experimental results on TTSA. It first reports the results for the standard version, which aims at producing the best possible schedules. The impact of the various components is then studied in detail. The next set of experimental results describe how the solution quality evolves over time. The section concludes with a discussion on a fast cooling version

n	Best (Nov. 2002)	min(D)	max(D)	mean(D)	std(D)
8	39721	39721	39721	39721	0
10	61608	**59583**	**59806**	**59605.96**	53.36
12	118955	**112800**	**114946**	**113853.00**	467.91
14	205894	**190368**	**195456**	**192931.86**	1188.08
16	281660	**267194**	**280925**	**275015.88**	2488.02

Table 1: Solution Quality of TTSA on the TTP

n	T_0	β	w_0	δ	θ	$maxC$	$maxP$	$maxR$	γ
8	400	0.9999	4000	1.04	1.04	5000	7100	10	2
10	400	0.9999	6000	1.04	1.04	5000	7100	10	2
12	600	0.9995	10000	1.03	1.03	4000	1385	50	1.6
14	600	0.9999	20000	1.03	1.03	4000	7100	30	1.8
16	700	0.9999	60000	1.05	1.05	10000	7100	50	2

Table 2: Parameter Values for the TTSA Instances

of TTSA, which improves TTSA's ability to find good solutions quickly. The section concludes with the best solutions found over the course of this research.

Quality and Performance of TTSA TTSA was applied to the National League benchmark described in [4, 10] (we did not consider $n = 6$, since TTSA always finds the optimal solution). We experimented with different values for the parameters described previously. The most successful version of the algorithm uses a very slowly cooling system ($\beta \simeq .9999$), a large number of phases (so that the system can reach low temperatures), and long phases. In order to avoid big oscillations in the value of the penalty weight w, the parameters δ and θ were chosen to be close to 1 ($\simeq 1.03$). For each instance set (i.e., for every value of n), in all 50 runs, the parameters had the same initial values.

Table 1 describes the quality of the results for 50 runs of TTSA on each of the instances with parameters as shown on Table 2. The first column gives the number of teams, the second column gives the best-known solutions at the time of writing (January 12, 2003) as shown in [10] (last update was November, 2002). These solutions are obtained using a variety of techniques, including constraint and integer programming and Lagrangian relaxation.[1] The next columns give the best solution found over the 50 runs, the worst solution, the average quality of the solutions. and the standard deviation.

TTSA improved all the best known solutions (at the time of the experiments) on instances with at least 10 teams. TTSA was the first algorithm to go lower than 60,000 on 10 teams, 200,000 for 14 teams and 280,000 for 16 teams. The improvements ranged between 2% to 5%. The table also shows that the worst solution of TTSA is always smaller than or equal to the best known solution, indicating the robustness of TTSA.

Table 3 also gives the CPU time in seconds needed by TTSA on an AMD Athlon(TM) at 1544 MHz. It gives the time to find the best solution, the average time, and the standard deviation over 50 runs.

[1] It is not clear however how some of these, and newer, results were obtained. See [10] for more details.

n	min(T)	mean(T)	std(T)
8	596.6	1639.33	332.38
10	8084.2	40268.62	45890.30
12	28526.0	68505.26	63455.32
14	418358.2	233578.35	179176.59
16	344633.4	192086.55	149711.85

Table 3: Computation Times of TTSA on the TTP

Method	Best	Worst	Mean	Std
TTSA	190,514	196,989	194,560	1,631
TTSA(PS)	191,145	197,383	194,694	1,304
TTSA(NR)	196,561	205,094	200,680	2,152
TTSA(150)	197,781	211,347	203,621	3,157
TTSA(300)	195,627	202,158	198,004	964
TTSA(450)	204,872	215,485	206,862	1,428
TTSA(600)	213,938	218,879	216,412	1,096

Table 4: Impact of TTSA Components on Solution Quality (14 Teams)

Impact of the Components TTSA includes a variety of components and it is interesting to measure how important they are in the performance and quality of the algorithm. Table 4 compares various versions of the algorithm on 14 teams with the parameters shown on table 5. Each version leaves out some component of TTSA: TTSA(PS) considers partial moves only, TTSA(NR) does not include reheats, and the TTSA(T) versions are not based on simulated annealing but on a Metropolis algorithm with temperature T. All versions were executed at least 35 times for 100,000 seconds. The table reports the minimum, maximum, and mean solution values, as well as the standard deviation. Observe that considering one of the partial moves only would degrade solution quality. It was apparent early on in our research that both moves brought benefits, since most of our best solutions were obtained when we added *PartialSwapTeams*.

It is interesting to observe that TTSA outperforms all other versions on these experiments. TTSA(PS) is slighly outperformed by TTSA, although the full moves can be thought as a combination of partial moves. The full moves seem to bring some benefit because of their ability to diversify the schedule more substantially. The use of reheats produce significant benefits. The performance of the algorithm degrades significantly when they are not used, raising the mean from about 194,000 to about 200,000. Similar observations hold for the Metropolis version which are largely dominated in general.

Since the results with and without full moves were rather close, another set of experiments was carried out to understand their effect more precisely. These results are shown in Table 6 which evaluates TTSA and TTSA(PS) when the time limit is varied. Interestingly, the results seem to indicate that full swaps are beneficial early in the search and become marginal when long runs are considered.

Solution Quality over Time TTSA is computationally intensive, at least to find very high-quality solutions as earlier results demonstrate. It is thus important to study how solution quality

Method	T_0	β	w_0	δ	θ	$maxC$	$maxP$	$maxR$	γ
TTSA	1100	0.999	18000	1.03	1.03	3000	710	1000	1.4
TTSA(PS)	1100	0.999	18000	1.03	1.03	3000	710	1000	1.4
TTSA(NR)	1100	0.9999	18000	1.03	1.03	3000000	∞	0	1
TTSA(150)	150	1	18000	1.03	1.03	∞	0	0	1
TTSA(300)	300	1	18000	1.03	1.03	∞	0	0	1
TTSA(450)	450	1	18000	1.03	1.03	∞	0	0	1
TTSA(600)	600	1	18000	1.03	1.03	∞	0	0	1

Table 5: Parameter Values for Experiments on the Impact of the Components (14 Teams)

Time	Method	Best	Worst	Mean	Std
50,000 sec	TTSA	192,040	198,140	195,349	1,311
	TTSA(PS)	193,144	202,435	196,112	1,755
100,000 sec	TTSA	190,514	196,989	194,560	1,631
	TTSA(PS)	191,145	197,383	194,694	1,304
150,000 sec	TTSA	190,514	196,989	194,186	1,550
	TTSA(PS)	191,060	196,665	194,300	1,289

Table 6: Impact of Full Moves on the Solution Quality of TTSA (14 Teams)

evolves over time in TTSA. Figures 4 and 5 depict the solution values over time for many runs with 12 and 14 teams. The figures depict the superposition of the curves for many runs. It is interesting to observe the sharp initial decline in the solution values which is followed by a long tail where improvements are very slow. Moreover, at the transition point between the decline and the tail, TTSA typically has improved the previous best solutions. In particular, TTSA takes about 1,000 seconds to beat the previous best results for for 12 teams, after which improvements proceed at a much slower rate. The same phenomenon arises for 14 teams.

Fast Cooling As mentioned earlier, the parameters of TTSA were chosen to find the best possible solutions without much concern about running times. The experimental results indicate that TTSA generally exhibits a sharp initial decline followed by a long tail. Hence it is intriguing to evaluate the potential of TTSA to find "good" solutions quickly by using a fast cooling.

Table 7 and Figure 6 depict the results of TTSA(FC), a fast cooling version of TTSA, with the parameters shown in Table 8. TTSA(FC) uses a cooling factor (beta) of .98, phases of length (maxC) 5000 and a number of non-improving phases before reheating (maxP) of 70. Table 7 compares TTSA and TTSA(FC) for a running time of 2 hours. The results clearly show the benefits of a fast cooling schedule for obtaining high-quality solutions quickly. Within two hours, the best run of TTSA(FC) outperforms the previous best solution for 16 teams, while its average solution is less than 2% above this solution. Figure 6 depicts the solution quality over time for best runs of TTSA and TTSA(FC) over time. Observe the sharper initial decline of TTSA(FC), which significantly outperforms TTSA for short runs. Of course, over time, TTSA(FC) is slighly dominated by TTSA. These results seem to indicate the versatility of TTSA and its potential to find high-quality solutions reasonably fast.

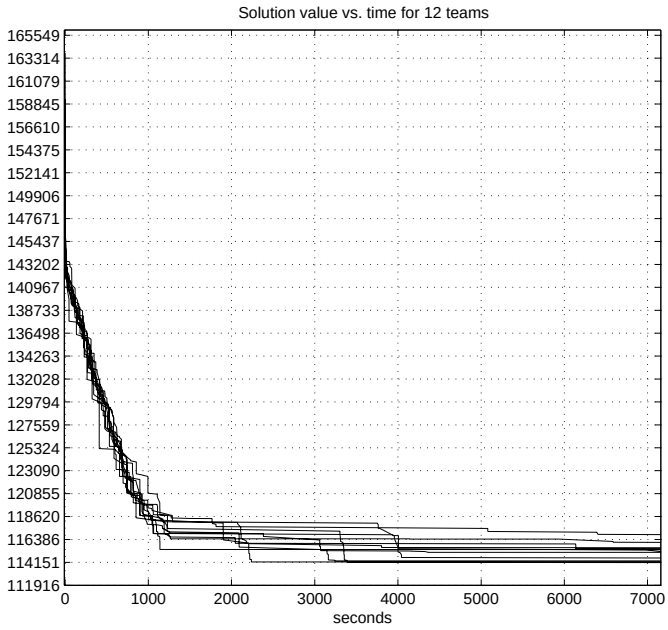

Figure 4: Solution Quality over Time for 12 Teams.

Method	Best	Worst	Mean	Std
TTSA	282,948	331,014	312,102	7,200
TTSA(FC)	277,626	295,299	286,527	4,125

Table 7: Solution Quality of TTSA and TTSA(FC) within 2 Hours on 16 Teams.

Best Solutions Since the Beginning of this Research Table 9 reports the evolution of the best solutions since we started this research in November 2002. Our currently best solution for $n = 12$ and $n = 14$, were produced by running TTSA at a fixed temperature, as shown in Table 10 which shows the parameters for each instance. The fixed temperature at each case was determined in the following way. For every n, we ran 50 experiments starting from different random points, all with the same parameters used in obtaining our previous best solution for that n. Then, we computed the average over the 50 experiments of the temperature at which the best solution was found for every experiment. This average gave us the starting temperature used in obtaining the latest best results. Note that, although we did not improve the best known solution for $n = 16$ using this approach, the results were pretty close to the value of the best known solution. In particular, after running 50 experiments, we got minimum cost 268137 and maximum cost 272376. Note that the best known solution not produced with TTSA is by Langford, January 2004, who gave a solution of value 272902.

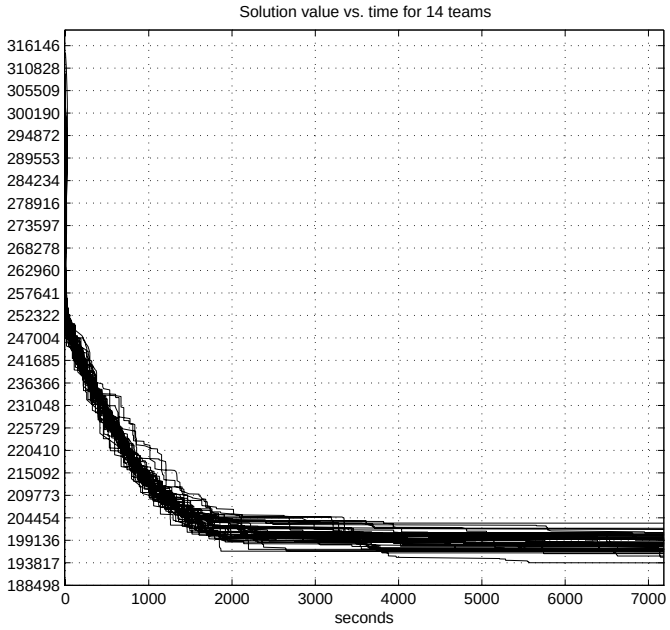

Figure 5: Solution Quality over Time for 14 Teams.

Method	T_0	β	w_0	δ	θ	$maxC$	$maxP$	$maxR$	γ
TTSA	700	0.9999	60000	1.05	1.05	10000	7100	50	2
TTSA(FC)	700	0.98	60000	1.05	1.05	5000	70	10000	2

Table 8: Parameter Values for Experiments on Fast Cooling (16 Teams)

4 Conclusion

Sport league scheduling has received considerable attention in recent years, since these applications involve significant revenues for television networks and generate challenging combinatorial optimization problems. This paper considers the traveling tournament problem (TTP) proposed in [10, 4] to abstract the salient features of major league baseball (MLB) in the United States. It described the simulated annealing algorithm TTSA for the TTP which represents the problem with hard and soft constraints, explores both feasible and infeasible schedules, uses a large neighborhood whose moves may have significant effects on the overall schedule, and incorporates advanced techniques such as strategic oscillation and reheats to balance the exploration of the feasible and infeasible regions and to escape local minima at very low temperatures. TTSA matches the best known solutions on the small instances of the TTP and produces significant improvements over previous approaches on the larger instances. TTSA is also shown robust, since its worst solution quality is always smaller or equal to the best known solutions. As a consequence, we believe that these results enhance our understanding of the effectiveness of various solution techniques on the TTP, which is a very hard combinatorial optimization problem.

There are a variety of open issues that need to be addressed. On the theoretical side, it would

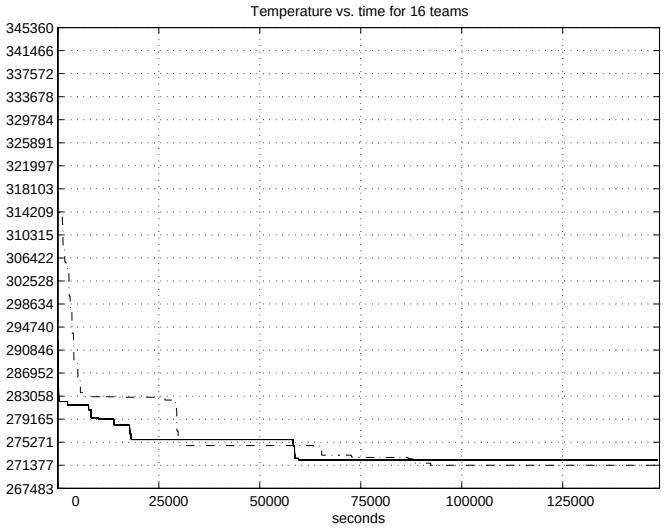

Figure 6: Solution Quality over Time for TTSA and TTSA(FC).

n	Nov 2002	Author	June 2003	Apr 2004	Author	TTSA	Sep 2005	Author
8	39721	Easton	39721	39721		39721	39721	
10	61608	Zhang	59583	59583		59583	59436	Langford
12	118955	Cardemil	112800	112298	Langford	**111248**	**111248**	
14	205894	Cardemil	190368	190056	Langford	**189766**	**189766**	
16	281660	Shen	**267194**	**267194**		**267194**	**267194**	

Table 9: Timeline of Best Known Solutions: Where no author is mentioned (including the Jun 2003 column), solutions were produced using TTSA. We show in bold face TTSA solutions that are the best known as of September 2005.

be interesting to determine if the neighborhood is connected. On the practical side, it would be interesting to explore other meta-heuristics that may improve the efficiency of the algorithm. This is a challenging task however, since it seems critical to consider a large neighborhood to obtain high-quality solutions. However, preliminary experimental results with fast cooling schedules are encouraging.

Acknowledgements

The authors are grateful to the four reviewers for their very interesting suggestions on the experimental results. This work is partially supported by by NSF ITR Awards DMI-0121495 and ACI-0121497.

n	T_0	β	w_0	δ	θ	$maxC$	$maxP$	$maxR$	γ
12	158	1	10000	1.03	1.03	∞	0	0	1
14	193	1	18000	1.03	1.03	∞	0	0	1

Table 10: Parameter Values for TTSA Instances

References

[1] T. Benoist, F. Laburthe, and B. Rottembourg. Lagrange Relaxation and Constraint Programming Collaborative Schemes for Travelling Tournament Problems. In *CP-AI-OR'2001*, Wye College (Imperial College), Ashford, Kent UK, April 2001.

[2] D.T. Connelly. General Purpose Simulated Annealing. *Journal of Operations Research*, 43, 1992.

[3] Juan A. Díaz and Elena Fernández. A tabu search heuristic for the generalized assignment problem. *European Journal of Operational Research*, 132(1):22–38, July 2001.

[4] K. Easton, G. Nemhauser, and M. Trick. The traveling tournament problem description and benchmarks. In *Seventh International Conference on the Principles and Practice of Constraint Programming (CP'99)*, pages 580–589, Paphos, Cyprus, 2001. Springer-Verlag, LNCS 2239.

[5] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[6] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[7] M. Laguna, J.P. Kelly, Gonzalez-Velarde, and F. Glover. Tabu search for the multilevel generalized assignment problems. *European Journal of Operational Research*, 42:677–687, 1995.

[8] I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.

[9] E. Pesch and F. Glover. TSP Ejection Chains. *Discrete Applied Mathematics*, 76:165–181, 1997.

[10] M. Trick. http://mat.gsia.cmu.edu/TOURN/, 2002.