EURO J Comput Optim
DOI 10.1007/s13675-013-0010-3

ORIGINAL PAPER

A complete search method for the relaxed traveling tournament problem

Filipe Brandão · João Pedro Pedroso

Received: 15 November 2012 / Accepted: 16 April 2013

Abstract The traveling tournament problem is a sports scheduling problem that includes two major issues in creating timetables: home/away pattern feasibility and travel distance. In this problem, the schedule must be compact: every team plays in every time slot. However, there are some sports leagues that have both home/away pattern restrictions and distance limits, but do not require a compact schedule. In such schedules, one or more teams can have a bye in any time slot. This leads us to a variant of the problem: the relaxed traveling tournament problem. We present a complete search method to solve this problem based on branch-and-bound, metaheuristics and dynamic programming.

Keywords Traveling tournament problem · Branch-and-bound · Metaheuristics · Dynamic programming

Mathematics Subject Classification 90-08 Computational methods · 90-XX Operations research, mathematical programming

Introduction

The advances in modeling the combinatorial structure of sports schedules and their solution, together with the increasing practical requirements for schedules by real sports leagues have increased the interest in computational methods for creating them.

F. Brandão (✉) · J. P. Pedroso
INESC TEC and Faculdade de Ciências, Universidade do Porto,
Rua do Campo Alegre , 4169-007 Porto, Portugal
e-mail: fdabrandao@dcc.fc.up.pt

J. P. Pedroso
e-mail: jpp@fc.up.pt

Published online: 30 April 2013

The key issues for constructing a schedule are travel distance and home/away pattern restrictions. While teams wish to reduce the total amount they travel, they are also concerned with more traditional issues with respect to home and away patterns.

The traveling tournament problem (TTP), which was proposed by Easton et al. (2001) abstracts the key issues in creating a schedule that combines home/away pattern constraints and travel distance minimization. Either home/away pattern constraints or travel distance minimization is reasonably easy to solve, but the combination of them makes this problem very difficult.

In TTP, the schedule must be compact: every team plays in every time slot; however, there are some sports leagues that have both home/away pattern restrictions and distance limits, but do not require a compact schedule. This leads us to a new problem: the relaxed traveling tournament problem (RTTP). This variant of the TTP was proposed by Bao (2006). As in this variant the schedule is not compact, teams have byes (i.e., slots where they do not play) in their schedule. The teams are allowed to have a fixed number K of byes and the objective is to minimize the travel distance.

A survey of scheduling problems arising in sports, as well as solution methods, is presented in Kendall et al. (2010).

The traveling tournament problem

In the TTP, there is an even number n of teams, each with a home venue. The teams wish to play a round robin tournament, whereby each team will play against every other team twice, once at each team's home venue. This means that $2(n-1)$ slots, or time periods, are required to play a double round robin tournament. There are exactly $2(n-1)$ time slots available to play these games, so every team plays in every time slot. Associated with a TTP instance, there is an n-by-n distance matrix D, where D_{ij} is the distance between the venues of team i and team j.

Each team begins at its home site and travels to play its games at the chosen venues. At the end of the schedule, each team returns to its home site.

Consecutive away games for a team constitute a road trip; consecutive home games are a home stand. The length of a road trip or home stand is defined as the number of opponents played (not the travel distance); both road trips and home stands have a lower bound l and an upper bound u on their length.

The TTP is defined as follows:

> Input: n, the number of teams; D, an n-by-n symmetrical distance matrix; l, u integer parameters.
> Output: A double round robin tournament on the n teams such that:
>
> - the length of every home stand and road trip is between l and u, inclusive;
> - games between the same opponents cannot happen in consecutive time slots; this is called the no repeater constraint;
> - the total distance traveled by the teams is minimized.

The parameters l and u define the trade-off between distance and pattern considerations. For $l = 1$ and $u = n-1$, a team may take a trip equivalent to a traveling salesman tour. For small u, teams must return home often, so the travel distance increases. Usually $l = 1$ and $u = 3$, which means that each team cannot play more than three consecutive home games or three consecutive away games.

The solution to the TTP has proved to be a computationally difficult challenge. For many years, the six-team instance NL6, available in Trick (2012), was the largest instance solved to optimality. In 2008, NL8 was solved; NL10 was solved in 2009. This leaves 12 teams as the next unsolved instance, which is a significantly small league size for such a simple problem description.

The relaxed traveling tournament problem

The goal in the TTP is to find a compact schedule: the number of time slots is equal to the number of games each team plays. This forces every team to play in every time slot. The input of the RTTP has an additional parameter K, specifying the number of byes allowed in the schedule. In this problem, the schedule is not required to be compact and teams are allowed to have a fixed number K of byes.

In this variant of the TTP, instead of fixing the schedule length to be $2(n-1)$, it is allowed to be $2(n-1)+K$ for some integer $K \geq 0$. For a given K, the problem is called K-RTTP. For $K = 0$, the RTTP is just the TTP. For $K > 0$, each team has K slots in which it does not play.

Byes are ignored in determining the length of a home stand or road trip, and in determining whether a repeater has occurred. This means that TTP's solutions are feasible for the K-RTTP for every $K \geq 0$. In fact, K_1-RTTP's solutions are feasible for K_2-RTTP if $K_1 \leq K_2$.

Solution methodology

For solving the RTTP, one has to deal with both feasibility concerns (the home and away pattern) and optimization concerns (the travel distance); this combination makes this problem very difficult to solve to optimality.

One of the most successful methods of solving the TTP is an algorithm that combines an iterative deepening algorithm with depth-first branch-and-bound (see, e.g., Uthus et al. 2009; for the iterative deepening algorithm see, e.g., Korf 1985). Other approaches include a simulated annealing metaheuristic (see, e.g., Anagnostopoulos et al. 2006), representing the problem with hard and soft constraints, and exploring both feasible and infeasible schedules based on a large neighborhood.

Our solution methodology for the RTTP is a complete search method, putting in place several tools: branch-and-bound (the main method), metaheuristics (for trying to improve bounds) and dynamic programming (to compute lower bounds quickly). The way we combined these tools is described below in Algorithm 1.

Algorithm 1 starts with an empty schedule, which corresponds to the root node in the stack, and with the upper bound UB set to infinity. While the stack is not empty (line 3), we pop the last node from the stack in line 4 and check if it is a leaf in line 5. If the node is as leaf, we apply a hill-climbing metaheuristic as described in Sect. 4.3; otherwise, we branch on the node if its lower bound does not exceed the upper bound (in line 10).

So far, the largest instance of the RTTP solved to optimality was NL4; our method allowed us to solve all the previously open instances with up to eight teams. For larger instances, our method was unable to reach solutions better than the best known solutions for the TTP.

Algorithm 1: Hybrid RTTP-Solver

```
UB ← ∞;
S ← [empty schedule];                                 // root node: empty schedule
while not empty(S) do
    u ← pop(S);
    if final(u) then                                  // final(u) is true if u is a leaf
        v ← hill-climbing(u);                         // as described in section 4.3
        if cost(v) < UB then
            UB ← cost(v);
        end
    else if cost(u) + ILB(u) < UB then                // else, the branch is pruned
        foreach v ∈ branch(u) do
            push(S, v);
        end
    end
end
```

Branch-and-bound

If solutions for the RTTP were generated team by team (i.e., fix all the games of a team before moving to the other team), it would become very difficult to check all the constraints of the problem. For example, when we fix a game for a team, we are also fixing a game for another team (the first's opponent) in the same round; however, we cannot apply, for instance, the restriction of home/away pattern to the opponent team, due to not having information about previous games.

Therefore, in our algorithm, solutions are generated round by round: all the games of one round are fixed before moving to the subsequent round. The advantage of this order is that we can verify restrictions earlier, avoiding the exploration of large parts of the branch-and-bound tree.

To enumerate solutions we use the following method:

1. start at the first round;
2. for each team, if a game is not scheduled yet, pick each possible opponent following the order of the input and try to schedule a game;
3. after trying all opponents, try to use a bye;
4. when the schedule for the current round is complete, repeat this process in the following round if the schedule is not yet complete.

For trimming off non-optimal candidates from the branch-and-bound tree, we use the current cost plus the independent lower bound (see Sect. 4.2) for the remaining games of each team.

Our implementation of the branch-and-bound procedure is based on a recursive depth-first search that only updates and restores global data structures. Since these data structures are updated quickly, it allows us to traverse the tree much faster than by using a stack.

Independent lower bound and dynamic programming

If we calculate the optimal schedule (that minimizes travel distance) for one team without taking into account the other teams' schedule, we have a lower bound to the distance traveled by that team. The sum over the n teams of the distances associated with their independent optimal schedule provides a simple but strong lower bound. This is called the independent lower bound (ILB), as was first proposed in Easton et al. (2002).

To calculate this lower bound, we need to know: the team, the current location, the number of remaining home games, the list of remaining away games and the current number of consecutive home/away games. This information can be used as a state in dynamic programming. Exploiting some symmetries, a small table suffices for holding this information; e.g., a 108 Mb table is sufficient for the 12 teams problem NL12, and it can be computed very quickly.

We identify each state by a tuple (team, location, number of remaining home games, key) where the key is composed of $n+2$ bits ($n-1$ bits to identify the remaining away games, each bit corresponding to an opponent, 1 bit to distinguish between home stand and road trip, and 2 bits for the length of the home stand/road trip). For instance, for $n = 16$, we have $16 \times 16 \times 16 \times 2^{16+2}$ states holding 32-bit integer values, resulting in a 4 Gb table. For instances with more than 16 teams, the size of this table grows very markedly and dynamic programming may not be so useful.

Before starting the branch-and-bound procedure, we compute the entire table using a recursive procedure with memoization, and later we just need to sum the lower bound over the n teams to obtain a lower bound for a given node.

Metaheuristics

Whenever we find a new solution inside the branch-and-bound tree, we apply a hill-climbing metaheuristic to try to improve bounds. When a local optimum is reached, random perturbations are applied to the solution; this perturbation and hill-climbing process is repeated a number of times (100, in our experiment).

To generate the neighbors for the current solution, we use three from the five transformations proposed in Anagnostopoulos et al. (2006). These movements are:

- `SwapHomes`(t_i, t_j): given two teams, their home/away roles in the two scheduled games between them are swapped;
- `SwapRounds`(r_k, r_l): this move swaps rounds r_k and r_l;
- `SwapTeams`(t_i, t_j): this move simply swaps the schedule of teams t_i and t_j.

Whenever applying a move leads to an invalid solution, the schedule is discarded. Moreover, these three moves are not sufficient for exploring the entire search space and, as a consequence, lead to suboptimal solutions. However, they require limited computational time and can lead to better solutions, thereby improving the upper bound and reducing the size of the branch-and-bound tree.

To perturb a solution when a local optimum is reached, we apply a `SwapHomes` move to two random teams, a `SwapRounds` move to two random rounds and finally a `SwapTeams` move to two random teams. In our experiment, we have repeated these perturbations ten times and the final solution is used as the next starting point for the hill-climbing procedure.

The use of this metaheuristic to improve bounds is particularly important for big instances, such as NL8, as it allows finding good solutions sooner and thus pruning more effectively the branch-and-bound tree. Small instances, such as NL6, can be solved without this component, as in this case the search tree (using only the ILB) is relatively small.

Data structures

To improve the run time of our branch-and-bound procedure, we use many auxiliary data structures so that every constraint can be checked quickly. For each team t, we have:

- cur_loc[t]-current location;
- last_op[t]-last opponent;
- byes[t]-current number of byes used;
- H[t]-current number of consecutive home games;
- A[t]-current number of consecutive away games.

We also have a matrix `G`[t_1][t_2] to check if the game between team t_1 and team t_2 (at t_1's venue) has already been scheduled. These auxiliary data structures can be updated quickly and allow us to check every constraint in constant time.

Computational results

The method proposed in this paper was tested on a subset of the benchmark instances available at Trick (2012). The results obtained for NL instances are reported in Table 1. The previous best known Table 2. Table 3 presents the results for other instances that are also available at Trick (2012). In these tables, n is the number of teams, K is the number of byes and ILB is the independent lower bound at the root node. For $n = 8$ and two byes, the solution for $K = 1$ was used as the initial upper bound ($*$); for $n = 8$ and three byes, the previous ($K = 2$) solution provided the initial upper bound ($\star$). CPU times were obtained with a (sequential) implementation in the C programming language, in a Quad-Core Intel Xeon at 2.66 GHz, running Mac OS X 10.6.6. The source code, optimal solutions and log files are available online[1].

To the best of our knowledge, Table 1 reports for the first time the proven optimal solutions to NL6 and NL8 instances. Note that even with $K = 0$, NL8 remained

[1] http://www.dcc.fc.up.pt/~fdabrandao/code.

Table 1 Results for NL instances

Name	n	K	ILB	Solution	Time
NL4	4	=0	8,044	8,276	0 s
NL4	4	=1	8,044	8,160	1 s
NL4	4	=2	8,044	8,160	0 s
NL4	4	≥3	8,044	8,044	0 s
NL6	6	=0	22,557	23,916	0 s
NL6	6	=1	22,557	23,124	6 s
NL6	6	≥2	22,557	22,557	1 s
NL8	8	=0	38,670	39,721	26 min
NL8	8	=1	38,670	39,128	44 h
NL8	8	=2	38,670	38,761	208 h*
NL8	8	≥3	38,670	38,670	92 h⋆

Table 2 Previous results for NL instances by Bao (2006)

Name	n	K	Solution
NL4	4	=1	8, 160
NL4	4	=2	8, 160
NL4	4	≥3	8, 044
NL6	6	=1	23, 791

open for 9 years. The proposed method takes less than half an hour to solve this instance to optimality. The relaxed version of the TTP seems to be harder to solve to optimality. Nevertheless, NL8 was solved to optimality in a few days. Table 3 presents the optimal solutions obtained with our method for additional instances, belonging to different classes, also previously unsolved.

Problem variants

The standard definition of the RTTP with byes does not model all the situations that may arise in practice. In some leagues, additional constraints on the byes are required. In some cases it may be necessary to impose a limit on the number of consecutive home or away byes. Moreover, to avoid speculation, it may be necessary to impose that every team has to play in the last time slot, i.e., no team ends its matches before the others. Even though these additional constraints are not part of the standard relaxed traveling tournament problem, our exact method is prepared to handle them. However, note that for a fixed number of byes K, these constraints easily lead to infeasible instances.

Concerning the performance of the method, for instances with up to six teams, the impact of these additional constraints on the run time is usually small. However, instances with constant distance matrix became harder to solve, likely due to the time required to prove optimality. Table 4 summarizes the results for RTTP with additional constraints for instances with up to six teams (we do not present results for con6 since it became harder to solve with the additional constraints and it was not possible to prove

Table 3 Results for other classes

Name	n	K	ILB	Solution	Time
con4	4	=0	16	17	0 s
con4	4	≥1	16	16	0 s
con6	6	=0	42	43	2 s
con6	6	≥1	42	42	0 s
con8	8	≥0	80	80	0 s
circ4	4	=0	16	20	0 s
circ4	4	=1	16	18	0 s
circ4	4	=2	16	18	0 s
circ4	4	≥3	16	16	0 s
circ6	6	=0	60	64	0 s
circ6	6	≥1	60	60	1 s
circ8	8	=0	128	132	18 min
circ8	8	≥1	128	128	23 min
super4	4	=0	63,192	63,405	0 s
super4	4	=1	63,192	63,334	0 s
super4	4	=2	63,192	63,263	0 s*
super4	4	≥3	63,192	63,192	0 s⋆
super6	6	=0	127,370	130,365	0 s
super6	6	=1	127,370	127,903	3 s
super6	6	≥2	127,370	127,370	3 s
super8	8	=0	177,258	182,409	18 min
super8	8	=1	177,258	178,115	5 h
super8	8	=2	177,258	177,406	195 h*
super8	8	≥3	177,258	177,258	14 h⋆
galaxy4	4	=0	412	416	0 s
galaxy4	4	=1	412	414	0 s
galaxy4	4	=2	412	413	0 s
galaxy4	4	≥3	412	412	0 s
galaxy6	6	=0	1,294	1,365	0 s
galaxy6	6	=1	1,294	1,330	6 s
galaxy6	6	≥2	1,294	1,294	0 s
galaxy8	8	=0	2,250	2,373	27 min
galaxy8	8	=1	2,250	2,298	16 h
galaxy8	8	=2	2,250	2,261	266 h*
galaxy8	8	≥3	2,250	2,250	2 h⋆

optimality in a reasonable amount of time; this is also the reason why we limited the number of byes to 4). We consider the standard RTTP and three variants: in RTTP-1, byes are only allowed during home stands; in RTTP-2, every team has to play in the last time slot; in RTTP-3 byes are only allowed during home stands and every team has to play in the last time slot.

Table 4 Results for RTTP with additional constraints

Name	K	RTTP		RTTP-1		RTTP-2		RTTP-3	
		Solution	Time (s)	Solution	Time (s)	Solution	Time (s)	Solution	Time (s)
NL4	=1	8,160	0	8,160	0	8,276	0	8,276	0
NL4	=2	8,160	0	8,160	0	8,160	0	8,276	0
NL4	=3	8,044	0	8,160	0	8,160	0	8,160	0
NL4	=4	8,044	0	8,160	0	8,160	0	8,160	1
NL6	=1	23,124	6	23,124	8	23,263	1	23,263	2
NL6	=2	22,557	1	22,557	1	22,890	2	22,968	5
NL6	=3	22,557	2	22,557	1	22,696	1	22,745	3
NL6	=4	22,557	0	22,557	2	22,604	2	22,604	3
con4	=1	16	0	17	0	16	0	17	0
con4	=2	16	0	17	0	16	0	17	0
con4	=3	16	0	17	0	16	0	17	0
con4	=4	16	0	17	1	16	0	17	0
circ4	=1	18	0	18	0	20	0	20	0
circ4	=2	18	0	18	0	18	0	20	0
circ4	=3	16	0	18	0	18	0	18	0
circ4	=4	16	0	18	0	18	0	18	0
circ6	=1	60	1	60	1	60	0	62	2
circ6	=2	60	0	60	0	60	0	60	1
circ6	=3	60	0	60	0	60	0	60	0
circ6	=4	60	0	60	0	60	0	60	1
super4	=1	63,334	0	63,334	0	63,334	0	63,405	0
super4	=2	63,263	0	63,334	0	63,263	0	63,405	0
super4	=3	63,192	0	63,334	0	63,263	0	63,334	0
super4	=4	63,192	0	63,334	0	63,263	0	63,334	0
super6	=1	127,903	4	127,903	4	128,187	1	128,187	1
super6	=2	127,370	3	127,472	43	127,477	1	127,614	3
super6	=3	127,370	2	127,370	7	127,370	1	127,472	5
super6	=4	127,370	1	127,370	39	127,370	1	127,472	19
galaxy4	=1	414	0	415	0	414	0	416	0
galaxy4	=2	413	0	415	0	413	0	416	0
galaxy4	=3	412	0	415	0	413	0	415	1
galaxy4	=4	412	0	415	0	413	0	415	0
galaxy6	=1	1,330	7	1,330	8	1,341	4	1,341	3
galaxy6	=2	1,294	1	1,294	0	1,314	2	1,317	6
galaxy6	=3	1,294	0	1,294	1	1,298	2	1,301	3
galaxy6	=4	1,294	1	1,294	2	1,295	1	1,297	4

As expected, the optimal objective value for problems with additional constraints is generally greater than the corresponding value in the standard problem. As the lower bounds are the same, this may explain the increase in the CPU time required for proving optimality in some of the variants studied in this section.

Conclusions

The solution to the traveling tournament problem proved to be a computationally difficult challenge. The combination of home/away pattern constraints and travel distance minimization makes this problem very difficult. Its relaxed version (RTTP) seems to be even harder to solve to optimality.

To tackle this problem, we combined different methods: branch-and-bound, dynamic programming and metaheuristics. These were combined in a careful computer implementation, allowing us to solve to optimality all the previously open instances with up to eight teams. The same algorithm was also used to solve some variants of this problem that may arise in practice.

This paper shows how important it is to combine different techniques in order to tackle this problem, since it combines feasibility and optimality issues. Besides that, another important contribution is the introduction of dynamic programming to compute the independent lower bound quickly.

References

Anagnostopoulos A, Michel L, Hentenryck PV, Vergados Y (2006) A simulated annealing approach to the traveling tournament problem. J Sched 9:177–193

Bao R (2006) Time relaxed round Robin Tournament and the NBA Scheduling Problem. Cleveland State University, Master's thesis

Easton K, Nemhauser G, Trick MA (2001) The traveling tournament problem description and benchmarks. Tepper School of Business

Easton K, Nemhauser GL, Trick MA (2002) Solving the travelling tournament problem: a combined integer programming and constraint programming approach. In: Burke EK, Causmaecker PD (eds) PATAT, volume 2740 of Lecture Notes in Computer Science. Springer, Berlin, pp 100–112

Kendall G, Knust S, Ribeiro C, Urrutia S (2010) Scheduling in sports: an annotated bibliography. Comput Oper Res 37:1–19

Korf RE (1985) Depth-first iterative-deepening: an optimal admissible tree search. Artif Intell 27(1):97–109

Trick M (2012) Challenge traveling tournament problems. http://mat.gsia.cmu.edu/TOURN. Accessed 15 Feb 2013

Uthus DC, Riddle PJ, Guesgen HW (2009) DFS* and the traveling tournament problem. In: van Hoeve WJ, Hooker JN (eds) CPAIOR, volume 5547 of Lecture Notes in Computer Science, Springer, Berlin, pp 279–293