A Constraint Programming Approach to the Traveling Tournament Problem with Predefined Venues

Gilles Pesant

the date of receipt and acceptance should be inserted later

Abstract The Traveling Tournament Problem with Predefined Venues (TTPPV) has been introduced as an abstraction of sports scheduling. Exact integer programming and heuristic approaches have been proposed so far. We investigate an exact constraint programming approach for this problem, discussing different models and search strategies. We report their respective performance on the standard set of benchmark instances and compare them to the current state of the art.

Keywords traveling tournament problem with predefined venues · constraint programming · sports scheduling

1 Introduction

The Traveling Tournament Problem with Predefined Venues (TTPPV) was introduced in [13] and consists of finding an optimal compact single round robin schedule for a sport tournament. Given a set of n teams, each team has to play once against every other team. In each game, a team is supposed to play either at home or away, however no team can play more than three consecutive times at home or away (in the rest of the paper, we will refer to this restriction as the *stretch* constraint). We seek to minimize the total distance traveled by all the teams. The main distinctive feature of this variant of the Traveling Tournament Problem (TTP) [4] is that the venue of each game is predefined, i.e. for the game in which team a plays against b it is already known whether it is going to be held at a's home or at b's home. A TTPPV instance is said to be balanced if the number of home games and the number of away games differ by at most one for each team; otherwise it is referred to

G. Pesant
École Polytechnique de Montréal, Canada
CIRRELT, Montreal, Canada
E-mail: Gilles.Pesant@cirrelt.ca

as non-balanced or random. TTPPV benchmark instances were created in [13] from existing TTP instances (the Circle instances, see [4]) by adding a venue for each game. The number of teams goes from 4 to 20 and there are twenty instances (ten balanced; ten random) of each size. Instances on n teams will be denoted CIRCn.

The integer programming models described in [13] solve to optimality instances with up to 8 teams but have great difficulty finding feasible solutions beyond 16-team instances. By removing the travel distance from the objective and replacing it with penalties associated with the (now relaxed) predefined venue and stretch constraints, they manage to generate feasible solutions for larger instances. More recently an iterated local search approach achieves much better solutions [2].

This paper's contribution is to show how to model and solve the TTPPV using constraint programming. This approach provides a concise formal model that can be used both in an exact or heuristic setting. It also offers the possibility to integrate side constraints easily. The rest of the paper is organized as follows: Section 2 gives a short introduction to constraint programming, Section 3 gradually describes CP models and search heuristics for the TTPPV, Section 4 presents search space exploration strategies and empirical results on instances of realistic size.

2 Constraint Programming

Constraint Programming (CP) is a powerful technique to solve combinatorial problems. It applies sophisticated inference to reduce the search space and a combination of variable- and value-selection heuristics to guide the exploration of that search space. The problem to solve is described through a formal model expressed using constraints from a rich set of modeling primitives. Each type of constraint encapsulates its own specialized inference algorithm.

2.1 CP Inference

To every variable of a CP model is associated a finite set called its *domain*: each value in that domain represents a possible value for the variable. Constraints on the variables forbid certain combinations of values. Picturing the model as a network whose vertices are the variables and whose (hyper)edges are the constraints provides insight into the basic algorithm used in CP. A vertex is labeled with the set of values in the domain of the corresponding variable and an edge is incident to those vertices representing the variables appearing in the associated constraint. Looking locally at a particular edge (constraint), the algorithm attempts to modify the label (reduce the domain) of the incident vertices (variables) by removing values which cannot be part of any solution because they would violate that individual constraint; this *local consistency* step can be performed efficiently. If every violating variable-value

pair is identified and removed, we achieve *domain consistency* which is the best we can do locally; sometimes achieving that level of consistency is computationally too costly and we will only remove values at both ends of a domain, achieving *bounds consistency* (typically for domains from a totally ordered set such as the integers).

The modification of a vertex's label triggers the inspection of all incident edges, which in turn may modify other labels. This recursive process stops when either all label modifications have been dealt with or the empty label is obtained, in which case no solution exists. The overall behavior is called *constraint propagation*.

2.2 CP Search

Since constraint propagation may stop with indeterminate variables (i.e. whose domain still contains several values) the solution process requires search, which can potentially take exponential time. It usually takes the form of a tree search in which branching corresponds to fixing a variable to a value in its domain, thus triggering more constraint propagation. We call *variable-selection heuristic* and *value-selection heuristic* the way one decides which variable to branch on and which value to try first, respectively. For combinatorial optimization problems, the tree search evolves into a branch-and-bound search in which branching is the same as before and lower bounds at tree nodes are obtained by various means.

2.3 CP for Sports Scheduling

The area of sports scheduling has already been quite successful for CP. For example it plays an important role in scheduling Major League Baseball in North America [5], it has been used to schedule the National Football League in the US [12], and has been shown to perform well for College Basketball [9]. In particular a CP model for the TTP is proposed in [10].

3 Modeling the TTPPV

In this section we present and evaluate empirically several models and search heuristics for the TTPPV. All tests were performed on a AMD Opteron 2.2GHz with 1GB of RAM and used the Ilog Solver 6.6 constraint programming language.

3.1 Initial Model

A CP model for the TTP was presented in [10] and can be partly transposed to the TTPPV. We describe an adaptation of it as our first model. For a

tournament with n teams, we will have $n-1$ rounds since it is built as a single round robin, as opposed to the TTP. We define *opponent* variables o_{ij}, $1 \leq i \leq n$, $1 \leq j \leq n-1$ to represent the opponent of team i in round j. We also define *home* variables h_{ij}, $1 \leq i \leq n$, $1 \leq j \leq n-1$ which are equal to 1 if team i plays at home in round j and zero otherwise.[1] The model is partly expressed as

$$\texttt{alldifferent}((o_{ij})_{1 \leq j \leq n-1}) \qquad 1 \leq i \leq n \tag{1}$$

$$o_{o_{ij},j} = i \qquad 1 \leq i \leq n, 1 \leq j \leq n-1 \tag{2}$$

$$o_{ij} \in \{1, \ldots, i-1, i+1, \ldots, n\} \qquad 1 \leq i \leq n, 1 \leq j \leq n-1 \tag{3}$$

$$h_{ij} \in \{0, 1\} \qquad 1 \leq i \leq n, 1 \leq j \leq n-1 \tag{4}$$

The `alldifferent` constraint, defined on a set of variables, enforces that these variables take on distinct values and a few different consistency levels (with corresponding filtering algorithms) can be achieved for it [11]. We will use domain consistency, the strongest possible for filtering variable domains. Constraints (1) state that each team plays exactly once against every other team (single round robin): all opponents must be different and there are as many opponents as there are rounds.[2] By definition, Constraints (3) guarantee that each team plays exactly one game in every round (compact tournament). However one must ensure that the schedule is consistent: Constraints (2) state that in any given round, the opponent of team i has its opponent variable set to i. This is an instance of the `element` constraint, which allows array indexing by finite-domain variables and maintains domain consistency [7]. Finally Constraints (4) express the choice of the venue for the game team i plays in round j.

To take into account the predefined venue of each game, we again use `element` with the $n \times n$ matrix V giving the venue of each game ($V[i,k] = 1$ if the game is hosted by i and 0 if it is hosted by k):

$$h_{ij} = V[i, o_{ij}] \qquad 1 \leq i \leq n, 1 \leq j \leq n-1 \tag{5}$$

Reference [10] is not very clear on how it handles the stretch constraint and there is only mention of "some inequalities". Since its publication there has been significant progress in modeling restrictive patterns on sequences of variables, notably the regular language membership (`regular`) constraint that takes as input an automaton describing the allowed patterns and achieves domain consistency [14]. We use it here:

$$\texttt{regular}((h_{ij})_{1 \leq j \leq n-1}, \mathcal{A}) \qquad 1 \leq i \leq n \tag{6}$$

The small automaton $\mathcal{A}$ for this constraint is depicted at Figure 1.

---

[1] *away* variables are also introduced in [10], but these are unnecessary since they are simply the opposite of the *home* variables.

[2] [10] uses the more general `cardinality` constraint since each opposing team is met twice in the TTP.

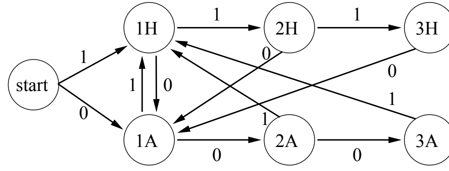

Fig. 1 Automaton for the "maximum 3 consecutive home or away games" restriction

Expressing the cost of a schedule is a bit tedious with this model because individual travel distances depend on pairs of consecutive variables. Let D represent the $n \times n$ travel distance matrix and variable d_{ij} the travel distance for team i to go play its game in round j. We follow [10] by considering four cases: two consecutive games for a team are either both played home, home then away, away then home, or both played away:

$$(h_{ij} = 1 \land h_{i,j+1} = 1) \Rightarrow d_{i,j+1} = 0 \qquad 1 \le i \le n, 1 \le j < n-1 \quad (7)$$

$$(h_{ij} = 1 \land h_{i,j+1} = 0) \Rightarrow d_{i,j+1} = D[i, o_{i,j+1}] \qquad 1 \le i \le n, 1 \le j < n-1 \quad (8)$$

$$(h_{ij} = 0 \land h_{i,j+1} = 1) \Rightarrow d_{i,j+1} = D[o_{ij}, i] \qquad 1 \le i \le n, 1 \le j < n-1 \quad (9)$$

$$(h_{ij} = 0 \land h_{i,j+1} = 0) \Rightarrow d_{i,j+1} = D[o_{ij}, o_{i,j+1}] \qquad 1 \le i \le n, 1 \le j < n-1 \quad (10)$$

$$(h_{i1} = 1) \Rightarrow d_{i1} = 0 \qquad 1 \le i \le n \quad (11)$$

$$(h_{i1} = 0) \Rightarrow d_{i1} = D[i, o_{i1}] \qquad 1 \le i \le n \quad (12)$$

$$(h_{i,n-1} = 1) \Rightarrow d_{in} = 0 \qquad 1 \le i \le n \quad (13)$$

$$(h_{i,n-1} = 0) \Rightarrow d_{in} = D[o_{i,n-1}, i] \qquad 1 \le i \le n \quad (14)$$

$$d_{ij} \in \{0\} \cup \{\min\{D\}, \ldots, \max\{D\}\} \qquad 1 \le i \le n, 1 \le j \le n \quad (15)$$

$$z = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \quad (16)$$

Constraints (7)-(10) state the four cases in terms of the *home* variables and correspondingly define the travel distance variables through indexing the travel distance matrix by the *opponent* variables. Constraints (11)-(14) handle the special cases of the first and last rounds. Note that these "$p \Rightarrow q$" constraints propagate in both directions: when p is satisfied then q is enforced; when q is violated then $\neg p$ is enforced. Constraint (16) sums the individual travel distances into z, the cost objective to be minimized.

Finally we must specify a search heuristic. [10] guides search by selecting uniformly at random the next variable to branch on (value selection is not mentioned). To be more precise, a team is first selected at random, then all rounds for that team are fixed in random order, where the *home* variable is fixed before the *opponent* variable. We initially do something similar, selecting uniformly at random the next *opponent* variable among those of smallest current domain size (a proven simple generic heuristic criterion) and selecting values randomly as well. We also report on the even simpler static heuristic selecting both variables and values lexicographically.

Table 1 Average computation time for Model (1)-(16) on the CIRC6 and CIRC8 instances

instance		time (sec)	
size	type	randomMinDom	lexico
6	random	0.07	0.10
	balanced	0.15	0.16
8	random	395.50	298.60
	balanced	–	–

Table 2 Average computation time for Model (1)-(17)

instance		time (sec)	
size	type	randomMinDom	lexico
8	random	–	567.8
	balanced	–	2926.4

On the TTP, [10] reported that he could solve the $n = 4$ and $n = 6$ instances but the latter required more than 10 minutes and 100 000 backtracks. Table 1 shows our results on the TTPPV $n = 6$ and $n = 8$ instances. The 6-team instances are easily solved whereas only the random 8-team instances are within reach with this model: neither search heuristic could solve a majority of the CIRC8 balanced instances. Note that one CIRC8 random instance could not be solved within the one-hour time limit and was therefore excluded from the average. The lexicographic search heuristic appears to do a little better than randomized smallest domain.

3.2 Adding Redundant Constraints

Even though our model accurately describes an optimal solution to the TTPPV, adding some constraints can help us to solve it faster. They will be redundant from a declarative point of view but the inference algorithms they encapsulate may filter out more values from the domains of variables and thus reduce the search space further, at the expense of extra computation.

As already pointed out in [10], the *opponent* variables in a given round must take distinct values:

$$\texttt{alldifferent}((o_{ij})_{1\leq i\leq n}) \qquad 1 \leq j \leq n-1 \tag{17}$$

Note that these do not replace Constraints (2), which are still needed.

Table 2 presents new results once redundant Constraints (17) are added. This time every 8-team instance is solved to optimality by the lexicographic heuristic, but not by the randomized-smallest-domain one. Henceforth we will use the former. There is still a sharp difference in performance between the random and balanced instances, probably due to the fact that the former are more constrained and therefore have a smaller search space.

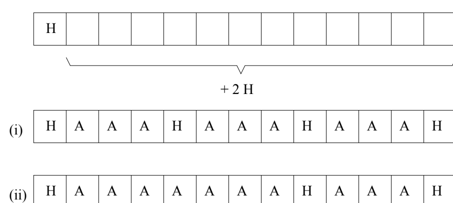

Fig. 2 14-team individual schedule with 3 predefined home games and starting with a home game. Both (i) and (ii) completed schedules are invalid.

Table 3 Average computation time for Model (1)-(5), (7)-(18) with the lexico search heuristic on the CIRC8 instances

instance		time (sec)
size	type	
8	random	413.2
	balanced	2666.4

3.3 Catching Infeasible Instances

In addition to proving optimality, another advantage of exact methods is identifying infeasibility. Arguably a successful tree search algorithm needs to be good at pruning infeasible subtrees. Among the CIRC instances, 14 are known to be infeasible. Using the previous model and search heuristic, five of these are proven without any search (the 6- and 8-team infeasible instances), one requires 367 backtracks, and another 2595142 backtracks in about 12 minutes. The other seven could not be proven within one hour.

One source of infeasibility is a team not having enough home (resp. away) games to separate long stretches of away (resp. home) games. A necessary condition is given in [13] and states that at least $\lfloor \frac{n-1}{4} \rfloor$ of each are needed. The inability of our model to detect this stems from Constraints (6) and (1) being handled separately: the former would need to know that games cannot be repeated. Alternatively we could make sure that we have the correct number of home games in each team's schedule. The `cost_regular` constraint [3], a variant of `regular` that additionally assigns an individual cost to each value taken by a variable and constrains their sum, can do just that:

$$\texttt{cost_regular}((h_{ij})_{1 \leq j \leq n-1},\ \mathcal{A},\ \textstyle\sum_{1 \leq k \leq n} V[i,k],\ [0,1]) \qquad 1 \leq i \leq n \quad (18)$$

The last argument attributes a cost to each value possibly taken by the h_{ij} variables: here an away game for i costs nothing whereas a home game costs 1, effectively counting the latter. The third argument counts the number of home games for team i according to matrix V and constrains the sum of individual costs to that value. This way the constraint simultaneously enforces the correct number of home/away games and the stretch requirement — it is also strictly stronger than the $\lfloor \frac{n-1}{4} \rfloor$ condition. Consider Figure 2: thirteen games are to be planned, three of them home games, and the first one being

Table 4 Average computation time for Model (1)-(5), (7)-(19) with simple static symmetry breaking and the lexico search heuristic on the CIRC8 instances

instance		time (sec)
size	type	
8	random	249.1
	balanced	1372.5

played home. Despite the $\lfloor \frac{n-1}{4} \rfloor$ condition being satisfied, there is no valid way to complete that schedule. For example, schedule (i) respects the stretch constraint but not the number of predefined home games, whereas schedule (ii) respects the latter but not the former. With this more powerful constraint replacing (6), the 14 instances are shown to be infeasible without any search (0 backtrack). This confirms that the source of infeasibility here is an insufficient number of home (or away) games for a team to respect the stretch restriction, now captured by a single constraint. Table 3 also shows that the additional inference improves the overall performance on feasible instances as well by pruning more infeasible subtrees.

3.4 Symmetry Breaking

The presence of symmetry in models can considerably slow down tree search approaches because the same infeasible subtree will be met repeatedly. This is especially true when we are solving an optimization problem and the whole tree must be traversed (even if only implicitly by pruning infeasible or provably suboptimal subtrees). Identifying and removing all symmetries is generally a very difficult task. Here there is one symmetry that is easy to see and remove: the mirror image of a schedule, going from the last round to the first. Such a transformation preserves the compact single round robin structure of the schedule and the stretch restriction. Games are still played at the same venues. And because travel distances are symmetric, its cost will be identical. We break that symmetry by selecting the first team, arbitrarily, and requiring that its first opponent be smaller than its last (according to team identifiers):

$$o_{11} < o_{1,n-1} \tag{19}$$

Table 4 shows breaking that symmetry improves the overall performance further, cutting the computation time almost by half. However solving the 10-team instances to optimality remains out of reach within one hour of computation time.

4 Exploring the Search Space

Given a CP model and search heuristics to dictate how we branch at a search tree node, there are still many ways we can explore the search space, even if it is all based on tree search. This section explores two possible avenues.

Table 5 Average solution value for the CIRC 10-team random instances with and without limited discrepancy search

LDS	1 sec.	1 min.	1 hour
no	168.9	163.4	158.9
yes	171.1	164.0	158.6

Table 6 Best solution value for the feasible CIRC 18-team random instances using different approaches

instance	CP without LDS			CP with LDS			IP	ILS	
	1 sec.	1 min.	1 hour	1 sec.	1 min.	1 hour	2 hours	1 sec.	2 hours
A	1040	1028	998	1054	986	962	1124	940	806
D	1052	1014	1006	–	1020	972	1060	914	800
E	1018	994	994	1038	1018	972	1092	922	804
F	986	970	962	1006	974	936	1098	956	802
G	1002	988	976	–	990	962	1098	924	782
H	996	972	972	–	1004	956	1110	944	804
I	1000	990	968	–	958	936	1104	932	818
J	944	928	918	–	970	896	1102	898	782

4.1 Adding Robustness to the Search Heuristic

It is well known for tree search that regardless of a search heuristic's quality, a depth-first traversal may take a very long time to undo a bad branching decision made early on. Our simple static search heuristic certainly is no exception. There are a few devices commonly used to add robustness to such search heuristics, e.g. randomized restarts and limited discrepancy search (LDS) [6]. The latter modifies the order in which the leaves of a search tree are visited according to how often the corresponding path deviates from the search heuristic's recommendation: first the leaf with 0 deviation, then those with $1k$ deviations, followed by those with $2k$ deviations, and so forth, for a given parameter k. This has the effect of more quickly changing decisions close to the root. We add LDS to the lexicographic search heuristic as its tree traversal strategy.

Table 5 compares the average value of solutions found for the 10-team random instances after 1 second, 1 minute, and one hour, with and without LDS (the behavior on the balanced instances is similar). Note that the quality starts out worse with LDS but it eventually catches up to and exceeds the performance of the heuristic without LDS. The improvement observed here is small but we next confirm it on instances of realistic size, comparing it at the same time to the state of the art.

The two previous papers on the TTPPV both use the CIRC 18- and 20-team instances. Table 6 and 7 report our results for $n = 18$ (Column 2 to 7) as well as the best ones for [13] (Column 8) and [2] (Column 9 and 10). Our solutions are considerably better than the ones obtained with IP or their polishing/enumerative heuristics, even after only 1 second of computation.

Table 7 Best solution value for the feasible CIRC 18-team balanced instances using different approaches

instance	CP without LDS			CP with LDS			IP	ILS	
	1 sec.	1 min.	1 hour	1 sec.	1 min.	1 hour	2 hours	1 sec.	2 hours
A	998	972	960	1036	980	932	1106	912	776
B	1012	1006	990	1036	972	924	1100	896	796
C	1068	1040	1022	1050	980	978	1038	892	794
D	1020	1002	986	–	1022	988	1096	882	788
E	1018	1012	1006	–	954	948	1074	892	784
F	1044	1030	1016	1034	1014	976	1060	910	792
G	972	948	942	1018	972	918	1100	894	784
H	986	964	956	–	984	918	1094	880	780
I	1004	994	978	1074	982	968	1102	894	778
J	1022	994	966	994	968	930	1078	878	780

Table 8 Best solution value for the feasible CIRC 20-team random instances using different approaches

instance	CP without LDS			CP with LDS			IP	ILS	
	1 sec.	1 min.	1 hour	1 sec.	1 min.	1 hour	2 hours	1 sec.	2 hours
A	1422	1410	1380	–	1418	1340	1502	1270	1106
B	1456	1454	1446	–	1436	1358	1522	1258	1082
C	–	–	1440	–	1424	1324	1488	1318	1096
D	1386	1376	1360	–	1368	1334	1510	1294	1136
E	1432	1410	1404	–	1432	1346	1574	1250	1100
G	1400	1380	1370	–	1386	1362	1540	1278	1078
I	1388	1366	1348	–	1360	1304	1516	1236	1082
J	1376	1360	1356	–	1344	1272	1516	1220	1070

For every instance but one, using LDS yields noticeably improved solutions after one hour. Our best solutions after one hour are comparable to those obtained by ILS after one second on the random instances but are inferior on the balanced instances. This may be explained by the larger search space of the latter for our exact approach. On none of the instances are we competitive with ILS given a reasonable amount of time (two hours).

Table 8 and 9 report our results for $n = 20$ (Column 2 to 7) as well as the best ones for [13] (Column 8) and [2] (Column 9 and 10). Again our solutions are much better than the ones from [13] and LDS systematically improves our exact algorithm. Unfortunately the gap between our performance and that of ILS widens.

4.2 Applying Large Neighborhood Search to the CP Model

Local search is often the method of choice to solve large combinatorial optimization problems and as we saw it is currently the best method for this problem. Several ways to combine CP and local search have been proposed in the literature, e.g. [15], [16], [8]. Large Neighborhood Search (LNS) iteratively

Table 9 Best solution value for the feasible CIRC 20-team balanced instances using different approaches

instance	CP without LDS			CP with LDS			IP	ILS	
	1 sec.	1 min.	1 hour	1 sec.	1 min.	1 hour	2 hours	1 sec.	2 hours
A	1380	1368	1360	1362	1356	1302	1520	1236	1094
B	1412	1386	1382	1432	1394	1346	1530	1252	1082
C	1408	1388	1370	1366	1362	1338	1470	1234	1072
D	1404	1400	1398	–	1430	1354	1464	1238	1100
E	1426	1416	1402	–	1414	1356	1526	1214	1076
F	1390	1372	1356	1440	1404	1340	1546	1236	1072
G	1348	1334	1316	–	1368	1296	1536	1210	1068
H	1446	1430	1410	1444	1358	1336	1516	1268	1094
I	1378	1356	1352	1422	1362	1310	1544	1238	1078
J	1410	1400	1368	–	1340	1308	1484	1222	1086

Table 10 Compared best solution value for the feasible CIRC 18-team random (left) and balanced (right) instances using LNS

	LDS	LNS	ILS	
	1 hour	1 hour	1 sec.	2 hours
A	962	894	940	806
D	972	902	914	800
E	972	882	922	804
F	936	902	956	802
G	962	896	924	782
H	956	868	944	804
I	936	888	932	818
J	896	870	898	782

	LDS	LNS	ILS	
	1 hour	1 hour	1 sec.	2 hours
A	932	890	912	776
B	924	896	896	796
C	978	910	892	794
D	988	900	882	788
E	948	866	892	784
F	976	918	910	792
G	918	882	894	784
H	918	856	880	780
I	968	914	894	778
J	930	874	878	780

freezes part of the current solution and explores the remaining search space (its potentially large neighborhood) by applying a (typically incomplete) CP tree search, benefiting from the usual inference and search heuristics. It is thus easy to transform an exact CP approach to one using LNS.

We tried a simple implementation of this idea. We run our exact CP algorithm for a few seconds in order to get a fair initial solution. We then freeze the schedule of a randomly selected small subset of the teams (here, 6 teams). We explore the neighborhood with the same exact CP algorithm, stopping at the first improving solution or until a time limit of 30 seconds is reached. We stop after 100 consecutive unsuccessful iterations or one hour. Admittedly this is a bare-bones representative of local search methods.

Table 10 reports empirical results on the $n = 18$ instances. It reiterates some of the results from previous tables and adds a column for LNS. On every instance LNS significantly improves our LDS results. On random instances LNS solution values now lie somewhere between the one-second and two-hours results of ILS. On balanced instances they are now comparable to the one-second results. Table 11 reports empirical results on the $n = 20$ instances.

Table 11 Compared best solution value for the feasible CIRC 20-team random (left) and balanced (right) instances using LNS

	LDS	LNS	ILS	
	1 hour	1 hour	1 sec.	2 hours
A	1340	1206	1270	1106
B	1358	1294	1258	1082
C	1324	1286	1318	1096
D	1334	1250	1294	1136
E	1346	1278	1250	1100
G	1362	1280	1278	1078
I	1304	1258	1236	1082
J	1272	1234	1220	1070

	LDS	LNS	ILS	
	1 hour	1 hour	1 sec.	2 hours
A	1302	1270	1236	1094
B	1346	1270	1252	1082
C	1338	1264	1234	1072
D	1354	1268	1238	1100
E	1356	1244	1214	1076
F	1340	1218	1236	1072
G	1296	1248	1210	1068
H	1336	1266	1268	1094
I	1310	1268	1238	1078
J	1308	1242	1222	1086

Again LNS dominates LDS. On both random and balanced instances it is often comparable to the one-second results of ILS.

5 Conclusion

We presented a constraint programming approach to the traveling tournament problem with predefined venues. A model was gradually refined and a few search heuristics and strategies were considered. On standard benchmark instances of realistic size, this approach outperforms a previous integer programming exact approach and its related heuristic variants but falls short of competing with the current best local search approach to this problem.

Much has yet to be tried with this CP approach and we believe there is real potential for improvement. This is especially true of our search heuristic which is currently static: it has performed well but a dynamic search heuristic tailored to the problem should perform better. There is definitely a lot of room for improvement in the local search approach with LNS which is currently very simple. As an exact algorithm it could be brought to solve the 10-team instances in reasonable time. Finally the easy integration of side constraints is also an asset for a CP approach in the real world of sports scheduling.

References

1. Burke, E.K., Trick, M.A. (eds.): Practice and Theory of Automated Timetabling V, 5th International Conference, PATAT 2004, Pittsburgh, PA, USA, August 18-20, 2004, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 3616. Springer (2005)
2. Costa, F., Urrutia, S., Ribeiro, C.: An ILS heuristic for the traveling tournament problem with predefined venues. Annals of Operations Research **194**(1), 137–150 (2012)
3. Demassey, S., Pesant, G., Rousseau, L.M.: A Cost-Regular Based Hybrid Column Generation Approach. Constraints **11**(4), 315–333 (2006)
4. Easton, K., Nemhauser, G.L., Trick, M.A.: The Traveling Tournament Problem Description and Benchmarks. In: T. Walsh (ed.) CP, *Lecture Notes in Computer Science*, vol. 2239, pp. 580–584. Springer (2001)

5. Easton, K., Nemhauser, G.L., Trick, M.A.: Solving the Travelling Tournament Problem: A Combined Integer Programming and Constraint Programming Approach. In: E.K. Burke, P.D. Causmaecker (eds.) PATAT, *Lecture Notes in Computer Science*, vol. 2740, pp. 100–112. Springer (2002)
6. Harvey, W.D., Ginsberg, M.L.: Limited Discrepancy Search. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI- 95, pp. 607–615. Morgan Kaufmann (1995)
7. Hentenryck, P.V., Carillon, J.P.: Generality versus Specificity: An Experience with AI and OR Techniques. In: H.E. Shrobe, T.M. Mitchell, R.G. Smith (eds.) AAAI, pp. 660–664. AAAI Press / The MIT Press (1988)
8. Hentenryck, P.V., Michel, L.: Constraint-based local search. MIT Press (2005)
9. Henz, M.: Scheduling a Major College Basketball Conference—Revisited. Operations Research **49**(1), 163–168 (2001)
10. Henz, M.: Playing with Constraint Programming and Large Neighborhood Search for Traveling Tournaments. In: Burke and Trick [1]
11. van Hoeve, W.J.: The alldifferent Constraint: A Survey. CoRR **cs.PL/0105015** (2001)
12. Lustig, I.: Scheduling the NFL with Constraint Programming. In: Burke and Trick [1]
13. Melo, R., Urrutia, S., Ribeiro, C.: The traveling tournament problem with predefined venues. Journal of Scheduling **12**(6), 607–622 (2009)
14. Pesant, G.: A Regular Language Membership Constraint for Finite Sequences of Variables. In: M. Wallace (ed.) CP, *Lecture Notes in Computer Science*, vol. 3258, pp. 482–495. Springer (2004)
15. Pesant, G., Gendreau, M.: A constraint programming framework for local search methods. J. Heuristics **5**(3), 255–279 (1999)
16. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: M.J. Maher, J.F. Puget (eds.) CP, *Lecture Notes in Computer Science*, vol. 1520, pp. 417–431. Springer (1998)