Solving the Traveling Tournament Problem: A Combined Integer Programming and Constraint Programming Approach

Kelly Easton[1], George Nemhauser[1], and Michael Trick[2]

[1] School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia USA,30332 {keaston,george.nemhauser}@isye.gatech.edu
[2] Graduate School of Industrial Administration, Carnegie Mellon, Pittsburgh, PA USA, 15213 trick@cmu.edu***

Abstract. The Traveling Tournament Problem is a sports timetabling problem requiring production of a minimum distance double round robin tournament for a group of n teams. Even small instances of this problem seem to be very difficult to solve. In this paper, we present the first provably optimal solution for an instance of 8 teams. The solution methodology is a parallel implementation of a branch and price algorithm that uses integer programming to solve the master problem and constraint programming to solve the pricing problem. Additionally, constraint programming is used as a primal heuristic.

1 Introduction

The Traveling Tournament Problem (TTP) represents the fundamental issues involved in creating a schedule for sports leagues where the amount of team travel is an issue. For many of these leagues, the scheduling problem includes a myriad of constraints based on thousands of games and hundreds of team idiosyncrasies that vary in their content and importance from year to year, but at its heart are two basic requirements. The first is a feasibility issue in that the home and away pattern must be sufficiently varied so as to avoid long home stands and road trips. The second is the goal of preventing excessive travel. For simplicity, we state this objective as minimize total travel distance.

While each issue has been addressed by either the integer programming or constraint programming communities (and sometimes both), their combination is a relatively new problem for both groups. Constraint programming has been successfully used to solve complex sets of home and away pattern constraints. Integer programming methods have been used to solve large traveling salesman and vehicle routing problems that require minimizing travel distance. It might seem that insights from these solution methodologies would make the TTP relatively easy to solve. However, even very small instances of the TTP have proven

*** The authors would like to thank CPLEX, a division of ILOG, for its software support. This work was supported, in part, by NSF grants DMI-0101020 and DMI-0121495 to the Georgia Institute of Technology.

difficult for traditional methods. Thus the TTP seems to be a good problem for a combined approach.

The TTP was introduced in a paper by Easton, Nemhauser, and Trick [6], in which we give results for some basic formulations. Here, we outline an approach that combines integer and constraint programming methods to solve larger problems much more quickly. This approach is appealing not just for its application to the TTP, but for the possibilities inherent for other difficult combinatorial problems.

In the following section, we give a formal definition for the Traveling Tournament Problem. In section 3, we discuss the single-team problem, show its complexity, and discuss how that problem can be used to bound solutions on the TTP. In section 4, we present our solution methodology. We then conclude with some computational results and a discussion of future research.

2 The Traveling Tournament Problem

Given n teams, n even, a double round robin tournament (DRRT) is a set of games in which each team plays each other team once at home and once away. A schedule for a DRRT is a mapping of games to slots, or time periods, such that each team plays exactly once in each slot. A DRRT schedule covers exactly $2(n-1)$ slots. The distances between the team venues are given by an n by n matrix D. For the distance calculations, it is important to note that each team starts and finishes the tournament at its home venue.

A road trip, or trip, is defined as a series of consecutive away games. Similarly, a home stand is defined as a series of consecutive home games. The length of a road trip or home stand is the number of games in the series (not the travel distance).

The TTP is defined as follows:

Input: A set of n teams $T = \{t_1, \ldots, t_n\}$ with n even; D a symmetric n by n integer distance matrix with elements d_{ij}; l, u integer parameters, $l \leq u$.
Output: A double round robin tournament on the teams in T such that

- The length of every home stand and road trip is between l and u inclusive, and
- The total distance traveled by the teams is minimized.

For $u = n - 1$, the maximum value for u, a team may visit every opponent without returning home, which is equivalent to a traveling salesman tour. For small u, a team must return home often, and consequently, its travel distance increases. For $u = 1$, the objective becomes constant, and the problem is solely one of feasibility. In practice, $l = 1$ and $u = 3$ or $u = 2$ are most commonly used. Additional background on these parameters and a description of other instances appear in [6].

Although we don't have a formal proof, we strongly believe that the TTP is $\mathcal{NP}$-hard. Indeed, computationally, it appears to be much harder than the

Traveling Salesman Problem (TSP). As shown in the next section, some simple relaxations of TTP are provably hard.

3 The Single Team Problem

Fundamental to our solution methodology is the idea of a tour. A tour is a vector that describes the travel of a single team. Each element in the vector is associated with a slot in the schedule and gives the venue for the game in that slot. For example, in an instance with $n = 6$, a tour for team 1 could be (1, 2, 3, 4, 1, 1, 5, 1, 6, 1). Note that each opponent's venue appears exactly once and the home venue appears exactly $n - 1$ times.

A team's optimal tour minimizes the travel distance for that team exclusive of the other teams in the schedule. The sum over n teams of the distances associated with their optimal tours provides a simple but strong lower bound on the TTP. We call this the Independent Lower Bound (ILB).

For most values of l and u, simply generating an optimal tour for a single team is $\mathcal{NP}$-hard, although there exist some polynomial solvable special cases. Here we study the complexity of generating a minimum distance tour for each (feasible) combination of l and u. We will assume that the triangle inequality holds for the distances. The decision form of the Single Team Problem (STP) is formally defined as:

Input: A set of n teams $T = \{t_1, \ldots, t_n\}$ with n even; a designated team $t_r \in T$; D a symmetric n by n integer distance matrix with elements d_{ij} satisfying the triangle inequality; l, u integer parameters; integer threshold k.
Output: Does there exist a tour for t_r such that

- the length of every home stand and road trip is between l and u inclusive, and
- the sum of the distance traveled by t_r is less than or equal to k?

For specified l and u, we use the notation STP(l,u).

At the extremes of l and u, the complexity of STP(l,u) is obvious. For STP(1,1), the objective is constant, and a solution can be obtained by alternating away and home games. For STP(l,n-1), then the problem is a TSP and is $\mathcal{NP}$-complete. In fact, as long as u is proportional to n^α for a fixed constant α, there is an immediate reduction from the TSP to prove completeness.

The STP is also $\mathcal{NP}$-complete for any constant $u > 2$ and $l \leq u$. The reduction is from the $\mathcal{NP}$-complete problem Partition into Isomorphic Subgraphs Restricted to Paths (PISP) [9]. The proof of this is given in [5].

In the special case where $u = 2$ ($l = 1$ is the only feasible assignment for l since n is even), a $O(n^3)$ algorithm can be obtained using weighted matchings to find the optimal tour. To see this first note that there will always be an optimal tour with the minimum number of trips. Two single team trips may be combined into one two team trip with a distance no more than the sum of the two single team trips by the triangle inequality. Secondly, since we are working with even

n, a minimum set of trips will always include exactly one single team trip. Given these observations, the following algorithm finds a minimum distance tour:

Algorithm STP(1,2)

1. Given team t_r, generate a complete graph $G = (V, E)$ with n nodes. Let vertex i represent team t_i for all $i \in \{1, \ldots, n\}$. Assign weights to the edges as follows: Let $w_{pq} = d_{rp} + d_{pq} + d_{qr}$ for all $p, q \in \{1, \ldots, n\} \setminus \{r\}, p \neq q$. Set $w_{rp} = 2 * d_{rp}$ for all $p \in \{1, \ldots, n\} \setminus \{r\}$.
2. Find a minimum weight perfect matching on G.
3. Create a minimum distance tour from the matching: For each matched pair of nodes (p, q) such that $p, q \in \{1, \ldots, n\} \setminus \{r\}$, create a two game road trip with t_r playing at t_p then at t_q. For the node p that is matched to r, create a single team trip in which t_r plays at t_p. In addition, create a set of home stands mimicking the trips, i.e. one home stand with one slot and $n/2 - 1$ home stands with 2 trips. Finally, schedule the tour by alternately selecting trips and home stands and scheduling them consecutively in the slots of the tournament.

Theorem 1 Algorithm STP(1,2) yields an optimal solution.
Proof Every feasible tour with a minimum number of trips corresponds to a perfect matching in G with the distance of the tour equal to the weight of the matching. Therefore, finding a minimum weight perfect matching in G yields an optimal solution to STP(1,2). □

The running time of this algorithm is dominated by the work required to solve the matching. A weighted perfect matching on a general graph can be found in $O(|E||V| + log(|V|)|V|^2) = O(n^3)$ [7]. Therefore, the algorithm runs in $O(n^3)$. This algorithm can be easily generalized in the case the triangle inequality does not hold by using b-matchings instead of perfect matchings. This use of matchings in the case where trips consist of at most a pair of teams is fundamental to the approaches of [2] and [11].

4 Solution Methodology

The TTP has feasibility elements (the home and away pattern) and optimization elements (the travel distance). Constraint programming has been successfully used to solve complex timetabling feasibility problems [8] while integer programming has been successfully used to solve difficult optimality problems like the TSP [1]. The combination of feasibility and optimality in a timetabling problem seems to be difficult for either method, thus we chose a combined approach.

Our solution methodology for the TTP is a branch-and-price (column generation) algorithm in which individual team tours are the columns. In branch-and-price, the linear programming (LP) relaxation at the root node of the branch and bound tree includes only a small subset of the columns. To check the LP objective, a subproblem, called a pricing problem, is solved to determine whether there

are any additional columns available to enter the basis. If the pricing problem returns one or more columns, the LP is reoptimized. If no more columns can be found to enter the basis and the LP solution is fractional, the algorithm branches. Branch-and-price is a generalization of branch-and-bound with LP relaxations. In our combined integer programming-constraint programming approach, we use constraint programming to solve the pricing problem.

Although the numbers of teams in the instances of the TTP that we have been able to solve seem small, these problems are actually quite large in terms of the numbers of possible columns. For an instance with 8 teams there are roughly a total of 4.5 million tours. In order to solve these instances in a reasonable amount of time, we run a parallel version of our branch-and-price algorithm. We use the master-slave paradigm. After generating an initial set of nodes, the master processor passes one node to each slave along with the best known solution (if any). Each slave evaluates a fixed number of nodes and passes back its best solution and all the nodes it has generated, specifying which nodes have been evaluated and which have not. If a slave completely fathoms its portion of the tree, no nodes are returned to the master. The master maintains the best solution reported by the slaves and a list of all unevaluated nodes. A single node from this list and the current best solution are passed to the now idle slave for processing. When the master's list is empty and all the processors are idle, the algorithm is complete and the current best solution is an optimum.

At the root node, we start with a small set of columns $P = \{1, 2, \ldots, m\}$ composed of the optimal tours for each team. Let the vector c represent the distances associated with the tours; T be the set of teams in the tournament and S be the set of slots. Let P_t be the indices of tours for team $t \in T$ and x be a binary decision variable that represents the tours. This leads to an integer program:

Minimize $\sum_{i \in P} c_i x_i$

subject to

$\sum_{i \in P_t} x_i = 1 \quad \forall t \in T$

$\sum_{\{i \in P : i \notin P_t \text{and } t \text{ is opponent in slot } s \text{ in } i\}} x_i +$
$\sum_{\{i \in P_t : t \text{ is away in slot } s \text{ in } i\}} x_i = 1 \quad \forall s \in S, \forall t \in T$

x_i binary for $i \in P$.

The first set of constraints forces exactly one tour to be selected for each team in the tournament. The second set of constraints requires each team to play exactly once in each slot.

In some branch-and-price algorithms, columns are generated until no more negative reduced cost columns remain. In others, columns are generated only if a node is about to be cut off. The branch and price routine described here falls into the latter category. One reason for this is that if fewer columns are generated, the master problem will solve more quickly. More importantly, however, the pricing problem is easier to solve in the latter case. So generating columns only

if necessary reduces the time it takes to solve both the master and the pricing problem. By not always generating all columns, however, we are unable to use a best bound node selection strategy. With parallel programming, however, it is difficult to assess the best bound at any given time, so this disadvantage is not key.

Prior research using branch and price algorithms indicates that it is most efficient to generate a pool of columns, select from this pool at each node, then generate a new pool when necessary [3]. Thus at each node in our search tree, we first check to see if there are any negative reduced cost columns in the pool. If so, we select up to 10 columns for each team. If not, we refill the pool. The pricing problem is solved using constraint programming.

When invoked, the constraint program for the subproblem is run once for each team. It generates all negative reduced cost tours. The variables are the venues at which the team's games are played in each slot. The domains of these variables are the teams in the tournament. Two dummy variables are added for slots 0 and $2(n-1)+1$. These variables are both set equal to the home team and are used for distance calculations. The distances between consecutive venues are also variables but are used strictly for calculating the objective or goal. Before the CP is run, the variable domains are reduced according to prior branching.

The constraints in this model are as follows (for the $l = 1, u = 3$ case):

- Each opponent venue appears exactly once.
- The home venue appears $n - 1$ times.
- No more than 3 home venues may appear consecutively.
- No more than 3 non-home venues may appear consecutively.
- The sum of the distances and the dual values associated with the games in the tour must be negative (equivalently, the reduced cost of the resulting tour must be negative).

Variables are selected in order of domain size. In the case that all uninstantiated variables have domains of equal size, the variable closest to an end of the tournament (either the first or the last uninstantiated slot) is selected. Ties in this second criteria are broken arbitrarily. Once a variable is selected, it is instantiated with the team in its domain that will make the most negative (or least positive) contribution to the total cost of the tour which includes the distances between consecutive venues and the reduced costs associated with opponents and slots.

A variable instantiation triggers domain reduction for the remaining uninstantiated variables. If an away game is being scheduled, the algorithm first eliminates that opponent from further consideration. Otherwise, a home game is being scheduled, and the home venue is eliminated from further consideration only if the home game tally reaches $n - 1$. Secondly, the pattern constraints are considered. Any members of the remaining domains that would create road trips or home stands longer than 3 slots are eliminated. Finally, the contribution to the objective of potential assignments is considered. Generating new columns is a fairly simple procedure, and we have observed no runtime savings from adding

a degree of look ahead to our algorithm. If a domain is reduced to no remaining elements, we backtrack to the last decision point. If a domain is reduced to one, the variable is marked. At the completion of an iteration of constraint propagation, the algorithm selects a marked variable (if there are any), instantiates it with the single member of its domain, and begins another iteration of constraint propagation. We use our own code to solve the pricing problem, but a model in OPL is given below for clarity.

```
   int home = 1;
   int  nbTeams = 8;
   range Teams 1..nbTeams;
   int nbSlots = 14;
   range Slots 1..nbSlots;
   float EPS = 0.0001;
   int Distance[Teams,Teams] = ...;
   float rc[Teams,Slots] = ...;
   var int travel[0..nbSlots] in 0..1380;
   var int venue[0..nbSlots+1] in 1..nbTeams;
   var float totcost;

   solve {

/* Venue variables */
     forall (i in Teams: i <> home) sum (s in Slots) (venue[s]=i)=1;
       sum (s in Slots) (venue[s]=home)=0.5*nbSlots;
       venue[0]=home;
       venue[nbSlots+1]=home;
       forall (s in 1..nbSlots-4) sum (j in 0..3) (venue[s+j]=home) <= 3;
       forall (s in 1..nbSlots-4) sum (j in 0..3) (venue[s+j] <> home) <= 3;

/* Distance calculations */
   forall (s in 0..nbSlots) travel[s]=Distance[venue[s],venue[s+1]];

/* Generate only negative reduced cost columns */
   totcost = sum (s in 0..nbSlots) travel[s] +sum (s in Slots) rc[venue[s],s];
   totcost <= -EPS;

  };
```

We have found that it is helpful to populate the column pool with "good" tours (relatively small distances) in addition to tours with negative reduced costs. Thus we add tours to the pool with distances within an increment of the ILB, increasing this increment iteratively. This reduces the number of times we need to refill the column pool.

As discussed above, parallel programming makes it difficult to determine the current best bound at any given time. For this reason, we use a simple depth first node selection strategy.

Rather than branching on a tour, we use higher order branching variables. Our higher order variables are the patterns indexed by team and slot. In other words, at any given node in which the master LP value is less than our current best solution, we divide the solution space into schedules in which team t is home in slot s and schedules in which team t is away in slot s. In order to select a higher order variable on which to branch, we use a strategy known as strong branching. We create a candidate list by selecting the 10 higher order variables with total weight closest to 0.5. For each variable in the candidate list, we do 50 iterations of the simplex method, setting the variable equal to 0 in one case and equal to 1 in a second case. We then select the variable with the largest total change in objective value summed over both cases. A similar strong branching strategy was used successfully in [10].

The final component of our solution methodology is a primal heuristic that makes use of an expanded version of the constraint program for the pricing problem. The primal heuristic works on the whole schedule, as opposed to just one tour, and the model is modified accordingly. Specifically, the variable arrays are expanded to include n teams, and the following constraints are added:

- No more than 2 teams can play at one venue in one slot.
- If a team plays away, its opponent must play at home.

The objective is to minimize the distances summed over all teams and slots.

Simply running this model does not produce "good" solutions quickly enough; however, by fixing certain elements we can generate solutions within 5-6 percent of the ILB in a reasonable amount of time. Specifically, we select $n/2$ teams that are each forced to play one of their optimal tours. Note this does not mean that we select $n/2$ columns to fix. A team's set of optimal tours includes all tours with minimum distance. "Fixing" a team amounts to adding a set of constraints to the model regarding trips the selected team may play. Once we have chosen a set of "fixed" teams, we run the constraint program for a specific time interval, outputing solutions and improved solutions as they are generated. Variable and value selection strategies are similar to those used in the pricing problem. We instantiate the variables associated with the fixed teams first. Domain reduction considers the inter-team constraints after the intra-team constraints and before considering objective contribution.

At the end of the set time interval, we chose a new set of four teams and run the model again. Fixed teams are chosen in order of proximity to the estimated center of the geographical region defined by the teams participating in the tournament. One processor is dedicated to running the primal heuristic. The master processor checks the primal heuristic output file at intervals that depend on the size of the instance. If a better integer solution has been found, it is sent to the slave processors along with the next nodes they are given to process.

5 Computational Results

The Traveling Tournament Problem was created to capture the essence of real sports leagues, Major League Baseball (MLB) in particular. Unfortunately, MLB has far too many teams for the current state-of-the-art for finding optimal solutions. MLB is divided into two leagues: the National League and the American League. Almost all of the games each team plays are against teams in its own league, so it is reasonable to limit analysis to an individual league.

We have generated the National League distance matrices by using "air distances" from the city centers. To generate smaller instances, we simply take subsets of the teams. In doing so, we create instances NL4, NL6, NL8, NL10, NL12, NL14 and NL16, in which the number indicates the number of teams in the instance. All of these instances are on the challenge page associated with this work: `http://mat.gsia.cmu.edu/TOURN`. In this work, however, we do not impose the "no back-to-back" games constraint, so the values here are not directly comparable to those on the web page.

Instances with $n = 4$ are nearly trivial to solve. Instances with $n = 6$ are more challenging. We have found several models that can solve these instances in a reasonable amount of time without parallel programming. When 20 processors are used to solve instances with $n = 6$, the computation time is on the order of minutes. Finally, we have found that it is necessary to use parallel programming to solve instances with $n = 8$ teams. On 20 processors, these problems take approximately 4 days.

The first table gives wall-clock times for finding and proving optimal solutions to NL4, NL6 and NL8. These problems were solved on a network of PCs with 300MHz Pentium II processors and 512MB RAM running Redhat Linux 7.1.

Name	l	u	ILB	Opt Sol'n	Procs	Time (sec)
NL4	1	3	8276	8276	1	30
NL6	1	3	22969	23552	20	912
NL8	1	3	38670	39479	20	362630

Table 1. Results for TTP Instances

Another approach for combining constraint programming and integer programming for this problem was presented in [4]. Their work was able to solve the size 6 instance but could not solve the size 8 instance.

Additionally, we have obtained the bounds for NL16 by running the constraint program from our primal heuristic on a single processor for 24 hours of computation time. We get a lower bound of 248,852 and an upper bound of 312,623.

6 Future Research

While the large increase in solution time between NL6 and NL8 leads us to believe that we will not be able to solve NL16 to optimality, we do hope to solve at least NL10. More broadly, however, we believe the framework underlying our solution methodology may be used to develop algorithms for solving a wide range of difficult timetabling and other scheduling problems with competing feasibility and optimality features.

References

1. Applegate, D. R. Bixby, V. Chvatal, and W. Cook. 1998. "On the solution of traveling salesman problems", Documenta Mathematica Journal der Deutschen Mathematiker-Verinigung International Congress of Mathemeticians, 645-656.
2. Ball, B.C. and D.B. Webster. 1977. "Optimal scheduling for even-numbered team athletic conferences", AIIE Transactions, 9, 161-169.
3. Barnhart, C., E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H.Vance. 1998. "Branch-and-Price: Column Generation for Huge Integer Programs," Operations Research, 46, 316-329.
4. Benoist, T., F. Laburthe, and B. Rottembourg, 2001. "Lagrange relaxation and constraint programming collaborative schemes for traveling tournament problems", CPAI-OR, Wye College, UK, 15-26.
5. Easton, K. "Using Integer and Constraint Programming to Solve Sports Scheduling Problems", doctoral dissertation, Georgia Institute of Technology (2002).
6. Easton, K., G.L. Nemhauser, and M.A. Trick. 2001. "The Traveling Tournament Problem: Description and Benchmarks, Principal and Practises of Constraint Programming - CP 2001, Springer Lecture Notes in Computer Science 2239, 580-585.
7. Gabow, H. N. 1990. "Data Structures for Weighted Matching and Nearest Common Ancestors with Linking," Proc. 1st Ann. ACM-SIAM Symp. On Discrete Algorithms, SIAM, Philadelphia, 434-443.
8. Henz, M. 2001. "Scheduling a Major College Basketball Conference: Revisted", Operations Research, 49, 163-168.
9. Kirkpatrick, D.G. and P, Hell. 1978. "On the Complexity of a Generalized Matching Problem," Proc. 10th Ann. ACM Symp. On Theory of Computing, Association for Computing Machinery, New York, 240-245.
10. Klabjan, D., E.L. Johnson, and G.L. Nemhauser. 2001. "Solving Large Airline Crew Scheduling Problems: Random Pairing Generation and Strong Branching," Computational Optimization and Applications, 20, 73-91.
11. Russell, R.A. and J.M Leung. 1994. "Devising a cost effective schedule for a baseball league", Operations Research, 42, 614-625.