Benders' Cuts Guided Large Neighborhood Search for the Traveling Umpire Problem

Michael A. Trick,[1] Hakan Yildiz[2]

[1] *Tepper School of Business, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213*

[2] *Eli Broad College of Business, Michigan State University, East Lansing, Michigan 48824*

Received 18 April 2009; revised 4 August 2011; accepted 10 August 2011
DOI 10.1002/nav.20482
Published online 18 October 2011 in Wiley Online Library (wileyonlinelibrary.com).

Abstract: This article introduces the use of Benders' cuts to guide a large neighborhood search to solve the traveling umpire problem, a sports scheduling problem inspired by the real-life needs of the officials of a sports league. At each time slot, a greedy matching heuristic is used to construct a schedule. When an infeasibility is recognized first a single step backtracking is tried to resolve the infeasibility. If unsuccessful, Benders' cuts are generated to guide a large neighborhood search to ensure feasibility and to improve the solution. Realizing the inherent symmetry present in the problem, a large family of cuts are generated and their effectiveness is tested. The resulting approach is able to find better solutions to many instances of this problem. © 2011 Wiley Periodicals, Inc. Naval Research Logistics 58: 771–781, 2011

Keywords: personnel scheduling; spots; benders cuts; large neighborhood search

1. INTRODUCTION AND PROBLEM DESCRIPTION

We consider a hybrid use of Benders' cuts, large neighborhood search and a greedy matching based heuristic to find good solutions to the traveling umpire problem (TUP), which was first introduced by [22] and then formally described by [23]. As shown in the subsequent sections, even finding a feasible solution to TUP is challenging, making it difficult to find good solutions in a reasonable amount of time. By combining the seemingly unrelated approaches of Benders' cuts, large neighborhood Search, and greedy heuristics, we are able to find good solutions quickly.

Like the traveling tournament problem (TTP) for league scheduling, which was introduced by Easton et al. [5], TUP is based on the most important features of a real sports scheduling problem, scheduling the umpires for major league baseball (MLB). There are 2430 games in MLB's schedule. Each game requires an umpire crew (other sports refer to these as referees). The "real" umpire scheduling problem consists of dozens of pages of constraints, including such idiosyncratic constraints as an umpire's preferred vacation dates. TUP limits the constraints to the key issues: no umpire crew should be assigned to a team too often in short succession, and every umpire crew should be assigned to every team some time in a season. Given these constraints, the primary objective is to minimize the travel of the umpires. In MLB, umpire crews are formed before the season begins and they stay together throughout the season. Thus, we do not need to worry about forming these crews while considering the scheduling problem. Because of that, in the rest of the article, we will use the word "umpire" instead of "umpire crew."

Formally, given a double round robin tournament, where every team plays against all other teams twice, on $2n$ teams ($4n - 2$ slots), we want to assign one of n umpires to each game. The constraints that need to be satisfied are:

1. Every game gets an umpire
2. Every umpire works exactly one game per slot
3. Every umpire sees every team at least once at the team's home
4. No umpire is in the same site more than once in any $n - d_1$ consecutive slots
5. No umpire sees a team more than once in any $\lfloor \frac{n}{2} \rfloor - d_2$ consecutive slots

Here, d_1 and d_2 are parameters that represent the level of constraint required. The structural results regarding the parameters in the constraints are presented in [26]. In summary, these results show that even though we do not want an umpire

Correspondence to: H. Yildiz (yildiz@msu.edu)

Table 1. Round robin tournament for four teams and a feasible umpire schedule for two umpires.

Slots	1	2	3	4	5	6
Umpire1	(1,3)	(3,4)	(1,4)	(3,1)	(4,3)	(2,3)
Umpire2	(2,4)	(1,2)	(3,2)	(4,2)	(2,1)	(4,1)

to see the same team at home frequently, there is a limit on the enforcement of this constraint even for a relaxation of TUP. This limit is at most n consecutive games for Constraint 4, in which case $d_1 = 0$. For Constraint 5, on the other hand, it is shown that the relaxation of TUP, consisting of Constraints 1,2, and 5, is always feasible when $d_2 = 0$. Setting these to d_1 and d_2 to 0 leads to the most constrained system, while setting them to n and $\lfloor \frac{n}{2} \rfloor$, respectively "turns off" the corresponding constraint.

We present an example of a round robin tournament for 4 teams and a feasible umpire schedule for $d_1 = d_2 = 0$ in Table 1. A game is represented as a pair (i, j) where i is the home team and j is the away team. Rows correspond to umpire schedules and columns correspond to games that are played in the corresponding time slots.

The literature contains many papers concerned with the scheduling of sports leagues (see the following surveys and books for more references: [1, 6, 14, 18]). However, papers concerned with the scheduling of sports officials are very few. A general referee assignment problem is considered by [4]. There is another study for scheduling umpires in the US Open, which is due to [9], and there are two papers on scheduling umpires for cricket leagues [24, 25]. There are a few studies related to scheduling umpires for MLB. References [7, 8] and [16] discuss scheduling issues and approaches for that period. A very recent paper [23] discusses more current issues and presents the method used to schedule the MLB umpires in 2006 and 2008–2010.

Our main contribution is to show how to generate Benders cuts during the execution of a simple greedy heuristic. These cuts enforce feasibility requirements that allow the greedy heuristic to avoid infeasibilities. We embed these Benders cuts in a very-large-neighborhood search method and show that by searching this neighborhood, guided by the Benders' cuts, we can obtain a good initial feasible solution which can then be improved by local search. Combining this with some problem-specific symmetry breaking leads to very good solutions to TUP instances.

The rest of the article is organized as follows: Section 2 discusses the mixed integer linear programming and constraint programming approaches to show the computational difficulty of TUP. We present the algorithms we used to solve TUP in Sections 3, 4, 5, and 6. Computational results are given in in Section 7. The conclusion is given in Section 8.

2. EXACT SOLUTION APPROACHES

TUP has many characteristics in common with the vehicle routing problem with time windows (VRPTW) due to the emphasis on minimizing total travel cost of multiple routes. In fact, if we ignore Constraints 3,4, and 5, of TUP, it becomes a special case of VRPTW. It is well known that VRP and almost all of its variants, including VRPTW, are NP-Hard [15] and exact solution approaches are ineffective in solving large instances. Since TUP has side constraints in addition to the routing constraints, it is even a more challenging problem to solve than VRP and its variants.

We formulated TUP both as a mixed integer linear program (IP) and a constraint program (CP), which are presented in the Appendix. The initial results for the IP and CP models are given in Table 2 for the smaller instances with $d_1 = d_2 = 0$, which are the settings that make the problems hardest to solve. These instances use TTP Tournaments as given in [20]. Using OPL 3.7 with the default settings, both models are able to find the optimum solutions for the 4, 6, and 8 team instances. The OPL IP solver can solve the 10 team instance to optimality in more than 13 hours, whereas the CP model is able to find a feasible solution but not the optimum solution in 24 hours. The IP model is unable to find a feasible solution for the 14 team instance, whereas the CP model is able to find a feasible solution in 24 hours. We present these initial results to demonstrate that the problem becomes very difficult to solve as we increase the number of teams. Computational results for larger instances are given in Section 7.

Table 2. IP and CP results for $d_1 = d_2 = 0$.

No. of teams	OPT distance	BEST distance		Time to prove OPT or find BEST	
		IP	CP	IP	CP
4	5176	5176	5176	0	0
6	14,077	14,077	14,077	0	0
8	34,311	34,311	34,311	2 secs	3 secs
10	48,942	48,942	49,648	13 hrs	24 hrs
12	Infeasible	—	—	—	—
14	Unknown	No solution	177,933	24 hrs	24 hrs

3. GREEDY MATCHING HEURISTIC AND A BENDERS' BASED MODIFICATION

Given the difficulty of the problem, we explore heuristic approaches to find good solutions. Our methods begin with a simple greedy heuristic which adequately handles the objective along with Constraints 1 and 2 but which runs into difficulties enforcing constraints 4 and 5. We then extend this approach through the use of logic-based Benders' cuts to handle those additional constraints.

Greedy matching heuristic (GMH) is a constructive heuristic, which builds the umpire schedules starting from Slot 1 and ending at Slot $4n - 2$. This approach is very similar to that of [7, 8] who scheduled the American League umpires for a number of years.

For every slot s, the heuristic assigns umpires to games such that all constraints, except Constraint 3, are satisfied and the best possible assignment is made to minimize the total umpire travel at Slot s. To do that, GMH solves a perfect matching problem on a bipartite graph (aka assignment problem) in every slot s. The partitions in this bipartite matching problem are the umpires and the games in slot s. An edge is placed between an umpire u and a game (i, j) if Constraints 4&5 are not violated by assigning u to game (i, j) in slot s, given the assignments from slot 1 to $s - 1$. Cost of edge $(u, (i, j)) = \text{Distance}(k, i)$. In this cost function, k is the venue that u is assigned in slot $s - 1$ and $\text{Distance}(k, i)$ is the distance between cities k and i.

In practice, GMH often gets stuck at some time slot because there may be no feasible perfect matching. In this case, we can identify a set of previous assignments that are causing this lack of perfect matching. At least one of those previous assignments must be changed in order to create a perfect matching. This set of assignments leads to a Logic-Based Benders' Cut in the terminology of [13].

While we could add the Benders' cuts to an integer programming formulation of this problem, in a standard master/subproblem approach to this problem. Instead we use these cuts to guide a large neighborhood search heuristic. Violation of Benders' cuts is penalized in the objective function with a large cost. Thus, any changes in the schedule that reduces the the number of violated Benders' cuts or reduces total umpire travel is accepted. When a solution that satisfies all the Benders' cuts is found, we stop the neighborhood search and solve the perfect matching problem for slot s again. If there is no feasible matching, we repeat the process. We explain the generation of Benders' cuts and the large neighborhood search algorithm in the next section.

3.1. Generating Benders' Cuts

Benders' cuts are generated when there is no feasible matching at a slot s during the course of GMH. Such an infeasibility implies that the partial schedule built until slot s can not be completed to obtain a feasible solution. We, then, examine the reasons for this infeasibility and this examination leads to constraints that exclude a number of partial solutions. These cuts are obtained through logical deduction. Such cuts are known as Logic-Based Benders' Cuts as defined by Hooker and Ottosson [13], where the Benders' Cuts are obtained from inference duals. Although the Benders' Cuts generated in this method are not formally obtained as classical Benders' Cuts, one can obtain the same cuts as classical Benders' cuts in an appropriate space.

The difference between the logic-based Benders' cuts and classical Benders' cuts is that no standard form exists for the logic based Benders' cuts and such cuts are generated by logical inference on the solution of the subproblem. This contrasts with classical cuts which are based on the linear programming dual solutions of the subproblem. When the subproblem is a feasibility problem, the inference dual is a condition which, when satisfied, implies that the master problem is infeasible. This condition can be used to obtain a Benders' cut for cutting off infeasible solutions. Logic based Benders' cuts are a special case of "nogoods," a well-known known idea in constraint programming literature, but they exploit problem structure in a way that nogoods generally do not [3]. Logic-based Benders' cuts have been successfully used in several studies [10–12, 17].

The Benders' cuts are generated for any set of umpires (or games) whose adjacency neighborhood has a cardinality less than the cardinality of the set itself. The adjacency neighborhood $N(A)$ of a set A is the set of nodes that are adjacent to a node in A. This condition is known as Hall's Theorem:

HALL'S THEOREM: Let $G = (V, E)$ be a bipartite graph with bipartitions X and Y. Then G has a perfect matching if and only if $|N(A)| \geq |A|$ for all $\text{A} \subseteq \text{X}$.

Because of Hall's theorem, if there is no perfect matching in a time slot, there is a subset of umpires A whose neighborhood $N(A)$ has cardinality smaller than $|A|$. We will generate a constraint that creates at least one edge between A and $Y \setminus N(A)$ as follows. For each pair x, y such that $x \in A$ and $y \in Y \setminus N(A)$ and $(x, y) \notin E$, there exists an already made game-umpire assignment in previous slots that prevents the edge to be in the matching problem. We find all such game-umpire assignments for all the missing edges between $x \in A$ and $y \in Y \setminus N(A)$. Then the corresponding Benders' Cut requires that at least one of these game-umpire assignments should be changed. These logical conditions are converted into linear inequalities, which is similar to Codato and Fischetti's approach in [2].

To obtain all possible Benders' Cuts, we identify Hall sets by checking all subsets of Umpires and their neighborhoods. For small instances, the number of cuts identified when an

Table 3. Partial schedule for eight teams and four umpires.

Slots	1	2	3	4
Umpire1	(7,5)	(2,4)	(5,7)	(2,1)
Umpire2	(1,8)	(3,6)	(4,1)	(4,5)
Umpire3	(2,6)	(1,7)	(6,8)	(6,3)
Umpire4	(4,3)	(5,8)	(3,2)	(8,7)

The first three slots are scheduled and the games for the fourth slot are in consideration for assignment.

infeasibility occurs is not too large, though for larger problems it would be necessary to generate only a limited number of cuts.

We present a partial schedule for an example instance for 8 teams and 4 umpires in Table 3. For this instance we assume that $d_1 = d_2 = 0$. Thus, the fourth constraint imposes that an umpire can not visit the same home venue more than once in any 4 consecutive games. The fifth constraint, on the other hand, imposes that an umpire can not see a team more than once in any two consecutive games. For this example the games for the first three slots are scheduled and we are considering the games in the fourth slot.

The matching problem that corresponds to slot 4 is given in Fig. 1. In this figure, set A and $N(A)$, the adjacency neighborhood of A, are circled with dashed lines. The cardinality of A is 4, whereas the cardinality of $N(A)$ is 3. Thus, there is no feasible perfect matching for this graph.

The way we obtain the Benders' Cut from set A is best demonstrated with the help of Fig. 2 and Table 4. The Benders' Cut states that at least one of the edges between the nodes in A, the umpires, and the complement of $N(A)$, Game (2,1), has to be in the graph. To write this cut in terms of the game-umpire assignments, we identify the game-umpire

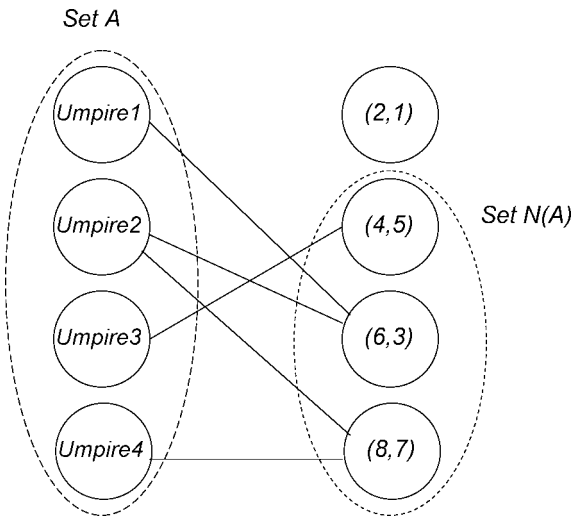

Figure 1. Bipartite matching problem for slot 4. The partitions are umpires and the games in slot 4.

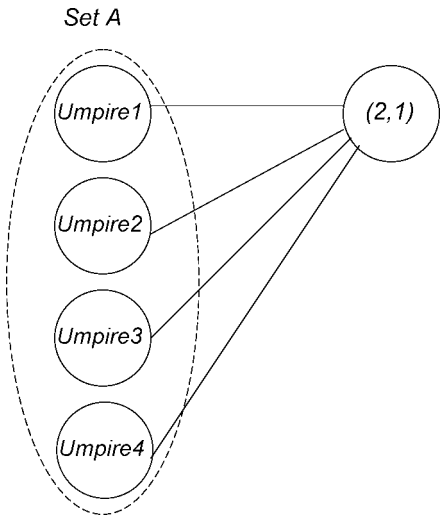

Figure 2. The missing edges between set A and the complement of N(A). Benders' cut requires at least one of these to be present in the bipartite perfect matching problem.

assignments in slots 1,2 and 3, which prevent the edges to be in the graph. For Umpire1, Game (2,4) in slot 2 is preventing the edge between Umpire1 and Game (2,1), because Umpire1 can not visit venue 2 twice in four consecutive games. Similarly, for Umpire2, Game (4,1) in slot 3 is preventing the edge between Umpire2 and Game (2,1); for Umpire3, Game (2,6) in slot 1 is preventing the edge between Umpire3 and Game (2,1); for Umpire4, Game (3,2) in slot 3 is preventing the edge between Umpire4 and Game (2,1).Thus the cut generated is

$$x_{221} + x_{432} + x_{213} + x_{334} \leq 3 \tag{1}$$

where, $x_{isu} = \begin{cases} 1, & \text{if umpire } u \text{ is at venue } i \text{ in slot } s \\ 0, & \text{otherwise.} \end{cases}$

3.2. Very Large Neighborhood Search

A neighborhood of a solution S is a set of solutions that are in some sense close to S, i.e., they can be easily computed from S or they share a significant amount of structure with S. An algorithm that starts at some initial solution and iteratively moves to solutions in the neighborhood of the current solution is called a neighborhood search algorithm or a local search algorithm.

Table 4. The bold values are the games that are in conflict with game (2,1) in slot 4.

Slots	1	2	3
Umpire1	(7,5)	**(2,4)**	(5,7)
Umpire2	(1,8)	(3,6)	**(4,1)**
Umpire3	**(2,6)**	(1,7)	(6,8)
Umpire4	(4,3)	(5,8)	**(3,2)**

The very large neighborhood search(VLNS) for TUP tries to improve the solution quality at each iteration. It is clear that the larger the neighborhood, the better is the quality of the solutions that can be reached in one single move. At the same time, the larger the neighborhood, the longer it takes to search the neighborhood at each iteration.

We introduce a very large neighborhood in the following section. We use this neighborhood in our search for a feasible solution with the use of Benders' cuts, when the greedy matching heuristic is unable to assign the games to umpires at a slot. Once an initial solution is found, we further use it to improve that solution. We stop this search when we reach a local optimum.

3.2.1. K-Umpire Neighborhood

We take the schedules of $K \leq n$ umpires and allow exchanges of game assignments within these schedules. The exchanges of game assignments are allowed only within the same slot, not across slots, as we need to make sure that Constraints 1&2 are always satisfied. In each slot, there are $K!$ different ways of assigning games to umpires. As there are $4n-2$ slots, the possible solutions that can be reached by one K-Umpire move is $(K!)^{4n-2}$.

The problem of finding the best move is done using the restricted IP for K-Umpires(RIP-U) for TUP with only the games for the K umpires in consideration. As this RIP-U is solved many times during the execution of the algorithm, the solution time should be very low to have the algorithm terminate in a reasonable amount of time. As the RIP-U becomes a very hard problem for $K \geq 4$ for even small instances, we take $K = 3$ and we look at all possible 3 combinations of the n umpires.

4. EXPLOITING THE SYMMETRY OF UMPIRES

Since umpires are essentially identical, this creates symmetry in the formulations of TUP and also allows us to create permutations of the Benders' cuts. We discuss these next.

4.1. Symmetry Breaking for IP and CP

By arbitrarily assigning the games to umpires in the first time slot, we can break the symmetry in the IP and CP models. When we tried this approach, the solution qualities for both the IP and CP models are improved and the solution times are reduced very significantly.

4.2. Permutation of Benders' Cuts

A Benders' Cut generated at a time slot s, as described in Section 3.1, may easily be satisfied by a local move. But even after satisfying all Benders' cuts, there may still be no feasible matching available at that time slot and GMH may still be stuck. To illustrate this, let's consider cut (1), which is generated by considering the infeasibility of the partial schedule presented in Table 4. We do the following changes to that partial schedule: Between Umpire 1 and 2, switch games (2,4) and (3,6) and switch games (5,7) and (4,1). This new partial schedule is presented in Table 5. This new schedule is no better than the previous schedule since there is still no feasible assignment of games to umpires in slot4, even though now (1) is satisfied:

Table 5. The umpire schedule, which is generated after a simple local move, that satisfies the Benders' cut (1).

Slots	1	2	3
Umpire1	(7,5)	(3,6)	**(4,1)**
Umpire2	(1,8)	**(2,4)**	(5,7)
Umpire3	**(2,6)**	(1,7)	(6,8)
Umpire4	(4,3)	(5,8)	**(3,2)**

$$x_{221} + x_{432} + x_{213} + x_{334} = 0 + 0 + 1 + 1 \leq 3$$

With this observation, we note that we can generate new cuts that are symmetric to (1) by permuting the decision variables in the cut between the umpires. For instance, when (1) is generated, $4! - 1 = 23$ additional cuts can be generated by permuting the first index of the variables. We generate one such cut here by switching the umpires of the first two decision variables in (1):

$$x_{222} + x_{431} + x_{213} + x_{334} \leq 3 \quad (2)$$

This cut is violated by the schedule presented in Table 5:

$$x_{222} + x_{431} + x_{213} + x_{334} = 1 + 1 + 1 + 1 > 3$$

Thus, once we include (2) in GMH, this new schedule will not be considered as better than the previous schedule as they both will be violating a cut.

5. ADDITIONAL FEATURES FOR GMH

5.1. Backtracking

When there is no feasible matching at a slot s during the course of GMH, before we generate Benders' cuts and do a local search, we first attempt to change the last game-umpire assignment to attain a feasible matching in slot s. For that, GMH backtracks to the previous slot $s-1$ and picks the second best matching, instead of the best one, and tries again for slot s. Backtracking is made at most once at each slot. This backtracking takes almost no time and may sometimes save us a significant amount of time, in search for a feasible matching at s, that would have been spend otherwise.

5.2. Incentive to Assign Unvisited Cities to Umpires

The only constraint that GMH is unable to explicitly handle is Constraint 3, because it spans the whole schedule and its

feasibility can not be ensured within a constructive heuristic such has GMH. One way of partially handling this constraint is done in the following way: While forming the matching problem at a time slot s, as described in Section 3, the cost of an assignment (u, i) is discounted by Incentive (u, i) if umpire u has never visited venue i in the previous slots. This reduction in distance guides GMH towards assigning umpires to cities that they have not visited yet, thus reducing the possibility of having a solution that violates Constraint 3 at the end of the execution of GMH. If the resulting solution violates Constraint 3, then the infeasibility is penalized in the objective function during the course of the Large Neighborhood Search Algorithm, which we explain in Section 3.2.

6. THE ALGORITHM

A high level pseudo code of GMH with Benders' cuts guided neighborhood search (GBNS) is presented in Algorithm 1. This algorithm is run multiple times for each instance until either the optimum solution is found (for instances with known optimum solutions) or until the time limit is reached. In each run a new value is used for Incentive(u, i). The values range from 0 to 100,000 and they are determined by empirical testing. The solution times of each run are added before reporting and the best solution obtained is reported.

Algorithm 1 Greedy matching heuristic with Benders' cuts guided neighborhood search (GBNS)

1. Arbitrarily assign umpires to games in slot 1
2. **for all** $1 < s \leq$ 4n-2 **do**
3. Construct a bipartite graph $G = (V, E)$ for the perfect matching problem
4. Find a minimum cost perfect matching on G
5. **if** there is no feasible perfect matching **then**
6. Backtrack to slot $s - 1$ and pick the second best matching
7. For slot s construct a bipartite graph $G = (V, E)$ and find a minimum cost perfect matching on G
8. **end if**
9. **if** there is no feasible perfect matching **then**
10. Find all Benders' Cuts
11. Set $Objective = (no\ of\ violated\ cuts) * (violation\ cost) + (total\ umpire\ sravel)$
12. Set $improvement = 1$
13. **while** at least one of the cuts is violated & $improvement > 0$ **do**
14. Do neighborhood search using 3-Ump Neighborhood
15. **if** Objective is not improved **then**
16. $improvement = 0$
17. **end if**
18. **end while**
19. **end if**
20. **end for**
21. Do neighborhood search using 3-Ump Neighborhood until either the time limit is reached or there is no further improvement.

7. COMPUTATIONAL RESULTS

We have tested the solution approaches presented in this article on a set of TUP instances and also on the 2006 MLB Schedule. We report these computational results in this section.

7.1. Instance Description

An instance of TUP has two matrices: The distance matrix, which stores the pairwise distances between cities and the Opponents Matrix, which stores the tournament information. We have used instances with 4 teams to 16 teams. The instances that have no letter but just a number in their names (e.g., 6) are original instances as given in [23], whereas the instances that has a letter in its name (e.g., 6A) are obtained by permuting team names of the original tournament. Thus, all the instances that have the same number of teams have the same tournament but a different distance matrix. This way of creating new instances actually creates quite an interesting situation somewhat linking TTP and TUP. In an optimal TTP solution, teams tend to visit close-by teams in order to minimize team travel. In the corresponding TUP solution, the umpire would also prefer to make that trip but is prohibited to do so since that would involve consecutive games seeing the same team. So the umpire is forced to visit a more distant team. In short, good TTP solutions are likely to lead to longer total umpire travel in TUP. The instances with 14 teams or fewer use TTP Tournaments as given in [20]. The 16 team instance use the distance matrix for the National Football League given in [20], and the game schedule is generated using a constraint program [19] that creates a round robin tournament. Since the 12 team instance used in [23] is infeasible, we ran the same constraint program to find a feasible 12 team instance but we were unsuccessful after generating more than 1 billion infeasible tournaments in more than 24 hours. Because of that, no 12 team instance is included in the computational results. All of the instances used in this study and additional ones are available at [21].

Depending on the choice of d_1 and d_2, the difficulty of the problem changes. Decreasing these parameters makes the problem harder to solve. Choosing a $d_1 = n-1$, which makes $n - d_1 = 1$, or a $d_2 = \lfloor \frac{n}{2} \rfloor - 1$, which makes $\lfloor \frac{n}{2} \rfloor - d_2 = 1$, simply means that Constraint 4 or Constraint 5 is not in effect.

Table 6. GBNS results for TUP instances with known optimal solutions for $d_1 = d_2 = 0$ and comparison to Improved IP, Improved CP and SA.

		Best Dist.				Time to prove OPT/find BEST			
Instance	OPT Dist.	IP	CP	SA	GBNS	IP	CP	SA	GBNS
4	5,176	5,176	5,176	5,176	5,176	0	0	0	0
6	14,077	14,077	14,077	14,077	14,077	0	0	0	0
6A	15,457	15,457	15,457	—	15,457	0	0	—	0
6B	16,716	16,716	16,716	—	16,716	0	0	—	1
6C	14,396	14,396	14,396	—	14,396	0	0	—	0
8	34,311	34,311	34,311	34,311	34,311	2 secs	2 secs	1 mins	1 mins
8A	31,490	31,490	31,490	—	31,490	1 secs	0	—	7 secs
8B	32,731	32,731	32,731		32,731	1 secs	0	—	15 secs
8C	29,879	29,879	29,879		29,879	1 secs	0	—	8 secs
10	48,942	48,942	49,595	50,196	49,528	5 mins	3 hrs	4 mins	1 mins
10A	46,551	46,551	46,927	—	46,822	23 secs	3 hrs	—	5 mins
10B	45,609	45,609	47,840	—	45,609	1 mins	3 hrs	—	1 mins
10C	43,149	43,149	43,149	—	43,533	2 hrs	3 hrs	—	5 mins

7.2. Summary of Results

We compare the results obtained by our algorithm (GBNS) with the results obtained by the integer programming (IP) and constraint programming (CP) models given in the Appendix. We also compared our results with the results obtained by a Simulated Annealing (SA) algorithm presented by [23]. ILOG OPL Studio 3.7 is used to solve the IP and CP models. We used the default settings for the IP except for the MIP emphasis. We used emphasis on feasibility rather than the

Table 7. GBNS results for TUP instances with unknown optimal solutions and comparison to Improved IP, Improved CP and SA.

				BEST Dist.			
Instance	$n - d_1$	$\lfloor \frac{n}{2} \rfloor - d_2$	Best bound	IP	CP	SA	GBNS
14	7	3	141,253	189,365	179,722	No sol.	184,117
14	6	3	141,064	182,930	183,511	180,697	176,290
14	5	3	141,134	174,859	183,470	169,173	167,146
14A	7	3	133,279	185,503	173,757	—	175,898
14A	6	3	133,194	174,070	184,385	—	172,764
14A	5	3	133,023	167,173	179,353	—	163,978
14B	7	3	131,373	191,784	173,893	—	172,302
14B	6	3	130,799	177,454	182,624	—	179,278
14B	5	3	130,628	167,236	177,805	—	160,717
14C	7	3	126,843	179,717	183,698	—	188,945
14C	6	3	126,613	166,395	186,941	—	172,243
14C	5	3	126,427	169,503	178,401	—	168,970
16	8	4	134,471	No sol.	No sol.	No sol.	No sol.
16	8	2	134,347	178,775	200,179	No sol.	179,548
16	7	3	121,933	No sol.	218,339	No sol.	193,498
16	7	2	121,670	172,064	201,121	176,527	166,274
16A	8	4	148,377	No sol.	No sol.	—	No sol.
16A	8	2	146,992	191,088	221,446	—	188,432
16A	7	3	137,178	No sol.	209,854	—	209,440
16A	7	2	137,806	190,953	222,619	—	177,857
16B	8	4	146,646	No sol.	No sol.	—	No sol.
16B	8	2	145,058	203,072	220,268	—	201,039
16B	7	3	139,833	No sol.	225,297	—	No sol.
16B	7	2	139,742	190,939	227,092	—	184,923
16C	8	4	145,012	No sol.	No sol.	—	No sol.
16C	8	2	144,398	206,796	216,845	—	202,023
16C	7	3	142,467	213,157	216,481	—	No sol.
16C	7	2	142,399	196,911	205,173	—	181,013

Table 8. Statistics about the execution of local search on 14 and 16 team instances.

Instance	$n-d_1$	$\lfloor \frac{n}{2} \rfloor - d_2$	LS initial phase Executed	LS initial phase Iterations	LS improvement Iterations
14	7	3	1	3	70
14	6	3	0	0	105
14	5	3	1	2	70
14A	7	3	4	252	105
14A	6	3	1	2	105
14A	5	3	0	0	70
14B	7	3	3	123	70
14B	6	3	1	70	105
14B	5	3	0	0	70
14C	7	3	1	16	70
14C	6	3	0	0	70
14C	5	3	0	0	70
16	8	4	—	—	—
16	8	2	6	700	56
16	7	3	2	211	224
16	7	2	1	280	112
16A	8	4	—	—	—
16A	8	2	6	747	168
16A	7	3	3	126	168
16A	7	2	0	0	392
16B	8	4	—	—	—
16B	8	2	4	616	112
16B	7	3	—	—	—
16B	7	2	2	10	280
16C	8	4	—	—	—
16C	8	2	8	1,274	224
16C	7	3	—	—	—
16C	7	2	0	0	168

default setting of balancing feasibility and optimality because finding a feasible solution for TUP can be very difficult and we observed that the new setting performs better than the default setting. Both the GA and the SA are implemented using the script language in the same software. Tests are performed on a Linux Server with an Intel(R) Xeon(TM) 3.2 GHz processor. In the following tables, empty entries for SA means that these instances were not attempted to be solved in [23].

Table 6 summarizes the results on the smallest instances with known optimum solutions for $d_1 = d_2 = 0$, which means Constraints 4 and 5 are most restricting. GBNS was able to solve all but three instances to optimality. For those three instances, the quality of the solutions found by GBNS was very close to the optimal solutions. In these instances GBNS performed better than the CP and SA.

Table 7 summarizes the results on the 14 and 16 team instances with three few different values of d_1 and d_2. We do not know the optimum solutions for these instances. We included the Best Bound found by the IP in those tables to illustrate the optimality gap. All the algorithms were allowed to run for 3 hours. At least one of the four methods was able find a feasible solution to 24 instances out of the 28 instances. Among those 24 instances, GBNS obtained the best solution for 16 instances. IP obtained the best solution for 5 instances. CP obtained the best solution for the remaining 3 instances.

For these larger instances, we also captured some statistics regarding some of the various ideas used in GBNS. This gives us a better understanding of the contribution of those ideas. Table 8 presents how often the local search (LS) procedure is executed in the initial solution construction phase and how many iterations are executed in the local search in both the initial solution construction phase and the improvement phase. Table 9 presents how often backtracking was successfully executed and how many Benders cuts are found and used, specified as original and total cuts including both the original cuts and the permutation cuts. Since the number of permutation cuts can easily become too large for larger instances, we restrict the number of permutation cuts generated for each slot by 10,000. Thus, the number of total cuts given in Table 9 are not necessarily equal to the number of all cuts that could be used, but instead it reflects the subset of cuts that have been used.

Table 9. Statistics about backtracking and Benders' cuts on 14 and 16 team instances.

Instance	$n-d_1$	$\lfloor \frac{n}{2} \rfloor - d_2$	Backtracking Success rate	Benders' cuts Original	Benders' cuts Total
14	7	3	3/4	1	2
14	6	3	2/2	0	0
14	5	3	0/1	3	72
14A	7	3	1/5	18	11,944
14A	6	3	0/1	1	6
14A	5	3	0/0	0	0
14B	7	3	0/3	14	930
14B	6	3	2/3	4	984
14B	5	3	0/0	0	0
14C	7	3	1/2	7	26
14C	6	3	1/1	0	0
14C	5	3	0/0	0	0
16	8	4	—	—	—
16	8	2	0/6	87	96,239
16	7	3	2/4	163	20,333
16	7	2	0/1	1	5040
16A	8	4	—	—	—
16A	8	2	3/9	90	78,818
16A	7	3	1/4	33	27,266
16A	7	2	2/2	0	0
16B	8	4	—	—	—
16B	8	2	1/5	15	21,562
16B	7	3	—	—	—
16B	7	2	1/3	2	5760
16C	8	4	—	—	—
16C	8	2	0/8	324	118,045
16C	7	3	—	—	—
16C	7	2	1/1	0	0

Table 10. Results for TUP on the MLB's 2006 game schedule with 30 teams.

			IP		CP		SA		GBNS	
Teams	$n-d_1$	$\lfloor \frac{n}{2} \rfloor - d_2$	Dist.	Time	Dist.	Time	Dist.	Time	Dist.	Time
30	5	5	No sol.	24 hrs	No sol.	24 hrs	581,363	5 hrs	517,143	5 hrs
30	5	5	No sol.	55 hrs	No sol.	55 hrs	—	—	494,062	55 hrs

As TUP is an abstraction of the real umpire scheduling problem for MLB, which is defined on a 30 team league and a 52 series schedule, we test our method on an instance of TUP with 30 teams and the 2006 MLB game schedule. The results are given in Table 10. We see that GBNS outperformed other methods. When we allowed the CP and GBNS to run for 55 hours, GBNS was able to improve the solution quality, whereas the CP was still unable to find a feasible solution.

8. CONCLUSION

In this article, we introduce a new approach to finding good solutions to the Traveling Umpire Problem, which is a highly constrained scheduling problem. We show that mixed integer linear programming and constraint programming models presented in this article are ineffective in solving large instances to optimality. Both models even have difficulty in finding feasible solutions for some instances.

This article integrates three seemingly unrelated streams of work: greedy heuristics, Benders' Cuts and large scale local search. This hybrid approach may be useful in similar problems where logical Benders cuts can be generated. For problems where a greedy heuristic gets "stuck" due to constraint violations, the general strategy of identifying Benders' constraints and then using large scale local search with penalties on those constraints may provide an effective approach.

While TUP's most obvious applications are in sports scheduling, there are other applications where there needs to be interaction between a resource and an existing underlying schedule. It is likely, for instance, that insights from this work would be useful in applications such as assigning judges or mediators to a series of cases or inspectors to the output of multiple production lines.

We would like to conclude by pointing out some future research directions. First, although we spent considerable amount of time trying to have better IP and CP formulations, further efforts, especially polyhedral investigation may be useful. Second, the Benders' cuts presented in this article may also be further strengthened and this can make the GBNS algorithm more effective. Third, using our algorithm as a primal heuristic within branch-and-cut may lead to better results. Finally, it would be interesting to apply the general approach to other problems where greedy heuristics have difficulty enforcing feasibility constraints.

APPENDIX A: IP FORMULATION FOR TUP

Problem Data

- S, T, U = sets of all slots, teams and umpires, respectively;
- $\text{OPP}[s,i] = \begin{cases} j, & \text{if team } i \text{ plays against team } j \text{ at venue } i \text{ in slot } s \\ -j, & \text{if team } i \text{ plays against team } j \text{ at venue } j \text{ in slot } s \end{cases}$
- d_{ij} = travel miles between venues i and j;

The following constants are defined to have a more readable model:

- $n_1 = n - d_1 - 1$
- $n_2 = \lfloor \frac{n}{2} \rfloor - d_2 - 1$;
- $N_1 = \{0, \ldots, n_1\}$;
- $N_2 = \{0, \ldots, n_2\}$;

Variables

- $x_{isu} = \begin{cases} 1, & \text{if game played at venue } i \in T \text{ in slot } s \in S \\ & \text{is assigned to umpire } u \in U \\ 0, & \text{otherwise.} \end{cases}$
- $z_{ijsu} = \begin{cases} 1, & \text{if umpire } u \text{ is at venue } i \text{ in slot } sandmovesto \\ & \text{venue } j \text{ in slot } s+1 \\ 0, & \text{otherwise.} \end{cases}$

The IP Model

$$\text{minimize} \sum_{i \in T} \sum_{j \in T} \sum_{u \in U} \sum_{s \in S: s < |S|} d_{ij} z_{ijsu}$$

subject to

$$\sum_{u \in U} x_{isu} = 1, \ \forall i \in T, s \in S : OPP[s,i] > 0 \tag{3}$$

$$\sum_{i \in T: OPP[s,i] > 0} x_{isu} = 1, \ \forall s \in S, \ u \in U \tag{4}$$

$$\sum_{s \in S: OPP[s,i] > 0} x_{isu} \geq 1, \ \forall i \in T, u \in U \tag{5}$$

$$\sum_{s_1 \in N_1} x_{i(s+s_1)u} \leq 1, \ \forall i \in T, \ u \in U, \ s \in S : s \leq |S| - n_1 \tag{6}$$

$$\sum_{s_2 \in N_2} \left(x_{i(s+s_2)u} + \sum_{k \in T: OPP[s+s_2,k] = i} x_{k(s+s_2)u} \right) \leq 1, \quad \forall i \in T, \ u \in U, \ s \in S : s \leq |S| - n_2 \tag{7}$$

$$x_{isu} + x_{j(s+1)u} - z_{ijsu} \leq 1, \ \forall i, j \in T, \ u \in U, \ s \in S : s \leq |S| \tag{8}$$

We strengthened the formulation with the following additional valid inequalities:

$$x_{isu} = 0,\ \forall\, i \in T,\ u \in U,\ s \in S : OPP[s,i] < 0 \tag{9}$$

$$z_{ijsu} - x_{isu} \le 0,\ \forall\, i,j \in T,\ u \in U,\ s \in S : s < |S| \tag{10}$$

$$z_{ijsu} - x_{j(s+1)u} \le 0,\ \forall\, i,j \in T,\ u \in U,\ s \in S : s < |S| \tag{11}$$

$$\sum_{i\in T} z_{ijsu} - \sum_{i\in T} z_{ji(s+1)u} = 0,\ \forall\, j \in T,\ u \in U,\ s \in S : s < |S| - 1 \tag{12}$$

$$\sum_{i\in T}\sum_{j\in T} z_{ijsu} = 1,\ \forall\, u \in U,\ s \in S : s < |S| \tag{13}$$

$$x_{isu},\ z_{ijsu}\ \text{binary}\ \forall\, i,j \in T,\ u \in U,\ s \in S \tag{14}$$

Here is a short description of the constraints. (3): every game must be assigned to a single umpire; (4): every umpire is assigned to exactly one game per slot; (5): every umpire sees every team at least once at the team's home; (6): no umpire should visit a venue more than once in any $n - d_1$ consecutive slots; (7): no umpire should see a team twice in any $\lfloor \frac{n}{2} \rfloor - d_2$ consecutive slots; (8): if umpire u is assigned to a game at venue i in slot s and to a game at venue j in slot $s+1$, then the umpire should move from i to j in slot s. (9): if team i plays away in slot s, no umpire can be assigned to the game at venue i; (10): if umpire u moves from venue i to venue j in slot s, it must be assigned to a game at venue i in s; (11): if umpire u moves from venue i to venue j in slot s, it must be assigned to a game at venue i in $s+1$; (12): number of umpires moving to venue j at slot s should be equal to the number of umpires moving from venue j at $s+1$; (13): every umpire must move in every slot.

APPENDIX B: CP FORMULATION FOR TUP IN OPL

Model Parameters

T = {1, . . . , 2n} is the set of teams;
S = {1, . . . , 4n − 2} is the set of slots;
U = {1, . . . , n} is the set of umpires;

$$\text{opponents[t, i]} = \begin{cases} j, & \text{if team } i \text{ plays against team } j \text{ at venue } i \\ & \text{in slot } t; \\ -j, & \text{if team } i \text{ plays against team } j \text{ at venue } j \\ & \text{in slot } t; \end{cases}$$

dist[i, j] = distance between venues i and j.

Decision Variables

team_assigned[u, t, 0] = the home team that umpire u sees in slot t.
team_assigned[u, t, 1] = the away team that umpire u sees in slot t.

The formulation in the OPL language is as follows:

```
minimize with linear relaxation
   sum (u in U, t in S: t < 4*n-2) dist[team_assigned[u,t,0],team_assigned[u,t+1,0]]

subject to {

forall(u in U, t in S)
    team_assigned[u,t,1] = opponents[t,team_assigned[u,t,0]];

//Constraints (1)&(2)
forall(t in S) {
    distribute( all(i in G) 1, all(j in T: opponents[t,j] < 0) -1*opponents[t,j],
                all(u in U) team_assigned[u,t,0]); };

//Constraint (3)
forall(u in U)
    atleast(all(i in T) 1, all(i in T) i, all(t in S) team_assigned[u,t,0]);

//Constraint (4)
forall(u in U, t in S: t<= (4*n-2)-(n-d1-1) )
   alldifferent(all(r in [0..n-d1-1]) team_assigned[u,t+r,0]);

//Constraint (5)
forall(u in U, t in S: t<= (4*n-2)-(floor(n/2)-d2-1) )
   alldifferent(all(r in [0..floor(n/2)-d2-1], y in [0..1]) team_assigned[u,t+r,y]);

search {
    forall(t in S)
      forall(u in U)
         tryall (i in T: opponents[t,i] > 0)
            team_assigned[u,t,0] = i; };
```

REFERENCES

[1] D. Briskorn, Sports leagues scheduling: Models, combinatorial properties, and optimization algorithms, Springer Verlag, Berlin, Germany, 2008.

[2] G. Codato and M. Fischetti, Combinatorial Benders' cuts for mixed-integer linear programming, Oper Res 54 (2006), 756.

[3] M.W. Dawande and J.N. Hooker, Inference-based sensitivity analysis for mixed integer/linear programming, Oper Res 48 (2000), 623–634.

[4] A.R. Duarte, C.C. Ribeiro, S. Urrutia, and E.H. Haeusler, Referee assignment in sports leagues, Lecture Notes Comput Sci 3867 (2007), 158.

[5] K. Easton, G. Nemhauser, and M. Trick, "The traveling tournament problem description and benchmarks," Seventh International Conference on the Principles and Practice of Constraint Programming (CP99), 2001, pp. 580–589.

[6] K. Easton, G.L. Nemhauser, and M.A. Trick, Sports Scheduling, Handbook of Scheduling: Algorithms, Models, and Performance Analysis, J.Y-T. Leung (Editor), CRC Press, Boca Raton, FL, 2004.

[7] J.R. Evans, A microcomputer-based decision support system for scheduling umpires in the American baseball league, Interfaces 18 (1988), 42–51.

[8] J.R. Evans, J.E. Hebert, and R.F. Deckro, Play ball!–the scheduling of sports officials, Perspect Comput: Appl Acad Sci Community 4 (1984), 18–29.

[9] A. Farmer, J.S. Smith, and L.T. Miller, Scheduling umpire crews for professional tennis tournaments, Interfaces 37 (2007), 187.

[10] I.E. Grossmann and V. Jain, Algorithms for hybrid milp/cp models for a class of optimization problems, INFORMS J Comput 13 (2001), 258–276.

[11] I. Harjunkoski and I.E. Grossmann, Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods, Comput Chem Eng 26 (2002), 1533–1552.

[12] J.N. Hooker, Planning and scheduling by logic-based Benders decomposition, Oper Res 55 (2007), 588.

[13] J.N. Hooker and G. Ottosson, Logic-based Benders decomposition, Math Program 96 (2003), 33–60.

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

[15] J.K. Lenstra and A.H.G. Rinnooy Kan, Complexity of vehicle routing and scheduling problems, Networks 11 (1981), 221–227.

[16] R.I.L.E. Ordonez, Solving the American League umpire crew scheduling problem using constraint logic programming, PhD thesis, Illinois Institute of Technology, 1997.

[17] R.V. Rasmussen and M.A. Trick, A Benders approach for the constrained minimum break problem, Eur J Oper Res 177 (2007), 198–213.

[18] R.V. Rasmussen and M.A. Trick, Round robin scheduling–a survey, Eur J Oper Res 188 (2008), 617–636.

[19] M.A. Trick, Integer and constraint programming approaches for round robin tournament scheduling, Lecture Notes Comput Sci 2740 (2003), 63–77.

[20] M.A. Trick, Challenge Traveling Tournament Instances. Availeblat at http://mat.tepper.cmu.edu/TOURN/, 2009.

[21] M.A. Trick, Traveling Umpire Problem. Available at http://mat.tepper.cmu.edu/TUP/, 2010.

[22] M.A. Trick and H. Yildiz, Bender's cuts guided large neighborhood search for the traveling umpire problem, Lecture Notes Comput Sci 4510 (2007), 332–345.

[23] M.A. Trick, H. Yildiz, and T. Yunes, Scheduling major league baseball umpires and the traveling umpire problem interfaces, (in press).

[24] M.B. Wright, Scheduling English cricket umpires, J Oper Res Soc 42 (1991), 447–452.

[25] M.B. Wright, "A rich model for scheduling umpires for an amateur cricket league," Lancaster University Management School Working Paper, 2004.

[26] H. Yildiz, Methodologies and applications for scheduling, routing and related problems, PhD thesis, Carnegie Mellon University, 2008.