European Journal of Operational Research 216 (2012) 286–292

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Locally Optimized Crossover for the Traveling Umpire Problem

Michael A. Trick [a], Hakan Yildiz [b,*]

[a] Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213, United States
[b] Eli Broad College of Business, Michigan State University, East Lansing, MI 48824, United States

ARTICLE INFO

Article history:
Received 24 June 2009
Accepted 27 July 2011
Available online 4 August 2011

Keywords:
Scheduling
Genetic algorithms
OR in sports

ABSTRACT

This paper presents a genetic algorithm (GA) to solve the Traveling Umpire Problem, which is a recently introduced sports scheduling problem that is based on the most important features of the real Major League Baseball umpire scheduling problem. In our GA, contrary to the traditional way of randomly obtaining new solutions from parent solutions, we obtain partially optimized solutions with a Locally Optimized Crossover operator. This operator also presents a link between the evolutionary mechanism on a population of solutions and the local search on a single solution. We present improved results over other methods on benchmark instances.

1. Introduction and problem description

In this paper we present a genetic algorithm (GA) to solve the Traveling Umpire Problem (TUP), a sports scheduling problem, which was first introduced by Trick and Yildiz (2007) and then formally described by Trick et al. (2011). Like the Traveling Tournament Problem for league scheduling, which was introduced by Easton et al. (2001), the TUP is based on the most important features of a real sports scheduling problem, scheduling the umpires for Major League Baseball (MLB).

The TUP extracts the most critical aspects of scheduling the umpires associated with Major League Baseball. 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 is complex and difficult to solve as it consists of dozens of pages of constraints, including such idiosyncratic constraints as an umpire's preferred vacation dates. In order to develop and test a solution approach for this problem, a testbed for experimentation is needed. Since there is only one schedule done per season, every year only one instance becomes available. Like the TTP, which has encouraged research on team scheduling approaches without getting caught up in idiosyncratic league details, the TUP provides that testbed.

The TUP limits the constraints to the key issues: no umpire should be assigned to a team too often in short succession, and every umpire should be assigned to every team some time in a season. Given these constraints, the primary objective is to minimize the total travel distances of the umpires. Unlike players, who have a home city where they play half of their games, MLB umpires travel from one city to another throughout the season. Thus, the total travel distance of an umpire is calculated by summing the distances between each consecutive city that the umpire visits throughout the season. 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 paper, we will use the word "umpire" instead of an "umpire crew".

Underlying the umpire schedule is the team schedule. For the TUP, the underlying team schedule is a double round robin tournament. There are $2n$ teams each with a home venue. The schedule lasts for $4n-2$ time slots, with each team playing one game against an opponent in each time slot. Every pair of teams plays exactly twice, once at the home venue of each team. This is known as a double round robin tournament. There are often further restrictions on the underlying team tournament. For instance, in a mirrored tournament, the schedule is divided into two halves with play in the second half identical to play in the first half except the venues are reversed. We will not assume any further structure on the underlying team schedule.

Given a team schedule for a double round robin tournament on $2n$ teams, the Traveling Umpire Problem is 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 a home 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.

* Corresponding author. Tel.: +1 517 4326439; fax: +1 517 4321112.
E-mail address: yildiz@msu.edu (H. Yildiz).

0377-2217/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2011.07.049

where d_1 and d_2 are integer parameters in the range $(0..n-1)$ and $(0..\lfloor \frac{n}{2} \rfloor - 1)$, respectively. The parameters for Constraints (4) and (5) are not chosen arbitrarily. The structural results regarding the parameters in the constraints are presented by Yildiz (2008). In summary, these results show that even though we do not want an umpire to see the same team at home frequently, there is a limit on the enforcement of this constraint even for a relaxation of the TUP, consisting of Constraint (1), (2) and (4). 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 the TUP, consisting of Constraint (1), (2) and (5), is always feasible when $d_2 = 0$. Here, d_1 and d_2 are parameters that represent the level of constraint required. Setting these 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.

Without the requirement that the underlying team schedule be a double round-robin tournament, the TUP can easily be shown to be NP-complete by reduction from the Traveling Salesman Problem. Simply have a team schedule where n teams are at home and n other teams visit them over n slots. If an umpire is not permitted to revisit a home team (but can revisit away teams as desired), then the optimal TUP solution will have all umpires travel the optimal tour through the home teams. So there is a suitable generalization of the TUP that is NP complete, and it seems unlikely that the specialization we have chosen is easy, though the complexity of the TUP as presented is open.

The literature contains many papers concerned with the scheduling of sports leagues (see the following surveys and books for more references: Kendall et al. (2010), Easton et al. (2004), Rasmussen and Trick (2008), and Briskorn (2008)). However, papers concerned with the scheduling of sports officials are very few. A general referee assignment problem is considered by Duarte et al. (2007). There is another study for scheduling umpires in the US Open, which is due to Farmer et al. (2007), and there are two papers on scheduling umpires for cricket leagues (Wright, 1991, 2004). There are a few studies related to scheduling umpires for MLB. Evans (1988), Evans et al. (1984), and Ordonez (1997) discuss scheduling issues and approaches for that period.

The rest of the paper is organized as follows: We present the algorithms we use to solve the TUP in Section 2. Computational results are given in Section 3. The conclusion is given in Section 4.

2. Solution strategy

In this study we develop a Locally Optimized Crossover operator and use it in our GA. A GA is an adaptive heuristic search method based on population genetics, and it borrows its vocabulary from that domain. The basic concepts of a GA were primarily developed by Holland (1975) and described later by Goldberg (1989) and after that many researchers contributed to the theory and application of GAs (see the following surveys and books for more references: Coello et al. (2007), Sastry et al. (2006), and Srinivas and Patnaik (1994)).

A GA uses a genetic representation for potential solutions, which are referred to as *chromosomes*. An initial set of chromosomes can be generated randomly or by using a heuristic. Each chromosome has an associated *fitness value*, which may depend partly on whether a solution satisfies all of the constraints and partly on its objective function value. The genetic operations applied to chromosomes are *mutation* and *crossover*. Traditionally, crossover operators in GAs generate offspring from two parent solutions in a random fashion and the resulting child is created without regard to fitness value. Hence the fitness of an offspring can be quite worse than the fitness of its parents. An alternative crossover approach, called *optimized crossover* is proposed by Aggarwal et al. (1997) for solving the independent set problem. In optimized crossover, genes of the parents are selected to optimize the fitness of the offspring so that the best offspring from all possible offspring is obtained. A few other studies used optimized crossover operators to solve other problems such as maximum cardinality and maximum weight clique problems (Balas and Niehaus, 1998), building phylogenetic trees (Ribeiro and Vianna, 2003), quadratic assignment problem (Ahuja et al., 2000), and bus driver scheduling problem (Lourenço et al., 2001). The potential use of optimized crossover operators to solve timetabling problems is highlighted by Meyers and Orlin (2007).

The GA we developed uses a special crossover operator that uses local optimization to generate offsprings. Thus, instead of obtaining a random solution from two parent solutions, we obtain a partially optimized solution. Since the offspring obtained is not necessarily the best possible solution that can be obtained from the two parents, we refer to this operator as *Locally-Optimized Crossover*.

2.1. Representation and genetic operators

A complete schedule of umpires is a natural representation for a solution. We present two example solutions for an eight team instance 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.

Since umpires are essentially identical, our representation scheme is inherently symmetric. Permuting the schedules among umpires yields equally good but different solutions. This symmetry is easily broken by arbitrarily assigning the games to umpires in the first time slot.

We use a crossover operator and a mutation operator to create new offspring solutions. These operators are discussed next.

2.1.1. Locally-Optimized Crossover

We pick two parent solutions and produce a new solution from them by applying a one-point crossover. The partial schedules from the parents are used as building blocks for local optimization, which then results in the final offspring.

We randomly pick a crossover slot t and copy the schedule of Parent1 for slots 1 to $t-1$ and copy the schedule of Parent2 for slots t to the last slot. Before finalizing the schedule of the offspring, we do the following: We fix the partial schedule copied from Parent1. Thus, an umpire i of the offspring has the same exact schedule as the umpire i of Parent1 for slots 1 through $t-1$. But, the rest of the schedule segments that are copied from Parent2 (from slots t through the last slot) are allowed to be exchanged among the umpires. To find the best assignment of these schedule segments, we solve a bipartite matching problem. In this matching problem, the partitions are Umpires (with fixed schedules for slots 1 to $t-1$) and partial schedule segments for slots t to the last slot. We place all edges between the partitions, which means all possible assignments are allowed. Cost of an edge between an Umpire u and a Segment s is $Distance(k,i) + Penalty(u,s)$. In this cost function, k is the venue that Umpire u is assigned in slot $t-1$ and i is the venue of the game in slot t of segment s. Thus, $Distance(k,i)$ is the distance between cities k and i. For every constraint violated by assigning Umpire u to Segment s, the cost of the edge is increased by a high penalty, which makes up the $Penalty(u,s)$ term in the cost function.

We illustrate how the crossover operator works in Table 2. The crossover operator is applied to the parent solutions in Table 1 at slot $t = 11$ to form this new solution. We copy Parent1's schedule for slots 1–10 and copy Parent2's schedule for slots 11–14. To finalize the new schedule we solve a matching problem on a bipartite graph as illustrated in Fig. 1. For this instance we assume that

Table 1
Two feasible solutions for eight teams and four umpires.

Slots	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Parent1														
Umpire 1	(1,5)	(2,8)	(5,6)	(3,7)	(8,1)	(4,6)	(7,5)	(2,4)	(6,8)	(4,5)	(3,1)	(8,6)	(1,3)	(4,2)
Umpire 2	(4,8)	(5,3)	(1,4)	(8,2)	(6,5)	(2,3)	(1,8)	(3,6)	(5,7)	(2,1)	(7,6)	(3,5)	(6,4)	(8,3)
Umpire 3	(6,2)	(4,7)	(3,8)	(5,4)	(7,2)	(8,5)	(2,6)	(5,8)	(4,1)	(6,3)	(8,4)	(1,2)	(7,8)	(5,1)
Umpire 4	(7,3)	(1,6)	(2,7)	(6,1)	(3,4)	(7,1)	(4,3)	(1,7)	(3,2)	(8,7)	(2,5)	(7,4)	(5,2)	(6,7)
Parent2														
Umpire 1	(1,5)	(2,8)	(5,6)	(8,2)	(3,4)	(7,1)	(4,3)	(1,7)	(3,2)	(8,7)	(2,5)	(7,4)	(5,2)	(6,7)
Umpire 2	(4,8)	(5,3)	(1,4)	(3,7)	(6,5)	(2,3)	(1,8)	(3,6)	(5,7)	(2,1)	(7,6)	(1,2)	(6,4)	(8,3)
Umpire 3	(6,2)	(4,7)	(3,8)	(5,4)	(8,1)	(4,6)	(7,5)	(2,4)	(6,8)	(4,5)	(3,1)	(8,6)	(1,3)	(4,2)
Umpire 4	(7,3)	(1,6)	(2,7)	(6,1)	(7,2)	(8,5)	(2,6)	(5,8)	(4,1)	(6,3)	(8,4)	(3,5)	(7,8)	(5,1)

Table 2
New solution is obtained by crossover at slot 11 and optimized by solving a matching problem at the crossover slot.

Slots	1	2	3	4	5	6	7	8	9	10		11	12	13	14
	Before optimization											Crossover at slot = 11			
Umpire 1	(1,5)	(2,8)	(5,6)	(3,7)	(8,1)	(4,6)	(7,5)	(2,4)	(6,8)	(4,5)	‖	(2,5)	(7,4)	(5,2)	(6,7)
Umpire 2	(4,8)	(5,3)	(1,4)	(8,2)	(6,5)	(2,3)	(1,8)	(3,6)	(5,7)	(2,1)	‖	(7,6)	(1,2)	(6,4)	(8,3)
Umpire 3	(6,2)	(4,7)	(3,8)	(5,4)	(7,2)	(8,5)	(2,6)	(5,8)	(4,1)	(6,3)	‖	(3,1)	(8,6)	(1,3)	(4,2)
Umpire 4	(7,3)	(1,6)	(2,7)	(6,1)	(3,4)	(7,1)	(4,3)	(1,7)	(3,2)	(8,7)	‖	(8,4)	(3,5)	(7,8)	(5,1)
	After optimization														
Umpire 1	(1,5)	(2,8)	(5,6)	(3,7)	(8,1)	(4,6)	(7,5)	(2,4)	(6,8)	(4,5)	‖	(3,1)	(8,6)	(1,3)	(4,2)
Umpire 2	(4,8)	(5,3)	(1,4)	(8,2)	(6,5)	(2,3)	(1,8)	(3,6)	(5,7)	(2,1)	‖	(7,6)	(1,2)	(6,4)	(8,3)
Umpire 3	(6,2)	(4,7)	(3,8)	(5,4)	(7,2)	(8,5)	(2,6)	(5,8)	(4,1)	(6,3)	‖	(8,4)	(3,5)	(7,8)	(5,1)
Umpire 4	(7,3)	(1,6)	(2,7)	(6,1)	(3,4)	(7,1)	(4,3)	(1,7)	(3,2)	(8,7)	‖	(2,5)	(7,4)	(5,2)	(6,7)

$d_1 = d_2 = 0$. Thus, *Constraint* (4) imposes that an umpire cannot visit the same home venue more than once in any four consecutive games. *Constraint* (5), on the other hand, imposes that an umpire cannot see a team more than once in any two consecutive games. In this figure, the solid edges are feasible assignments, whereas the dashed edges are penalized for their constraint violations. For instance, the edge between Umpire 1 and the first segment ((2,5),(7,4),(5,2),(6,7)) has a cost equal to *Distance*(4,2) + *Penalty* (*Umpire*1,*Segment*1), where *Distance*(4,2) is the distance between cities 4 and 2, which are the venues Umpire 1 would be visiting in slots 10 and 11, respectively. There are two constraint violations in this assignment that are penalized. First, Umpire 1 would visit city 2 in both slots 8 and 11, which violates *Constraint* (4). Second, Umpire 1 would see team 5 in slots 10 and 11, which violates *Constraint* (5). *Constraint* (3) is not violated as Umpire 1 will be visiting all the cities in the new solution.

The new offspring generated is the one illustrated in the lower part of Table 2. Fortunately, we were able to obtain a feasible matching that consists of only solid edges in Fig. 1. The total cost of the parent solutions are 34,432 and 35,097, respectively and the cost of the new solution after optimization is 34,773. Before optimization, the new solution would be an infeasible solution violating the constraints several times.

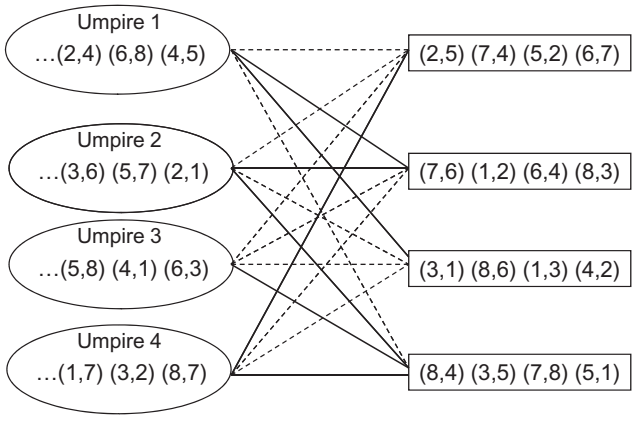

Fig. 1. The matching problem that is solved. The solid edges are feasible assignments, whereas the dashed lines are penalized for their constraint violations.

2.1.2. Mutation

We randomly select two games in a randomly selected slot t and flip the umpires assigned to these games. The mutation operator introduces a certain amount of randomness to the search. It can help identify solutions that crossover alone might not.

2.2. Initial population of solutions

The initial set of solutions are formed by using a Probabilistic Greedy Matching Heuristic (PGM), which builds the umpire schedules starting from the first slot and ending at the last slot. This approach is very similar to that of Evans (1988), Evans et al. (1984), who scheduled the American League umpires for a number of years.

For every slot t, the heuristic assigns umpires to games such that all constraints, except Constraint 3, are satisfied and games in that slot are assigned to umpires using either the best or second best matching with equal probability. This probabilistic assignment allows us to form a diverse set of quality solutions.

To make the assignments, PGM solves a perfect matching problem on a bipartite graph in every slot t. This bipartite graph has the *umpires* on one side and the *games* of slot t on the other side of the partition. An edge is placed between an umpire u and a game (i,j) if Constraints (4) and (5) are not violated by assigning u to game (i,j) in slot t, given the assignments from slot 1 to $t-1$. Cost of an edge $(u,(i,j)) = Distance(k,i)$. In this cost function, k is the venue that u is assigned in slot $t-1$ and $Distance(k,i)$ is the distance between cities k and i.

We find the second best matching by first solving the matching problem and finding the optimum solution. We then resolve the matching problem, this time by including a constraint that forces

the new solution to have a strictly greater objective function value than the cost of the best matching.

If in a slot t, there is no feasible matching available, we backtrack to the previous slot $t-1$ and switch our initial choice (from best to second best or vice versa), and try again. Backtracking is made at most once at each slot.

In the initial population, umpires are assigned to the same games in the first slot in every solution. After that in each slot, either the best matching or second best matching is selected with equal probability. This inherently creates an imbalance of variety among the slots in the population: there are only two different columns for slot 2 in the whole population after initialization, for column 3 there may be up to 4, for column 4 there may be up to 8 and so forth. Moreover, in the current form of the crossover operator, the slots after the crossover point are exchanged among umpires. The slots before the crossover point are fixed. This also inherently may add to the already existing potential imbalance of variety. With this observation, we tried to remove these potential imbalances by choosing randomly whether we exchange rows in the range of slots before or after the crossover point. Unfortunately, in general this did not help. Although it helped with a few instances presented in Section 3, the original method performed better with the rest of the instances. It appears that the mixing provided by the optimized crossover along with the mutation operator is sufficient to keep enough variety, even in the early slots that lack such variety.

2.3. Evaluation function

Clearly, PGM does not guarantee to find feasible solutions. Because of that constraint violations are penalized and reflected in the fitness of a solution. We use an evaluation function that consists of two parts. One is the total travel distance of umpires. The other is the penalties for the constraint violations. Each time a pair of games violates a constraint, a fixed penalty cost is incurred. Thus, more violations make a solution more inferior, thus such solutions are either improved during the course of the evolutionary algorithm by fixing these violations or naturally eliminated from the population by the selection process.

2.4. The algorithm

1. Set *time* = 0.
2. Form $N_{initial}$ solutions to form the set *CurrentSolutions*(*time*).
3. While the *Stopping Criteria* is not met do:
   (a) Set *time* = *time* + 1.
   (b) Create $N_{initial}/2$ new solutions by:
      i. Randomly select two solutions from the current population of solutions.
      ii. Randomly select a crossover slot.
      iii. Apply the crossover on the two selected solutions to create a new solution.
      iv. If the new solution created exists in *TempSolutions* (*time*) or in *CurrentSolutions*(*time* − 1), discard it and goto Step 3(b). Otherwise add the new solution into the set *TempSolutions*(*time*) .

   (c) Replace the inferior solutions in *CurrentSolutions*(*time* − 1) with the superior solutions in *TempSolutions*(*time*) and create *CurrentSolutions*(*time*).
   (d) For every solution S in *CurrentSolutions*(*time* − 1) apply Mutation to S with *MutationProbability*.

The parameters of the GA have been set empirically (by trying out a few different alternatives on some of the instances): We used an initial population size $N_{initial} = 500$. At each generation we created 250 new solutions by applying the Crossover Operator to the current solutions. We used *MutationProbability* = 0.05. We allowed the algorithm to run for 3 h, except for the 30 team instance, which is run for 5 h. The algorithm is stopped if either the optimum solution is found, which can only happen for instances with known optimal solutions, or the time limit is reached.

3. Computational results

We now present and discuss computational results for the GA we presented in this paper. We have performed our tests on a set of TUP instances and also on the 2006 MLB Schedule.

3.1. Instance description

An instance of the 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 a number of teams ranging from 4 to 16. The instances that have no letter but just a number in their names (e.g. 6) are original instances as given in Trick et al. (2011), whereas the instances that have a letter in their names (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 the TTP and the 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 the TTP Tournaments as given in Trick (2009). The 16 team instance use the distance matrix for the National Football League given in Trick (2009), and the game schedule is generated using a constraint program (Trick, 2003) that creates a round robin tournament. Since the 12 team instance used in Trick et al. (2011) 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 h. 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 http://mat.tepper.cmu.edu/TUP.

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.

3.2. Summary of results

We compare the results obtained by the GA with the results obtained by the Integer Programming (IP) and Constraint Programming (CP) models given in the Appendices. We also compared our results with the results obtained by a Simulated Annealing (SA) algorithm presented by Trick et al. (2011). 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 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 pro-

Table 3
GA results for TUP instances with known optimal solutions for $d_1 = d_2 = 0$ and comparison to IP, CP and SA.

Instance	OPT distance	Best distance				Time to prove OPT/find BEST			
		IP	CP	SA	GA	IP	CP	SA	GA
4	5176	5176	5176	5176	5176	0	0	0	0
6	14,077	14,077	14,077	14,077	14,077	0	0	0	1 seconds
6A	15,457	15,457	15,457	–	15,457	0	0	–	1 seconds
6B	16,716	16,716	16,716	–	16,716	0	0	–	1 seconds
6C	14,396	14,396	14,396	–	14,396	0	0	–	1 seconds
8	34,311	34,311	34,311	34,311	34,311	2 seconds	2 seconds	1 minutes	5 seconds
8A	31,490	31,490	31,490	–	31,490	1 seconds	0	–	12 seconds
8B	32,731	32,731	32,731	–	32,731	1 seconds	0	–	5 seconds
8C	29,879	29,879	29,879	–	29,879	1 seconds	0	–	4 seconds
10	48,942	48,942	49,595	50,196	48,942	5 minutes	3 h	4 minutes	3 minutes
10A	46,551	46,551	46,927	–	46,632	23 minutes	3 h	–	2 minutes
10B	45,609	45,609	47,840	–	45,609	1 minutes	3 h	–	2 minutes
10C	43,149	43,149	43,149	–	43,149	2 hours	3 h	–	20 minutes

Table 4
GA results for TUP instances with unknown optimal solutions for $d_1 = d_2 = 0$ and comparison to IP, CP and SA.

Instance	Best bound	Best distance			
		IP	CP	SA	GA
14	141,253	189,365	179,722	No sol.	186,890
14A	133,279	185,503	173,757	–	175,487
14B	131,373	191,784	173,893	–	172,514
14C	126,843	179,717	183,698	–	175,536
16	134,471	No sol.	No sol.	No sol.	No sol.
16A	148,377	No sol.	No sol.	–	No sol.
16B	146,646	No sol.	No sol.	–	No sol.
16C	145,012	No sol.	No sol.	–	No sol.

cessor. In the following tables, empty entries for SA means that (Trick et al., 2011) did not attempt to solve those instances.

Table 3 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. The GA was able to solve all but one instances to optimality. It was unable to solve the 10A instance to optimality but the quality of the solution found by the GA is very close to the optimal solution and it is better than the solution found by the CP.

Table 4 summarizes the results on the 14 and 16 team instances with $d_1 = d_2 = 0$. 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. For the 14 team instances, the GA was able to find feasible solutions better than the ones found by the IP. For 2 of those instances, CP was able to find the best solution and for the remaining 2, GA found the best solutions. None of the four methods were able to find a feasible solution for the 16 team instances.

Based on the results we obtained using the different methods presented, it is clear that the 14 and 16 team instances are very difficult instances to solve when we have $d_1 = d_2 = 0$. To further investigate the GA's performance on the TUP, we solve the relaxations of those instances by increasing the values of d_1 and d_2. The results for these relaxations are given in Table 5. We also compare the results to the results obtained by the IP, CP and SA. For all 14 team relaxation instances, we see that the GA obtains a better solution than all three other methods. For the 16 team instances, GA was able to obtain feasible solutions for all but one instance. For 7 out of 12 instances, GA obtained the best solution. CP was able to find a feasible solution for all of the instances, but none of them are the best solution. IP was able to find a feasible solution to all but three instances. For 5 out of 12 instances, IP obtained the best solution. SA was able to find a feasible solution to one of the three 16 team instances that SA attempted to solve.

Table 5
Results for GA on relaxations of 14 and 16 team instances, where $d_1 + d_2 > 0$, and comparison to results of the IP, CP and SA.

Instance	$n - d_1$	$\lfloor \frac{n}{2} \rfloor - d_2$	Best bound	Best distance			
				IP	CP	SA	GA
14	6	3	141,064	182,930	183,511	180,697	173,681
14	5	3	141,134	174,859	183,470	169,173	165,558
14A	6	3	133,194	174,070	184,385	–	164,857
14A	5	3	133,023	167,173	179,353	–	162,380
14B	6	3	130,799	177,454	182,624	–	168,476
14B	5	3	130,628	167,236	177,805	–	160,443
14C	6	3	126,613	166,395	186,941	–	171,788
14C	5	3	126,427	169,503	178,401	–	163,662
16	8	2	134,347	178,775	200,179	No sol.	No sol.
16	7	3	121,933	No sol.	218,339	No sol.	185,966
16	7	2	121,670	172,064	201,121	176,527	166,114
16A	8	2	146,992	191,088	221,446	–	194,639
16A	7	3	137,178	No sol.	209,854	–	199,016
16A	7	2	137,806	190,953	222,619	–	172,728
16B	8	2	145,058	203,072	220,268	–	208,539
16B	7	3	139,833	No sol.	225,297	–	202,395
16B	7	2	139,742	190,939	227,092	–	185,005
16C	8	2	144,398	206,796	216,845	–	209,967
16C	7	3	142,467	213,157	216,481	–	216,896
16C	7	2	142,399	196,911	205,173	–	181,385

Table 6
Results for TUP on the MLB's 2006 game schedule with 30 teams.

Teams	$n-d_1$	$\lfloor \frac{n}{2} \rfloor - d_2$	IP		CP		SA		GA	
			Distance	Time	Distance	Time	Distance	Time	Distance	Time
30	5	5	No sol.	24 h	No sol.	24 h	581,363	5 h	483,224	5 h

As stated before, TUP is an abstraction of the real umpire scheduling problem for MLB, which is defined on a 30 team league and a 52 week game schedule. To resonate the two, we test our method on an instance of TUP with 30 teams and the 2006 MLB game schedule. For this instance, we needed to make a small modification in the way we formed our initial population by the GA. Because of the increase in the size of the problem, finding the second best matching as described in Section 2.2 consumes too much time. To speed up the algorithm, we instead look for a good matching that is different than the best matching. We do this by simply forbidding the most expensive edge in the optimal matching solution and solving the resulting assignment problem. This reduces the computational time very significantly and still yields a good and diverse initial population. The results are given in Table 6. We see that GA outperformed both IP and CP on this instance. GA also found a much better solution than SA.

4. Conclusion

In this paper, we introduce a new approach that uses a locally optimized crossover operator to find good solutions to the Traveling Umpire Problem, which is a highly constrained scheduling problem.

We demonstrate that the GA is simple to implement and provides very good results in very short time. The crossover operator we use can also be used as a local search operator on a single solution. This fact is an interesting resemblance between two different solution methods.

There is no doubt that other approaches may also work well on the TUP. The instances we used are available at http://mat.tepper.cmu.edu/TUP along with the best solutions and lower bounds to date.

Appendix A. IP formulation for the TUP

A.1. Problem data

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

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

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

A.2. Variables

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

A.3. The IP model

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

subject to

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

$$\sum_{i\in T: OPP[s,i]>0} x_{isu} = 1, \quad \forall\, s \in S,\ u \in U \tag{2}$$

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

$$\sum_{s_1\in N_1: OPP[s+s_1,i]>0} x_{i(s+s_1)u} \leqslant 1, \quad \forall\, i \in T,\ u \in U,\ s \in S : s \leqslant |S| - n_1 \tag{4}$$

$$\sum_{s_2\in N_2} \left(x_{i(s+s_2)u} + \sum_{k\in T: OPP[s+s_2,k]=i} x_{k(s+s_2)u} \right) \leqslant 1, \quad \forall i \in T,\ u \in U,\ s \in S : s \leqslant |S| - n_2 \tag{5}$$

$$x_{isu} + x_{j(s+1)u} - z_{ijsu} \leqslant 1, \quad \forall\, i,j \in T,\ u \in U,\ s \in S : s \leqslant |S| \tag{6}$$

We strengthened the formulation with the following additional valid inequalities:

$$x_{isu} = 0, \quad \forall\, i \in T,\ u \in U,\ s \in S : OPP[s,i] < 0 \tag{7}$$

$$z_{ijsu} - x_{isu} \leqslant 0, \quad \forall\, i,j \in T,\ u \in U,\ s \in S : s < |S| \tag{8}$$

$$z_{ijsu} - x_{j(s+1)u} \leqslant 0, \quad \forall i,j \in T,\ u \in U,\ s \in S : s < |S| \tag{9}$$

$$\sum_{i\in T} z_{ijsu} - \sum_{i\in T} z_{ji(s+1)u} = 0, \quad \forall j \in T,\ u \in U,\ s \in S : s < |S| - 1 \tag{10}$$

$$\sum_{i\in T}\sum_{j\in T} z_{ijsu} = 1, \quad \forall u \in U,\ s \in S : s < |S| \tag{11}$$

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

Appendix B. CP formulation for the 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;

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

$d[i,j]$ = distance between venues i and j.

Decision variables

team_assigned$[u,s,0]$ = the home team that umpire u sees in slot s.
team_assigned$[u,s,1]$ = the away team that umpire u sees in slot s.
The formulation in the OPL language is as follows:

```
minimize with linear relaxation
  sum(u in U, s in S: s < 4 * n - 2)d[team_assigned
  [u,s,0], team_assigned[u,s + 1,0]]
subject to {
forall(u in U, s in S)
  team_assigned[u,s,1] = OPP[s,team_assigned
  [u,s,0]];
//Constraints(1)& (2)
forall (s in S) {
  distribute (all (i in G) l, all (j in T:OPP[s,j]
   < 0) - l * OPP[s,j],
      all (u in U) team_assigned[u,s,0]);};
//Constraint (3)

forall (u in U)
      at least(all (i in T) l, all (i in T) i, all
  (s in S) team_assigned[u,s,0]); //Constraint (4)

forall (u in U, s in S: s <= (4 * n - 2) - (n - dl - 1))
      all different (all (r in [0..n - dl - 1])
  team_assigned[u,s + r,0]);
//Constraint (5)

forall (u in U, s in S: s <= (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,s + r,y]);
  search {
  forall (s in S)
    forall (u in U)
      tryall (i in T: OPP[s,i] > 0)
        team_assigned[u,s,0] = i;};
```

References

Aggarwal, C.C., Orlin, J.B., Tai, R.P., 1997. Optimized crossover for the independent set problem. Operations Research, 226–234.

Ahuja, R.K., Orlin, J.B., Tiwari, A., 2000. A greedy genetic algorithm for the quadratic assignment problem. Computers and Operations Research 27 (10), 917–934.

Balas, E., Niehaus, W., 1998. Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems. Journal of Heuristics 4 (2), 107–122.

Briskorn, D., 2008. Sports Leagues Scheduling: Models, Combinatorial Properties, and Optimization Algorithms. Springer Verlag.

Coello, C.A.C., Lamont, G.B., Van Veldhuizen, D.A., 2007. Evolutionary algorithms for solving multi-objective problems. Springer-Verlag New York Inc..

Duarte, A.R., Ribeiro, C.C., Urrutia, S., Haeusler, E.H., 2007. Referee assignment in sports leagues. Lecture Notes in Computer Science 3867, 158.

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

Easton, K., Nemhauser, G., Trick, M., 2004. Sports Scheduling. Handbook of Scheduling: Algorithms, Models, and Performance Analysis

Evans, JR., 1988. A microcomputer-based decision support system for scheduling umpires in the American baseball league. Interfaces 18 (6), 42–51.

Evans, J.R., Hebert, J.E., Deckro, R.F., 1984. Play Ball! – the scheduling of sports officials. Perspectives in Computing: Applications in the Academic and Scientific Community 4 (1), 18–29.

Farmer, A., Smith, J.S., Miller, L.T., 2007. Scheduling umpire crews for professional tennis tournaments. Interfaces 37 (2), 187.

Goldberg, D.E., 1989. Genetic Algorithms in Search. Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.

Kendall, G., Knust, S., Ribeiro, C.C., Urrutia, S., 2010. Scheduling in sports: an annotated bibliography. Computers and Operations Research 37 (1), 1–19.

Lourenço, H.R., Paixão, J.P., Portugal, R., 2001. Multiobjective metaheuristics for the bus driver scheduling problem. Transportation Science 35 (3), 331.

Meyers, C., Orlin, J.B., 2007. Very large-scale neighborhood search techniques in timetabling problems. Lecture Notes in Computer Science 3867, 24.

Ordonez, R.I.L.E., 1997. Solving the American League umpire crew scheduling problem using constraint logic programming. Ph.D. Thesis, Illinois Institute of Technology.

Rasmussen, R.V., Trick, M.A., 2008. Round robin scheduling – a survey. European Journal of Operational Research 188 (3), 617–636.

Ribeiro, C.C., Vianna, D.S., 2003. A genetic algorithm for the phylogeny problem using an optimized crossover strategy based on path-relinking. In: Proceedings of 2nd Brasil Workshop on Bioinformatics, pp. 97–102.

Sastry, K., Goldberg, D., Kendall, G., 2006. Genetic algorithms. 2005. Introductory Tutorials in Optimisation, Decision Support and Search Methodology. ISBN 387234608 97-125.

Srinivas, M., Patnaik, L.M., 1994. Genetic algorithms: a survey. Computer 27 (6), 17–26.

Trick, M.A., 2003. Integer and constraint programming approaches for round robin tournament scheduling. Lecture Notes in Computer Science 2740, 63–77.

Trick, M.A., 2009. Challenge Traveling Tournament Instances, <http://mat.tepper.cmu.edu/TOURN/>.

Trick, M.A., Yildiz, H., 2007. Bender's cuts guided large neighborhood search for the Traveling Umpire Problem. Lecture Notes in Computer Science 4510, 332–345.

Trick, M.A., Yildiz, H., Yunes, T., 2011. Scheduling major league baseball umpires and the traveling umpire problem. Interfaces. doi:10.1287/inte.1100.0514. <http://interfaces.journal.informs.org/content/early/2011/06/01/inte. 1100.0514.abstract>.

Wright, M.B., 1991. Scheduling English cricket umpires. Journal of the Operational Research Society 42 (6), 447–452.

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

Yildiz, H., 2008. Methodologies and applications for scheduling, routing and related problems. Ph.D. Thesis, Carnegie Mellon University.