Branch-and-Price and Improved Bounds to the Traveling Umpire Problem

**T´ulio A. M. Toffolo · Sam Van Malderen·
Tony Wauters · Greet Vanden Berghe**

Abstract The present paper proposes a branch-and-price approach to the Traveling Umpire Problem (TUP). In this hard combinatorial optimization problem, umpires (or referees) have to be assigned to games in a double round robin tournament. The objective is to obtain a solution with minimum total travel distance over all umpires, while respecting several hard constraints. Dantzig-Wolfe decomposition is applied to an existing Integer Programming formulation to be used in a branch-and-price framework. The pricing problems are solved using a specialized branch-and-bound algorithm, which applies multiple pruning techniques. Two branching strategies (best-first and depth-first) were employed and result in many improved lower bounds compared to the previous best known. In addition, five new best solutions were found and four instances with 16 teams were proven to be infeasible.

Keywords Traveling Umpire Problem· Branch and Price· Column Generation· Decomposition Strategies · Integer Programming

1 Introduction

The Traveling Umpire Problem (TUP) is a sports timetabling problem that considers the assignment of n umpires (or referees) to games in a double round robin tournament (e.g. a baseball championship). The tournament schedule is given as input with $4n - 2$ rounds (or slots), where the $2n$ teams play twice against each other; once in their home venue and once away. The objective is to minimize the total travel distance of all umpires. In order to obtain a fair assignment, several hard constraints are imposed:

a) every game in the tournament is officiated by exactly one umpire.

T´ulio A. M. Toffolo[1,2] · Sam Van Malderen[1] · Tony Wauters[1] · Greet Vanden Berghe[1]
1 KU Leuven, Department of Computer Science, CODeS & iMinds-ITEC
2 Federal University of Ouro Preto - Brazil, Department of Computing
Emails: tulio.toffolo@kuleuven.be, sam.vanmalderen@cs.kuleuven.be, tony.wauters@cs.kuleuven.be and greet.vandenberghe@cs.kuleuven.be

b) every umpire must work in every round.
c) every umpire should visit the home of every team at least once.
d) no umpire is in the same venue more than once in any $q_1 = n - d_1$ consecutive rounds.
e) no umpire officiates a game with the same team more than once in any $q_2 = \binom{n}{2} - d_2$ consecutive rounds (this constraint is similar to the previous one, but also takes the 'away team' into consideration).

The values d_1 and d_2 range from 0 to n and 0 to $\binom{n}{2}$, respectively.

The Traveling Umpire Problem (TUP) was first introduced by Trick and Yildiz (2007). Their work was extended by Trick and Yildiz (2011), where a Benders cuts guided large neighborhood search is proposed. These papers also provide an Integer Programming (IP) and Constraint Programming (CP) formulation for the problem. A greedy matching heuristic and a simulated annealing approach with a two-exchange neighbourhood are described by Trick et al (2012). Trick and Yildiz (2012) present a Genetic Algorithm (GA) with a locally optimized crossover procedure. A stronger IP formulation and a relax-and-fix heuristic are proposed in (de Oliveira et al, 2013), which improve both lower and upper bounds. Wauters et al (2014) present an enhanced iterative deepening search with leaf node improvements (IDLI), an iterated local search (ILS) and a new decomposition based lower bound methodology. Many improved solutions and lower bounds were found.

In this work, we present a branch-and-price approach to the TUP. By applying the Dantzig-Wolfe decomposition on an existing formulation of the problem, we obtain a Restricted Master Problem (RMP) and pricing subproblems. The RMP is a set partition problem, and its relaxation can be solved by linear programming algorithms such as Simplex. The pricing subproblems are solved by a specialized branch-and-bound. The branch-and-price can be seen as a branch-and-bound employing a column generation scheme to solve the relaxation in each node. We considered two branching and node selection strategies, one for improving the lower bounds and another for obtaining feasible solutions.

The following section presents the formulation introduced by de Oliveira et al (2013) for the TUP. Section 3 details the reformulation of the original model. The strategies considered in the branch-and-price framework are discussed in Section 4. Section 5 presents computational experiments considering both lower and upper bounds and, finally, Section 6 summarizes the conclusions and proposes future work.

2 Integer Programming Formulation for the TUP Problem

The first formulation for the TUP was proposed by Trick and Yildiz (2007). This formulation was then improved by de Oliveira et al (2013). We apply the Dantzig-Wolfe decomposition on the latter model. Following, we present this formulation. For that, consider the following input data:

U : set of umpires, such that $U = \{1, ..., n\}$;
T : set of teams, such that $T = \{1, ..., 2n\}$;
R : set of rounds, such that $R = \{1, ..., 4n - 2\}$;

$H(r)$: set of teams i hosting a game in round r;
$\delta(i)$: set of rounds $r \in R$ in which the team i is hosting a game;
$A(i, r)$: function that returns the team playing against team i in round r;
d_{ij} : distance between the home of teams i and j;
$CV(r)$: set of rounds $\{r,\ldots,r + q_1 - 1\} \in R$ defined for $r \in \{1,\ldots, |R| - q_1 - 1\}$;
$CT(r)$: set of rounds $\{r,\ldots,r + q_2 - 1\} \in R$ defined for $r \in \{1,\ldots, |R| - q_2 - 1\}$;

The decision variables are:

$$x(i, j, r, u) = \begin{cases} 1 \text{ if umpire } u \text{ is assigned to venue } i \text{ in round } r \text{ and to } j \text{ in round } r + 1 \\ 0 \text{ otherwise} \end{cases}$$

The formulation is presented by constraints (1)-(9).

$$\min \sum_{i \in T} \sum_{j \in T} \sum_{r \in R} \sum_{u \in U} d_{ij} x_{ijru} \quad (1)$$

$$\text{s.t.} \sum_{u \in U} \sum_{j \in T} x_{ijru} = 1 \quad \forall i \in T, r \in \delta(i) \quad (2)$$

$$\sum_{i \in H(r)} \sum_{j \in T} x_{ijru} = 1 \quad \forall r \in R, u \in U \quad (3)$$

$$\sum_{r \in \delta(i)} \sum_{j \in T} x_{ijru} \geq 1 \quad \forall i \in T, u \in U \quad (4)$$

$$\sum_{c \in CV(r): c \in \delta(i)} \sum_{j \in T} x(i,j, c, u) \leq 1 \quad \forall i \in T, u \in U, r \in R(:5) \quad \checkmark \quad r < |R| - q_1 - 1$$

$$x(i,j, c, u) + \sum_{c \in CT(r)} \sum_{j \in T} x(k, j, c,u) \leq 1 \quad \forall i \in T, u \in U, r \in R(:6) \quad \checkmark \quad r < |R| - q_2 - 1$$

where $k \in T$ otherwise

$$\sum_{j \in T} x(i, j, r, u) = 0 \quad \forall i \in T, u \in U, r \in R \setminus \delta(i) \quad (7)$$

$$\sum_{j \in T} x_{jiru} - \sum_{j \in T} x_{ij(r+1)u} = 0 \quad \forall i \in T, u \in U, r \in R(:8) \quad r < |R| - 1$$

$$x_{ijru} \in \{0, 1\} \quad \forall i \in T, j \in T, r \in R, \quad (9)$$

$$x(i, j, r, u) = \begin{cases} x_{ijru} & \text{if } r = 1 \\ x_{ji(r-1)u} \end{cases}$$

Constraints (2) and (3) ascertain that every game can be officiated by only one umpire and that every umpire may officiate only one game per round, respectively. Constraints (4) state that every umpire should visit every team at least once during the season. Constraints (5) and (6) specify that every umpire must wait $q_1 - 1$ days to revisit the same home location and that every umpire must wait $q_2 - 1$ days to revisit the same team, respectively. Constraints (7) enforce that an umpire can only travel to the home location of a team if that team hosts a game in that certain round. If an umpire is at the location of a team in round r, the umpire must leave from the same location in round $r + 1$ as specified by constraints (8). Finally, the objective function, given by equation (1), is to minimize the total travel distances of the umpires.

3 Dantzig-Wolfe Reformulation

In order to obtain stronger bounds, we reformulate the model presented in the previous section by applying the Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960). The original problem is decomposed into a master problem and n pricing problems, one for each umpire.

Figure 1 shows the structure of the linear program (LP) of a TUP instance considering the formulation presented in Section 2. This figure presents the coefficient matrix of the original LP (left image) and the same LP after sorting the rows and columns by umpire (right image). The dots indicate non-zero coefficients in the constraint matrix. The required block structure for the Dantzig-Wolfe decomposition is easily identified in the right image (sorted model). In this image, each square block forms a pricing problem containing the constraints and variables corresponding to a single umpire.

In formulation (1)-(9), constraints (3)-(9) are umpire-oriented and form the pricing problems. The remaining constraints, given by equation (2), are the coupling (or linking) constraints. These constraints correspond to the wide block at the bottom of the sorted LP in Figure 1.

The pricing problem can be stated as the problem of finding the optimal schedule for one umpire with the consideration of dual costs.

The master problem is a set partition problem, whose formulation is given by equations (10)-(13). In this formulation, $(1$ is the set of columns (possible schedules for the umpires), $(1_u$ represents the subset of $(1$ containing all columns of umpire $u \in U$, d_s is the cost (travel distance) of column $s \in (1$, λ_s is a binary variable that indicates whether the column $s \in (1$ is selected or not and, finally, a_{irs} is a binary coefficient denoting whether the umpire is assigned to game hosted by team $i \in T$ in round $r \in R$ in column $s \in (1$. Constraints (11) guarantee that only one column is chosen per umpire while constraints (12) are the coupling constraints inherited from the original problem (2), and ensure that each game in each round is officiated by exactly one umpire.

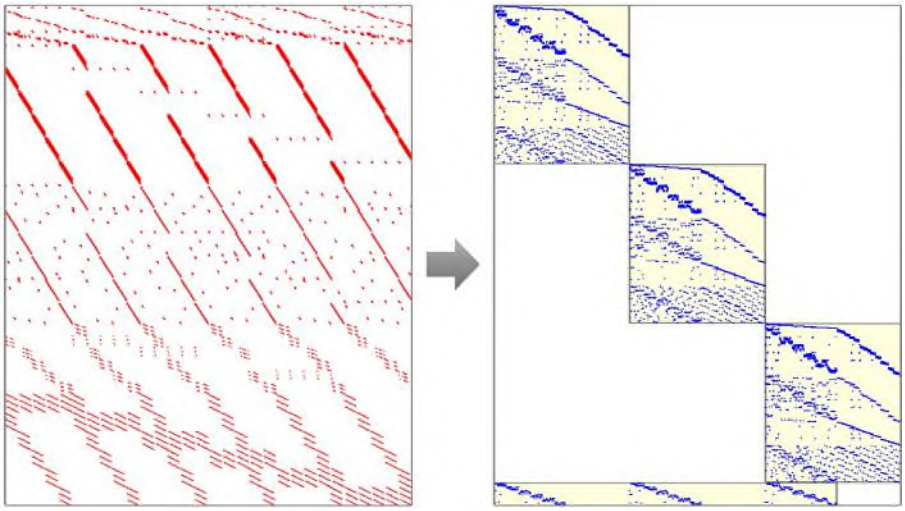


Fig. 1 Representation of the applied decomposition

$$\min \sum_{u \in U} \sum_{s \in \Omega_u} d_s \lambda_s \tag{10}$$

$$\text{s.t.} \sum_{s \in \Omega_u} \lambda_s = 1 \quad \forall u \in U \tag{11}$$

$$\sum_{u \in U} \sum_{s \in \Omega_u} a_{irs} \lambda_s = 1 \quad \forall i \in T, \forall r \in R \tag{12}$$

$$\lambda_s \in \{0, 1\} \quad \forall u \in U, \forall s \in (1_u \tag{13}$$

The column generation approach (L¨ubbecke and Desrosiers, 2005; Vanderbeck and Wolsey, 2010) is solved iteratively. The linear relaxation of the master problem is solved first. In every iterations, the pricing problems are solved to obtain reduced cost columns. A reduced cost column for umpire u is a column $s \in (1_u$ for which $v_u + \sum_{i \in T} \sum_{r \in R} a_{irs} w_{ir} > d_s$, where v_u and w_{ir} represent the dual variables corresponding to constraints (11) and (12), respectively. If such columns are found, they are added to the master problem, which is subsequently re-solved. The algorithm continues until no reduced cost columns exists, in which case the relaxation of the reduced master problem is solved.

3.1 Symmetry breaking

In order to speed up the pricing solver, we preallocate the games assigned to the umpires in the first round. This reduces symmetry (Yildiz, 2008) in the original problem,

as otherwise the umpires would have similar coefficients in the constraint matrix. Pre-allocation can be easily achieved by adding constraints (14) to the formulation (1)-(9). In these constraints, we use the notation $H_k(r)$ to represent the home team of the k-th game of round r, with the games in lexicographical order.

$$\sum_{j \in T} x_{ij1u} = 1 \quad \forall u \in U, i = H_u(1) \tag{14}$$

Constraints (14) are umpire-oriented and can be included in the pricing problems of the column generation scheme. Adding these constraints results in a reduction of the pricing problem size by one round.

3.2 Specialized pricing problem solver

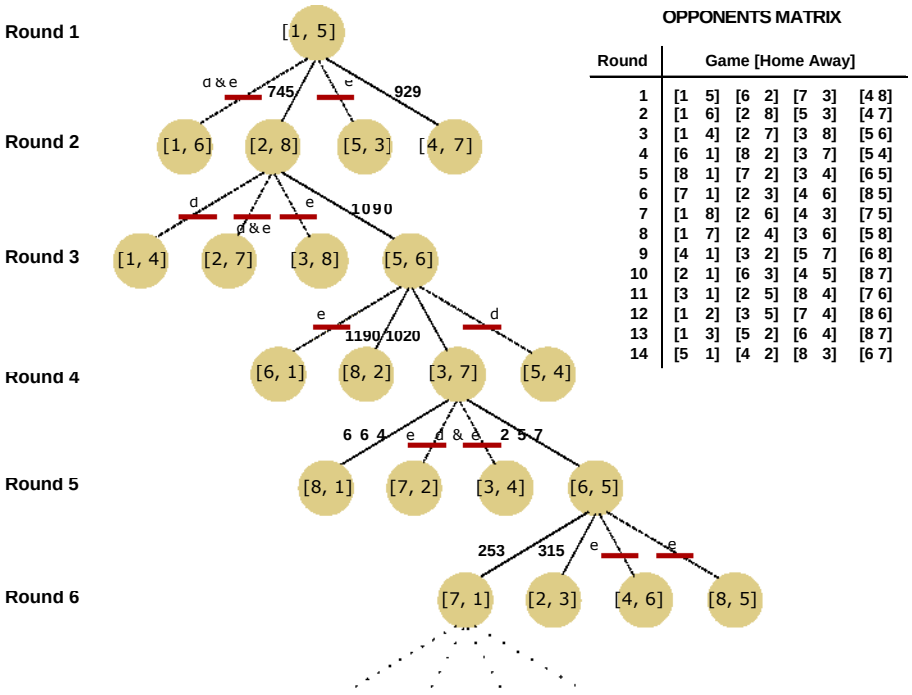
A branch-and-bound pricing solver is used to generate a predefined maximum number c of reduced cost columns. Starting in the first round, the algorithm assigns games to the umpire, round after round until the last round. An assigment of a game to an umpire in a round is feasible if (i) the umpire did not visit the same location in the previous $q_1 - 1$ rounds and (ii) the umpire did not officiate any of the teams during the previous $q_2 - 1$ rounds. Whenever multiple games can be assigned in a round, the algorithm chooses the assignment incurring the smallest increase in the travel distance.

Figure 2 shows an example of the branch-and-bound procedure for an 8-team (4-umpire) problem instance. The table inside the figure shows the considered game schedule (opponents matrix). The example considers the pricing for the first umpire using parameter values $q_1 = 4$ and $q_2 = 2$. As detailed in section 3.1, the assignment in the first round is fixed.

The umpire can neither officiate game [5,3] nor game [1,6] in the second round due to constraint *e* (presented in section 1), since a game played by teams 1 and 5 has already been officiated by the umpire during the first round. Moreover, the umpire cannot officiate game [1,6] due to constraint *d*, since the home location of team 1 has already been visited in the previous round. The only possibilities left in round two are game [2,8] and game [4,7]. The branch-and-bound prefers to assign the umpire to game [2,8] because the travel distance between the home location of teams one and two is smaller than the distance between the home locations of teams one and four.

If no valid assignment can be found in a certain round, the procedure returns to the previous round and chooses the game with the second smallest travel distance. This procedure continues until a valid assignment has been found in the last round. If the resulting solution does not violate constraint *c*, it is feasible and serves as an upper bound for pruning when exploring the rest of the search tree.

We implemented several extensions to improve the performance of the branch-and-bound algorithm. In section 3.2.1, we explain these pruning strategies.



OPPONENTS MATRIX

Round	Game [Home Away]							
1	[1	5]	[6	2]	[7	3]	[4	8]
2	[1	6]	[2	8]	[5	3]	[4	7]
3	[1	4]	[2	7]	[3	8]	[5	6]
4	[6	1]	[8	2]	[3	7]	[5	4]
5	[8	1]	[7	2]	[3	4]	[6	5]
6	[7	1]	[2	3]	[4	6]	[8	5]
7	[1	8]	[2	6]	[4	3]	[7	5]
8	[1	7]	[2	4]	[3	6]	[5	8]
9	[4	1]	[3	2]	[5	7]	[6	8]
10	[2	1]	[6	3]	[4	5]	[8	7]
11	[3	1]	[2	5]	[8	4]	[7	6]
12	[1	2]	[3	5]	[7	4]	[8	6]
13	[1	3]	[5	2]	[6	4]	[8	7]
14	[5	1]	[4	2]	[8	3]	[6	7]

Fig. 2 Specialized branch-and-bound example for a 8-team (4-umpire) problem instance.

3.2.1 Pruning the search tree

Multiple strategies exist to prune unfavorable parts of the search tree. First of all, the branch-and-bound algorithm prunes the parts of the search tree where no optimal solution can reside based on the lower and upper bound on the travel distance. Once the branch-and-bound algorithm has obtained a feasible solution, it can be used as an upper bound on the minimum travel distance of the umpire.

For each game in every round, a shortest path exists to any of the games in the last round. The shortest path serves as a lower bound for the branch-and-bound procedure.When trying to assign a game in a round, the algorithm evaluates whether the current travel distance together with the lower bound exceeds the currently best known upper bound. If so, the branch-and-bound need not consider that assignment anymore, since it will not improve the current upper bound.

It is impossible to evaluate constraint c before a complete path has been generated for the umpire. Nevertheless, a second pruning strategy is possible. If in a certain round, the number of unvisited home locations exceeds the remaining number of rounds, it is impossible to obtain a solution satisfying constraint c, given the assignments in the previous rounds. The branch-and- bound algorithm should therefore return to a previous round and explore other assigments.

4 Branch-and-Price

Section 3 presented the column generation scheme. Since this algorithm only solves the LP-relaxed version of the problem, it may be necessary to branch on the variables to find an integer solution. In that case, we apply branch-and-price (Barnhart et al, 1998; Vanderbeck, 2000), which is a variant of branch-and-bound where the relaxation is solved by column generation in each node of the search tree.

The branch-and-price algorithm branches on the variables x_{ijru} from the original formulation. Since the branching tree is too large, we consider two different strategies for branching, each one pursuing a different goal. The first branching strategy aims at providing good lower bounds by conducting a best-first search (BFS) in the branching tree. The second strategy executes depth-first search (DFS) and focuses on finding integral solutions.

In each iteration, the BFS strategy selects variables for branching based on the following criterion: variables with the most fractional value of the earliest available round are selected first. Fixing variables of the earliest available round impacts the performance of the pricing solver considerably. The specialized branch-and-bound constructs the solution from the first to the last round, in lexicographical order. Hence, if a variable of the last round were selected first, processing time may be wasted searching in infeasible subtrees. Since the fixation of a variable renders several subtrees infeasible, it is better to detect the infeasibility as soon as possible during the branching process. If this strategy were not used, the detection of infeasible subtrees would be delayed, consuming a considerable amount of processing time.

The DFS strategy aims to obtain feasible solutions as soon as possible. Therefore, in each node the variable with least fractional value of the earliest possible round is selected to be branched first. By doing so in a depth-first search manner, the fixations are directed to iteratively build a feasible solution using the information provided by the column generation.

5 Computational Experiments

The approach applies SCIP/GCG (Achterberg, 2009). This open source framework provides a well structured platform for developing branch-and-price algorithms. The branching scheme, node rules and pricing solver were coded in Java, using Java Native Interface to exchange information between Java and C. CPLEX was used to solve the linear relaxation of the Restricted Master Problem.

The experiments were executed on a Intel(R) Xeon(R) CPU E5-2650 @ 2.60GHz computer with 128Gb of RAM memory running Linux Mint 16. CPLEX version 12.6 and Java Virtual Machine 1.7 were used.

The benchmark set used within the experiments is available online [1], together with the currently best known solution values in literature. We also consider the most recent bounds found by Wauters et al (2014) for comparison. Further information about the instances can be found in (Trick et al, 2012).

1 http://mat.gsia.cmu.edu/TUP/, last accessed June 8, 2014

The discussion of experiments focuses on two evaluations. First we consider the dual bounds obtained by the BFS branching scheme in the branch-and-price. Afterwards we present the feasible solutions obtained by the DFS branching strategy. In both situations, we compare our results with the best known in the literature.

5.1 BFS Strategy

The results of the branch-and-price with the BFS strategy are presented in Table 1. This table shows the lower bound (LB_0) obtained with the column generation (in the root node), the best lower bound obtained (LB) and the best lower bound found in the literature. The table also shows the time in which the lower bound was found by the branch-and-price algorithm. The last column presents the gap between the best known bound and the obtained bound. The cells marked with indicate some improvement over the best known bound. Considering the imposed time limit of 3 hours, no feasible solutions were found for any of the instances with more than 10 teams with the BFS strategy. We omitted the results for the small instances, as they can be easily solved to optimality in few seconds.

Table 1 shows that the column generation approach already improves 8 best known lower bounds. By applying the branch-and-price with BFS, 15 other instances have their best known dual bound improved. This result corroborates the expected strong bound from column generation.

Table 1 also shows the influence of the pricing solver on the total processing time of the column generation approach. Consider, for example, the difference in time required for solving the column generation in the root bound (given by column LB_0) for instances ‘16A-7,2’ and ‘16A-7,3’. Column generation for instance ‘16A-7,2’ required much more computation time than for ‘16-A-7,3’. This is mainly due to the value $q = 2$ in the first instance, which is less constrained than the second one, with $q = 3$. Small values of q negatively impact the performance of our specialized branch-and-bound, since it provides fewer pruning opportunities.

5.2 DFS Strategy

The results of the branch-and-price with the DFS strategy are presented in Table 2. This table shows the value of the first feasible solution found (UB_0), the best solution found by the branch-and-price (UB) and the best solution in the literature. The table also shows the total runtime to find the solutions with the branch-and-price and the gap between the best known solution and the obtained solution. The experiments were restricted to 3 hours of processing time. As in Table 1, cells marked with indicate some improvement over the best known solution. It is important to note that the bounds for these instances have been updated repeatedly over the years. Best bounds were hard to trace.

Table 2 shows that the DFS strategy provides feasible solutions in a small amount of time for most instances. Even considering the total available runtime, the branch-and-price was able to improve five upper bounds. For 7 instances, the branch-and-price was not able to produce feasible solutions within the time limit.

Table 1 Experiments with the BFS strategy in branch-and-price

Inst.	q_1, q_2	Bounds		Time (seconds)		LB*	gap(%)
		LB*	LB	LB*	LB		
14	7, 3	156439.3	157812.8	42.1	10500.0	159797	1.24
14	6, 3	154439.9	155570.4	40.8	10560.0	156551	0.63
14	5, 3	152941.3	® 153759.6	50.3	10740.0	153066	-0.45
14A	7, 3	149992.7	151243.5	40.7	10740.0	153199	1.28
14A	6, 3	148168.7	149285.4	45.5	10680.0	150998	1.13
14A	5, 3	147097.5	147966.4	47.8	10620.0	148299	0.22
14B	7, 3	149767.0	® 151165.8	43.5	10620.0	151059	-0.07
14B	6, 3	148243.9	149208.6	49.6	10620.0	149267	0.04
14B	5, 3	146846.2	® 147638.3	55.7	10800.0	147534	-0.07
14C	7, 3	148613.2	150101.6	44.6	10380.0	151581	0.98
14C	6, 3	146774.6	147820.0	47.7	10320.0	148728	0.61
14C	5, 3	145794.4	146622.1	49.5	10620.0	146764	0.10
16	8, 4	184187.6	® 193457.1	172.0	10260.0	185939	-3.89
16	8, 2	® 155045.2	® 155045.2	7092.0	7092.0	151481	-2.82
16	7, 3	158257.4	® 158586.0	10500.0	10500.0	158480	-0.07
16	7, 2	® 148341.8	® 148341.8	10102.0	10102.0	147138	-0.81
16A	8, 4	® 198969.7	® 200648.5	172.0	10260.0	185119	-11.28
16A	8, 2	® 166575.5	® 166624.1	5403.0	10410.0	162788	-2.77
16A	7, 3	170575.1	172420.1	371.0	10560.0	172964	0.31
16A	7, 2	161571.2	161571.2	7476.0	7476.0	161640	0.04
16B	8, 4	207505.4	® 209346.5	202.0	10440.0	208418	-4.55
16B	8, 2	® 169363.4	® 170092.6	5162.0	10162.0	167768	-1.37
16B	7, 3	170632.5	172058.0	880.0	10560.0	173023	0.56
16B	7, 2	163539.7	163649.6	9021.2	11298.3	164012	0.29
16C	8, 4	® 200682.6	® 205643.8	234.0	10380.0	188561	-8.31
16C	8, 2	® 168783.6	® 168783.6	7380.0	7380.0	166001	-1.77
16C	7, 3	171216.0	® 171767.6	449.0	10740.0	171377	-0.23
16C	7, 2	® 163850.8	® 163850.8	10578.0	10578.0	163305	-0.33

Considering that the developed approach tends to perform better on more constrained instances, one would probably expect better results for the very constrained '16-8,4', '16A-8,4', '16B-8,4' and '16C-8,4' instances. The branch-and-price was not able to find any feasible solution after several hours of processing time. This result, coupled with the fact that there are no known solution for these instances, motivated us to investigate the strong indication that they may be infeasible. In the next section we discuss over the infeasibility of these instances.

5.3 A note on the feasibility of TUP instances

It was already shown that TUP instances with $q1 > n$ and $q2 > \lceil \frac{n}{2} \rceil$ are infeasible (Yildiz, 2008). Instance '12-6,3' was also proven infeasible. This instance belongs to the special class of TUP instances with constraint values $q1 = n$ and $q2 = \lfloor \frac{n}{2} \rfloor$, further denoted as TUP^0_0, referring to TUP^{d1}_{d2} with $d1 = 0$ and $d2 = 0$. Other instances from TUP^0_0 with $n \leq 7$ were shown to contain at least one feasible solution. No feasible solution was found for TUP^0_0 instances with $n > 7$.

An adapted version of the branch-and-bound procedure presented in Section 3.2, considering all umpires simultaneously, enables proving that instance '16-8,4', be-

Table 2 Experiments with the DFS strategy in branch-and-price

Inst.	q1, q2	Bounds		Time (seconds)		UB*	gap(%)
		UB0	UB	UB0	UB		
6	3, 1	14077	14077	2.0	2.0	14077	0.00
6A	3, 1	16918	15457	1.3	1.8	15457	0.00
6B	3, 1	16716	16716	2.2	2.2	16716	0.00
6C	3, 1	14396	14396	1.8	1.8	14396	0.00
10	5, 2	49154	48942	7.6	8.1	48942	0.00
10A	5, 2	46551	46551	7.6	7.6	46551	0.00
10B	5, 2	45609	45609	5.7	5.7	45609	0.00
10C	5, 2	43149	43149	6.9	6.9	43149	0.00
14	7, 3	174373	166942	84.0	546.6	164440	1.50
14	6, 3	163488	159808	88.7	2462.4	159505	1.67
14	5, 3	156406	® 155392	99.9	1214.8	155439	-0.03
14A	7, 3	172737	160856	91.4	7500.1	158760	1.30
14A	6, 3	160599	154637	89.7	2813.5	153216	0.92
14A	5, 3	157249	150386	933.6	4110.2	149331	0.07
14B	7, 3	170180	162677	88.3	1560.1	157884	2.95
14B	6, 3	164212	155817	94.0	5662.4	152740	1.97
14B	5, 3	154425	149866	100.8	1579.2	149621	0.16
14C	7, 3	173962	159815	87.2	6071.9	154913	3.07
14C	6, 3	155918	152696	93.2	6877.4	150858	1.20
14C	5, 3	155218	® 149482	108.6	9218.9	149662	-0.12
16	8, 4	-	-	-	-	-	-
16	8, 2	162720	161999	6612.6	9918.9	160705	0.80
16	7, 3	176576	170293	950.8	7799.8	168860	0.84
16	7, 2	-	-	-	-	153978	-
16A	8, 4	-	-	-	-	-	-
16A	8, 2	175796	® 171882	7065.8	8016.7	172966	-0.63
16A	7, 3	190715	187686	1020.5	2979.9	179960	4.12
16A	7, 2	165931	165766	9562.6	9759.0	164620	0.69
16B	8, 4	-	-	-	-	-	-
16B	8, 2	189564	® 180728	9283.3	10717.5	180888	-0.09
16B	7, 3	192188	186429	1176.3	1378.1	181565	2.61
16B	7, 2	-	-	-	-	170194	-
16C	8, 4	-	-	-	-	-	-
16C	8, 2	191461	® 179939	8949.0	9285.8	180221	-0.16
16C	7, 3	191859	187310	822.4	2234.5	184181	1.67
16C	7, 2	-	-	-	-	169184	-

longing to TUP , has no feasible solution. At the same time, instance '16A-8,4', instance '16B-8,4' and instance '16C-8,4' are proven to be infeasible since their opponents matrix is equal to that of '16-8,4'. Table 3 summarizes the feasibility of the instances from TUP up to $n = 10$. The table reports for each instance, the feasibility, the number of nodes and time (in milliseconds) needed by the branch-and-bound to prove (in)feasibility. The feasibility of instances from TUP with $n > 8$ remains unknown after 48 hours of running time of the branch-and-bound algorithm.

6 Conclusions and future work

This work introduced a branch-and-price approach to the Traveling Umpire Problem, devoting attention to both computation of strong dual bounds and production of good

Table 3 Feasibility of instances from $\mathbf{TUP}^0_0$ up to n = 10, and the number of nodes and time (ms) needed to prove (in)feasiblity.

Inst.	$q1, q2$	Feasibility	Nodes	Time(ms)
4	2,1	feasible	12	1
6	3,1	feasible	35	2
8	4,2	feasible	1, 129	5
10	5,2	feasible	27, 179	63
12	6,3	infeasible	901, 228	309
14	7,3	feasible	172, 552	77
16	8,4	**infeasible**	$35,696 \times 10^6$	3h
18	9,4	unknown	-	48h
20	10,5	unknown	-	48h

feasible solutions. We presented a pricing solver and branching rules, optimized to speed up the resolution of the pricing problem.

We were able to improve several lower bounds. In addition, five improved solutions have been obtained. We also employed a modified version of the pricing solver to prove infeasibility of some instances. Tight runtime limits are sufficient for the branch-and-price to generate competitive feasible solutions for the most constrained problem instances.

As suggestions for future work, we point at the development of heuristics that use the information from the column generation approach to produce good solutions. In addition, other branching rules and approaches to speed up the resolution of the pricing problem may yield further improvements.

7 Acknowledgements

Work supported by the Belgian Science Policy Office (BELSPO) in the Interuniversity Attraction Pole COMEX (http://comex.ulb.ac.be).

References

Achterberg T (2009) Scip: Solving constraint integer programs. Mathematical Programming Computation 1(1):1–41

Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MWP, Vance PH (1998) Branch-and-price: column generation for solving huge integer programs. Operations Research 46:316–329

Dantzig GB, Wolfe P (1960) Decomposition principle for linear programs. Operations Research 8(1):101–111

L¨ubbecke ME, Desrosiers J (2005) Selected Topics in Column Generation. Operations Research 53(6):1007–1023

de Oliveira L, de Souza CC, Yunes T (2013) Improved bounds for the traveling umpire problem: A stronger formulation and a relax-and-fix heuristic. European Journal of Operational Research In press

Trick MA, Yildiz H (2007) Bender's cuts guided large neighborhood search for the traveling umpire problem. In: Van Hentenryck P, Wolsey L (eds) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, no. 4510 in Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp 332–345

Trick MA, Yildiz H (2011) Benders' cuts guided large neighborhood search for the traveling umpire problem. Naval Research Logistics (NRL) 58(8):771 – 781

Trick MA, Yildiz H (2012) Locally optimized crossover for the traveling umpire problem. European Journal of Operational Research 216(2):286 – 292

Trick MA, Yildiz H, Yunes T (2012) Scheduling major league baseball umpires and the traveling umpire problem. Interfaces 42:232 – 244

Vanderbeck F (2000) On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. Operations Research 48(1):111–128

Vanderbeck F, Wolsey L (2010) Reformulation and decomposition of integer programs. In: J¨unger M, Liebling TM, Naddef D, Nemhauser GL, Pulleyblank WR, Reinelt G, Rinaldi G, Wolsey LA (eds) 50 Years of Integer Programming 1958-2008, Springer Berlin Heidelberg, pp 431–502

Wauters T, Malderen SV, Vanden Berghe G (2014) Decomposition and local search based methods for the traveling umpire problem. European Journal of Operational Research

Yildiz H (2008) Methodologies and applications for scheduling, routing & related problems. PhD thesis, Carnegie Mellon University