Lower Bounds for Large Traveling Umpire Instances: New Valid Inequalities and a Branch-and-Cut Algorithm

Lucas de Oliveira[a], Cid C. de Souza[a], Tallys Yunes[b,*]

[a]*Institute of Computing, University of Campinas (UNICAMP), Campinas, SP, Brazil*
[b]*School of Business Administration, University of Miami, Coral Gables, FL, USA*

Abstract

Given a double round-robin tournament, the Traveling Umpire Problem (TUP) seeks to assign umpires to the games of the tournament while minimizing the total distance traveled by the umpires. The assignment must satisfy constraints that prevent umpires from seeing teams and venues too often, while making sure all games have umpires in every round, and all umpires visit all venues. We propose a new integer programming model for the TUP that generalizes the two best existing models, introduce new families of strong valid inequalities, and implement a branch-and-cut algorithm to solve instances from the TUP benchmark. When compared against published state-of-the-art methods, our algorithm significantly improves all best known lower bounds for large TUP instances (with 20 or more teams).

Keywords:
sports scheduling, traveling umpire problem, integer programming, branch-and-cut, OR in sports

1. Introduction

The field of sports scheduling is rich with interesting and difficult problems that arise from the design of fair competitions. The assignment of officials (judges, referees, umpires, etc.) to the games of a competition is a well-known and challenging problem in this field. Typically, a myriad of conditions have to be imposed to guarantee the fairness of refereeing over the entire event, while minimizing some measure of cost. Several studies have been published dealing with specific details of different sports, such as: baseball [1, 2, 3, 4, 5, 6, 7, 8, 9], cricket [10], football [11], and tennis [12]. A variety of other sports scheduling problems can be found in [13, 14, 15].

We focus on the Traveling Umpire Problem (TUP), which is an abstraction that incorporates the main issues behind the assignment of umpire crews (umpires, henceforth for short) to the games of Major League Baseball (MLB). This problem was first introduced in [16] and recently proved to be NP-Complete (under certain conditions) in [17].

The TUP receives as input a double round-robin tournament with $2n$ teams and $4n-2$ rounds, the distances between the home venues of each pair of teams, and two integers $0 \leq d_1 < n$ and $0 \leq d_2 < \lfloor \frac{n}{2} \rfloor$. A feasible solution to the TUP is an assignment of n umpires to the games of the tournament that satisfies the following constraints:

[☆]This research was supported by *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (CNPq/Brazil) under grants 304727/2014-8, 142278/2013-0, and 477692/2012-5.

[*]Corresponding author. Postal address: School of Business Administration, University of Miami. 5250 University Drive, Room KE405, Coral Gables, FL 33146, USA. Phone: +1 305-284-5107. Fax: +1 305-284-2321.

Email addresses: lucas.oliveira@ic.unicamp.br (Lucas de Oliveira), cid@ic.unicamp.br (Cid C. de Souza), tallys@miami.edu (Tallys Yunes)

(i) Each game is refereed by exactly one umpire;
(ii) Each umpire is assigned to exactly one game per round;
(iii) Each umpire visits the home venue of each team at least once;
(iv) Each umpire visits any given venue at most once during any $q_1 = n - d_1$ consecutive rounds;
(v) Each umpire sees any given team at most once during any $q_2 = \lfloor \frac{n}{2} \rfloor - d_2$ consecutive rounds.

The TUP's objective function is to minimize the total distance traveled by the umpires throughout the entire tournament.

Our main contributions are: (a) we present an integer programming model for the TUP that generalizes the two best models in literature; (b) we introduce new families of strong valid inequalities for this model; and (c) we improve the best known lower bounds for all large instances in the TUP benchmark [18, 23] with 20 or more teams by solving our optimization model with a branch-and-cut algorithm.

The remainder of this paper is organized as follows. The next section presents a literature review of the TUP, while Section 3 describes our optimization model and the new valid inequalities. Section 4 details the separation routines used in our branch-and-cut algorithm, and Section 5 analyzes our computational results. Finally, we conclude and discuss ideas for future work in Section 6.

2. Previous Work

It is evident from several years of computational experience with the TUP that it is a very difficult problem to solve. Even finding feasible solutions without regard to quality can be quite a challenging task. In this section we summarize some of the most successful approaches from the TUP literature.

In [3], the authors introduce a set of benchmark instances having between 4 and 32 teams. These instances are available for download at [18, 23], and have become the standard benchmark set for all published research on the TUP. Both an integer programming (IP) and a constraint programming (CP) model for the TUP were proposed in [3]. Exact solvers were able to solve these models to optimality for instances with up to 10 teams, but had difficulty finding feasible solutions to larger problems. Therefore, also in [3], the authors proposed a greedy matching heuristic to generate good solutions. When this heuristic gets stuck with an infeasible partial solution, a large neighborhood search guided by Benders cuts takes place to fix it, allowing the heuristic to resume execution. This approach successfully found many solutions that were better than those found by exact methods for instances with 14, 16, and 30 teams.

The real-life MLB umpire scheduling problem (MLB-USP) is described in [4], but the IP model proposed therein cannot be solved due its large number of variables and constraints. Hence, the TUP is highlighted an abstraction of MLB-USP that captures its most important features and ignores minor details. A simulated annealing (SA) algorithm was proposed in [4] to obtain good solutions for both MLB-USP and TUP, finding better schedules than those adopted by MLB. The solutions found by the SA for the TUP, however, were inferior to those obtained by the heuristic proposed in [3]. Continuing on the heuristic front, a genetic algorithm (GA) was proposed in [5] employing a sophisticated crossover operator tailored to recombine two solutions in a way that the offspring is locally optimized by solving a matching problem. Several new best solutions were found by this GA for instances with 14, 16, and 30 teams. Later on, a stronger IP model based on the one proposed in [3] was presented in [6]. This new model was more compact with respect to the number of variables and constraints and also included new constraints. It led to improvements in all known lower bounds for the benchmark instances and, for the first time, provided lower bounds for instances with more than 16 teams. Additionally, [6] introduced a relax-and-fix heuristic based on their IP model that managed to improve the quality of almost all solutions known at the time.

An iterative deepening search (IDS) and an iterative local search (ILS) were proposed in [7]. These methods are complementary in the sense that IDS found many improved solutions to medium-sized instances (with 14 and 16 teams), whereas ILS obtained new better solutions to larger instances (with 26 or more teams). A decomposition approach to derive strong lower bounds for the TUP is also proposed in [7]. This approach subdivides the tournament into smaller pieces, solving each one with a modified version of the IP model from [6] that corresponds to a relaxed version of TUP. This method improved all of the best lower bounds known at the time.

In [8], the authors introduce a set partitioning model whose variables represent an umpire's complete schedule, visiting a game in each one of the $4n-2$ rounds of the tournament. A branch-and-price (BP) algorithm was developed to solve this model since it has an exponential number of variables. Its pricing routine uses branch-and-bound to solve a constrained shortest path problem. Following a best-first search strategy, this BP improved several lower bounds for instances with 14 and 16 teams and, using a depth-first search strategy, it obtained some new best solutions to instances with 14 and 16 teams.

A network flow model and a set partitioning model that is equivalent to the one in [8] were presented in [9]. The network flow model is optimized via a branch-and-bound (BB) algorithm which solves a Lagrangian relaxation at every node of the search tree. This BB algorithm improved several lower bounds for instances with more than 18 teams. Their set partitioning model is strengthened by the addition of cutting planes and solved with a branch-and-price-and-cut (BPC) algorithm. The BPC improved many lower bounds for instances with up to 18 teams, and was the first method to solve instances with 14 teams to optimality.

A branch-and-bound algorithm combined with a parallel routine to generate strong lower bounds was recently proposed in [19]. The branch-and-bound rapidly enumerates the nodes in the search tree and uses the lower bounds calculated concurrently to prune as many nodes as possible. The lower bound calculation comprises a bottom-up algorithm inspired by the decomposition scheme presented in [7]. This method solved to optimality all 14-team benchmark instances within a few minutes and was the first to obtain provably optimal solutions for 16-team instances. Additionally, lower and upper bounds were improved for the 16-team instances. Despite these remarkable results, this method does not appear to scale well for instances with 18 or more teams.

3. Optimization Model

We now present an integer programming (IP) model for the TUP that generalizes two existing models. In [9], the authors describe a network-flow model (NFM) whose variables represent trips made by umpires between consecutive rounds in the tournament. Its linear relaxation can be solved quickly and produces good lower bounds. Still in [9], and also in [8], a stronger set-partitioning model (SPM) is used whose variables represent an umpire's entire sequence of trips through all $4n-2$ rounds of the tournament, while satisfying constraints *(iii)–(v)*. Although SPM's linear relaxation produces significantly stronger lower bounds than NFM's linear relaxation, the time required to solve it increases quickly as the number of teams increases, which makes it impractical to use SPM with more than 18 teams.

Here is how our model generalizes the previous ones. While NFM's and SPM's variables represent umpires' trip sequences with lengths of 2 and $4n-2$ rounds, respectively, the length of the trip sequences our model's variables represent is a parameter that can fall anywhere between 2 and $4n-2$. This flexibility allows us to empirically study the trade-off between relaxation solution speed (an advantage of NFM) and lower bound strength (an advantage of SPM).

Let $2 \leq w \leq 4n-2$ be the sequence-length parameter mentioned above. For a fixed value of w, we create variables by dividing the input tournament T into sections indexed by $S = \{1, 2, \ldots, \lceil \frac{4n-3}{w-1} \rceil\}$ as follows. For any $s \in S$, the s-th section of T, denoted T_s, consists of consecutive rounds $(s-1)(w-1)+1$

through $\min\{s(w-1)+1, 4n-2\}$. Note that all sections have exactly w rounds, except for the last one, which could be shorter. Figure 1 illustrates a tournament with four teams and six rounds being subdivided into sections for $w = 2$, 3, 4, and 6.

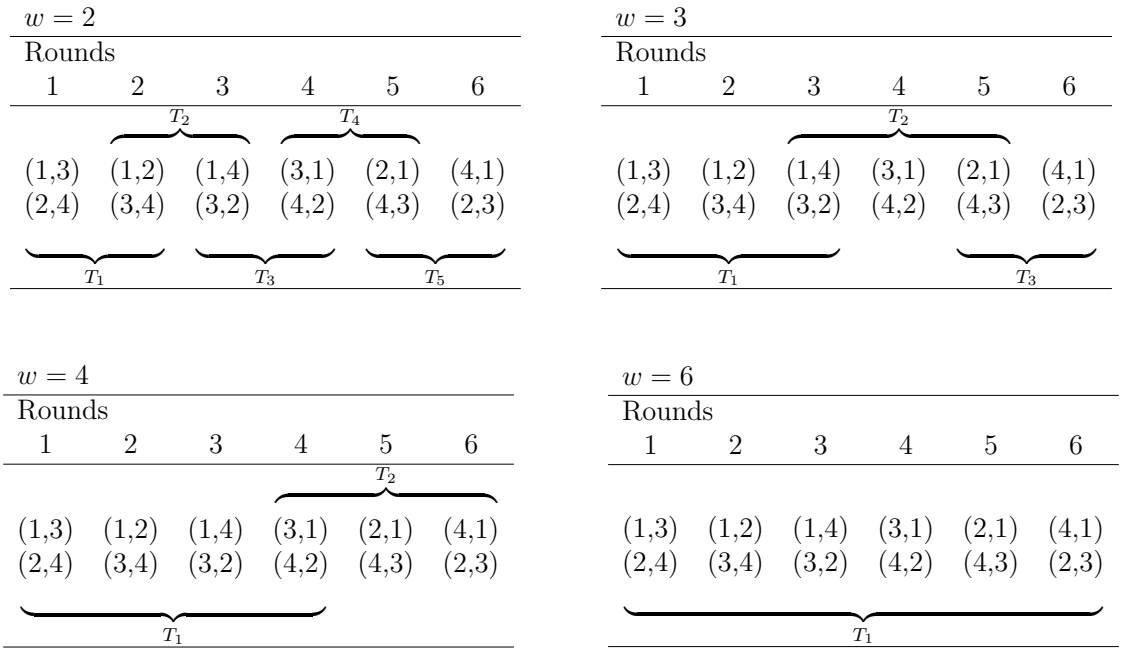

Figure 1: Sections of a 4-team, 6-round tournament for $w = 2$, 3, 4, and 6.

For each section $s \in S$, our model contains variables to represent every trip sequence that visits all of the rounds in T_s and satisfies TUP constraints *(iv)* and *(v)*. Because only one section exists when $w = 4n-2$, trip sequences are also required to satisfy constraint *(iii)* in this particular case. When $2 \leq w < 4n-2$, we cannot impose constraint *(iii)*. Note, however, that consecutive sections have one round in common (Figure 1), which allows us to connect their trip sequences to create a longer sequence. In the next section we introduce our mathematical model and detail the constraints that ensure trip sequences get properly combined to create feasible travel schedules for the n umpires.

3.1. Initial Integer Programming Formulation

For a fixed w and any $s \in S$, let P_s be the set of trip sequences in T_s that visit all of its rounds and satisfy constraints *(iv)* and *(v)*. (When $w = 4n-2$, we have only P_1 and require that its sequences satisfy constraint *(iii)*.) For each $p \in P = \bigcup_{s \in S} P_s$, create a binary variable x_p that is equal to one when p is part of the solution, and equal to zero otherwise. We denote the distance traveled by the trips in p by d_p. Let G_s be the set of games in T_s's rounds, and let P_{sg} be the set of all trip sequences in P_s that contain a given game $g \in G_s$. From now on, we will use the term *simple route* to refer to any trip sequence in P, and the term *route* to refer to an ordered sequence of simple routes $r_1, \ldots, r_m$ such that r_i and r_{i+1} come from consecutive sections, and the last game of r_i is the same as the first game of r_{i+1}, for any $i = 1, \ldots, m-1$. Given a route Q, we denote by $P(Q)$ the set of all simple routes in Q. A *complete route* is a route that visits every round of the tournament, that is, it contains one simple route from each section of T. A route is said to be infeasible when it contains two or more games that violate constraints *(iv)* or *(v)*, or when it is a complete route and violates constraint *(iii)*. Finally, we denote the set of all infeasible routes by $\mathbb{U}$. We are now ready to present our mathematical model.

$$\min \sum_{p \in P} d_p x_p \tag{1}$$

Subject to:

$$\sum_{p \in P_{sg}} x_p = 1, \quad \forall\, s \in S, g \in G_s, \tag{2}$$

$$\sum_{p \in P(U)} x_p \leq |P(U)| - 1, \quad \forall\, U \in \mathbb{U}, \tag{3}$$

$$x_p \in \{0, 1\}, \quad \forall\, p \in P. \tag{4}$$

The objective function (1) minimizes the total distance traveled by the umpires, and (2) ensures that all games in each section are visited by a simple route. Note that a game in a round shared by two consecutive sections is visited by two simple routes; one ending and one starting at that game. Constraints (2) and (4) together guarantee that a feasible solution consists of n complete routes satisfying TUP constraints *(i)* and *(ii)*. TUP constraints *(iii)–(v)* are respected because of (3), which prevents infeasible routes from being part of the solution by excluding at least one of their constituent simple routes. From now on, let $\mathcal{T}$ denote the TUP polytope, that is, the convex hull of the feasible solutions to (1)–(4).

3.2. Strong Valid Inequalities

The linear relaxation of (1)–(4) does not provide strong lower bounds, mostly because (3) turns out to be a weak constraint. In [9], the authors propose to strengthen (3) via a lifting procedure from [20], which we explain next. Let $U = (u_1, u_2, \ldots)$ be an infeasible route, and let $H^+(U) = P(U) \cup \{p \in P \mid (u_1, u_2, \ldots, u_i, p)$ is an infeasible route for some $i = 1, \ldots, |P(U)| - 1\}$. Then, (5) is a stronger version of (3)

$$\sum_{p \in H^+(U)} x_p \leq |P(U)| - 1, \quad \forall\, U \in \mathbb{U}. \tag{5}$$

The validity of (5) stems from the fact that, by construction, any $|P(U)|$ simple routes in $H^+(U)$ that satisfy (2) contain an infeasible route. Alternatively, validity proofs for similar inequalities for the vehicle routing problem with time windows shown in [20] can also be applied to (5).

To obtain additional valid inequalities for $\mathcal{T}$, we exploit some of the TUP's inherent symmetry. As we reverse the order of rounds in a tournament, turning round r into round $4n - 1 - r$, for all $1 \leq r \leq 4n - 2$, we obtain a modified instance of the problem that is equivalent to the original instance. The fundamental difference is that the umpires travel routes in the reverse direction. The sections of the tournament are also reversed, that is section $s' = |S| - s + 1$ of the modified instance contains round $r' = 4n - 1 - r$ if, and only if, section s of the original instance contains round r. Therefore, $P'_{s'}$, the set of simple routes in section s' of the modified instance, contains the reversed simple routes that belong to P_s in the original instance. As a consequence, variables in the formulation of the modified instance are equivalent to the variables for the corresponding reversed route in the formulation of the original instance. Applying this equivalence to the version of (5) for the modified instance, we obtain (6), which is valid for $\mathcal{T}$ in the original instance.

$$\sum_{p \in H^-(U)} x_p \leq |P(U)| - 1, \quad \forall\, U \in \mathbb{U}, \tag{6}$$

where $H^-(U) = P(U) \cup \{p \in P \mid (p, u_i, u_{i+1}, \ldots, u_{|P(U)|})$ is an infeasible route for some $i = 2, \ldots, |P(U)|\}$. Inequalities (5) and (6) are linearly independent, and hence not redundant together. In fact, the computational results in Section 5 indicate that the addition of (6) significantly strengthens the linear relaxation of (1)–(4).

Next, we obtain two additional families of valid inequalities for $\mathcal{T}$ derived from cliques in conflict graphs. Let $s \in S$, $s \neq |S|$, $g \in G_s \cap G_{s+1}$, and define A_{sg} as the graph whose vertices correspond to the simple routes in P_s that end with game g, as well as the simple routes in P_{s+1} that start with g. We denote the vertex of A_{sg} that corresponds to a given simple route p by $v^A_{sg}(p)$. Two vertices of A_{sg}, $v^A_{sg}(p_1)$ and $v^A_{sg}(p_2)$ are adjacent if, and only if, p_1 and p_2 either belong to the same section, or constitute an infeasible route when put together. If we denote the set of cliques in A_{sg} by $\mathbb{A}_{sg}$, (7) is clearly valid for $\mathcal{T}$.

$$\sum_{p \mid v^A_{sg}(p) \in C} x_p \leq 1, \quad \forall\, s \in S, s \neq |S|, g \in G_s \cap G_{s+1}, C \in \mathbb{A}_{sg}. \tag{7}$$

Similarly, let B_s be a graph whose vertices correspond to the simple routes in P_s for a given $s \in S$. We denote the vertex of B_s that corresponds to a given simple route p by $v^B_s(p)$. Two vertices in B_s, $v^B_s(p_1)$ and $v^B_s(p_2)$, are adjacent if, and only if, p_1 and p_2 have a game in common. If we denote the set of cliques in B_s by $\mathbb{B}_s$, (8) is valid for $\mathcal{T}$ because of (2).

$$\sum_{p \mid v^B_s(p) \in C} x_p \leq 1, \quad \forall\, s \in S, C \in \mathbb{B}_s. \tag{8}$$

Constraints (5) and (6) are called *path inequalities*, whereas (7) and (8) are referred to as *clique inequalities*.

4. Separation Routines for Path and Clique Inequalities

Because the number of path and clique inequalities grows exponentially with n, it is impractical to add them all to the model. Instead, we develop separation routines to detect the violation of these inequalities and use them as cutting planes. We start with a few auxiliary results that improve the separation of path inequalities, and describe the two separation routines afterward.

4.1. Auxiliary Results for Path Inequalities

We call an infeasible route *right-minimal* (*left-minimal*) if it becomes feasible once its rightmost (leftmost) simple route is removed. An infeasible route is called *minimal* if it is both left- and right-minimal.

Proposition 1. *If U is not a right-minimal (left-minimal) infeasible route, its corresponding inequality (5) (respectively, (6)) is redundant.*

Proof. Let U be an infeasible route that is not right-minimal, and let Z be the minimal set of simple routes in U whose removal would make it into a right-minimal route U'. If we sum together, for each $p \in Z$, equalities (2) with s being p's section and g being p's first game, and add the result to the inequality (5) corresponding to U', we end up with the inequality (5) corresponding to U. The proof for the left-minimal case is analogous. □

Note that inequalities (5) corresponding to two right-minimal infeasible routes that differ only in their last (rightmost) simple route, are identical. Likewise, two left-minimal infeasible routes that differ only in their first (leftmost) simple route give rise to the same inequality (6). Therefore, we now present modified versions of (5) and (6) that prevent our separation algorithm from generating repeated inequalities. Let $\mathbb{F}$ be the set of feasible routes. Let $\mathbb{F}' = \{F = (f_1, f_2, \dots) \in \mathbb{F} \mid f_{|P(F)|} \notin P_{|S|}\}$ and $\mathbb{F}'' = \{(f_1, f_2, \dots) \in \mathbb{F} \mid f_1 \notin P_1\}$ be the sets of feasible routes that exclude simple routes from the last

and first sections of T, respectively. Consider $F' = (f'_1, f'_2, \dots) \in \mathbb{F}'$, $F'' = (f''_1, f''_2, \dots) \in \mathbb{F}''$, and define $K^+(F') = P(F') \cup \{p \in P \mid (f'_1, f'_2, \dots, f'_i, p)$ is an infeasible route for some $i = 1, \dots, |P(F')|\}$ and $K^-(F'') = P(F'') \cup \{p \in P \mid (p, f''_i, f''_{i+1}, \dots, f''_{|P(F'')|})$ is an infeasible route for some $i = 1, \dots, |P(F'')|\}$. Instead of using (5) and (6), we use (9) and (10), respectively.

$$\sum_{p \in K^+(F)} x_p \leq |P(F)|, \quad \forall\, F \in \mathbb{F}', \tag{9}$$

$$\sum_{p \in K^-(F)} x_p \leq |P(F)|, \quad \forall\, F \in \mathbb{F}''. \tag{10}$$

Notice that (9) and (10) are respectively equivalent to (5) and (6), but the former are defined in terms of feasible routes, whereas the later are defined in terms of infeasible routes. For instance, given a right-minimal (resp. left-minimal) infeasible route U, inequality (5) (resp. (6)) for U is equal to (9) (resp. (10)) for the feasible route obtained by removing the last (resp. first) game in U. In addition, inequality (9) (resp. (10)) for a given F eliminates only right-minimal (resp. left-minimal) infeasible routes (see Proposition 1) and there is a one-to-one correspondence between an inequality (9) or (10) and a feasible route F from $\mathbb{F}'$ or $\mathbb{F}''$ because $K^+(F)$ and $K^-(F)$ are uniquely determined from F.

Although our separation routines look for violations of (9) and (10), these cuts can be dense, potentially leading to decreased computational performance. Therefore, we add equivalent, sparser versions of (9) and (10) to the formulation, which are given by (11) and (12), respectively.

$$\sum_{p \in \tilde{K}^+(F')} x_p - x_{f'_1} \geq 0, \quad \forall\, F' \in \mathbb{F}', \tag{11}$$

$$\sum_{p \in \tilde{K}^-(F'')} x_p - x_{f''_{|P(F'')|}} \geq 0, \quad \forall\, F'' \in \mathbb{F}'', \tag{12}$$

where $\tilde{K}^+(F') = \{p \in (P \setminus P(F')) \mid (f'_1, f'_2, \dots, f'_i, p)$ is a feasible route for some $i = 1, \dots, |P(F')|\}$, and $\tilde{K}^-(F'') = \{p \in (P \setminus P(F'')) \mid (p, f''_i, f''_{i+1}, \dots, f''_{|P(F'')|})$ is a feasible route for some $i = 1, \dots, |P(F'')|\}$. Intuitively, (11)–(12) are sparser than (9)–(10) because the $\tilde{K}^+$ and $\tilde{K}^-$ sets used in the former contain simple routes that yield feasibility, which tend to be less numerous than the infeasibility-inducing simple routes of the K^+ and K^- sets used in the latter. Given F', we obtain (11) by multiplying (9) by -1 and adding to the result, for all $i = 1, \dots, |P(F')|$, equalities (2) with $s = i + 1$ and g equal to the last game in f'_i. Analogously, we can combine the negation of (10) with (2) to obtain (12).

4.2. Separation Routine for Path Inequalities

Algorithm 1 describes the separation routine for (9) for routes that violate TUP constraints *(iv)* or *(v)*. Given a solution x^* (e.g. from the linear relaxation of the current branch-and-bound node), we enumerate the routes in $\mathbb{F}'$ looking for violations of (9) by calling the procedure Sep-Frwd-Freq-Rec for each section $s \in S$, except for the last one. Sep-Frwd-Freq-Rec recursively checks inequalities (9) for routes that start in s, which could take exponential time. Hence, we strategically skip some routes, as described next.

Typically, most x variables are either zero or very close to zero in the input solution x^*, contributing very little to a potential violation of (9). Hence, we disregard routes whose variables have values below 0.001 by using the following sets inside Sep-Frwd-Freq-Rec: $P^+ = \{p \in P \mid x^*_p \geq 0.001\}$, $P^+_s = P_s \cap P^+$, and $P^+_{sg} = P_{sg} \cap P^+_s$. The steps in lines 8–16 of Algorithm 1 enumerate all feasible routes obtained by adding simple routes from P^+_{sg} (or P^+_s when F is empty) to the end of F (creating F'). If

F' violates (9) (line 17) by at least 0.009 (to promote reasonable progress in the lower bound value), the corresponding inequality (11) is added to the formulation (line 18). In lines 20 and 21, routes derived from F' are enumerated in a recursive fashion only when the next section is not the last ($s+1 < |S|$) and when $\sum_{p' \in K^+(F') \cap P^+} x^*_{p'} > |P(F')| - 1 + 0.009$. If the previous inequality is not satisfied, routes that extend F' cannot satisfy the condition in line 17. To see why, consider a route F'' obtained by adding ℓ simple routes at the end of F'. The inequality in line 17 for F'' will have a right-hand side equal to $|P(F')| + \ell + 0.009$, and its left-hand side will have variables from the routes in $K^+(F') \cap P^+$ plus additional variables whose values in x^* add up to no more than $\ell + 1$, which results in the inequality of line 20 after canceling ℓ on both sides. This check prevents the unnecessary enumeration of a large number of routes. Summations calculated in lines 17 and 20 are available to subsequent recursive calls to allow for incremental updates, saving additional computation time.

Algorithm 1 Separation routine for (9) for routes that violate TUP constraints *(iv)* or *(v)*.

1: **procedure** SEP-FRWD-FREQ(solution x^*)
2: **for all** $s \in S, s \neq |S|$ **do**
3: SEP-FRWD-FREQ-REC(x^*, (), s);
4: **end for**
5: **end procedure**
6:
7: **procedure** SEP-FRWD-FREQ-REC(solution x^*, route F, section s)
8: **if** $F = ()$ **then**
9: Let $E = P^+_s$;
10: **else**
11: Let g be the last game in F;
12: Let $E = P^+_{sg}$;
13: **end if**
14: **for all** $p \in E$ **do**
15: Let F' be F with p added to its end;
16: **if** F' is a feasible route **then**
17: **if** $\sum_{p' \in K^+(F') \cap P^+} x^*_{p'} > |P(F')| + 0.009$ **then**
18: Add to the formulation inequality (11) for F';
19: **end if**
20: **if** $s+1 < |S|$ and $\sum_{p' \in K^+(F') \cap P^+} x^*_{p'} > |P(F')| - 1 + 0.009$ **then**
21: SEP-FRWD-FREQ-REC(x^*, F', $s+1$);
22: **end if**
23: **end if**
24: **end for**
25: **end procedure**

We separate (10) for routes that violate TUP constraints *(iv)* or *(v)* with simple modifications to SEP-FRWD-FREQ and SEP-FRWD-FREQ-REC, creating their respective counterparts SEP-BCWD-FREQ and SEP-BCWD-FREQ-REC. SEP-BCWD-FREQ calls SEP-BCWD-FREQ-REC for each section $s \in S$, except for the first. SEP-BCWD-FREQ-REC also enumerates routes, but considering sections in reverse order, with the following modifications to the steps in Algorithm 1. Game g becomes the first game of F in line 11. Route p gets inserted at the beginning of F in line 15. Inequalities in lines 17, 18, and

20 are modified to be consistent with (10) and (12). The first condition in line 20 becomes $s-1>1$. Finally, the third parameter of the recursive call in line 21 becomes $s-1$.

Empirically, inequalities (9) and (10) that eliminate long infeasible routes are not worth separating, when it comes to violations of *(iv)* or *(v)*, unless they are minimal (i.e. both left-minimal and right-minimal). Let $\tilde{U}=(\tilde{u}_1,\tilde{u}_2,\dots)$ be an infeasible route that only violates either *(iv)* or *(v)*. If its internal route $(\tilde{u}_2,\cdots,\tilde{u}_{|P(\tilde{U})|-1})$ traverses at least $q_{\max}-1$ rounds, $\tilde{U}$ cannot be minimal, where $q_{\max}=\max\{q_1,q_2\}$. Therefore, we define subsets of $\mathbb{F}'$ and $\mathbb{F}''$ for (9) and (10), respectively, which exclude inequalities that only eliminate non-minimal routes violating *(iv)* or *(v)*. Given a route $Q=(q_1,q_2,\dots)$, let $I'(Q)$ and $I''(Q)$ be the number of rounds visited by routes $(q_2,q_3,\dots,q_{|P(Q)|})$ and $(q_1,q_2,\dots,q_{|P(Q)|-1})$, respectively. We define $\tilde{\mathbb{F}}'=\{F\in\mathbb{F}'\mid I'(F)<q_{\max}-1\}$ and $\tilde{\mathbb{F}}''=\{F\in\mathbb{F}''\mid I''(F)<q_{\max}-1\}$, and implement separation routines Sep-Frwd-Freq-Mnl and Sep-Frwd-Freq-Mnl-Rec (resp. Sep-Bcwd-Freq-Mnl and Sep-Bcwd-Freq-Mnl-Rec) to separate inequalities (9) (resp. (10)) for routes in $\tilde{\mathbb{F}}'$ (resp. $\tilde{\mathbb{F}}''$). Routine Sep-Frwd-Freq-Mnl-Rec is obtained by modifying Sep-Frwd-Freq-Rec, as follows. The extra condition $I'(F')+w<q_{\max}-1$ is added to the "if" in line 20 and, of course, the recursive call in line 21 becomes Sep-Frwd-Freq-Mnl-Rec. Routine Sep-Bcwd-Freq-Mnl-Rec is similarly obtained from Sep-Bcwd-Freq-Rec by adding the condition $I''(F')+w<q_{\max}-1$ before its recursive call. Routines Sep-Frwd-Freq-Mnl and Sep-Bcwd-Freq-Mnl are similar to Sep-Frwd-Freq and Sep-Bcwd-Freq, but call Sep-Frwd-Freq-Mnl-Rec and Sep-Bcwd-Freq-Mnl-Rec, respectively.

We now turn our attention to violations of TUP constraint *(iii)*. The feasible routes in $\mathbb{F}'$ (resp. $\mathbb{F}''$) corresponding to inequalities (9) (resp. (10)) that eliminate routes violating *(iii)* are those that miss the home venue of at least one team and include simple routes from each of the sections $1,2,\dots,|S|-1$ (resp. $2,3,\dots,|S|$). These inequalities can be separated by defining routine Sep-Frwd-Visit-Rec (resp. Sep-Bcwd-Visit-Rec) as a variation of Sep-Frwd-Freq-Rec (resp. Sep-Bcwd-Freq-Rec). Essentially, the violated inequality should only be added to the model when $s=|S|-1$ (resp. $s=2$) and F' excludes the home venue of at least one team. Empirically, however, the amount of improvement to the lower bound obtained by separating (9) and (10) for routes that violate *(iii)* was not worth the extra time required by the separation routines. Therefore, we decided to separate (13), instead:

$$\sum_{p\in K'(F')} x_p \le |F'|, \quad \forall\ F'=(f'_1,f'_2,\dots)\in\mathbb{F}', \tag{13}$$

where $K'(F')=P(F')\cup\{p\in P\mid (f'_1,f'_2,\dots,f'_{|P(F')|},p)\text{ is an infeasible route}\}$. Despite being weaker than (9)–(10) (since $K'(F')\subset K^+(F')$ when $|F'|\ge 2$), (13) can be separated by enumerating a lot fewer routes, which reduces computational effort considerably. Algorithm 2 describes the separation routine for (13). It is similar to Sep-Frwd-Visit-Rec, differing with respect to the summations calculated in lines 12 and 16 of Algorithm 2. Even though it looks for violations of (13), Sep-Frwd-Visit-Weak-Rec adds to the formulation the stronger inequality (11), as is done in Algorithm 1.

Algorithm 2 Separation routine for (13) for routes that violate TUP constraint *(iii)*.

1: **procedure** SEP-FRWD-VISIT-WEAK-REC(solution x^*, route F, section s)
2: **if** $F = ()$ **then**
3: Let $E = P_s^+$;
4: **else**
5: Let g be the last game in F;
6: Let $E = P_{sg}^+$;
7: **end if**
8: **for all** $p \in E$ **do**
9: Let F' be F with p added to its end;
10: **if** F' is a feasible route **then**
11: **if** $s = |S| - 1$ and F' does not visit a team at home **then**
12: **if** $\sum_{p' \in K'(F') \cap P^+} x^*_{p'} > |F'| + 0.009$ **then**
13: Add to the formulation inequality (11) for F';
14: **end if**
15: **end if**
16: **if** $s + 1 < |S|$ and $\sum_{p' \in K'(F') \cap P^+} x^*_{p'} > |F'| - 1 + 0.009$ **then**
17: SEP-FRWD-VISIT-WEAK-REC(x^*, F', $s + 1$);
18: **end if**
19: **end if**
20: **end for**
21: **end procedure**

Although the separation routines described so far are recursive, which makes them easier to understand, they are implemented as non-recursive procedures to improve their running time.

4.3. Separation Routine for Clique Inequalities

We separate (7) and (8) as follows. Given a solution x^*, we start by building graphs A_{sg} and B_s (see Section 3.2). After assigning weight x^*_p to each vertex $v^A_{sg}(p)$ and $v^B_s(p)$, we look for a maximum-weight clique in either graph. A violated inequality exists if, and only if, the maximum-weight clique found has total weight greater than 1. Algorithms 3 and 4 describe the separation routines for (7) and (8), respectively. We use the Cliquer solver [21] to look for maximum-weight cliques. Because this is an NP-hard problem [22], we reduce the sizes of our two graphs by only creating vertices for simple routes p with $x^*_p \geq 0.01$. The subgraphs of A_{sg} and B_s obtained this way are denoted by $\tilde{A}_{sg}$ and $\tilde{B}_s$. Additionally, as Cliquer only works with integer weights, the weight of vertices $v^{\tilde{A}}_{sg}(p)$ and $v^{\tilde{B}}_s(p)$ is converted to $\lfloor 100 x^*_p \rfloor$, and since finding the maximum-weight clique can be very time-consuming, we stop running Cliquer as soon as a clique of weight greater than or equal to 101 is found. The strongest inequalities (7) and (8) are those associated with maximal cliques in A_{sg} and B_s, respectively. Therefore, once a clique C is found in one of the subgraphs, we scan the corresponding original graph looking for vertices not in C that happen to be adjacent to all vertices of C. If such a vertex exists, it is included in C and the procedure continues for the remaining unverified vertices and the updated C. After scanning all vertices, the violated inequality is added to the formulation. (See lines 8-12 in Algorithm 3, and lines 7-11 in Algorithm 4.)

Algorithm 3 Separation routine for (7).

procedure SEP-CLIQUE-ADJT-SECTION(solution x^*)
  for all $s \in S, s \neq |S|$ **do**
    for all $g \in G_s \cap G_{s+1}$ **do**
      Build graph $\tilde{A}_{sg}$ for routes $p \in P_{sg} \cup P_{(s+1)g}$ with $x^*_p \geq 0.01$;
      Assign weight $\lfloor 100 x^*_p \rfloor$ to each vertex $v^{\tilde{A}}_{sg}(p)$ of $\tilde{A}_{sg}$;
      Run the Cliquer solver on the weighted graph $\tilde{A}_{sg}$;
      if a clique C with weight greater than or equal to 101 is found **then**
        for all $p \in P_{sg} \cup P_{(s+1)g}$ **do**
          if $v^{\tilde{A}}_{sg}(p) \notin C$ and $v^{A}_{sg}(p)$ is adjacent, in A_{sg}, to all vertices of C **then**
            Add $v^{\tilde{A}}_{sg}(p)$ to C;
          end if
        end for
        Add to the formulation inequality (7) for C;
      end if
    end for
  end for
end procedure

Algorithm 4 Separation routine for (8).

procedure SEP-CLIQUE-SAME-SECTION(solution x^*)
  for all $s \in S$ **do**
    Build graph $\tilde{B}_s$ for routes $p \in P_s$ with $x^*_p \geq 0.01$;
    Assign weight $\lfloor 100 x^*_p \rfloor$ to each vertex $v^{\tilde{B}}_s(p)$ of $\tilde{B}_s$;
    Run the Cliquer solver on the weighted graph $\tilde{B}_s$;
    if a clique C with weight greater than or equal to 101 is found **then**
      for all $p \in P_s$ **do**
        if $v^{\tilde{B}}_s(p) \notin C$ and $v^{B}_s(p)$ is adjacent, in B_s, to all vertices of C **then**
          Add $v^{\tilde{B}}_s(p)$ to C;
        end if
      end for
      Add to the formulation inequality (8) for C;
    end if
  end for
end procedure

5. Computational Results

We perform computational experiments to show the relevance of the cuts from Section 3.2, to assess the impact of parameter w (the length of umpire trip sequences) on the lower bounds produced by the relaxation of our IP model, and to compare the performance of our branch-and-cut algorithm with other methods in literature.

Our implementation is done in C++ using ILOG CPLEX's Callable Library version 12.6.1, with GCC 4.6.3 as the compiler. All experiments are carried out on a machine equipped with an Intel Xeon X3430 2.40GHz processor and 8GB of RAM, running Linux Ubuntu 12.04.3.

The problem instances we use come from the TUP benchmark [18], also present in the recently created automated benchmark [23], which includes tournaments ranging from 4 to 32 teams. We do not consider instances with fewer than 14 teams because they are easily solved by the current state-of-the-art methods. Instance names start with the number of teams in the tournament, optionally followed by a letter. The presence of a letter indicates a variation of the original instance (without the letter), keeping the same tournament but changing the distance matrix. We consider the usual values of q_1 and q_2 adopted in the TUP literature and, in addition, include $q_1 = q_2 = 5$ for the instances with 26, 28, 30, and 32 teams, which are also studied in [7].

Before we proceed, two aspects are worth emphasizing. First, although the number of variables in our formulation grows exponentially in w, we enumerate all of them *a priori* and add them to the model from the beginning (see Algorithm 5), rather than resorting to on-the-fly variable generation (see Appendix A for the number of variables in our test instances). The time spent with this enumeration is already included in the solution times reported in this section and never exceeds 15 seconds.

Algorithm 5 Enumeration of the model's variables.

procedure ENUM-VARS
  for all $s \in S$ **do**
    ENUM-VARS-REC(s, (), 0); ▷ generates all trips in P_s
  end for
end procedure

procedure ENUM-VARS-REC(section s, simple route p, simple route length ℓ);
  ▷ Append games to the simple route p of length ℓ until it reaches the size of section s
  if $\ell = w$ **or** $s(w-1) + 1 + \ell > 4n - 2$ **then**
    Add variable x_p to the formulation;
  else
    for all games g in round $s(w-1) + 1 + \ell$ **do**
      Let p' be p with g appended to it;
      if p' is a feasible simple route **then**
        ENUM-VARS-REC(s, p', $\ell + 1$);
      end if
    end for
  end if
end procedure

A second relevant aspect refers to the way we compare our running times against those in [7, 8, 9], as their experiments were conducted in computational environments different from ours. Rather than trying to establish a reliable speed ratio between two distinct CPUs (a very difficult task), for the purpose of assessing our results it suffices to know that the machine we used is slower than all of the others, as can be verified, for example, on the following web site: www.cpubenchmark.net (accessed in July, 2015). Therefore, when we say that "we found a better lower bound, and X times faster, than the one in [*citation*]", it actually means that the true speed-up is even greater than X. If the exact CPU speed ratio was used in our comparisons, the conclusions could only become more favorable to our method. With these observations in mind, we continue with the analysis of the results.

5.1. The Impact of Our Valid Inequalities

We start by evaluating different combinations of the valid inequalities presented in Section 3.2 to assess their impact on solution times and lower bound strength. We solve the linear relaxation of our IP model six times for each instance, each time using a procedure consisting of distinct ordered subsets of the separation routines from Section 4, chosen empirically, as follows:

Sep1: Sep-Frwd-Freq, Sep-Frwd-Visit-Rec.

Sep2: Sep-FB-Freq, Sep-FB-Visit-Rec.

Sep3: Sep-FB-Freq-Mnl, Sep-FB-Visit-Rec.

Sep4: Sep-FB-Freq-Mnl, Sep-Frwd-Visit-Weak-Rec.

Sep5: Sep-FB-Freq-Mnl, Sep-Frwd-Visit-Weak-Rec, Sep-Clique-Adjt-Section.

Sep6: Sep-FB-Freq-Mnl, Sep-Frwd-Visit-Weak-Rec, Sep-Clique-Adjt-Section, Sep-Clique-Same-Section.

Routine Sep-FB-Freq corresponds to the execution of Sep-Frwd-Freq followed by Sep-Bcwd-Freq, Sep-FB-Freq-Mnl corresponds to Sep-Frwd-Freq-Mnl followed by Sep-Bcwd-Freq-Mnl, and Sep-FB-Visit-Rec corresponds to Sep-Frwd-Visit-Rec followed by Sep-Bcwd-Visit-Rec.

The above procedures (combinations of separation routines) are used in a cutting plane algorithm as follows. We start by solving the linear programming (LP) relaxation of a model that only includes (1) and (2). Then, given $i \in [1, 6]$, procedure Sep$i$ is applied to the optimal solution found, with its separation routines executed in the order in which they appear above. When a routine inside Sep$i$ finishes its execution, the next routine is executed only if the previous one did not add any violated inequalities to the model. Otherwise, Sep$i$ terminates, the model is re-optimized (with the dual Simplex method), and Sep$i$ is called again. This process is repeated until no more violated inequalities are found. Because Sep-FB-Freq, Sep-FB-Freq-Mnl, and Sep-FB-Visit-Rec consist of two routines each, they receive special treatment: their second routine is always executed, even when their first routine adds inequalities to the model.

Given a w, for each $i \in [1, 6]$ we denote by $\mathcal{M}_w^i$ the IP model comprising (1), (2), (4), and all the inequalities separated by Sep$i$. The linear relaxation of $\mathcal{M}_w^i$, obtained by dropping (4), is denoted by $\mathcal{M}_w^{Ri}$. We solve each linear relaxation $\mathcal{M}_w^{Ri}$ with our cutting plane algorithm and report the lower bounds and solution times (limited to 3 hours) in Table 1. In these experiments we use the same values for w adopted in the branch-and-cut experiments (see Section 5.3 for explanations).

We now compare the different linear relaxations based on the results in Table 1. $\mathcal{M}_w^{R1}$ comprises (1), (2) and (9), whereas $\mathcal{M}_w^{R2}$ is equal to $\mathcal{M}_w^{R1}$ plus (10). Including (10) significantly improves the lower bounds, increasing them by about 1300 to 5900 miles (0.9 to 3.2%) on instances with up to 16 teams, and by about 5100 to 12600 miles (1.8 to 3%) on instances with more than 16 teams. On the other hand, solution time increases up to 6.23 times on all but one 16-team instance, and up to 3.6 times on the remaining instances. Instance 16 with $q_1 = 8$ and $q_2 = 2$ ends up taking 13.7 times longer to solve once (10) is included. $\mathcal{M}_w^{R3}$ differs from $\mathcal{M}_w^{R2}$ only with respect to (9) and (10). In $\mathcal{M}_w^{R3}$, we disregard some inequalities in (9) and (10) that eliminate non-minimal paths violating *(iv)* or *(v)*, as described in Section 4. $\mathcal{M}_w^{R3}$ solves up to twice as fast as $\mathcal{M}_w^{R2}$ (1.37 times faster on average), whereas the lower bounds given by the former are at most 174 miles less than those of the latter, which is negligible. Because our heuristic separation routine disregards variables with value less than 0.001, $\mathcal{M}_w^{R3}$ actually

Inst.	q_1	q_2	w	Lower bound						Solution time (seconds)					
				$\mathcal{M}_w^{R1}$	$\mathcal{M}_w^{R2}$	$\mathcal{M}_w^{R3}$	$\mathcal{M}_w^{R4}$	$\mathcal{M}_w^{R5}$	$\mathcal{M}_w^{R6}$	$\mathcal{M}_w^{R1}$	$\mathcal{M}_w^{R2}$	$\mathcal{M}_w^{R3}$	$\mathcal{M}_w^{R4}$	$\mathcal{M}_w^{R5}$	$\mathcal{M}_w^{R6}$
14	7	3	4	151371.3	153680.4	153621.9	153621.9	154863.3	**155041.5**	0.28	0.74	0.46	0.46	1.78	3.07
14	6	3	4	150721.9	152670.6	152670.5	152670.5	153927.7	**154187.7**	0.21	0.34	0.25	0.24	0.86	1.48
14	5	3	4	149963.0	151792.7	151792.7	151792.7	152928.3	**153095.0**	0.11	0.20	0.12	0.12	0.26	0.44
14A	7	3	4	145138.1	147900.5	147872.8	147872.8	148706.3	**149081.4**	0.24	0.51	0.31	0.30	1.00	2.62
14A	6	3	4	144470.1	147039.7	147036.9	147036.9	147937.8	**148428.9**	0.14	0.30	0.20	0.20	0.52	1.59
14A	5	3	4	143973.3	146259.4	146259.4	146259.4	147065.8	**147421.0**	0.09	0.18	0.10	0.09	0.21	0.57
14B	7	3	4	145404.2	147263.1	147203.7	147201.9	148609.1	**148776.4**	0.23	0.77	0.46	0.42	1.57	2.49
14B	6	3	4	144787.5	146422.8	146422.8	146422.0	147785.0	**147959.1**	0.17	0.42	0.27	0.25	0.84	1.42
14B	5	3	4	144191.9	145529.6	145529.6	145529.6	146819.5	**147138.0**	0.09	0.20	0.10	0.10	0.27	0.55
14C	7	3	4	143650.8	146213.7	146039.7	146036.3	147415.5	**147686.6**	0.47	0.77	0.47	0.44	1.83	3.55
14C	6	3	4	142792.7	145005.7	145005.5	145005.5	146477.5	**146644.6**	0.15	0.37	0.26	0.26	1.05	2.02
14C	5	3	4	142150.3	144305.7	144305.8	144305.8	145776.0	**145953.5**	0.11	0.14	0.11	0.11	0.31	0.55
16	8	4	10	171712.1	176495.3	176495.7	176391.6	181095.3	**182696.7**	20.88	130.07	108.95	91.91	867.52	2877.17
16	8	2	8	147603.9	150068.3	150065.1	150055.5	**152853.0**	**152853.0**	106.53	1455.42	1184.43	978.76	10800.00	10800.00
16	7	3	4	148826.5	151856.2	151838.8	151838.8	157223.4	**157377.2**	0.86	2.44	1.37	1.35	10.26	13.34
16	7	2	4	141176.4	143872.7	143864.3	143864.3	147669.1	**147853.3**	0.65	1.64	1.22	1.20	7.84	10.14
16A	8	4	10	184979.7	188739.3	188739.9	188649.9	194032.1	**195581.3**	26.15	148.27	127.81	101.74	1389.07	3151.16
16A	8	2	8	157507.5	159312.5	159312.4	159311.4	**164893.2**	**164893.2**	151.92	626.02	515.39	501.40	10800.00	10800.00
16A	7	3	4	164939.1	167559.7	167539.7	167539.7	169767.1	**169970.9**	0.93	2.28	1.29	1.28	8.39	12.56
16A	7	2	4	155891.5	158068.2	158028.1	158028.1	160733.1	**160905.6**	0.60	1.32	0.92	0.91	6.57	9.73
16B	8	4	10	192098.3	197791.3	197790.3	197764.2	202768.2	**203952.1**	33.71	161.88	130.53	118.79	1262.69	2759.63
16B	8	2	8	160446.3	162696.2	162696.2	162696.2	**167241.3**	**167241.3**	139.06	282.94	253.69	253.35	10800.00	10800.00
16B	7	3	4	161769.3	165175.7	165144.9	165144.9	169537.3	**169617.5**	0.87	2.02	1.25	1.25	10.78	12.85
16B	7	2	4	155095.8	157774.9	157755.7	157755.7	162711.1	**162936.6**	0.81	1.82	1.25	1.24	7.70	12.24
16C	8	4	10	184697.1	190615.5	190615.7	190556.8	195863.7	**197220.7**	28.44	96.56	78.94	66.35	595.89	1511.01
16C	8	2	8	161294.0	164240.8	164241.2	164233.7	167196.7	**167341.3**	123.97	400.58	355.47	327.12	7996.15	10800.00
16C	7	3	4	164417.6	166988.9	166930.6	166930.6	169894.2	**169967.0**	0.96	2.57	1.54	1.53	10.13	12.91
16C	7	2	4	157474.4	159745.2	159733.8	159733.8	162813.5	**162903.3**	0.72	1.63	1.28	1.26	9.10	12.26
18	9	4	9	196674.4	201793.5	201795.9	201772.9	205489.9	**205743.8**	359.99	1294.34	1168.20	1086.19	7949.31	10800.00
20	10	5	10	238778.9	245908.8	**245960.6**	245907.2	245907.2	245907.2	3582.13	10800.00	10800.00	9134.04	10800.00	10800.00
22	11	5	7	260340.7	**266460.9**	266423.7	266423.5	266423.5	266423.5	3413.92	10800.00	9790.81	9764.14	10800.00	10800.00
24	12	6	7	292168.5	**297586.7**	297556.7	297556.7	297556.7	297556.7	10800.00	10800.00	10800.00	10800.00	10800.00	10800.00
26	13	6	6	327716.1	333517.8	**333678.9**	**333678.9**	**333678.9**	**333678.9**	8498.39	10800.00	10800.00	10800.00	10800.00	10800.00
26	5	5	4	307130.4	313683.9	313683.7	313683.7	323346.4	**323684.2**	45.25	121.77	76.61	76.51	823.52	1140.39
28	14	7	5	364692.1	374601.6	**374619.7**	**374619.7**	**374619.7**	**374619.7**	6116.04	10800.00	10800.00	10800.00	10800.00	10800.00
28	5	5	4	348811.5	355275.6	355275.5	355275.5	362132.7	**362585.2**	76.80	182.79	133.63	133.56	1140.83	1836.80
30	15	7	5	413149.5	421985.0	**422012.1**	**422012.1**	**422012.1**	**422012.1**	10119.78	10800.00	10800.00	10800.00	10800.00	10800.00
30	5	5	4	397669.4	404757.8	404757.8	404757.8	414403.9	**414865.9**	131.91	304.63	238.15	238.06	2305.95	3232.17
32	16	8	5	453106.0	**462944.9**	**462944.9**	**462944.9**	**462944.9**	**462944.9**	10800.00	10800.00	10800.00	10800.00	10800.00	10800.00
32	5	5	4	429802.6	442418.9	442418.9	442418.9	455453.6	**455885.7**	266.27	675.47	532.11	531.13	5654.81	7369.47

Table 1: Lower bounds and solution times for the linear relaxations $\mathcal{M}_w^{Ri}$ for $i \in [1, 6]$ (best lower bounds in bold).

yields greater lower bounds than $\mathcal{M}_w^{R2}$ on some instances (e.g. 16C with $q_1 = 8$ and $q_2 = 2$, and instance 18). $\mathcal{M}_w^{R4}$ includes all the constraints in $\mathcal{M}_w^{R3}$, except for those inequalities in (9) and (10) that eliminate paths violating *(iii)*, which are replaced by the inequalities in (11) that induce the satisfaction of (13). $\mathcal{M}_w^{R4}$ solves slightly faster than $\mathcal{M}_w^{R3}$ (1.05 times on average), while the lower bounds produced by the former are at most 105 miles shorter than those by the latter, which is negligible. $\mathcal{M}_w^{R5}$ is equal to $\mathcal{M}_w^{R4}$ plus (7). Adding (7) leads to significant improvements to the lower bounds, increasing them by about 800 to 5500 miles (0.6 to 3.5%) on the instances with at most 16 teams, and by about 3700 to 13000 miles (1.8 to 3%) on the instances with more than 16 teams, with some exceptions: instances with 20 or more teams and $q_1 = n$, whose lower bounds remain the same because no inequalities (7) are found to be violated within the 3-hour time limit. In terms of solution time, however, $\mathcal{M}_w^{R5}$ solves up to 4.2 times slower than $\mathcal{M}_w^{R4}$ on 14-team instances, from 11 to 42.6 times slower on 16-team instances with $q_1 = 8$ and $q_2 = 2$, and 13.7 times slower on the remaining instances. $\mathcal{M}_w^{R6}$ includes all the inequalities in $\mathcal{M}_w^{R5}$ plus (8). Almost all of our best lower bounds come from $\mathcal{M}_w^{R6}$, except when its execution reaches the time limit, where it performs as well as $\mathcal{M}_w^{R4}$ or $\mathcal{M}_w^{R5}$, or slightly better/worse than $\mathcal{M}_w^{R2}$ or $\mathcal{M}_w^{R3}$, since the latter include different sets of inequalities from (9) and (10). Compared with $\mathcal{M}_w^{R5}$, $\mathcal{M}_w^{R6}$'s lower bounds are up to 1600 miles (0.9%) greater on 16-team instances with $q_1 = 8$ and $q_2 = 4$, and up to 490 miles (0.3%) greater on the remaining instances. At first sight, since $\mathcal{M}_w^{R6}$ solves up to 3.32 times slower

than $\mathcal{M}_w^{R5}$ and yields small lower-bound improvements, there seems to be no reason to advocate using (8). Nevertheless, preliminary experiments with the branch-and-cut presented in Section 5.3 indicate that (8) contributes to a significant reduction in the size of the search tree. As a consequence, we decide to focus on $\mathcal{M}_w^{R6}$ in our subsequent experiments.

5.2. The Impact of Parameter w

Tables 2 and 3 present, respectively, the lower bounds and solution times for $\mathcal{M}_w^{R6}$ as w varies between 2 and 10. We only solve LP models with up to 5 million variables and set a time limit of 3 hours. (The number of variables in the model for each instance and value of w tested can be found in Appendix A.) Tables 2 and 3 also include, on the right-hand side, the lower bounds and solution times for the relaxations of the best IP models published at the time of this writing: the dual ascent method in [9] that solves a Lagrangian relaxation of a network flow model (RNFM), the column generation method in [9] and [8] that solves the linear relaxation of a set partitioning model (RSPM), and the column generation method in [9] that solves the same set partitioning relaxation, but with additional cutting planes (RSPCM). Although the lower bounds reported in [9] and [8] for the RSPM are the same, the corresponding solution times are different and appear, respectively, in columns RSPM[9] and RSPM[8] of Table 3.

Inst.	q_1	q_2	$\mathcal{M}_2^{R6}$	$\mathcal{M}_3^{R6}$	$\mathcal{M}_4^{R6}$	$\mathcal{M}_5^{R6}$	$\mathcal{M}_6^{R6}$	$\mathcal{M}_7^{R6}$	$\mathcal{M}_8^{R6}$	$\mathcal{M}_9^{R6}$	$\mathcal{M}_{10}^{R6}$	RNFM	RSPM	RSPCM
14	7	3	152596.7	153805.9	155041.5	156029.2	156644.5	156728.2	**156799.3**	156718.5	156751.4	150934.5	156439.3	157016.3
14	6	3	152423.5	153401.3	154187.7	154715.3	154851.6	154925.8	154925.2	**154975.5**	154924.1	150909.7	154439.9	155252.6
14	5	3	151916.2	152707.7	153095.0	153173.7	153210.3	153196.0	**153250.3**	153230.9	153216.0	150621.1	152941.3	153486.5
14A	7	3	146776.2	147937.8	149081.4	149982.8	150408.5	**150548.7**	150469.7	150443.3	150471.1	145049.6	149992.7	150707.9
14A	6	3	146685.9	147345.6	148428.9	148655.2	148977.7	**149096.8**	148980.4	148962.8	149054.9	145018.1	148168.7	149283.7
14A	5	3	146362.1	146727.2	147421.0	147528.0	147704.9	**147728.1**	147592.5	147616.4	147622.2	144884.9	147097.5	147999.3
14B	7	3	146648.9	147822.9	148776.4	149706.9	150163.7	150390.9	150296.3	**150533.7**	150488.5	144053.6	149767.0	150699.7
14B	6	3	146531.2	147337.4	147959.1	148733.0	148842.6	148938.8	148880.3	148964.4	**149070.3**	144054.1	148243.9	149240.8
14B	5	3	146136.8	146801.8	147138.0	147369.5	147455.9	147366.1	**147617.9**	147463.4	147598.1	143866.8	146846.2	147784.8
14C	7	3	145208.9	146438.2	147686.6	148550.6	148934.8	149106.3	149076.5	**149179.0**	149138.1	143276.4	148613.2	149489.7
14C	6	3	145150.2	145900.2	146644.6	147075.3	147140.5	147137.9	147186.3	**147250.3**	147236.6	143233.8	146774.6	147644.9
14C	5	3	144847.6	145536.4	145953.5	146105.9	146161.2	146210.1	**146230.1**	146212.4	146145.2	143149.5	145794.4	146597.9
16	8	4	157573.8	163238.6	167294.2	174986.5	178375.3	179509.2	181402.9	181063.4	**182696.7**	152507.6	184187.6	185056.8
16	8	2	144224.4	146585.4	148506.6	150717.2	152104.1	152027.0	**152853.0**	149662.8		142134.8	155045.2	155712.5
16	7	3	153900.5	155923.5	157377.2	158188.3	158460.0	158466.9	**158635.7**	158626.9	158222.0	150532.2	158257.4	158883.4
16	7	2	144176.9	146475.8	147853.3	148369.0	148534.3	148574.7	**148629.7**	147865.3		142145.2	148341.8	148980.7
16A	8	4	170424.0	176358.3	179870.1	187519.0	190972.0	192643.0	194233.8	194015.0	**195581.3**	164945.9	198969.7	200007.6
16A	8	2	157765.9	159929.7	161300.4	164116.1	164511.7	164512.8	**164893.2**	163339.0		155641.5	166575.5	167360.0
16A	7	3	166719.6	168505.7	169970.9	170615.1	170880.4	171103.7	**171251.7**	171205.0	170763.1	162700.0	170575.1	171426.6
16A	7	2	157682.6	159756.9	160905.6	161525.5	161617.4	161715.0	**161731.9**	160634.4		155963.5	161571.2	161975.1
16B	8	4	170001.6	177606.1	182450.8	192774.4	196594.2	198726.5	201609.1	202655.0	**203952.1**	165008.4	207505.4	208496.8
16B	8	2	157967.4	161411.3	163379.5	166207.6	166892.4	166889.7	**167241.3**	165511.6		156402.2	169363.4	170040.3
16B	7	3	165010.3	167814.3	169617.5	170351.2	170837.9	170993.9	**171110.0**	171083.2	170860.4	162073.7	170632.5	171280.6
16B	7	2	157936.9	161267.6	162936.6	163536.6	163812.7	163814.9	**163894.9**	162738.2		156442.1	163539.7	164160.9
16C	8	4	171801.3	176480.1	180684.9	187282.9	191314.0	192331.8	195932.0	195866.4	**197220.7**	167256.9	200682.6	201107.5
16C	8	2	160069.3	161761.6	163307.9	165357.6	166335.5	166488.9	**167341.3**	165813.3		158947.2	168783.6	169270.9
16C	7	3	166754.7	168194.5	169967.0	170841.1	171373.7	171366.7	**171596.4**	171454.5	171225.4	164380.8	171216.0	171827.6
16C	7	2	160006.2	161596.2	162903.3	163393.5	163858.2	**163907.0**	163771.9	163122.1		158906.2	163850.8	164182.8
18	9	4	187132.4	192865.6	196085.7	200213.7	201992.9	203102.9	203813.8	**205743.8**	204027.0	181430.7	212121.6	212793.6
20	10	5	220179.1	229339.6	234897.0	237819.0	243686.7	244967.0	242752.5	244388.4	**245907.2**	213513.3		
22	11	5	248369.7	257481.4	261951.9	263768.8	264970.6	**266423.5**	265699.9			241909.2		
24	12	6	277716.5	286579.0	293662.7	295812.5	295828.3	**297556.7**				270662.0		
26	13	6	317247.7	325892.7	331844.8	331456.4	**333678.9**					310366.9		
26	5	5	314581.9	321305.4	323684.2	**323843.5**	323070.6							
28	14	7	355413.1	366501.5	371985.2	**374619.7**						348059.2		
28	5	5	352797.1	360574.9	**362585.2**	362571.0								
30	15	7	406137.0	417363.9	420846.9	**422012.1**						396222.1		
30	5	5	404198.1	412133.4	**414865.9**	413756.4								
32	16	8	439408.5	460082.6	459192.4	**462944.9**						427436.2		
32	5	5	435769.2	452052.4	**455885.7**	450956.9								

Table 2: Lower bounds obtained with $\mathcal{M}_w^{R6}$ for $w \in [2, 10]$ (best values in bold), and with the best models from literature (RNFM, RSPM, and RSPCM).

Inst.	q_1	q_2	$\mathcal{M}_2^{R6}$	$\mathcal{M}_3^{R6}$	$\mathcal{M}_4^{R6}$	$\mathcal{M}_5^{R6}$	$\mathcal{M}_6^{R6}$	$\mathcal{M}_7^{R6}$	$\mathcal{M}_8^{R6}$	$\mathcal{M}_9^{R6}$	$\mathcal{M}_{10}^{R6}$	RNFM	RSPM[9]	RSPM[8]	RSPCM
14	7	3	0.14	0.51	3.07	10.42	24.54	43.15	87.16	258.95	630.87	0.59	526.56	42.10	742.86
14	6	3	0.07	0.29	1.48	3.28	11.65	31.77	90.46	551.29	1741.88	0.54	108.70	40.80	443.61
14	5	3	0.05	0.16	0.44	1.05	3.74	17.80	57.12	587.51	1998.03	0.52	40.50	50.30	216.54
14A	7	3	0.12	0.63	2.62	11.13	27.78	53.05	91.42	273.05	881.75	0.58	107.57	40.70	638.13
14A	6	3	0.08	0.40	1.59	4.27	11.18	47.92	113.57	527.70	2090.22	0.55	77.54	45.50	450.26
14A	5	3	0.06	0.26	0.57	1.97	6.80	28.70	134.03	792.61	4513.95	0.54	42.13	47.80	220.36
14B	7	3	0.14	0.68	2.49	8.73	22.14	58.06	84.32	364.78	879.30	0.60	377.83	43.50	842.78
14B	6	3	0.12	0.45	1.42	4.36	11.67	35.78	126.73	401.60	1939.73	0.54	52.16	49.60	462.99
14B	5	3	0.07	0.29	0.55	1.78	6.82	29.42	160.98	1035.51	5777.11	0.53	31.68	55.70	480.79
14C	7	3	0.13	0.72	3.55	11.52	22.21	63.54	103.36	297.25	879.67	0.56	442.33	44.60	806.03
14C	6	3	0.12	0.30	2.02	3.72	9.11	27.48	72.84	370.68	1534.01	0.56	326.59	47.70	796.43
14C	5	3	0.09	0.18	0.55	1.39	4.49	28.97	111.04	818.00	5163.33	0.52	54.40	49.50	396.86
16	8	4	0.81	14.29	42.07	201.03	343.69	431.65	1032.18	2385.00	2877.17	0.86	457.10	172.00	771.61
16	8	2	0.18	1.73	14.24	272.94	1843.58	4820.36	10800.00	10800.00		0.60	50247.89	7092.00	59186.31
16	7	3	0.34	2.48	13.34	59.54	154.72	402.81	691.82	3445.89	10800.00	0.71	2207.89	10500.00	2959.44
16	7	2	0.15	1.58	10.14	114.91	515.94	1902.23	6051.61	10800.00		0.69	2598.89	10102.00	5113.53
16A	8	4	1.20	15.51	32.32	198.92	386.43	551.95	781.06	1816.88	3151.16	0.81	373.20	172.00	923.43
16A	8	2	0.15	1.58	12.42	352.65	1400.26	4330.80	10800.00	10800.00		0.58	14548.00	5403.00	17380.10
16A	7	3	0.44	2.91	12.56	61.85	172.06	491.18	1381.38	5342.18	10800.00	0.58	3266.34	371.00	4303.05
16A	7	2	0.13	1.21	9.73	93.21	348.10	1415.13	5790.52	10800.00		0.56	3918.92	7476.00	6044.11
16B	8	4	1.01	11.28	39.69	179.43	349.11	467.83	782.94	2449.49	2759.63	0.81	342.27	202.00	771.07
16B	8	2	0.16	1.67	15.07	287.66	1459.32	4453.18	10800.00	10800.00		0.74	42129.13	5162.00	53974.07
16B	7	3	0.38	2.38	12.85	58.67	141.48	389.96	1051.29	4142.73	10800.00	0.73	3236.07	880.00	4053.36
16B	7	2	0.15	1.33	12.24	110.03	515.78	1699.03	7627.61	10800.00		0.72	3077.02	9021.20	5781.35
16C	8	4	0.99	10.14	40.07	142.23	236.83	246.88	663.24	1037.80	1511.01	0.86	201.13	234.00	586.44
16C	8	2	0.19	1.76	14.95	161.89	1003.26	2501.33	10800.00	10800.00		0.74	13634.98	7380.00	22101.70
16C	7	3	0.33	2.76	12.91	61.77	164.26	411.04	1012.05	3788.13	10800.00	0.79	851.66	449.00	1658.40
16C	7	2	0.15	1.45	12.26	68.82	353.68	1518.86	10800.00	10800.00		0.71	3319.74	10578.00	4790.26
18	9	4	1.65	32.74	165.18	686.79	1856.79	2428.94	6430.24	10800.00	10800.00	1.07	17834.91		22980.33
20	10	5	3.61	109.79	955.08	2038.08	4351.67	7870.05	10800.00	10800.00	10800.00	1.53			
22	11	5	4.52	197.52	2233.87	7476.42	10800.00	10800.00	10800.00			1.95			
24	12	6	11.68	387.45	10800.00	10800.00	10800.00	10800.00				2.90			
26	13	6	11.92	621.74	10800.00	10800.00	10800.00					3.75			
26	5	5	3.54	100.85	1140.39	5094.29	10800.00								
28	14	7	16.98	973.06	10800.00	10800.00						4.29			
28	5	5	5.08	134.21	1836.80	9919.03									
30	15	7	23.43	1588.45	10800.00	10800.00						4.16			
30	5	5	6.87	211.68	3232.17	10800.00									
32	16	8	34.30	3920.92	10800.00	10800.00						7.09			
32	5	5	9.60	452.96	7369.47	10800.00									

Table 3: Solution times (in seconds) for linear relaxations $\mathcal{M}_w^{R6}$ with $w \in [2, 10]$, and for the relaxations of the best models from literature.

Tables 2 and 3 show that, although the solution time for $\mathcal{M}_w^{R6}$ increases considerably as w increases, the lower bound does not always improve significantly and, sometimes, can even worsen. For example, consider instance 14 with $q_1 = 7$ and $q_2 = 3$. The lower bound with $w = 6$ takes 24.54 seconds to calculate but is never more than 155 miles below those obtained with $w > 6$, which take between 43.15 and 630.87 seconds to calculate. In addition, the lower bound found for this instance with $w = 8$ is greater than those found with $w > 8$. A possible cause for the deterioration the $\mathcal{M}_w^{R6}$ lower bounds as w increases is the increase in model size. Because the linear relaxations of larger models take longer to solve and our running time is limited, fewer iterations of the cutting-plane algorithm are executed. With fewer cuts, the dual bounds are expected to decrease in quality. This behavior indicates that we must be careful when choosing the value of w to use in our branch-and-cut algorithm.

We now compare $\mathcal{M}_w^{R6}$ against RNFM, RSPM, and RSPCM. Recall that the variables of RNFM are equivalent to those of $\mathcal{M}_2^{R6}$, but the latter includes additional valid inequalities. On the 28 instances with at most 16 teams, the lower bounds produced by $\mathcal{M}_2^{R6}$ are between 1100 and 5478 miles greater than those produced by RNFM. Moreover, 25 out of these 28 improved bounds require less time to calculate with $\mathcal{M}_2^{R6}$ than with RNFM. On instances with 18 or more teams, even though $\mathcal{M}_2^{R6}$ can be up to 5.6 times slower than RNFM, the lower bounds produced by $\mathcal{M}_2^{R6}$ are between 5701 and

11972 miles greater than those produced by RNFM. The variables of RSPM and RSPCM represent all complete routes that satisfy *(iii)–(iv)*, which are equivalent to the variables of $\mathcal{M}^{R6}_{4n-2}$, i.e. a much larger value of w than the largest one we consider. Note, however, that the lower bounds obtained by $\mathcal{M}^{R6}_{6}$ on 16-team instances with $q_1 = 7$ and on all 14-team instances are already better than those obtained by RSPM. Furthermore, on these instances, $\mathcal{M}^{R6}_{6}$ solves between 3.9 and 35.8 times faster than RSPM is solved in [9], and between 1.5 and 67.9 times faster than RSPM is solved in [8]. These $\mathcal{M}^{R6}_{6}$ bounds are at most 555 miles shorter than those obtained by RSPCM on the same instances, while still taking less time to solve (between 9.9 and 88.4 times faster than RSPCM). Despite these good results, $\mathcal{M}^{R6}_{w}$ does not perform well on 16-team instances with $q_1 = n = 8$. $\mathcal{M}^{R6}_{w}$'s best lower bounds on these instances are between 1442 and 3553 miles shorter, and between 1929 and 4544 miles shorter than those obtained by RSPM and RSPCM, respectively, while solving between 1.5 and 18.3 times more slowly than RSPM, as reported in [8]. We believe that RSPM and RSPCM tend to outperform $\mathcal{M}^{R6}_{w}$ as q_1 and q_2 increase because this leads to an increase in the number of routes that are forbidden in RSPM and RSPCM which would have, otherwise, been part of valid fractional solutions to $\mathcal{M}^{R6}_{w}$. As a consequence, $\mathcal{M}^{R6}_{w}$'s optimal solutions may contain such routes, leading to weaker dual bounds. On instance 18, although RSPM's and RSPCM's bounds obtained in [9] are 6378 miles (3.1%) and 7050 miles (3.4%) better than the best $\mathcal{M}^{R6}_{w}$ bound, they were obtained in 4 hours and 57 minutes and in 6 hours and 23 minutes, respectively.

$\mathcal{M}^{R6}_{w}$'s advantage becomes more pronounced as the problem size increases, as evidenced by Tables 2 and 3. Because RSPM and RSPCM have an exponential number of variables and the corresponding pricing problem is time-consuming, these relaxations take too long to solve for instances with more than 18 teams. In addition to achieving good results on 16-team instances with $q_1 = 7$ and on all 14-team instances, $\mathcal{M}^{R6}_{w}$ can not only be solved within 3 hours for all instances with more than 18 teams, but also produces the best lower bounds known to date for these instances.

5.3. Branch-and-Cut Results

Based on our earlier experiments, we develop a branch-and-cut algorithm to solve $\mathcal{M}^{6}_{w}$ due to the good performance of $\mathcal{M}^{R6}_{w}$. Because (5)–(8) are exponential in number, we initialize our model with (1), (2), and (4) only, and introduce (5)–(8) during the search as they become violated. We invoke CPLEX callbacks at each node in the search tree to perform the separations in procedure Sep6.

We use the following parameter settings in CPLEX's branch-and-cut algorithm. Preliminary experiments show that CPLEX's general primal heuristics do not find good solutions to $\mathcal{M}^{6}_{w}$. Therefore, we focus on finding good lower bounds and on optimality by setting the MIP emphasis parameter to "best bound" and disabling primal heuristics. We also modify the MIP probing level parameter to force the algorithm to run a moderate probing on variables, since the time-consuming aggressive probing does not improve the results. In particular, we noticed probing spent too much time picking a branching variable on instances whose linear relaxation takes a long time (over 1000 seconds) to solve. (See column $\mathcal{M}^{R6}_{w}$ in Table 1 to identify these instances.) Hence, on these instances only, we set the MIP variable selection strategy parameter to choose the variable whose value is farthest from integer. This speeds up branching and increases the number of explored nodes within the given time limit, which led to better results. Finally, we disable the generation of all CPLEX's cuts to better assess the impact of our own cuts.

Next, we conduct preliminary experiments to determine which value of w to use for each instance in the benchmark based on the speed/strength trade-offs identified in Section 5.2. Our tests indicate that the branch-and-cut obtains better results by setting $w = 4$ for all instances with $q_1 < n$ and for 14-team instances with $q_1 = n$. The $\mathcal{M}^{R6}_{4}$ lower bound for these instances is not too far from the best one in Table 2 and it is calculated more quickly, allowing the enumeration of many more nodes. For the

remaining instances with $q_1 = n$ it is worth using time-consuming relaxations because of the improved lower bounds. Therefore, these instances are solved with the value of w that yields the best bound (bold numbers) in Table 2.

We execute our branch-and-cut algorithm with time limits of 3 and 24 hours to allow a fair comparison between our results and existing ones in the literature. We report the best lower bounds obtained within each time limit in Table 4. Column "Lower bound" contains the final (best) lower bound value, "Iterations" stand for Simplex iterations, and "Cuts" are the total number of violated inequalities added by Sep6. Instances with lower bounds displayed in bold and marked with an "*" were solved to optimality by our algorithm. These are the only upper bounds found within the given time limits.

Inst.	q_1	q_2	w	Time limit									
				3 hours					24 hours				
				Lower bound	Time (sec.)	Iterations	Nodes	Cuts	Lower bound	Time (sec.)	Iterations	Nodes	Cuts
14	7	3	4	158578.5	10800	6578554	5462	16004	159271.8	86400	34265950	25830	26417
14	6	3	4	157469.3	10800	8680509	12259	14585	158037.6	86400	41153400	52618	23008
14	5	3	4	**154962.0***	1823.44	1951142	10980	7622					
14A	7	3	4	152537.1	10800	7231231	7036	15132	153257.5	86400	36554874	33291	25296
14A	6	3	4	151611.1	10800	10066365	15628	12439	152169.5	86400	50868260	68271	20215
14A	5	3	4	**149331.0***	524.79	836360	3300	4342					
14B	7	3	4	152647.1	10800	7157927	5776	15660	153191.1	86400	34391926	23233	27720
14B	6	3	4	151360.5	10800	9863357	14392	12906	151821.9	86400	45898826	58131	22350
14B	5	3	4	**149455.0***	1003.7	1668752	8887	5211					
14C	7	3	4	151129.1	10800	6314167	4764	16719	151791	86400	30767570	20898	28766
14C	6	3	4	149820.4	10800	9316255	13735	13601	150286.6	86400	44207534	54463	22375
14C	5	3	4	148333.2	10800	8106602	27195	13572	**148349.0***	11457.49	8330667	29379	13705
16	8	4	10	183386.2	10800	1093933	8	9802	185936.7	86400	7361031	54	27141
16	8	2	8	152569.6	10800	206239	1	5230	153725	86400	1214144	13	9179
16	7	3	4	159841.1	10800	3261959	1649	13424	160664	86400	28910558	13240	28292
16	7	2	4	149560.8	10800	2634818	1573	13536	149988.5	86400	23603021	12705	30384
16A	8	4	10	196183.3	10800	1145764	12	13048	198330.5	86400	7415309	75	27001
16A	8	2	8	164625.8	10800	207790	1	4917	165915.3	86400	1256325	12	9541
16A	7	3	4	173028.2	10800	3065182	1052	12339	174226.3	86400	28361114	9054	28039
16A	7	2	4	162675.1	10800	2817934	1696	13326	163052.0	86400	26985443	14253	27689
16B	8	4	10	205073.4	10800	1126908	8	12883	207781.1	86400	7296067	61	31270
16B	8	2	8	167241.6	10800	190363	1	4021	168223.9	86400	1991551	7	7452
16B	7	3	4	172131.7	10800	2971901	1235	13041	173178.0	86400	29106668	10769	27111
16B	7	2	4	164978.2	10800	3988299	2986	13180	165581.1	86400	33172881	24074	25088
16C	8	4	10	198274.6	10800	1190729	11	11403	202369.4	86400	7259649	76	22285
16C	8	2	8	167339.8	10800	178530	1	4245	167530.7	86400	1124098	2	4983
16C	7	3	4	172377.7	10800	2787489	1072	13455	173273.9	86400	26730153	9351	29042
16C	7	2	4	164531.3	10800	3519161	1998	13513	165125.2	86400	27695874	13998	28036
18	9	4	9	205781.9	10800	257650	1	9386	206759.4	86400	1681072	16	12350
20	10	5	10	245897.4	10800	93440	1	9842	250372.6	86400	572624	1	17704
22	11	5	7	266415.6	10800	189486	1	20364	269735.5	86400	1062045	5	33425
24	12	6	7	297898.6	10800	107241	1	19403	301441.0	86400	671652	1	38858
26	13	6	6	333504.9	10800	127822	1	19216	336854.4	86400	730203	1	34010
26	5	5	4	324387.1	10800	1409902	247	11564	324753.2	86400	11158237	2770	22082
28	14	7	5	374630.6	10800	220783	1	26780	377356.2	86400	1149468	1	36817
28	5	5	4	363072.1	10800	916593	91	10571	363541.4	86400	7865382	1422	20455
30	15	7	5	422026.0	10800	144276	1	20906	424537.6	86400	749184	1	34822
30	5	5	4	415296.4	10800	614318	43	9642	415747.5	86400	5650380	1063	18417
32	16	8	5	462894.6	10800	99418	1	14476	468803.5	86400	590596	1	32946
32	5	5	4	455836.4	10800	376627	1	9909	456685.9	86400	3843703	492	17838

Table 4: Lower bounds obtained with the branch-and-cut algorithm. Asterisk indicates proven optimality.

As seen in Table 4, we solve to optimality all of the 14-team instances with $q_1 = 5$ and $q_2 = 3$, and all but one of them within 3 hours. The only previously published method capable of optimally solving instances with more than 12 teams is the branch-and-price-and-cut presented in [9]. It found optimal

solutions to instances 14 and 14A with $q_1 = 5$ and $q_2 = 3$ after 34:45h and 11:24h, respectively, whereas we solve all 14-team instances with $q_1 = 5$ and $q_2 = 3$ in no more than 3:10h (14, 14A, and 14B only require 31, 9, and 17 minutes, respectively). The results reveal that the majority of the improvement in the lower bound is achieved by the branch-and-cut within the first three hours of computation. Extending the time limit to 24 hours only produces an increase of 1368 miles in the dual bound on average, although the gain largely varies from an instance to another, as its standard deviation is of 1413 miles.

To complement the information in Table 4, we now present the percentage of separated cuts that come from each family of inequalities on average (followed by ± its standard deviation). With execution times limited to 3 hours, the averages are: 18.5%±14.3% from (7), 12.2%±12.8% from (8), 30.6%±11.6% from (9), 36.3% ± 15.5% from (10), and 2.4% ± 8.4% from (13). With execution times limited to 24 hours, the figures are similar: 21.2% ± 16.9% from (7), 11.9% ± 9.6% from (8), 30.2% ± 11.1% from (9), 35.4% ± 14.5% from (10), and 1.3% ± 7.1% from (13).

In Table 5 we compare, with matching times, the lower bounds found by our branch-and-cut algorithm (BC) with the best lower bounds available, which were obtained by the following methods: the decomposition approach (DA) in [7], the branch-and-price (BP) in [8], the branch-and-bound (BB) and branch-and-price-and-cut (BPC) in [9], and the branch-and-bound with decomposition-based lower bounds in [19] (BB-DLB). In [7], DA results are reported for two time limits: up to 3 hours (which we call DA3), and over 3 hours (which we call DA+). Methods BB and BP were limited to run for 3 hours, whereas BPC and BB-DLB were limited to 48 hours. Unlike the other methods, BB-DLB found many optimal solutions and infeasibility proofs before reaching the time limit. Therefore, for those results, we include BB-DLB's corresponding execution times in the last row of Table 5. We compare the lower bounds found by BC within 3 hours with those obtained by BB, BP, DA3, and BB-DLB within 3 hours, and the ones found by BC within 24 hours with those obtained by DA+, BPC and BB-DLB in more than 3 hours. As before, lower bounds marked with an "*" are optimal. A lower bound appears in bold if no better one was found within the time the former one was obtained.

According to Table 5, it seems that BB-DLB is better suited for smaller instances, whereas BC is more appropriate for larger ones. To see this, we divide our analysis in two complementary groups of instances. The first (SMALL) is composed of instances having up to 18 teams, while the second (LARGE) contains the remaining instances (with 20 or more teams).

For 20 of the 29 instances in the SMALL group, the BB-DLB lower bounds are strictly greater than those found by the other methods. BB-DLB solves 19 instances to optimality and produces 4 proofs of infeasibility not known before. BPC and BC only solve 2 and 4 instances to optimality, respectively.

We now focus on the 11 instances in the LARGE group, whose sizes come closer to the actual number of teams in MLB. Not all methods can handle instances this big and, therefore, several results are missing for many of them. Results for BB, DA3, DA+, and BB-DLB are only available for 7, 1, 4, and 7 instances in the LARGE group, respectively. These results, as well those for BC, appear in the last 11 rows of Table 5. We start with the results obtained within 3 hours of computation. Under this limit, the data for BC, BB, and DA3 are available: BC produces results for all 11 instances, BB for 7, and DA3 for just one instance. BC is clearly the winner as it computes the best lower bound for all the instances in LARGE. The average/max/min improvement in the lower bound values is of 23548.4 / 32379.7 / 11571.4 miles, with a standard deviation of 6718.9 miles.

The advantage of BC over the other methods in dealing with instances in the LARGE group is confirmed when we extend the analysis to the results obtained with more than 3 hours of computation. In this case, two other methods are considered in addition to BC: DA+ and BB-DLB. These two methods can be viewed as complementary with respect to LARGE in the sense that there are results reported for exactly one of them for each instance in this group, 7 for BB-DLB and 4 for DA+. BC's

Inst.	q_1	q_2	Time limit / Method								
			3 hours				24 hours	> 3 hours	48 hours		
			BC	BB	BP	DA3	BC	DA+	BPC	BB-DLB	
14	7	3	158578.5	154175.6	157812.8	156536	159271.8	159797	158900.2	**164440.0***	(228.6)
14	6	3	157469.3	154036.7	155570.4	156551	158037.6	156551	157083.4	**158875.0***	(51.6)
14	5	3	154962.0*	153318.8	153759.6	153066		153066	154962.0*	**154962.0***	(130.2)
14A	7	3	152537.1	147866.4	151243.5	151406	153257.5	153199	152635.7	**158760.0***	(123.0)
14A	6	3	151611.1	147773.1	149285.4	150998	152169.5	150998	151043.2	**152981.0***	(30.0)
14A	5	3	149331.0*	147358.3	147966.4	148299		148299	149331.0*	**149331.0***	(67.2)
14B	7	3	152647.1	147159.6	151165.8	149910	153191.1	151059	152517.6	**157884.0***	(241.2)
14B	6	3	151360.5	147031.5	149208.6	149267	151821.9	149267	150941.3	**152740.0***	(103.2)
14B	5	3	149455.0*	146606.1	147638.3	147534		147534	149311.6	**149455.0***	(63.0)
14C	7	3	151129.1	146104.6	150101.6	151122	151791	151581	150925.9	**154913.0***	(45.6)
14C	6	3	149820.4	145982.2	147820	148728	150286.6	148728	148986.5	**150858.0***	(100.2)
14C	5	3	148333.2	145598.1	146622.1	146764	148349.0*	146764	147902.9	**148349.0***	(764.4)
16	8	4	183386.2	156206.4	**193457.1**	168847	185936.7	185939	191458	**infeas.**	(13977.6)
16	8	2	152569.6	145829.7	**155045.2**	151481	153725	151481	**156088.1**	145531	
16	7	3	**159841.1**	153649.4	158586	155707	160664	158480	160161.3	**165765.0***	(24296.4)
16	7	2	**149560.8**	145787	148341.8	147138	149988.5	147138	149488	**150433.0***	(66118.8)
16A	8	4	196183.3	168882.5	**200648.5**	185119	198330.5	185119	206141.2	**infeas.**	(13549.2)
16A	8	2	164625.8	158645.6	**166624.1**	162788	165915.3	162788	**168274.4**	160739	
16A	7	3	**173028.2**	166459.3	172420.1	170342	174226.3	172964	172471.4	**178511.0***	(15101.4)
16A	7	2	**162675.1**	158621.8	161571.2	161640	163052	161640	162621.7	**163709.0***	(57922.2)
16B	8	4	205073.4	169684.4	**209346.5**	188195	207781.1	208418	215520.6	**infeas.**	(13764.6)
16B	8	2	167241.6	159525.2	**170092.6**	167768	168223.9	167768	**170384.4**	165737	
16B	7	3	**172131.7**	165753.2	172058	170940	**173178.0**	173023	172695.9	**180204.0***	(136216.8)
16B	7	2	**164978.2**	159538.6	163649.6	164012	**165581.1**	164012	164816	**167190.0***	(138118.8)
16C	8	4	198274.6	170370.6	**205643.8**	179213	202369.4	188561	206368.8	**infeas.**	(14216.4)
16C	8	2	167339.8	161296.6	**168783.6**	163543	167530.7	166001	**169697.7**	164541	
16C	7	3	**172377.7**	166562.3	171767.6	170133	**173273.9**	171377	172754.6	**176161.0**	
16C	7	2	**164531.3**	161241.1	163850.8	163305	**165125.2**	163305	164625.7	**166479.0***	(135509.4)
18	9	4	**205781.9**	184222			206759.4		**213805.5**	193632	
20	10	5	**245897.4**	216462.6			**250372.6**			220907	
22	11	5	**266415.6**	245030.5			**269735.5**			243052	
24	12	6	**297898.6**	272970			**301441.0**			250590	
26	13	6	**333504.9**	312705.5			**336854.4**			289651	
26	5	5	**324387.1**				**324753.2**	318690			
28	14	7	**374630.6**	350290.9			**377356.2**			322208	
28	5	5	**363072.1**				**363541.4**	358593			
30	15	7	**422026.0**				**424537.6**			339331	
30	5	5	**415296.4**	398032.9		403725	**415747.5**	413103			
32	16	8	**462894.6**	430514.9			**468803.5**			369695	
32	5	5	**455836.4**				**456685.9**	443281			

Table 5: Comparison between branch-and-cut lower bounds and best lower bounds from the literature. The time (in seconds) spent by BB-DLB to prove optimally or infeasibility appears between parentheses in the last column.

lower bounds are the best in all 11 cases. The average/max/min improvement in the lower bound values is of 38248.0 / 99108.5 / 2644.5 miles, with a standard deviation of 32671.3 miles. Furthermore, note that BC lower bounds remain the largest ones even when it is restricted to run for no more than 3 hours, while the other methods are allowed to run for longer periods of time. One possible explanation could be that BB-DLB seems to suffer from scalability problems, as its good performance on SMALL instances does not carry over to LARGE. In fact, the BB-DLB lower bounds for LARGE instances turn out to be worse than those generated by BB in 3 hours (we disregard here the 30-team instance with $q_1 = 15$ and $q_2 = 7$ for which no BB bound is available).

Finally, we assess whether or not it is advantageous to allow more computation time to BC in terms of lower bound improvement. Comparing the results in columns 4 and 8 for the last eleven rows of

Table 5, we see that the average/max/min increase in the lower bound, when going from 3 to 24 hours, is of 2542.6 / 5908.9 / 366.1 miles, with a standard deviation of 1833.9 miles. These figures are roughly one order of magnitude smaller than those coming from the comparison between BC and the other methods. This is an indication that no substantial lower bound gains are likely if we keep running BC for much longer.

6. Conclusions and Future Work

We introduce a parametrized IP model for the TUP that generalizes the two best existing models, which are based on network flows and set partitioning. Our parametrization determines the length w of umpires' trip sequences, which range from 2 to $4n - 2$ games and are represented as binary decision variables in the model. This flexibility allows us to explore the trade-off between solution speed (when trip sequences are short) and lower bound strength (when trip sequences are long). This model is further strengthened by new families of strong valid inequalities, which are added to the formulation as they are found to be violated inside a branch-and-cut (BC) algorithm.

Our computational results attest the relevance and impact of our inequalities and confirm the speed/strength trade-off as a function of w. BC was developed with the goal of solving instances of realistic size. Our experiments show that it scales better than existing alternatives because it continues to find strong lower bounds even for instances with 20 or more teams, improving all best known lower bounds for these instances. Although smaller instances were not the focus of this work, it is remarkable that only one method performed better than BC on instances having between 14 and 18 teams. Because of its robustness in producing high-quality bounds for both small and large instances, we believe that BC currently ranks as one of most competitive methods for the TUP.

As future work, we intend to study primal heuristics that can be embedded in our BC algorithm to help prune the search tree more quickly. In addition, instead of including all of our variables a priori, we plan on pricing them into the formulation dynamically (as in a branch-and-cut-and-price algorithm) to improve solution speed. We foresee the pricing problem to be challenging because it needs to account for our specific cutting planes, but we believe the ability to solve smaller linear relaxations will more than compensate for the extra pricing effort.

Appendix A. Number of Variables in the Optimization Model

Table A.6 shows the number of variables in the model presented in Section 3 for all instances and values of w between 2 and 10. Empty entries indicate that the given pair (instance, w) would produce a model with more than 5 million variables, which we do not consider in our experiments. Because instances with letters in their names have the same tournament and, therefore, the same variables as the original instances, they are omitted from Table A.6.

References

[1] J. R. Evans, J. E. Hebert, R. F. Deckro, Play ball – the scheduling of sports officials, Perspectives in Computing 4 (1) (1984) 18–29.

[2] J. R. Evans, A microcomputer-based decision support system for scheduling umpires in the American Baseball League, Interfaces 18 (6) (1988) 42–51.

[3] M. A. Trick, H. Yildiz, Benders' cuts guided large neighborhood search for the traveling umpire problem, Naval Research Logistics 58 (8) (2011) 771–781.

Inst.	q_1	q_2	Parameter w								
			2	3	4	5	6	7	8	9	10
14	7	3	875	1463	3124	6707	14480	26097	43858	92909	157212
14	6	3	875	1463	3124	6707	14480	30345	56849	140195	272418
14	5	3	875	1463	3124	6707	16354	37921	79866	224807	465183
16	8	4	1397	2961	5654	10844	18305	29823	37045	57076	97664
16	8	2	1397	3679	11624	38436	117394	331902	844815	2378813	7202046
16	7	3	1397	2961	7368	19089	43920	98326	204509	512763	1327904
16	7	2	1397	3679	11624	38436	117394	331902	973859	3031348	10546030
18	9	4	2081	5384	13994	34561	81585	171573	345990	621342	1211598
20	10	5	2962	9069	28332	72276	172373	393620	818194	1492658	2417177
22	11	5	4063	14405	53264	171979	535731	1497634	4036925	10939472	
24	12	6	5407	21810	97332	368098	1167827	3219784	8889449		
26	13	6	7009	31677	158375	717269	2615617	9823065			
26	5	5	7009	31677	158375	717269	3329528	17481485			
28	14	7	8909	44638	248893	1318194	5688863				
28	5	5	8909	44638	248893	1318194	7072643				
30	15	7	11124	61206	391728	2282757	11618198				
30	5	5	11124	61206	391728	2282757	14162234				
32	16	8	13673	81972	568954	3777946	22280158				
32	5	5	13673	81972	568954	3777946	26687469				

Table A.6: Number of variables in our optimization models with w varying from 2 to 10.

[4] M. A. Trick, H. Yildiz, T. Yunes, Scheduling Major League Baseball Umpires and the Traveling Umpire Problem, Interfaces 42 (3) (2012) 232–244.

[5] M. A. Trick, H. Yildiz, Locally Optimized Crossover for the Traveling Umpire Problem, European Journal of Operational Research 216 (2) (2012) 286–292.

[6] L. Oliveira, C. C. de Souza, T. Yunes, Improved bounds for the traveling umpire problem: A stronger formulation and a relax-and-fix heuristic, European Journal of Operational Research 236 (2) (2014) 592–600.

[7] T. Wauters, S. V. Malderen, G. V. Berghe, Decomposition and local search based methods for the traveling umpire problem, European Journal of Operational Research 238 (2014) 886–898.

[8] T. A. M. Toffolo, S. V. Malderen, T. Wauters, G. V. Berghe, Branch-and-price and improved bounds to the traveling umpire problem, in: Proceedings of the 10th International Conference of the Practice and Theory of Automated Timetabling, 420–432, 2014.

[9] L. Xue, L. Z., A. Lim, Two exact algorithms for the traveling umpire problem, European Journal of Operational Research 243 (3) (2015) 932–943.

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

[11] M. Yavuz, U. H. İnan, A. Fığlalı, Fair referee assignments for professional football leagues, Computers & Operations Research 35 (9) (2008) 2937–2951.

[12] A. Farmer, J. S. Smith, M. L. T., Scheduling umpire crews for professional tennis tournaments, Interfaces 37 (2) (2007) 187–196.

[13] R. V. Rasmussen, M. A. Trick, Round Robin Scheduling – A Survey, European Journal of Operational Research 188 (2008) 617–636.

[14] G. Kendall, S. Knust, C. C. Ribeiro, S. Urrutia, Scheduling in sports: An annotated bibliography, Computers & Operations Research 37 (1) (2010) 1–19.

[15] M. J. Fry, J. W. Ohlmann, Introduction to the Special Issue on Analytics in Sports, Part II: Sports Scheduling Applications, Interfaces 42 (3) (2012) 229–231.

[16] M. A. Trick, H. Yildiz, Benders' cuts guided large neighborhood search for the traveling umpire problem, in: P. Van Hentenryck, L. Wolsey (Eds.), Proceedings of the Fourth Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR), vol. 4510 of *Lecture Notes in Computer Science*, Springer-Verlag, 332–345, 2007.

[17] L. Oliveira, C. C. de Souza, T. Yunes, On the complexity of the traveling umpire problem, Theoretical Computer Science 562 (2015) 101–111.

[18] M. A. Trick, Traveling Umpire Problem: Data Sets and Results, available at `http://mat.tepper.cmu.edu/TUP`, 2015.

[19] T. A. M. Toffolo, T. Wauters, S. V. Malderen, G. V. Berghe, Branch-and-bound with decomposition-based lower bounds for the Traveling Umpire Problem, European Journal of Operational Research (forthcoming).

[20] B. Kallehauge, N. Boland, O. B. G. Madsen, Path inequalities for the vehicle routing problem with time windows, Networks 49 (4) (2007) 273–293.

[21] S. Niskanen, P. R. J. Östergård, Cliquer user's guide, version 1.0, Tech. Rep. T48, Communications Laboratory, Helsinki University of Technology, Espoo, Finland, 2003.

[22] R. M. Karp, Reducibility Among Combinatorial Problems, in: R. E. Miller, J. W. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, 85–103, 1972.

[23] T. A. M. Toffolo, T. Wauters, M. Trick, An automated benchmark website for the Traveling Umpire Problem, URL `http://gent.cs.kuleuven.be/tup`, 2015.