Computers & Operations Research 72 (2016) 147–159

Contents lists available at ScienceDirect

Computers & Operations Research

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

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, 5250 University Drive, Room KE405, Coral Gables, FL 33146, USA

ARTICLE INFO

Available online 3 March 2016

Keywords:
Sports scheduling
Traveling umpire problem
Integer programming
Branch-and-cut
OR in sports

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).

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 [1–9], cricket [10], football [11], and tennis [12]. A variety of other sports scheduling problems can be found in [13–15].

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 $0 \le d_1 < n$ and $0 \le d_2 < \lfloor \frac{n}{2} \rfloor$. A feasible solution to the TUP is an assignment of n umpires to the games of the tournament that satisfies the following constraints:

(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 [18,19] 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.

☆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. Tel.: +1 305 284 5107; fax: +1 305 284 2321.
E-mail addresses: lucas.oliveira@ic.unicamp.br (L.d. Oliveira), cid@ic.unicamp.br (C.C. de Souza), tallys@miami.edu (T. Yunes).

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 [18,19], 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 to its large number of variables and constraints. Hence, the TUP is highlighted as 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 [20]. 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 represented by the variables of our model 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 $2 \le w \le 4n-2$ 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. Fig. 1 illustrates a tournament with four teams and six rounds being subdivided into sections 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 $2 \le w < 4n-2$, we cannot impose constraint (iii). Note, however, that consecutive sections have one round in common (Fig. 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.

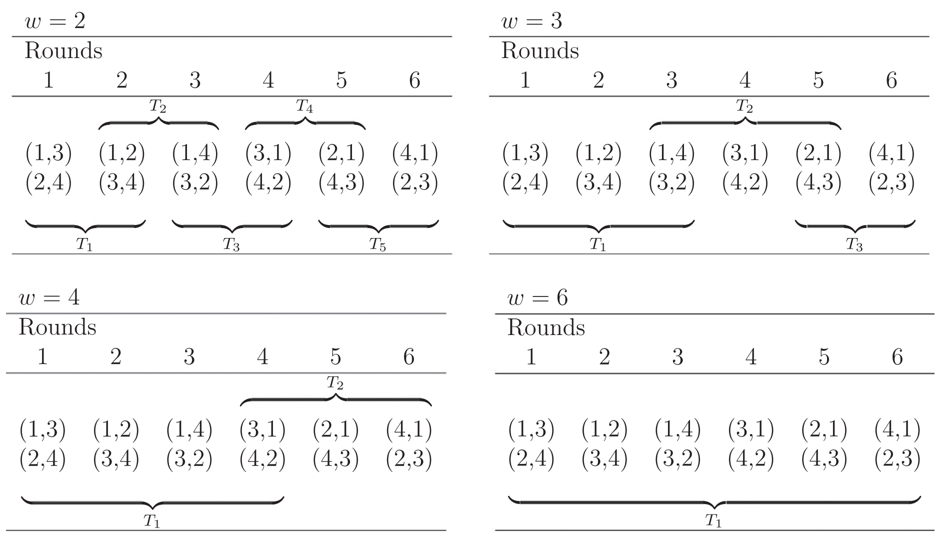

Fig. 1. Sections of a 4-team, 6-round tournament for w=2, 3, 4, and 6.

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 $p \in P = \bigcup_{s \in S} P_s$, let x_p be a binary variable 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:

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

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

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

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 (2)–(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 [21], which we explain next. Let $U = (u_1, u_2, \ldots)$ be an infeasible route, and let $H^+(U) = P(U) \cup \{p \in P \mid (u_1, u_2, \ldots, u_i, p)$ be an infeasible route for some $i = 1, \ldots, |P(U)| - 1\}$. Then, (5) is a stronger version of (3)

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

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 [21] 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 $1 \le r \le 4n-2$, 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.

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

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 $i = 2, \ldots, |P(U)|\}$. 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 $\mathcal{T}$.

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

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).

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

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 $\mathbb{F}' = \{F = (f_1, f_2, \ldots) \in \mathbb{F} \mid f_{|P(F)|} \neq P_{|S|}\}$ and $\mathbb{F}'' = \{(f_1, f_2, \ldots) \in \mathbb{F} \mid f_1 \neq P_1\}$ be the sets of feasible routes that exclude simple routes from the last and first sections of T, respectively. Consider $F' = (f'_1, f'_2, \ldots) \in \mathbb{F}'$, $F'' = (f''_1, f''_2, \ldots) \in \mathbb{F}''$, and define $K^+(F') = P(F') \cup \{p \in P \mid (f'_1, f'_2, \ldots, f'_i, p)$ is an infeasible route for some $i = 1, \ldots, |P(F')|\}$ and $K^-(F'') = P(F'') \cup \{p \in P \mid (p, f''_i, f''_{i+1}, \ldots, f''_{|P(F'')|})$ is an infeasible route for some $i = 1, \ldots, |P(F'')|\}$. Instead of using (5) and (6), we use (9) and (10), respectively.

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

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

Notice that (9) and (10) are respectively equivalent to (5) and (6), but the former are defined in terms of feasible routes, whereas the latter 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.

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

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

where $\tilde{K}^+(F') = \{p \in (P \backslash P(F')) \mid (f'_1, f'_2, \ldots, f'_i, p)$ is a feasible route for some $i = 1, \ldots, |P(F')|\}$, and $\tilde{K}^-(F'') = \{p \in (P \backslash P(F'')) \mid (p, f''_i, f''_{i+1}, \ldots, f''_{|P(F'')|})$ is a feasible route for some $i = 1, \ldots, |P(F'')|\}$. 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 $i = 1, \ldots, |P(F')|$, 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: $P^+ = \{p \in P \mid x^*_p \ge 0.001\}$, $P^+_s = P_s \cap P^+$, 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 $\tilde{U}=(\tilde{u}_1,\tilde{u}_2,\ldots)$ be an infeasible route that only violates either (iv) or (v). If its internal route $(\tilde{u}_2,\ldots,\tilde{u}_{|P(\tilde{U})|-1})$ 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 $Q=(q_1,q_2,\ldots)$, let $I'(Q)$ and $I''(Q)$ be the number of rounds visited by routes $(q_2,q_3,\ldots,q_{|P(Q)|})$ and $(q_1,q_2,\ldots,q_{|P(Q)|-1})$, 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 andSep-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 $1,2,\ldots,|S|-1$ (resp. $2,3,\ldots,|S|$). 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:

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

where $K'(F')=P(F')\cup\{p\in P \mid (f'_1,f'_2,\ldots,f'_{|P(F')|},p)$ 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 $v_{sg}^A(p)$ and $v_s^B(p)$, 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 [22] to look for maximum-weight cliques. Because this is an NP-hard problem [23], 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 $v_{sg}^{\tilde{A}}(p)$ and $v_s^{\tilde{B}}(p)$ is converted to $\lfloor 100x_p^* \rfloor$, 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).

```
 1: procedure SEP-CLIQUE-ADJT-SECTION (solution $x^*$)
 2:   for all $s \in S, s \neq |S|$ do
 3:     for all $g \in G_s \cap G_{s+1}$ do
 4:       Build graph $\tilde{A}_{sg}$ for routes $p \in P_{sg} \cup P_{(s+1)g}$ with $x_p^* \geq 0.01$;
 5:       Assign weight $\lfloor 100x_p^* \rfloor$ to each vertex $v_{sg}^{\tilde{A}}(p)$ of $\tilde{A}_{sg}$;
 6:       Run the Cliquer solver on the weighted graph $\tilde{A}_{sg}$;
 7:       if a clique $C$ with weight greater than or equal to 101 is found then
 8:         for all $p \in P_{sg} \cup P_{(s+1)g}$ do
 9:           if $v_{sg}^{\tilde{A}}(p) \neq C$ and $v_{sg}^{A}(p)$ is adjacent, in $A_{sg}$, to all vertices of $C$ then
10:             Add $v_{sg}^{\tilde{A}}(p)$ to $C$;
11:           end if
12:         end for
13:         Add to the formulation inequality (7) for $C$;
14:       end if
15:     end for
16:   end for
17: end procedure
```

Algorithm 4. Separation routine for (8).

```
 1: procedure SEP-CLIQUE-SAME-SECTION(solution $x^*$)
 2:   for all $s \in S$ do
 3:     Build graph $\tilde{B}_s$ for routes $p \in P_s$ with $x_p^* \geq 0.01$;
 4:     Assign weight $\lfloor 100x_p^* \rfloor$ to each vertex $v_s^{\tilde{B}}(p)$ of $\tilde{B}_s$;
 5:     Run the Cliquer solver on the weighted graph $\tilde{B}_s$;
 6:     if a clique $C$ with weight greater than or equal to 101 is found then
 7:       for all $p \in P_s$ do
 8:         if $v_s^{\tilde{B}}(p) \neq C$ and $v_s^B(p)$ is adjacent, in $B_s$, to all vertices of $C$ then
 9:           Add $v_s^{\tilde{B}}(p)$ to $C$;
10:         end if
11:       end for
12:       Add to the formulation inequality (8) for $C$;
13:     end if
14:   end for
15: 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 the 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 2.40 GHz processor and 8 GB 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 [19], 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 s.

Algorithm 5. Enumeration of the model's variables.

```
 1: procedure ENUM-VARS
 2:   for all $s \in S$ do
 3:     ENUM-VARS-REC($s$, (), 0);   ▹ generates all trips in $P_s$
 4:   end for
 5: end procedure
 6:
 7: procedure ENUM-VARS-REC(section $s$, simple route $p$, simple route length $\ell$);
 8:   ▹ Append games to the simple route $p$ of length $\ell$ until it reaches the size of section $s$
 9:   if $\ell = w$ or $s(w-1)+1+\ell > 4n-2$ then
10:     Add variable $x_p$ to the formulation;
11:   else
12:     for all games $g$ in round $s(w-1)+1+\ell$ do
13:       Let $p'$ be $p$ with $g$ appended to it;
14:       if $p'$ is a feasible simple route then
15:         ENUM-VARS-REC($s$, $p'$, $\ell+1$);
```

```
16:             end if
17:         end for
18:     end if
19: end procedure
```

A second relevant aspect refers to the way we compare our running times against those in [7–9], 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

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).

Inst.	q_1	q_2	w	Lower bound						Solution time (s)					
				$\mathcal{M}_w^{R1}$	$\mathcal{M}_w^{R2}$	$\mathcal{M}_w^{R3}$	$\mathcal{M}_w^{R4}$	$\mathcal{M}_w^{R5}$	$\mathcal{M}_w^{R6}$	$\mathcal{M}_w^{R1}$	$\mathcal{M}_w^{R2}$	$\mathcal{M}_w^{R3}$	$\mathcal{M}_w^{R4}$	$\mathcal{M}_w^{R5}$	$\mathcal{M}_w^{R6}$
14	7	3	4	151 371.3	153 680.4	153 621.9	153 621.9	154 863.3	**155 041.5**	0.28	0.74	0.46	0.46	1.78	3.07
14	6	3	4	150 721.9	152 670.6	152 670.5	152 670.5	153 927.7	**154 187.7**	0.21	0.34	0.25	0.24	0.86	1.48
14	5	3	4	149 963.0	151 792.7	151 792.7	151 792.7	152 928.3	**153 095.0**	0.11	0.20	0.12	0.12	0.26	0.44
14A	7	3	4	145 138.1	147 900.5	147 872.8	147 872.8	148 706.3	**149 081.4**	0.24	0.51	0.31	0.30	1.00	2.62
14A	6	3	4	144 470.1	147 039.7	147 036.9	147 036.9	147 937.8	**148 428.9**	0.14	0.30	0.20	0.20	0.52	1.59
14A	5	3	4	143 973.3	146 259.4	146 259.4	146 259.4	147 065.8	**147 421.0**	0.09	0.18	0.10	0.09	0.21	0.57
14B	7	3	4	145 404.2	147 263.1	147 203.7	147 201.9	148 609.1	**148 776.4**	0.23	0.77	0.46	0.42	1.57	2.49
14B	6	3	4	144 787.5	146 422.8	146 422.8	146 422.0	147 785.0	**147 959.1**	0.17	0.42	0.27	0.25	0.84	1.42
14B	5	3	4	144 191.9	145 529.6	145 529.6	145 529.6	146 819.5	**147 138.0**	0.09	0.20	0.10	0.10	0.27	0.55
14C	7	3	4	143 650.8	146 213.7	146 039.7	146 036.3	147 415.5	**147 686.6**	0.47	0.77	0.47	0.44	1.83	3.55
14C	6	3	4	142 792.7	145 005.7	145 005.5	145 005.5	146 477.5	**146 644.6**	0.15	0.37	0.26	0.26	1.05	2.02
14C	5	3	4	142 150.3	144 305.7	144 305.8	144 305.8	145 776.0	**145 953.5**	0.11	0.14	0.11	0.11	0.31	0.55
16	8	4	10	171 712.1	176 495.3	176 495.7	176 391.6	181 095.3	**182 696.7**	20.88	130.07	108.95	91.91	867.52	2877.17
16	8	2	8	147 603.9	150 068.3	150 065.1	150 055.5	**152 853.0**	**152 853.0**	106.53	1455.42	1184.43	978.76	10 800.00	10 800.00
16	7	3	4	148 826.5	151 856.2	151 838.8	151 838.8	157 223.4	**157 377.2**	0.86	2.44	1.37	1.35	10.26	13.34
16	7	2	4	141 176.4	143 872.7	143 864.3	143 864.3	147 669.1	**147 853.3**	0.65	1.64	1.22	1.20	7.84	10.14
16A	8	4	10	184 979.7	188 739.3	188 739.9	188 649.9	194 032.1	**195 581.3**	26.15	148.27	127.81	101.74	1389.07	3151.16
16A	8	2	8	157 507.5	159 312.5	159 312.4	159 311.4	**164 893.2**	**164 893.2**	151.92	626.02	515.39	501.40	10 800.00	10 800.00
16A	7	3	4	164 939.1	167 559.7	167 539.7	167 539.7	169 767.1	**169 970.9**	0.93	2.28	1.29	1.28	8.39	12.56
16A	7	2	4	155 891.5	158 068.2	158 028.1	158 028.1	160 733.1	**160 905.6**	0.60	1.32	0.92	0.91	6.57	9.73
16B	8	4	10	192 098.3	197 791.3	197 790.3	197 764.2	202 768.2	**203 952.1**	33.71	161.88	130.53	118.79	1262.69	2759.63
16B	8	2	8	160 446.3	162 696.2	162 696.2	162 696.2	**167 241.3**	**167 241.3**	139.06	282.94	253.69	253.35	10 800.00	10 800.00
16B	7	3	4	161 769.3	165 175.7	165 144.9	165 144.9	169 537.3	**169 617.5**	0.87	2.02	1.25	1.25	10.78	12.85
16B	7	2	4	155 095.8	157 774.9	157 755.7	157 755.7	162 711.1	**162 936.6**	0.81	1.82	1.25	1.24	7.70	12.24
16C	8	4	10	184 697.1	190 615.5	190 615.7	190 556.8	195 863.7	**197 220.7**	28.44	96.56	78.94	66.35	595.89	1511.01
16C	8	2	8	161 294.0	164 240.8	164 241.2	164 233.7	167 196.7	**167 341.3**	123.97	400.58	355.47	327.12	7996.15	10 800.00
16C	7	3	4	164 417.6	166 988.9	166 930.6	166 930.6	169 894.2	**169 967.0**	0.96	2.57	1.54	1.53	10.13	12.91
16C	7	2	4	157 474.4	159 745.2	159 733.8	159 733.8	162 813.5	**162 903.3**	0.72	1.63	1.28	1.26	9.10	12.26
18	9	4	9	196 674.4	201 793.5	201 795.9	201 772.9	205 489.9	**205 743.8**	359.99	1294.34	1168.20	1086.19	7949.31	10 800.00
20	10	5	10	238 778.9	245 908.8	**245 960.6**	245 907.2	245 907.2	245 907.2	3582.13	10 800.00	10 800.00	9134.04	10 800.00	10 800.00
22	11	5	7	260 340.7	**266 460.9**	266 423.7	266 423.5	266 423.5	266 423.5	3413.92	10 800.00	9790.81	9764.14	10 800.00	10 800.00
24	12	6	7	292 168.5	**297 586.7**	297 556.7	297 556.7	297 556.7	297 556.7	10 800.00	10 800.00	10 800.00	10 800.00	10 800.00	10 800.00
26	13	6	6	327 716.1	333 517.8	**333 678.9**	**333 678.9**	**333 678.9**	**333 678.9**	8498.39	10 800.00	10 800.00	10 800.00	10 800.00	10 800.00
26	5	5	4	307 130.4	313 683.9	313 683.7	313 683.7	323 346.4	**323 684.2**	45.25	121.77	76.61	76.51	823.52	1140.39
28	14	7	5	364 692.1	374 601.6	**374 619.7**	**374 619.7**	**374 619.7**	**374 619.7**	6116.04	10 800.00	10 800.00	10 800.00	10 800.00	10 800.00
28	5	5	4	348 811.5	355 275.6	355 275.5	355 275.5	362 132.7	**362 585.2**	76.80	182.79	133.63	133.56	1140.83	1836.80
30	15	7	5	413 149.5	421 985.0	**422 012.1**	**422 012.1**	**422 012.1**	**422 012.1**	10 119.78	10 800.00	10 800.00	10 800.00	10 800.00	10 800.00
30	5	5	4	397 669.4	404 757.8	404 757.8	404 757.8	414 403.9	**414 865.9**	131.91	304.63	238.15	238.06	2305.95	3232.17
32	16	8	5	453 106.0	**462 944.9**	**462 944.9**	**462 944.9**	**462 944.9**	**462 944.9**	10 800.00	10 800.00	10 800.00	10 800.00	10 800.00	10 800.00
32	5	5	4	429 802.6	442 418.9	442 418.9	442 418.9	455 453.6	**455 885.7**	266.27	675.47	532.11	531.13	5654.81	7369.47

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	RSPCM
14	7	3	152 596.7	153 805.9	155 041.5	156 029.2	156 644.5	156 728.2	**156 799.3**	156 718.5	156 751.4	150 934.5	156 439.3	157 016.3
14	6	3	152 423.5	153 401.3	154 187.7	154 715.3	154 851.6	154 925.8	154 925.2	**154 975.5**	154 924.1	150 909.7	154 439.9	155 252.6
14	5	3	151 916.2	152 707.7	153 095.0	153 173.7	153 210.3	153 196.0	**153 250.3**	153 230.9	153 216.0	150 621.1	152 941.3	153 486.5
14A	7	3	146 776.2	147 937.8	149 081.4	149 982.8	150 408.5	**150 548.7**	150 469.7	150 443.3	150 471.1	145 049.6	149 992.7	150 707.9
14A	6	3	146 685.9	147 345.6	148 428.9	148 655.2	148 977.7	**149 096.8**	148 980.4	148 962.8	149 054.9	145 018.1	148 168.7	149 283.7
14A	5	3	146 362.1	146 727.2	147 421.0	147 528.0	147 704.9	**147 728.1**	147 592.5	147 616.4	147 622.2	144 884.9	147 097.5	147 999.3
14B	7	3	146 648.9	147 822.9	148 776.4	149 706.9	150 163.7	150 390.9	150 296.3	**150 533.7**	150 488.5	144 053.6	149 767.0	150 699.7
14B	6	3	146 531.2	147 337.4	147 959.1	148 733.0	148 842.6	148 938.8	148 880.3	148 964.4	**149 070.3**	144 054.1	148 243.9	149 240.8
14B	5	3	146 136.8	146 801.8	147 138.0	147 369.5	147 455.9	147 366.1	**147 617.9**	147 463.4	147 598.1	143 866.8	146 846.2	147 784.8
14C	7	3	145 208.9	146 438.2	147 686.6	148 550.6	148 934.8	149 106.3	149 076.5	**149 179.0**	149 138.1	143 276.4	148 613.2	149 489.7
14C	6	3	145 150.2	145 900.2	146 644.6	147 075.3	147 140.5	147 137.9	147 186.3	**147 250.3**	147 236.6	143 233.8	146 774.6	147 644.9
14C	5	3	144 847.6	145 536.4	145 953.5	146 105.9	146 161.2	146 210.1	**146 230.1**	146 212.4	146 145.2	143 149.5	145 794.4	146 597.9
16	8	4	157 573.8	163 238.6	167 294.2	174 986.5	178 375.3	179 509.2	181 402.9	181 063.4	**182 696.7**	152 507.6	184 187.6	185 056.8
16	8	2	144 224.4	146 585.4	148 506.6	150 717.2	152 104.1	152 027.0	**152 853.0**	149 662.8		142 134.8	155 045.2	155 712.5
16	7	3	153 900.5	155 923.5	157 377.2	158 188.3	158 460.0	158 466.9	**158 635.7**	158 626.9	158 222.0	150 532.2	158 257.4	158 883.4
16	7	2	144 176.9	146 475.8	147 853.3	148 369.0	148 534.3	148 574.7	**148 629.7**	147 865.3		142 145.2	148 341.8	148 980.7
16A	8	4	170 424.0	176 358.3	179 870.1	187 519.0	190 972.0	192 643.0	194 233.8	194 015.0	**195 581.3**	164 945.9	198 969.7	200 007.6
16A	8	2	157 765.9	159 929.7	161 300.4	164 116.1	164 511.7	164 512.8	**164 893.2**	163 339.0		155 641.5	166 575.5	167 360.0
16A	7	3	166 719.6	168 505.7	169 970.9	170 615.1	170 880.4	171 103.7	**171 251.7**	171 205.0	170 763.1	162 700.0	170 575.1	171 426.6
16A	7	2	157 682.6	159 756.9	160 905.6	161 525.5	161 617.4	161 715.0	**161 731.9**	160 634.4		155 963.5	161 571.2	161 975.1
16B	8	4	170 001.6	177 606.1	182 450.8	192 774.4	196 594.2	198 726.5	201 609.1	202 655.0	**203 952.1**	165 008.4	207 505.4	208 496.8
16B	8	2	157 967.4	161 411.3	163 379.5	166 207.6	166 892.4	166 889.7	**167 241.3**	165 511.6		156 402.2	169 363.4	170 040.3
16B	7	3	165 010.3	167 814.3	169 617.5	170 351.2	170 837.9	170 993.9	**171 110.0**	171 083.2	170 860.4	162 073.7	170 632.5	171 280.6
16B	7	2	157 936.9	161 267.6	162 936.6	163 536.6	163 812.7	163 814.9	**163 894.9**	162 738.2		156 442.1	163 539.7	164 160.9
16C	8	4	171 801.3	176 480.1	180 684.9	187 282.9	191 314.0	192 331.8	195 932.0	195 866.4	**197 220.7**	167 256.9	200 682.6	201 107.5
16C	8	2	160 069.3	161 761.6	163 307.9	165 357.6	166 335.5	166 488.9	**167 341.3**	165 813.3		158 947.2	168 783.6	169 270.9
16C	7	3	166 754.7	168 194.5	169 967.0	170 841.1	171 373.7	171 366.7	**171 596.4**	171 454.5	171 225.4	164 380.8	171 216.0	171 827.6
16C	7	2	160 006.2	161 596.2	162 903.3	163 393.5	163 858.2	**163 907.0**	163 771.9	163 122.1		158 906.2	163 850.8	164 182.8
18	9	4	187 132.4	192 865.6	196 085.7	200 213.7	201 992.9	203 102.9	203 813.8	**205 743.8**	204 027.0	181 430.7	212 121.6	212 793.6
20	10	5	220 179.1	229 339.6	234 897.0	237 819.0	243 686.7	244 967.0	242 752.5	244 388.4	**245 907.2**	213 513.3		
22	11	5	248 369.7	257 481.4	261 951.9	263 768.8	264 970.6	**266 423.5**	265 699.9			241 909.2		
24	12	6	277 716.5	286 579.0	293 662.7	295 812.5	295 828.3	**297 556.7**				270 662.0		
26	13	6	317 247.7	325 892.7	331 844.8	331 456.4	**333 678.9**					310 366.9		
26	5	5	314 581.9	321 305.4	323 684.2	**323 843.5**	323 070.6							
28	14	7	355 413.1	366 501.5	371 985.2	**374 619.7**						348 059.2		
28	5	5	352 797.1	360 574.9	**362 585.2**	362 571.0								
30	15	7	406 137.0	417 363.9	420 846.9	**422 012.1**						396 222.1		
30	5	5	404 198.1	412 133.4	**414 865.9**	413 756.4								
32	16	8	439 408.5	460 082.6	459 192.4	**462 944.9**						427 436.2		
32	5	5	435 769.2	452 052.4	**455 885.7**	450 956.9								

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 h) 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 1300–5900 miles (0.9 to 3.2%) on instances with up to 16 teams, and by about 5100–12 600 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 a value less than 0.001, $\mathcal{M}_w^{R3}$ actually 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 13 000 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-h 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 h. (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 [9,8] 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 [9,8] 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.

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 s to calculate but is never more than 155 miles below those obtained with $w > 6$, which take between 43.15 and 630.87 s 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 of 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 11 972 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 $\mathcal{M}_{4n-2}^{R6}$, i.e. a much larger value of w than the largest one we consider. Note, however, that the lower bounds obtained by $\mathcal{M}_6^{R6}$ 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, $\mathcal{M}_6^{R6}$ solves between 3.9 and 35.8 times faster

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.

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	10 800.00	10 800.00		0.60	50 247.89	7092.00	59 186.31
16	7	3	0.34	2.48	13.34	59.54	154.72	402.81	691.82	3445.89	10 800.00	0.71	2207.89	10 500.00	2959.44
16	7	2	0.15	1.58	10.14	114.91	515.94	1902.23	6051.61	10 800.00		0.69	2598.89	10 102.00	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	10 800.00	10 800.00		0.58	14 548.00	5403.00	17 380.10
16A	7	3	0.44	2.91	12.56	61.85	172.06	491.18	1381.38	5342.18	10 800.00	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	10 800.00		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	10 800.00	10 800.00		0.74	42 129.13	5162.00	53 974.07
16B	7	3	0.38	2.38	12.85	58.67	141.48	389.96	1051.29	4142.73	10 800.00	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	10 800.00		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	10 800.00	10 800.00		0.74	13 634.98	7380.00	22 101.70
16C	7	3	0.33	2.76	12.91	61.77	164.26	411.04	1012.05	3788.13	10 800.00	0.79	851.66	449.00	1658.40
16C	7	2	0.15	1.45	12.26	68.82	353.68	1518.86	10 800.00	10 800.00		0.71	3319.74	10 578.00	4790.26
18	9	4	1.65	32.74	165.18	686.79	1856.79	2428.94	6430.24	10 800.00	10 800.00	1.07	17 834.91		22 980.33
20	10	5	3.61	109.79	955.08	2038.08	4351.67	7870.05	10 800.00	10 800.00	10 800.00	1.53			
22	11	5	4.52	197.52	2233.87	7476.42	10 800.00	10 800.00	10 800.00			1.95			
24	12	6	11.68	387.45	10 800.00	10 800.00	10 800.00	10 800.00				2.90			
26	13	6	11.92	621.74	10 800.00	10 800.00	10 800.00					3.75			
26	5	5	3.54	100.85	1140.39	5094.29	10 800.00								
28	14	7	16.98	973.06	10 800.00	10 800.00						4.29			
28	5	5	5.08	134.21	1836.80	9919.03									
30	15	7	23.43	1588.45	10 800.00	10 800.00						4.16			
30	5	5	6.87	211.68	3232.17	10 800.00									
32	16	8	34.30	3920.92	10 800.00	10 800.00						7.09			
32	5	5	9.60	452.96	7369.47	10 800.00									

than RSPM is solved in [9], and between 1.5 and 67.9 times faster than RSPM is solved in [8]. These $\mathcal{M}_6^{R6}$ 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, $\mathcal{M}_w^{R6}$ does not perform well on 16-team instances with $q_1 = n = 8$. $\mathcal{M}_w^{R6}$'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 $\mathcal{M}_w^{R6}$ 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, otherwise, would have been part of valid fractional solutions to $\mathcal{M}_w^{R6}$. As a consequence, $\mathcal{M}_w^{R6}$'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 $\mathcal{M}_w^{R6}$ bound, they were obtained in 4 h and 57 min and in 6 h and 23 min, respectively.

$\mathcal{M}_w^{R6}$'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, $\mathcal{M}_w^{R6}$ can not only be solved within 3 h 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 $\mathcal{M}_w^{6}$ due to the good performance of $\mathcal{M}_w^{R6}$. 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 $\mathcal{M}_w^{6}$. 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 s) to solve. (See column $\mathcal{M}_w^{R6}$ 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 $\mathcal{M}_4^{R6}$ 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 h 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.

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 h. 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 34:45 h and 11:24 h, respectively, whereas we solve all 14-team instances with $q_1 = 5$ and $q_2 = 3$ in no more than 3:10 h (14, 14A, and 14B only require 31, 9, and 17 min, 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 h 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 $\pm$ its standard deviation). With execution times limited to 3 h, 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 h, 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 [20] (BB-DLB). In [7], DA results are reported for two time limits: up to 3 h (which we call DA3), and over 3 h (which we call DA+). Methods BB and BP were limited to run for 3 h, whereas BPC and BB-DLB were limited to 48 h. 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 h with those obtained by BB, BP, DA3, and BB-DLB within 3 h, and the ones found by BC within 24 h with those obtained by DA+, BPC and BB-DLB in more than 3 h. 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

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

Inst.	q_1	q_2	w	Time limit									
				3 h					24 h				
				Lower bound	Time (s)	Iterations	Nodes	Cuts	Lower bound	Time (s)	Iterations	Nodes	Cuts
14	7	3	4	158 578.5	10 800	657 8554	5462	16 004	159 271.8	86 400	342 65 950	25 830	26 417
14	6	3	4	157 469.3	10 800	868 0509	12 259	14 585	158 037.6	86 400	411 53 400	52 618	23 008
14	5	3	4	**154 962.0***	1823.44	195 1142	10 980	7622					
14A	7	3	4	152 537.1	10 800	723 1231	7036	15 132	153 257.5	86 400	365 54 874	33 291	25 296
14A	6	3	4	151 611.1	10 800	100 66 365	15 628	12 439	152 169.5	86 400	508 68 260	68 271	20 215
14A	5	3	4	**149 331.0***	524.79	836 360	3300	4342					
14B	7	3	4	152 647.1	10 800	715 7927	5776	15 660	153 191.1	86 400	343 91 926	23 233	27 720
14B	6	3	4	151 360.5	10 800	986 3357	14 392	12 906	151 821.9	86 400	458 98 826	58 131	22 350
14B	5	3	4	**149 455.0***	1003.7	166 8752	8887	5211					
14C	7	3	4	151 129.1	10 800	631 4167	4764	16 719	151 791	86 400	307 67 570	20 898	28 766
14C	6	3	4	149 820.4	10 800	931 6255	13 735	13 601	150 286.6	86 400	442 07 534	54 463	22 375
14C	5	3	4	148 333.2	10 800	810 6602	27 195	13 572	**148 349.0***	11 457.49	833 0667	29 379	13 705
16	8	4	10	183 386.2	10 800	109 3933	8	9802	185 936.7	86 400	736 1031	54	27 141
16	8	2	8	152 569.6	10 800	206 239	1	5230	153 725	86 400	121 4144	13	9179
16	7	3	4	159 841.1	10 800	326 1959	1649	13 424	160 664	86 400	289 10 558	13 240	28 292
16	7	2	4	149 560.8	10 800	263 4818	1573	13 536	149 988.5	86 400	236 03 021	12 705	30 384
16A	8	4	10	196 183.3	10 800	114 5764	12	13 048	198 330.5	86 400	741 5309	75	27 001
16A	8	2	8	164 625.8	10 800	207 790	1	4917	165 915.3	86 400	125 6325	12	9541
16A	7	3	4	173 028.2	10 800	306 5182	1052	12 339	174 226.3	86 400	283 61 114	9054	28 039
16A	7	2	4	162 675.1	10 800	281 7934	1696	13 326	163 052.0	86 400	269 85 443	14 253	27 689
16B	8	4	10	205 073.4	10 800	112 6908	8	12 883	207 781.1	86 400	729 6067	61	31 270
16B	8	2	8	167 241.6	10 800	190 363	1	4021	168 223.9	86 400	199 1551	7	7452
16B	7	3	4	172 131.7	10 800	297 1901	1235	13 041	173 178.0	86 400	291 06 668	10 769	27 111
16B	7	2	4	164 978.2	10 800	398 8299	2986	13 180	165 581.1	86 400	331 72 881	24 074	25 088
16C	8	4	10	198 274.6	10 800	119 0729	11	11 403	202 369.4	86 400	725 9649	76	22 285
16C	8	2	8	167 339.8	10 800	178 530	1	4245	167 530.7	86 400	112 4098	2	4983
16C	7	3	4	172 377.7	10 800	278 7489	1072	13 455	173 273.9	86 400	267 30 153	9351	29 042
16C	7	2	4	164 531.3	10 800	351 9161	1998	13 513	165 125.2	86 400	276 95 874	13 998	28 036
18	9	4	9	205 781.9	10 800	257 650	1	9386	206 759.4	86 400	168 1072	16	12 350
20	10	5	10	245 897.4	10 800	93 440	1	9842	250 372.6	86 400	572 624	1	17 704
22	11	5	7	266 415.6	10 800	189 486	1	20 364	269 735.5	86 400	106 2045	5	33 425
24	12	6	7	297 898.6	10 800	107 241	1	19 403	301 441.0	86 400	671 652	1	38 858
26	13	6	6	333 504.9	10 800	127 822	1	19 216	336 854.4	86 400	730 203	1	34 010
26	5	5	4	324 387.1	10 800	140 9902	247	11 564	324 753.2	86 400	111 58 237	2770	22 082
28	14	7	5	374 630.6	10 800	220 783	1	26 780	377 356.2	86 400	114 9468	1	36 817
28	5	5	4	363 072.1	10 800	916 593	91	10 571	363 541.4	86 400	786 5382	1422	20 455
30	15	7	5	422 026.0	10 800	144 276	1	20 906	424 537.6	86 400	749 184	1	34 822
30	5	5	4	415 296.4	10 800	614 318	43	9642	415 747.5	86 400	565 0380	1063	18 417
32	16	8	5	462 894.6	10 800	99 418	1	14 476	468 803.5	86 400	590 596	1	32 946
32	5	5	4	455 836.4	10 800	376 627	1	9909	456 685.9	86 400	384 3703	492	17 838

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 as those for BC, appear in the last 11 rows of Table 5. We start with the results obtained within 3 h 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 23 548.4/32 379.7/11 571.4 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 h 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 lower bounds are the best in all 11 cases. The average/max/min improvement in the lower bound values is of 38 248.0/99 108.5/2644.5 miles, with a standard deviation of 32 671.3 miles. Furthermore, note that BC lower bounds remain the largest ones even when it is restricted to run for no more than 3 h, 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 h (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 h, 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.

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.

Inst.	q_1	q_2	Time limit/method								
			3 h				24 h	> 3 h	48 h		
			BC	BB	BP	DA3	BC	DA+	BPC	BB-DLB	
14	7	3	158 578.5	154 175.6	157 812.8	156 536	159 271.8	159 797	158 900.2	**164 440.0***	(228.6)
14	6	3	157 469.3	154 036.7	155 570.4	156 551	158 037.6	156 551	157 083.4	**158 875.0***	(51.6)
14	5	3	154 962.0*	153 318.8	153 759.6	153 066		153 066	154 962.0*	**154 962.0***	(130.2)
14A	7	3	152 537.1	147 866.4	151 243.5	151 406	153 257.5	153 199	152 635.7	**158 760.0***	(123.0)
14A	6	3	151 611.1	147 773.1	149 285.4	150 998	152 169.5	150 998	151 043.2	**152 981.0***	(30.0)
14A	5	3	149 331.0*	147 358.3	147 966.4	148 299		148 299	149 331.0*	**149 331.0***	(67.2)
14B	7	3	152 647.1	147 159.6	151 165.8	149 910	153 191.1	151 059	152 517.6	**157 884.0***	(241.2)
14B	6	3	151 360.5	147 031.5	149 208.6	149 267	151 821.9	149 267	150 941.3	**152 740.0***	(103.2)
14B	5	3	149 455.0*	146 606.1	147 638.3	147 534		147 534	149 311.6	**149 455.0***	(63.0)
14C	7	3	151 129.1	146 104.6	150 101.6	151 122	151 791	151 581	150 925.9	**154 913.0***	(45.6)
14C	6	3	149 820.4	145 982.2	147 820	148 728	150 286.6	148 728	148 986.5	**150 858.0***	(100.2)
14C	5	3	148 333.2	145 598.1	146 622.1	146 764	148 349.0*	146 764	147 902.9	**148 349.0***	(764.4)
16	8	4	183 386.2	156 206.4	**193 457.1**	168 847	185 936.7	185 939	191 458	**infeas.**	(13 977.6)
16	8	2	152 569.6	145 829.7	**155 045.2**	151 481	153 725	151 481	**156 088.1**	145 531	
16	7	3	**159 841.1**	153 649.4	158 586	155 707	160 664	158 480	160 161.3	**165 765.0***	(24 296.4)
16	7	2	**149 560.8**	145 787	148 341.8	147 138	149 988.5	147 138	149 488	**150 433.0***	(66 118.8)
16A	8	4	196 183.3	168 882.5	**200 648.5**	185 119	198 330.5	185 119	206 141.2	**infeas.**	(13 549.2)
16A	8	2	164 625.8	158 645.6	**166 624.1**	162 788	165 915.3	162 788	**168 274.4**	160 739	
16A	7	3	**173 028.2**	166 459.3	172 420.1	170 342	174 226.3	172 964	172 471.4	**178 511.0***	(15 101.4)
16A	7	2	**162 675.1**	158 621.8	161 571.2	161 640	163 052	161 640	162 621.7	**163 709.0***	(57 922.2)
16B	8	4	205 073.4	169 684.4	**209 346.5**	188 195	207 781.1	208 418	215 520.6	**infeas.**	(13 764.6)
16B	8	2	167 241.6	159 525.2	**170 092.6**	167 768	168 223.9	167 768	**170 384.4**	165 737	
16B	7	3	**172 131.7**	165 753.2	172 058	170 940	**173 178.0**	173 023	172 695.9	**180 204.0***	(136 216.8)
16B	7	2	**164 978.2**	159 538.6	163 649.6	164 012	**165 581.1**	164 012	164 816	**167 190.0***	(138 118.8)
16C	8	4	198 274.6	170 370.6	**205 643.8**	179 213	202 369.4	188 561	206 368.8	**infeas.**	(14 216.4)
16C	8	2	167 339.8	161 296.6	**168 783.6**	163 543	167 530.7	166 001	**169 697.7**	164 541	
16C	7	3	**172 377.7**	166 562.3	171 767.6	170 133	**173 273.9**	171 377	172 754.6	**176 161.0**	
16C	7	2	**164 531.3**	161 241.1	163 850.8	163 305	**165 125.2**	163 305	164 625.7	**166 479.0***	(135 509.4)
18	9	4	**205 781.9**	184 222			206 759.4		**213 805.5**	193 632	
20	10	5	**245 897.4**	216 462.6			**250 372.6**			220 907	
22	11	5	**266 415.6**	245 030.5			**269 735.5**			243 052	
24	12	6	**297 898.6**	272 970			**301 441.0**			250 590	
26	13	6	**333 504.9**	312 705.5			**336 854.4**			289 651	
26	5	5	**324 387.1**				**324 753.2**	318 690			
28	14	7	**374 630.6**	350 290.9			**377 356.2**			322 208	
28	5	5	**363 072.1**				**363 541.4**	358 593			
30	15	7	**422 026.0**				**424 537.6**			339 331	
30	5	5	**415 296.4**	398 032.9		403 725	**415 747.5**	413 103			
32	16	8	**462 894.6**	430 514.9			**468 803.5**			369 695	
32	5	5	**455 836.4**				**456 685.9**	443 281			

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 A1 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 A1.

Table A1
Number of variables in our optimization models with w varying from 2 to 10.

Inst.	q_1	q_2	Parameter w								
			2	3	4	5	6	7	8	9	10
14	7	3	875	1463	3124	6707	14 480	26 097	43 858	92 909	157 212
14	6	3	875	1463	3124	6707	14 480	30 345	56 849	140 195	272 418
14	5	3	875	1463	3124	6707	16 354	37 921	79 866	224 807	465 183
16	8	4	1397	2961	5654	10 844	18 305	29 823	37 045	57 076	97 664
16	8	2	1397	3679	11 624	38 436	117 394	331 902	844 815	237 8813	7 202 046
16	7	3	1397	2961	7368	19 089	43 920	98 326	204 509	512 763	132 7904
16	7	2	1397	3679	11 624	38 436	117 394	331 902	973 859	303 1348	10 546 030
18	9	4	2081	5384	13 994	34 561	81 585	171 573	345 990	621 342	121 1598
20	10	5	2962	9069	28 332	72 276	172 373	393 620	818 194	1 492 658	2 417 177
22	11	5	4063	14 405	53 264	171979	535 731	1 497 634	4 036 925	10 939 472	
24	12	6	5407	21 810	97 332	368 098	1 167 827	3 219 784	8 889 449		
26	13	6	7009	31 677	158 375	717 269	2 615 617	9 823 065			
26	5	5	7009	31 677	158 375	717 269	3 329 528	17 481 485			
28	14	7	8909	44 638	248 893	1 318 194	5 688 863				
28	5	5	8909	44 638	248 893	1 318 194	7 072 643				
30	15	7	11 124	61 206	391 728	2 282 757	11 618 198				
30	5	5	11 124	61 206	391 728	2 282 757	14 162 234				
32	16	8	13 673	81 972	568 954	3 777 946	22 280 158				
32	5	5	13 673	81 972	568 954	3 777 946	26 687 469				

References

[1] Evans JR, Hebert JE, Deckro RF. Play ball—the scheduling of sports officials. Perspect Comput 1984;4(1):18–29.

[2] Evans JR. A microcomputer-based decision support system for scheduling umpires in the american baseball league. Interfaces 1988;18(6):42–51.

[3] Trick MA, Yildiz H. Benders' cuts guided large neighborhood search for the traveling umpire problem. Nav Res Logist 2011;58(8):771–81.

[4] Trick MA, Yildiz H, Yunes T. Scheduling major league baseball umpires and the traveling umpire problem. Interfaces 2012;42(3):232–44.

[5] Trick MA, Yildiz H. Locally optimized crossover for the traveling umpire problem. Eur J Oper Res 2012;216(2):286–92.

[6] Oliveira L, de Souza CC, Yunes T. Improved bounds for the traveling umpire problem: a stronger formulation and a relax-and-fix heuristic. Eur J Oper Res 2014;236(2):592–600.

[7] Wauters T, Malderen SV, Berghe GV. Decomposition and local search based methods for the traveling umpire problem. Eur J Oper Res 2014;238:886–98.

[8] Toffolo TAM, Malderen SV, Wauters T, Berghe GV. 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, 2014. p. 420–32.

[9] Xue L, Luo Z, Lim A. Two exact algorithms for the traveling umpire problem. Eur J Oper Res 2015;243(3):932–43.

[10] Wright MB. Scheduling english cricket umpires. J Oper Res Soc 1991;42 (6):447–52.

[11] Yavuz M, İnan UH, Fığlalı A. Fair referee assignments for professional football leagues. Comput Oper Res 2008;35(9):2937–51.

[12] Farmer A, Smith JS, MLT. Scheduling umpire crews for professional tennis tournaments. Interfaces 2007;37(2):187–96.

[13] Rasmussen RV, Trick MA. Round robin scheduling—a survey. Eur J Oper Res 2008;188:617–36.

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

[15] Fry MJ, Ohlmann JW. Introduction to the special issue on analytics in sports, part II: sports scheduling applications. Interfaces 2012;42(3):229–31.

[16] Trick MA, Yildiz H. Benders' cuts guided large neighborhood search for the traveling umpire problem. In: Van Hentenryck P, Wolsey L, editors. Proceedings of the fourth conference on integration of AI and OR techniques in constraint programming for combinatorial optimization problems (CP-AI-OR), Lecture notes in computer science, vol. 4510. Brussels, Belgium: Springer-Verlag; 2007. p. 332–45.

[17] Oliveira L, de Souza CC, Yunes T. On the complexity of the traveling umpire problem. Theor Comput Sci 2015;562:101–11.

[18] Trick MA. Traveling umpire problem: data sets and results. Available at ⟨http://mat.tepper.cmu.edu/TUP⟩, 2015.

[19] Toffolo TAM, Wauters T, Trick M. An automated benchmark website for the Traveling Umpire Problem. ⟨http://gent.cs.kuleuven.be/tup⟩, 2015.

[20] Toffolo TAM, Wauters T, Malderen SV, Berghe GV. Branch-and-bound with decomposition-based lower bounds for the Traveling Umpire Problem. Eur J Oper Res 2016;250:737–44. http://dx.doi.org/10.1016/j.ejor.2015.10.004.

[21] Kallehauge B, Boland N, Madsen OBG. Path inequalities for the vehicle routing problem with time windows. Networks 2007;49(4):273–93.

[22] Niskanen S, Östergård PRJ. Cliquer user's guide. version 1.0, Technical Report T48. Communications Laboratory, Helsinki University of Technology, Espoo, Finland, 2003.

[23] Karp RM. Reducibility among combinatorial problems. In: Miller RE, Thatcher JW, editors. Complexity of computer computations. Plenum Press; 1972. p. 85–103.