European Journal of Operational Research 204 (2010) 218–228

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A new branch-and-price algorithm for the traveling tournament problem

Stefan Irnich *

Chair of Logistics Management, Johannes Gutenberg University Mainz, Jakob-Welder-Weg 9, D-55128 Mainz, Germany

ARTICLE INFO

Article history:
Received 29 April 2009
Accepted 27 October 2009
Available online 1 November 2009

Keywords:
Timetabling
Sports league scheduling
Traveling tournament problem
Column generation
Branch-and-price

ABSTRACT

The traveling tournament problem (TTP) consists of finding a distance-minimal double round-robin tournament where the number of consecutive breaks is bounded. For solving the problem exactly, we propose a new branch-and-price approach. The starting point is a new compact formulation for the TTP. The corresponding extensive formulation resulting from a Dantzig-Wolfe decomposition is identical to one given by Easton, K., Nemhauser, G., Trick, M., 2003. Solving the traveling tournament problem: a combined interger programming and constraint programming approach. In: Burke, E., De Causmaecker, P. (Eds.), Practice and Theory of Automated Timetabling IV, Volume 2740 of Lecture Notes in Computer Science, Springer Verlag Berlin/Heidelberg, pp. 100–109, who suggest to solve the tour-generation subproblem by constraint programming. In contrast to their approach, our method explicitly utilizes the network structure of the compact formulation: First, the column-generation subproblem is a shortest-path problem with additional resource and task-elementarity constraints. We show that this problem can be reformulated as an ordinary shortest-path problem over an expanded network and, thus, be solved much faster. An exact variable elimination procedure then allows the reduction of the expanded networks while still guaranteeing optimality. Second, the compact formulation gives rise to supplemental branching rules, which are needed, since existing rules do not ensure integrality in all cases. Third, non-repeater constraints are added dynamically to the master problem only when violated. The result is a fast exact algorithm, which improves many lower bounds of knowingly hard TTP instances from the literature. For some instances, solutions are proven optimal for the first time.

1. Introduction

The traveling tournament problem is the problem of finding a double round-robin schedule that minimizes the overall distance traveled by all teams such that, for each team, the number of consecutive home stands and consecutive away games is bounded. The TTP was introduced by Easton et al. (2001) as an artificial sports league scheduling problem. Since then, it has attracted numerous researchers, probably because of its fast growing difficulty.

Formally, an even number $n \in 2\mathbb{N}$ of teams is given. Let $T := \{1, 2, \ldots, n\}$ denote the set of teams. In a single round-robin tournament, each team t plays against each of its opponent teams $T_{-t} := T \setminus \{t\}$ once. Assuming that the tournament takes place on a minimum number of matchdays (in the following called "time slots"), there are $n/2$ games in each of the $\bar{n} := n - 1$ time slots. In a double round-robin tournament, each team plays against each other team twice, once at home and once away. Consequently, there are $2\bar{n}$ time slots with again $n/2$ games in each slot. In the following, the time slots $S = \{1, 2, \ldots, 2\bar{n}\}$ are indexed by s.

For each team, the sequence of consecutive games played (home or at an opponent's venue) implies a tour: We identify teams and their venues and use indices $i, j \in T$ to refer to venues. A tour $p = (i_1, i_2, \ldots, i_{2\bar{n}}) = (i_s)_{s \in S}$ of team $t \in T$ contains each opponent venue $i \in T_{-t}$ exactly once (away games) and the home venue $i = t$ exactly $\bar{n}$ times (home games). A *break* occurs if a home game is followed by another home game or if an away game is followed by another away game, i.e., $i_s = i_{s+1} = t$ or $i_s, i_{s+1} \in T_{-t}$ for a time slot $s < 2\bar{n}$. In the TTP, the number of consecutive home stands and consecutive away games is bounded by L and U, i.e., the number of consecutive breaks is bounded by $L - 1$ and $U - 1$. Since all instances from the literature have $L = 1$, we solely focus on the upper bound U. Moreover, there are (optional) no-repeater constraints (NRCs) stating that the game t against t' must not be followed by the return game t' against t for any pair of teams $t, t' \in T$.

The objective of the TTP is distance minimization over all teams. Distances $D = (d_{ij})_{i,j \in T}$ between the venues are assumed symmetric and non-negative. Because each team t initially starts at home ($i_0 = t$) and finally returns home ($i_{2\bar{n}+1} = t$), the distance traveled along a tour $p = (i_1, i_2, \ldots, i_{2\bar{n}})$ is $\sum_{s=0}^{2\bar{n}} d_{i_s, i_{s+1}}$. Summing up, an instance of TTP is defined by distances $D = (d_{ij})$, an integer U, and optional NRCs. The task is to compute a distance-minimal set of

* Tel.: +49 61313922007; fax: +49 61313922097.
E-mail address: irnich@uni-mainz.de

0377-2217/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2009.10.024

break-feasible tours that compose a double round-robin tournament (without repeater games).

The recent survey by Rasmussen and Trick (2008) devotes a full section to the TTP and gives a comprehensive overview of state-of-the-art approaches. While there is a large variety of metaheuristics available (see survey), the literature on exact algorithms is scarce: Based on the so-called independent lower bound (ILB) relaxation, Easton et al. (2001) are able to solve TTP instances with $n = 4$ and $n = 6$ teams. The same authors present in (Easton et al., 2003) a column-generation approach for TTP without NRCs, where the subproblems, one for each team, consist of generating least-cost (=reduced cost) tours. Subproblems are solved by constraint programming (CP), and integrality of the overall solution is enforced by branch-and-bound. A method to improve lower bounds is presented by Urrutia et al. (2007), but (to the best of our knowledge) no exact approach has been implemented based on this idea. The special case of mirrored TTP with uniform distances is treated in (Urrutia and Ribeiro, 2004), where instances with up to $n = 12$ teams are solved to optimality. Cheung (2008) is able to solve small ($n \leqslant 8$) mirrored TTP benchmark problems from http://mat.tepper.cmu.edu/TOURN (with non-constant distances) using a two-phase approach: First, all different one-factorizations are computed and then, for each of them, a timetable-constrained distance minimization problem is solved afterwards. Cheung (2009) proposes a Benders decomposition approach, which improves the lower bound for larger instances of the same type, i.e., for n between 10 and 24. It seems rather unlikely that these methods are successfully applicable to general or larger TTP instances. This motivated our research on fast exact approaches for the TTP.

This paper is structured as follows: Section 2 presents the new compact formulation for the TTP. The proposed Dantzig-Wolfe decomposition of this model and the resulting master and pricing problems are derived in Section 3. Section 4 devises the corresponding solution methods for the integer programming master. Computational results are discussed in Section 5 and final conclusions are drawn in Section 6.

2. Compact formulation

In integer column generation, a well-structured compact (=original) formulation is extremely important for devising branching rules and adding valid inequalities to the extensive (=column-generation) formulation (cf. Lübbecke and Desrosiers, 2005; Spoorendonk, 2008). The basic idea of the new compact formulation for the TTP is to represent the movement of each team, from venue to venue, by a path in a time-discrete network. Fig. 1 depicts the network for a TTP with $n = 4$ teams. For each time slot $s \in S$, a given team $t \in T$ visits one of the venues $i \in T$, i.e., its own home venue or any opponent's venue. The nodes of the network of team t are therefore

$$V^t = \{v_{is} : i \in T, s \in S\} \cup \{v_{t0}, v_{t,2\bar{n}+1}\}.$$

The two extra nodes v_{t0} and $v_{t,2\bar{n}+1}$ are source and sink. They model the fact that team t always starts at home and returns home at the end. The possible movements in space and time are given by arcs $(v_{is}, v_{j,s+1})$ with the meaning that team t is traveling from venue i at time s to venue j, where the next game takes places at time $s + 1$. To lighten the notation, arcs $(v_{is}, v_{j,s+1})$ are encoded by triplets (i,j,s), where the set of all feasible triplets for team t is

$$A^t = \{(t,j,0) : j \in T\} \cup \{(i,t,2\bar{n}) : i \in T\} \cup \{(i,j,s) : i,j \in T, (i \neq j \text{ or } i = j = t), s \in S \setminus \{2\bar{n}\}\}$$

The subset $B^t = \{(i,j,s) \in A^t : (i = j = t, s \neq 0, 2\bar{n}) \text{ or } i,j \in T_{-t}\} \subset A^t$ represents home stands and consecutive away games and, thus, defines the set of *break arcs*. Note that a home game in $s = 1$ or $s = 2\bar{n}$ does *not* impose a break. We use A^t to refer to arcs of the network $\mathcal{N}^t = (V^t, A^t)$ and also to index the corresponding decision variables $x^t_{ijs} \in \{0,1\}$ of the following compact formulation:

$$z_{ttp} = \min \sum_{t \in T} \sum_{(i,j,s) \in A^t} d_{ij} x^t_{ijs} \tag{1}$$

$$\text{s.t.} \quad \sum_{i:(i,j,s-1) \in A^t} x^t_{ij,s-1} - \sum_{i:(j,i,s) \in A^t} x^t_{jis} = 0 \quad \text{for all } t,j \in T, s \in S \tag{2}$$

$$\sum_{s \in S} \sum_{j:(i,j,s) \in A^t} x^t_{ijs} = 1 \quad \text{for all } t \in T, i \in T_{-t} \tag{3}$$

$$\sum_{u=0}^{U-1} \sum_{(i,j,s+u) \in B^t} x^t_{ij,s+u} \leqslant U - 1 \quad \text{for all } t \in T, s \in S : s \leqslant 2\bar{n} - U \tag{4}$$

$$\sum_{i \in T_{-t}} \sum_{j:(i,j,s) \in A^t} x^t_{ijs} + \sum_{t' \in T_{-t}} \sum_{j:(t,j,s) \in A^{t'}} x^{t'}_{tjs} = 1 \quad \text{for all } t \in T, s \in S \tag{5}$$

$$x^t_{ijs} \in \{0,1\} \quad \text{for all } t \in T, (i,j,s) \in A^t \tag{6}$$

The objective (1) is the minimization of the overall distance traveled by all teams. Flow conservation for each team is implied by (2), constraints (3) state that all teams must visit all opponent venues exactly once, and constraints (4) limit the number of consecutive breaks. The coupling constraints (5) are the crucial part of the model: They guarantee that each team t plays a game in each time slot s, either playing away against an opponent t' (first sum) or playing home as the opponent of another team t' (second sum).

One advantage of this formulation is that NRCs are simple to add: $x^t_{tt's} + x^{t'}_{tt's} \leqslant 1$ must hold for all $t \in T, t' \in T_{-t}, s \in S, s < 2\bar{n}$. It means that teams t and t' are not allowed to play against each other in consecutive time slots, first in slot s home at t, directly followed by the return game in slot $s + 1$ home at t'. By swapping the role of t and t' and anticipating that the four corresponding arcs are pairwise incompatible, NRCs can be lifted to

$$x^t_{tt's} + x^{t'}_{tt's} + x^{t'}_{t'ts} + x^t_{t'ts} \leqslant 1 \quad \text{for all } t,t' \in T, t < t'; \quad s \in S, s \neq 2\bar{n}. \tag{7}$$

These are $n\bar{n}(2\bar{n} - 1)/2 = \mathcal{O}(n^3)$ lifted NRCs. Thus, (1)–(7) is the compact formulation for the TTP with NRCs.

3. Extensive formulation

The extensive formulation consists of two parts: First, we briefly state the master program, which is identical to one used in Easton et al. (2003). The new aspect is the incorporation of the NRCs, which are directly derived from the compact formulation presented above. Second, we discuss the structure of the subproblems.

3.1. Master problem

The application of the Dantzig–Wolfe decomposition principle to the model (1)–(7) is straightforward: Note that the only constraints involving more than one team are the coupling constraints (5) and the NRCs (7). Therefore, constraints 2, 3, 4 and (6) define the domains of the subproblems. They decompose into n domains and corresponding subproblems, one for each team $t \in T$: Let $P^t = \{(x^t_{ijs}, (i,j,s) \in A^t$: satisfying 2, 3, 4 and (6)$\}$. The set P^t is the set of feasible paths from source v_{t0} to sink $v_{t,2\bar{n}+1}$ in the network $\mathcal{N}^t$. Such a path must be break-feasible and visit each opponent venue exactly once. The cost of a tour $p = \left(\bar{x}^t_{ijs}\right) \in P^t$ is $c_p = \sum_{(i,j,s) \in A^t} d_{ij} \bar{x}^t_{ijs}$.

Easton et al. (2003) were the first to present a column-generation formulation based on the tour variables $\lambda^t_p, p \in P^t$ for the TTP, but without deriving it from an original compact formulation. We define $P^t_{t's}$ to be the subset of tours in P^t, where team t plays away in slot s against team t', i.e., $p \in P^t_{t's}$ visits venue of t' in slot s (p touches the node $v_{t's}$). The extensive formulation is as follows:

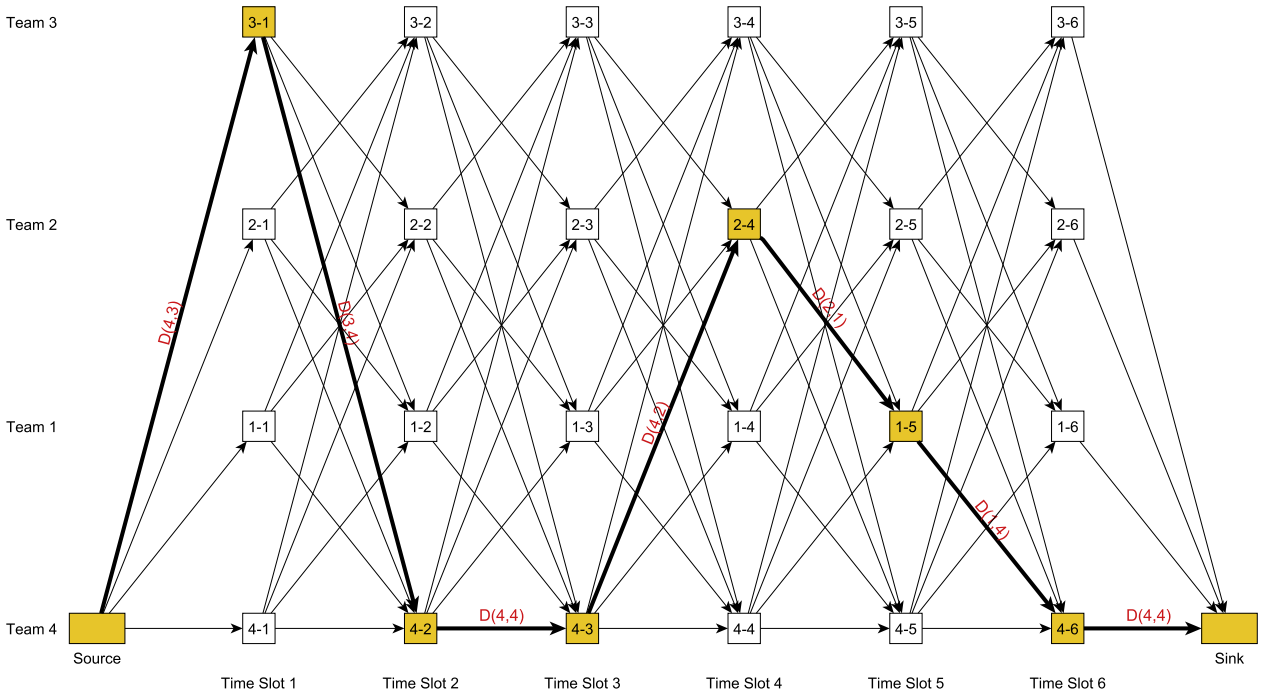

Fig. 1. Network $\mathcal{N}^4$ of team $t = 4$ for a TTP with $n = 4$ teams.

$$\min \quad \sum_{t\in T}\sum_{p\in P^t} c_p \lambda_p^t \tag{8}$$

$$\text{s.t.} \quad \sum_{t'\in T_{-t}}\sum_{p\in P^t_{t's}} \lambda_p^t + \sum_{t'\in T_{-t}}\sum_{p\in P^{t'}_{ts}} \lambda_p^{t'} = 1 \quad \text{for all } t\in T, s\in S \tag{9}$$

$$\sum_{p\in P^t} \lambda_p^t = 1 \quad \text{for all } t\in T \tag{10}$$

$$\lambda_p^t \in \{0,1\} \quad \text{for all } t\in T, p\in P^t \tag{11}$$

The overall distance traveled by all teams is minimized by (8). The reformulation of the coupling constraints (5) in the tour variables is given by (9). Finally, the convexity constraints (10) state that one tour has to be selected for each team. The binary requirements on the original arc variables x_{ijs}^t imply integrality of the path variables λ_p^t (cf. Desaulniers et al., 1998, p. 75). It is easy to see that equalities in (9) can be replaced by $\geqslant$, which gives a better stabilized master.

Defining P_{ijs}^t as the subset of paths of P^t that contain the arc $(i,j,s)\in A^t$, the reformulation of the NRCs is

$$\sum_{p\in\left(P^t_{tt's}\cup P^t_{t'ts}\right)} \lambda_p^t + \sum_{p\in\left(P^{t'}_{tt's}\cup P^{t'}_{t'ts}\right)} \lambda_p^{t'} \leqslant 1 \quad \text{for all } t,t'\in T, t<t'; s\in S, s\neq 2\bar{n}. \tag{12}$$

In the following, we refer to the LP-relaxation of (8)–(11) or (8)–(12) as the *master program (MP)*. The LP over a subset of the path variables λ_p^t is called the *restricted master program (RMP)*.

3.2. Subproblems

There is one subproblem (=pricing problem) for each team $t\in T$. The task of the t-th subproblem is to determine a tour for team t and the corresponding master program variable λ_p^t with minimum reduced cost. We consider a tour $p = (i_0, i_1, \ldots, i_{2\bar{n}}, i_{2\bar{n}+1})$ and recall that $i_0 = i_{2\bar{n}+1} = t$ holds. For $(i,j,s)\in A^t$ we write $(i,j,s)\in p$ if and only if $i = i_s$ and $j = i_{s+1}$. The dual variables associated with the constraints of the master problem are $\pi = (\pi_{ts})$ for the coupling constraints (9), $\mu = (\mu_t)$ for the convexity constraints (10), and, if present, $\beta = (\beta_{tt's})$ for the NRCs (12). With these definitions, the reduced cost of the tour variable λ_p^t is $\tilde{c}_p^t = \sum_{(i,j,s)\in p}\tilde{c}_{ijs}^t - \mu_t$, where the coefficients $\tilde{c}_{ijs}^t$ are defined by

$$\tilde{c}_{ijs}^t(\pi,\beta) = d_{ij} - \begin{cases} \pi_{ts}+\pi_{is}, & \text{if } i\in T_{-t} \\ 0, & \text{otherwise} \end{cases} - \begin{cases} \beta_{ijs}, & \text{if } i<j, s\neq 2\bar{n} \\ \beta_{jis}, & \text{if } i>j, s\neq 2\bar{n} \\ 0, & \text{otherwise} \end{cases}. \tag{13}$$

This representation shows that the pricing problem is in fact a shortest-path problem over the network $\mathcal{N}^t$ (we leave out the index t for the decision variables):

$$\zeta_{PP}^t(\pi,\beta) = \min \sum_{(i,j,s)\in A^t} \tilde{c}_{ijs}^t(\pi,\beta) x_{ijs}$$

$$\text{s.t.} \quad \sum_{i:(i,j,s-1)\in A^t} x_{ij,s-1} - \sum_{i:(j,i,s)\in A^t} x_{jis} = 0 \quad \text{for all } j\in T, s\in S \tag{14}$$

$$\sum_{s\in S}\sum_{j:(i,j,s)\in A^t} x_{ijs} = 1 \quad \text{for all } i\in T_{-t} \tag{15}$$

$$\sum_{u=0}^{U-1}\sum_{(i,j,s+u)\in B^t} x_{ij,s+u} \leqslant U-1 \quad \text{for all } s\in S: s\leqslant 2\bar{n}-U \tag{16}$$

$$x_{ijs}\in\{0,1\} \quad \text{for all } (i,j,s)\in A^t \tag{17}$$

The flow conservation constraints (14) together with the requirements to visit each opponent venue (15) imply that the solution is a path from v_{t0} to $v_{t,2\bar{n}+1}$. Moreover, this path must be break-feasible, which is implied by the constraints (16).

4. Solution of the integer master problem

We now derive fast solutions methods for the TTP based on the decomposition into master and pricing problems. The crucial part for efficiency is the representation of the pricing problems as ordinary shortest-path problems.

4.1. Solution of the subproblems

Not every path from v_{t0} to $v_{t,2\bar{n}+1}$ in the network $\mathcal{N}^t$ represents a feasible tour. The covering of each opponent venue and break-feasibility must be guaranteed by additional constraints. We show that these constraints can be handled by defining resources so that the subproblem is a shortest-path problem with resource constraints (SPPRC) (cf. Irnich and Desaulniers, 2005). Break-feasibility can be guaranteed with one constrained resource $b \in \{0,1,\dots,U-1\}$ that is increased whenever a break occurs, and is reset to zero otherwise. Note that both operations, incrementation and reset, are non-decreasing *resource extension functions (REFs)* (cf. Desaulniers et al., 1998, 2008).

Covering each opponent $i \in T_{-t}$ exactly once is a path-structural constraint (Irnich and Desaulniers, 2005, p. 38f). Since the tour represents a tight schedule with exactly $\bar{n}$ home games and the same number of away games, these covering constraints are already fulfilled if at most one node from each node set $V_i^t := \{v_{is} : s \in S\}$ is visited (for each $i \in T_{-t}$) and at most $\bar{n}$ nodes from $V_t^t := \{v_{ts} : s \in S\}$. A maximum number of visits can easily be handled by *visiting counters*, one for each team $i \in T$. For counting breaks and the $\bar{n}+1$ different visits, we define $1+n = 1+(\bar{n}+1)$ resources with feasible domain $\mathscr{D} := \{0,1,\dots,U-1\} \times (\{0,1\}^{\bar{n}} \times \{0,1,\dots,\bar{n}\}) \subset \mathbb{Z}_+^{1+n}$. At the source node v_{t0}, all resources are their lower bounds $(0,(\mathbf{0}))$. The resource update along an arc $(i,j,s) \in A^t$ is given by the following REF:

$$f_{ijs}^t(b,(n_k)_{k\in T},\tilde{c}) := \begin{cases} \Big(b+1,(n_k+\delta_{jk}(1-\delta_{s,2\bar{n}}))_{k\in T},\tilde{c}+\tilde{c}_{ijs}^t\Big), \\ \quad \text{if } (i,j,s) \in B^t \\ \Big(0,(n_k+\delta_{jk}(1-\delta_{s,2\bar{n}}))_{k\in T},\tilde{c}+\tilde{c}_{ijs}^t\Big), \\ \quad \text{otherwise} \end{cases} \tag{18}$$

Herein, δ_{pq} is the Kronecker symbol, i.e., $\delta_{pq}=1$ for $p=q$ and 0 otherwise. In the first component, the number of breaks is increased or reset depending on whether the arc is a break arc. For the second component, note that the node $j = v_{t,2\bar{n}+1}$ does not represent a home game and that, when entering this node, n_t must not be incremented. This explains the factor $(1-\delta_{s,2\bar{n}})$ in the above REF. The third component is simply the addition of the arc reduced costs as defined by (13).

Summing up, a partial path in $\mathcal{N}^t$, ending at node v_{is} with label $(b,(n_k),\tilde{c})$ can be extended to node $v_{j,s+1}$ if and only if $f_{ijs}^t(b,(n_k),\tilde{c}) \in \mathscr{D} \times \mathbb{R}$ holds. This establishes a resource-based definition of feasible tours.

4.1.1. Limited dominance and network expansion

The standard approach for solving SPPRCs is dynamic programming using a labeling algorithm, i.e., systematically building new paths, starting from the trivial path (v_{t0}) at the source node v_{t0}, by extending them one-by-one into feasible directions. Typically, labels (reached states together with minimum costs to reach them) are stored at each node $v_{is} \in V^t$. A dominance algorithm identifies a subset of these labels that are provably non-useful for the generation of a least-cost path, and if so, discards them. For a comprehensive survey on SPPRC labeling algorithms and alternative solution methods ([see]Irnich and Desaulniers, 2005).

For the TTP subproblem, assume that two labels $\ell_1 = (b_1,(n_{1k}),\tilde{c}_1)$ and $\ell_2 = (b_2,(n_{2k}),\tilde{c}_2)$ with associated partial paths ending at the same node $v_{is} \in V^t$ are given. The standard resource-based dominance between labels requires a component-wise comparison. If $\ell_1 \leqslant \ell_2$, then ℓ_1 dominates ℓ_2, implying that ℓ_2 can be discarded. However, this dominance relation is weak for TTP subproblems because only labels with identical visiting counters are possible candidates, as can be seen as follows: Labels that refer to the same node v_{is} have $s = \sum_{t\in T} n_{1t} = \sum_{t\in T} n_{2t}$. If $n_{1i} < n_{2i}$ in one component i, then there must exist another component j with $n_{1j} > n_{2j}$, i.e., the two labels are incomparable.

As a consequence, dominance applies only to labels with identical visiting counters. Comparable labels can only differ in their break counter and reduced cost. Thus, by dominance no more than $U-1$ labels associated to the U different states $(b,(n_k))$, $b \in \{0,1,\dots,U-1\}$ can be discarded. Since U is typically a small integer, mostly $U=3$ for the benchmark problems, we propose ignoring this limited dominance. This avoids a complex management (addition, comparison, and deletion) of labels as needed in standard labeling approaches. Instead the pricing problem can be solved over an expanded acyclic network $\mathcal{N}_{ex}^t = (V_{ex}^t, A_{ex}^t)$, but as an ordinary shortest-path problem (SPP). The node set V_{ex}^t of the expanded network consists of the reachable states $(b,(n_k)) \in \mathscr{D}$, each associated to an original node $v_{is} \in V^t$. To lighten the notation and to distinguish between states associated to different nodes v_{is}, states are denoted by (i,s,b,N) in the following. Herein, the set $N \subseteq T_{-t}$ represents the visited opponent venues, i.e., $N = \{i \in T_{-t} : n_i = 1\}$. Note that a state must necessarily fulfill $s = \sum_{k\in T} n_k$. Thus, knowing the integer s and the set N is equivalent to knowing all visiting counters because $n_i = 1$ if and only if $i \in N$ for $i \neq t$ and $n_t = s - |N|$.

In the expanded network, arcs exist between nodes (i,s,b,N) and $(j,s+1,b',N')$ if $(i,j,s) \in A^t$ and $f_{ijs}^t(b,(n_k)) = (b',(n'_k)) \in \mathscr{D}$, where N and N' correspond to the visiting counters (n_k) and (n'_k), respectively. Starting from node $(t,0,0,\emptyset) \in V_{ex}^t$, arcs emanate to the nodes associated to time slot 1, i.e., to the nodes $(i,1,0,\{i\})$ for $i \in T_{-t}$ and to node $(t,1,0,\emptyset)$. Fig. 2 depicts parts of the network $\mathcal{N}_{ex}^4 = \left(V_{ex}^4, A_{ex}^4\right)$ and the path associated with the tour $(3,t,t,2,1,t)$ of team $t=4$ for a TTP with $n=4$ teams.

4.1.2. Bidirectional labeling

Righini and Salani (2006) have shown that bidirectional labeling can accelerate the solution of SPPRCs. Their observation was that often fewer labels are created if paths are built as concatenations of forward and backward paths that are merged "in the middle". In general, there exist more feasible $s-t$-paths of length $\leqslant L$ (from a source s to a sink t) than feasible forward and backward paths of length $\leqslant L/2$ (starting at s or ending at t). The crucial point for the efficacy of bidirectional labeling is the ability to bound the extension of forward and backward paths, while still guaranteeing that each feasible path is the concatenation of a generated forward and backward path. In case of the TTP subproblem, the bounding is trivial because the networks are layered: the middle is the connection of nodes of slot $s = \bar{n}$ with those in slot $s' = \bar{n}+1$. Hence, forward paths are associated to the nodes v_{is} with $s \in \{0,1,\dots,\bar{n}\}$ and backward paths to the nodes $v_{is'}$ with $s' \in \{\bar{n}+1,\bar{n}+2,\dots,2\bar{n}+1\}$.

For the implementation, we exploit the fact that forward and backward extension of labels is symmetric in the following sense: All labels are of the form $(b,(n_k)_{k\in T},\tilde{c})$ and the REF (18) can be used for a forward extension along the arc $(i,j,s) \in A^t$ or a backward extension along an arc $(j,i,2\bar{n}-s) \in A^t$. The only difference is the cost component, where in the backward case $\tilde{c}_{ji,2\bar{n}-s}^t$ has to be added. Thus, only the digraph induced by the nodes of the slots $s \in \{0,1,\dots,\bar{n}\}$ and the connecting arcs between slots $\bar{n}$ and $\bar{n}+1$ have to be stored. Let $V_{s,ex}^t$ for $s \in \{0,1,\dots,2\bar{n}+1\}$ denote the nodes of slot s, let $V_{\leqslant \bar{n},ex}^t := \bigcup_{s=0}^{\bar{n}} V_{s,ex}^t$ be the nodes "in the first half", and let $A_{s,ex}^t$ be the arcs connecting slot $s-1$ with s.

The merging step, i.e., the concatenation of a feasible forward label $(b,(n_k)_{k\in T},\tilde{c})$ (associated to node $v_{i,\bar{n}}$) with a feasible backward label $(b',(n'_k)_{k\in T},\tilde{c}')$ (associated to node $v_{j,\bar{n}+1}$) must proceed in two steps. First, feasibility has to be checked: The concatenation is a feasible path if and only if $n_k + n'_k = 1$ for all $k \in T_{-t}$ and the resulting path is break-feasible. Break-feasibility is fulfilled if

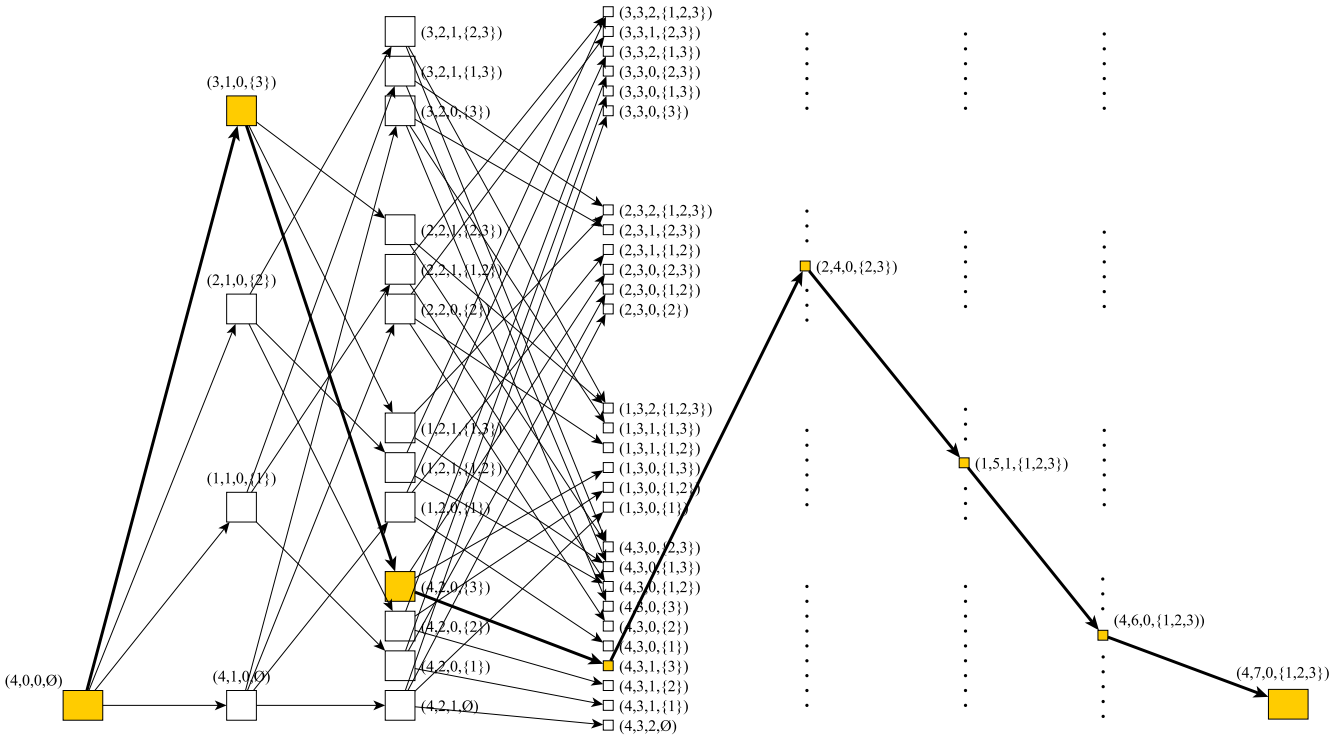

Fig. 2. Expanded network $\mathcal{N}_{ex}^4$ with tour $(3,t,t,2,1,t)$ for team $t=4$; A state (i,s,b,N) is associated to node v_{is}, has b breaks and opponent venues $N \subseteq T_{-t}$ already visited.

$(i,j,s) \notin B^t$ or $b+b'+1 \leqslant U-1$. Second, the cost of the concatenation is $\tilde{c}+\tilde{c}_{ij\bar{n}}^t+\tilde{c}'$.

Note that all break-feasible concatenations of forward and backward labels between nodes $V_{\bar{n},ex}^t$ and $V_{\bar{n}+1,ex}^t$ are *not* given by the arcs $A_{\bar{n}+1,ex}^t$. For instance, taking $U=3$ and $n=t=6$, the labels $(b,\{1,2,3\},\tilde{c})$ associated with node v_{35} (away at team 3 in slot 5) can be connected with labels $(b',\{4,5,6\},\tilde{c}')$ associated with node v_{66} (home in slot 6) independent of the values $b,b' \in \{0,1,2\}$. These are nine feasible connections, three for each associated state. In contrary, for each of the three states $(b,\{1,2,3\}), b \in \{0,1,2\}$ (associated to node v_{35}) there is one unique arc to the state (0,{1,2,3}) associated to node v_{66}.

For the implementation of bidirectional labeling, we store the first half of the network $\mathcal{N}_{ex}^t$, i.e., the graph induced by $V_{\leqslant \bar{n},ex}^t$. We store the nodes and arcs as well as two cost labels at each node, one for forward labeling and the other one for backward labeling (associated with the second half). The break-feasible concatenations between nodes of $V_{\leqslant \bar{n},ex}^t$ are stored (as pairs) $A_{ex,mid}^t \subset V_{\leqslant \bar{n},ex}^t \times V_{\leqslant \bar{n},ex}^t$. The reduced costs $\tilde{c}_{ijs}^t$, which change from iteration to iteration, can be stored within and retrieved from the original network. Thus, no cost needs to be stored and updated in the extended network $\mathcal{N}_{ex}^t$. This further reduces the computational effort for pricing.

4.1.2.1. Comparison of network sizes. The advantage of bidirectional labeling can be illustrated by comparing the sizes of the networks. Our goal is to precisely specify the memory consumption and effort of labeling in both the mono- and the bidirectional case. Table 1 shows the number of nodes and arcs of the monodirectional expanded network $\mathcal{N}_{ex}^t$, itemized by slots s. The number of arcs between slot $\bar{n}$ and $\bar{n}+1$ is printed in bold type. Moreover, for the bidirectional approach, the number $\left|A_{ex,mid}^t\right|$ of break-feasible connections is listed for different values of n.

It is obvious that in the monodirectional expanded network $\mathcal{N}_{ex}^t$ the second half is always larger than the first. For $n=4$ teams, there are $1+4+13+28=46$ nodes in the first half, but $1+12+32+40=85$ in the second. There are $\left|A_{4,ex}^t\right|=63$ arcs between slot 3 with 4 (the monodirectional case) compared to $\left|A_{ex,mid}^t\right|=96$ break-feasible connections in the middle (bidirectional case).

For a comparison of the mono- and the bidirectional case, data taken from Table 1 are further condensed, and presented in Table 2. The first columns summarize the network size in the monodirectional case, by taking the sum of the corresponding values in Table 1. The bidirectional case is shown in the second division of Table 2. The number of nodes is twice that of those in the first half. Clearly, there is always a smaller number of nodes in the bidirectional network, and the reduction is between approx. 15% and 30%. The number of

Table 1
Sizes of the monodirectional networks $\mathcal{N}_{ex}^t$ for $L=1$ and $U=3$.

n	Number of nodes in slot s/arcs to slot s		$\left\|A_{ex,mid}^t\right\|$
4	$\left\|V_{s,ex}^t\right\|$	(1, 4, 13, 28, 40, 32, 12, 1)	
	$\left\|A_{s,ex}^t\right\|$	(4, 13, 34, **63**, 70, 32, 12)	96
6	$\left\|V_{s,ex}^t\right\|$	(1, 6, 31, 96, 180, 255, 255, 226, 182, 93, 18, 1)	
	$\left\|A_{s,ex}^t\right\|$	(6, 31, 136, 310, 575, **710**, 640, 546, 322, 72, 18)	1480
8	$\left\|V_{s,ex}^t\right\|$	(1, 8, 57, 232, 504, 973, 1204, 1295, 1435, 1442, 1282, 933, 549, 171, 24, 1)	
	$\left\|A_{s,ex}^t\right\|$	(8, 57, 358, 889, 2275, 3766, 4165, **4571**, 4879, 4662, 3655, 2326, 814, 128, 24)	9072
10	$\left\|V_{s,ex}^t\right\|$	(1, 10, 91, 460, 1092, 2715, 4104, 5229, 6564, 7458, 7434, 7020, 7048, 6554, 5169, 2901, 1137, 273, 30, 1)	
	$\left\|A_{s,ex}^t\right\|$	(10, 91, 748, 1944, 6411, 13,722, 18,105, 22,812, 28,368, **30,396**, 28,920, 28,407, 27,844, 23,570, 14,498, 6266, 1658, 200, 30)	61,488
12	$\left\|V_{s,ex}^t\right\|$	(1, 12, 133, 804, 2024, 6193, 11,187, 16,555, 23,342, 31,086, 33,968, 34,903, 36,388, 36,443, 35,046, 32,429, 30,131, 24,126, 14,424, 6504, 2049, 399, 36, 1)	
	$\left\|A_{s,ex}^t\right\|$	(12, 133, 1354, 3619, 14,663, 38,962, 60,181, 84,381, 12,5851, 155,705, 161,260, **167,475**, 172,095, 168,850, 156,794, 149,898, 128,196, 82,920, 40,680, 13,950, 2950, 288, 36)	164,604

Table 2
Comparison of network sizes for the mono- and the bidirectional case; $L = 1$ and $U = 3$.

Monodirectional			Bidirectional		$2\lvert A(V^t_{\leqslant \bar{n},ex})\rvert$		Bidir. "First Half"		$\lvert A(V^t_{\leqslant \bar{n},ex})\rvert$	
n	$\lvert V^t_{ex}\rvert$	$\lvert A^t_{ex}\rvert$	$2\lvert V^t_{\leqslant \bar{n},ex}\rvert$		$+\lvert A^t_{ex,mid}\rvert$		$\lvert V^t_{\leqslant \bar{n},ex}\rvert$		$+\lvert A^t_{ex,mid}\rvert$	
4	131	228	92	−29.8%	198	−13.2%	46	−64.9%	147	−35,5%
6	1344	3366	1138	−15.3%	3596	+6.8%	569	−57.7%	2538	−24.6%
8	10,111	32,577	8548	−15.5%	32,108	−1.4%	4274	−57.7%	20,590	−36.8%
10	65,291	254,000	55,448	−15.1%	245,910	−3.2%	27,724	−57.5%	153,749	−39.5%
12	378,184	1,730,253	320,416	−15.3%	1,621,450	−6.3%	160,208	−57.6%	810,725	−53.1%

arcs and connections in the bidirectional case is twice that of the arcs from the first half plus the connections $|A^t_{ex,mid}|$. For $n = 4$ team, e.g., this is $2 \cdot (4 + 13 + 34) + 96 = 198$. However, the number of connections $|A^t_{ex,mid}|$ is for $n < 12$ larger than the number of arcs $|A_{\bar{n}+1,ex}|$. As a result, there is no clear picture when comparing the number of arcs and connections in the mono- and the bidirectional case, indicated also by the positive and negative relative changes between -13.2% and +6.8%. The number of operations in monodirectional labeling is proportional to $|A^t_{ex}|$, while for bidirectional labeling proportional to $2\left|A\left(V^t_{\leqslant \bar{n},ex}\right)\right| + \left|A^t_{ex,mid}\right|$. Thus, except for $n = 6$, the number of labeling operations is smaller in the bidirectional case.

The last four columns of Table 2 display how large the first half of the network is. This is particularly interesting because it relates to the amount of computer memory necessary to store the expanded networks. Here, the bidirectional approach saves between approx. 25% and 53% of main memory for n between 4 and 12. Since the expanded networks are rather large, the resulting reduction for the use of memory is another important detail for the viability of the approach.

4.1.2.2. Comparison with complete enumeration. A comparison of the proposed shortest-path pricing procedure with the constraint programming (CP) approach used by Easton et al. (2003) is hard. We are not able to precisely assess the impact of their (or any other) domain-reduction algorithms performed by a CP solver. Thus, we compare with the worst case, i.e., complete enumeration of all tours, which only occurs if the CP domain-reduction completely fails.

For a comparison with the worst case, we now determine the number $|P^t|$ of all possible tours of a team. Let $p = (i_1, i_2, \ldots, i_{2\bar{n}})$ be a tour. The associated *home-away pattern (HAP)* $h = (h_1, h_2, \ldots, h_{2\bar{n}})$ is defined by $h_i \in \{H, A\}$, $h_t = A$ for $i_t \in T_{-t}$ and $h_t = H$ for $i_t = t$. Clearly, there are $\bar{n}!$ tours that share the same home-away pattern. Hence, $|P^t| = \bar{n}! \cdot |H^t|$, where H^t denotes the set of all feasible HAPs.

What remains to do is to determine the cardinality of H^t, using the following recursively defined numbers: Let $\left]\begin{matrix} m \\ \ell \end{matrix}\right[$ be the number of ℓ-tuples with entries in $\{L, L+1, \ldots, U\}$ that sum up to m, i.e.,

$$\left]\begin{matrix} m \\ \ell \end{matrix}\right[= \left|\left\{(n_1, \ldots, n_\ell) \in \mathbb{N} : L \leqslant n_t \leqslant U, m = \sum_{t=1}^{\ell} n_t\right\}\right|.$$

Then,

$$\left]\begin{matrix} m \\ 1 \end{matrix}\right[= \begin{cases} 1, & \text{if } m \in \{L, \ldots, U\} \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad \left]\begin{matrix} m \\ \ell \end{matrix}\right[= \sum_{j=L}^{U} \left]\begin{matrix} m-j \\ \ell-1 \end{matrix}\right[, \text{for } \ell \geqslant 2.$$

For small values of ℓ and m, the values $\left]\begin{matrix} m \\ \ell \end{matrix}\right[$ with $L = 1$ and $U = 3$ are given in Table 3. For example, there are $\left]\begin{matrix} 4 \\ 2 \end{matrix}\right[= 3$ ways to decompose $m = 4$ into $\ell = 2$ summands between 1 and 3, namely $4 = 1 + 3 = 2 + 2 = 3 + 1$. Obviously, each HAP decomposes into subsequences of home games and away games, where each subsequence has $n \in \{L, \ldots, U\}$ identical elements. All home subsequences together are of length $\bar{n}$, as holds for away subsequences. Consequently, there are

$$|H^t| = 2 \cdot \sum_{\ell=1}^{\bar{n}} \left(\left]\begin{matrix} \bar{n} \\ \ell \end{matrix}\right[\cdot \left]\begin{matrix} \bar{n} \\ \ell \end{matrix}\right[+ \left]\begin{matrix} \bar{n} \\ \ell \end{matrix}\right[\cdot \left]\begin{matrix} \bar{n} \\ \ell - 1 \end{matrix}\right[\right) \tag{19}$$

different HAPs for a team, where the factor 2 arises from the fact that HAPs either start with a home or an away subsequence. Moreover, the number of home subsequences must be identical to the number of away subsequences $\left(\text{term} \left]\begin{matrix} \bar{n} \\ \ell \end{matrix}\right[\right] \left]\begin{matrix} \bar{n} \\ \ell \end{matrix}\right[\Big)$ or differ by 1 $\left(\text{term} \left]\begin{matrix} \bar{n} \\ \ell \end{matrix}\right[\right] \left]\begin{matrix} \bar{n} \\ \ell - 1 \end{matrix}\right[\Big)$. By combining $|P^t| = \bar{n}! \cdot |H^t|$ and (19), we get

$$|P^t| = \bar{n}! \cdot 2 \cdot \sum_{\ell=1}^{\bar{n}} \left(\left]\begin{matrix} \bar{n} \\ \ell \end{matrix}\right[\cdot \left]\bar{n}\ell\right[+ \left]\begin{matrix} \bar{n} \\ \ell \end{matrix}\right[\cdot \left]\begin{matrix} \bar{n} \\ \ell - 1 \end{matrix}\right[\right). \tag{20}$$

The ratio of the number of tours to the number of arcs in the expanded network exactly describes what is gained by replacing a simple enumerative approach by the shortest-path based pricing. This ratio is known as the *combinatorial leverage* ([see]Glover and Punnen, 1994). Table 4 shows the numerical results. With factors greater than 300 for $n \geqslant 8$, the shortest-path pricing is superior to simple enumerative approaches.

4.1.3. Reduced cost based arc elimination

Another advantage of the subproblem's network structure is that exact arc eliminations procedures become applicable. First, an upper bound ub to the TTP instance under consideration is needed. Metaheuristics, as surveyed in Rasmussen and Trick (2008), can provide such a bound. Following the ideas presented in Irnich et al. (2007), for a given dual feasible solution and an associated lower bound lb, an arc is redundant and can be eliminated if the reduced cost of any path using this particular arc exceeds the optimality gap $ub - lb$. This idea can be utilized in the TTP case either for the original networks $\mathcal{N}^t$ or the expanded networks $\mathcal{N}^t_{ex}$ of all teams $t \in T$. Since the original network is an aggregation of the expanded network, the finer level of detail in the expanded network allows the elimination of more arcs.

Moreover, the implementation of the method presented in (Irnich et al., 2007) is straightforward. After solving the root node

Table 3
Values $\left]\begin{matrix} m \\ \ell \end{matrix}\right[\neq 0$; number of ways to split m into ℓ values between $L = 1$ and $U = 3$.

$\left]\begin{matrix} m \\ \ell \end{matrix}\right[$	$m = 1$	2	3	4	5	6	7
$\ell = 1$	1	1	1				
2		1	2	3	2	1	
3			1	3	6	7	6
4				1	4	10	16
5					1	5	15
6						1	6
7							1

Table 4
Number of HAPs, tours, and columns in the master program for $L = 1$ and $U = 3$; combinatorial leverage comparing shortest-path pricing and tour enumeration.

n	Nb. HAPs $\lvert H^t\rvert$	Nb. Tours $\lvert P^t\rvert = \bar{n}! \cdot \lvert H^t\rvert$	Nb. Cols $n \cdot \lvert P^t\rvert = n! \cdot \lvert H^t\rvert$	Arcs $\lvert A^t_{ex}\rvert$	Comb. leverage
4	20	120	480	228	≈ 0.52
6	194	23,280	139,680	3366	≈ 6.9
8	1972	9,938,880	79,511,040	32,577	≈ 305
10	20,498	7,438,314,240	74,383,142,400	254,000	$\approx 29,285$
12	216,352	8,636,079,513,600	103,632,954,163,200	1,730,253	$\approx 4,991,224$

in branch-and-price, a feasible dual solution is known. Bidirectional labeling *without* bounding in the middle creates two minimum reduced-cost labels at each state associated to $\mathcal{N}^t_{ex}$. One label is for paths starting at the source node (forward label) and the other label is for paths ending at the sink node (backward label). For two states (i,s,b,N) and $(j,s+1,b',N')$, the first having forward reduced cost $\tilde{c}^{fw}$ and the second backward reduced cost $\tilde{c}^{bw}$, the term $\tilde{c}^{fw} + \tilde{c}^t_{ijs} + \tilde{c}^{bw}$ is the reduced cost of the associated path. In fact, due to Bellman's optimality principle, this reduced cost is the minimum reduced cost of all paths using this particular arc connecting the two states.

Therefore, forward and backward labeling needs to be performed only once in order to compute $\tilde{c}^{fw} + \tilde{c}^t_{ijs} + \tilde{c}^{bw}$ for all arcs between states. Whenever the term exceeds the optimality gap, the arc can be eliminated. Section 5.2 quantifies what can be gained from this exact arc elimination procedure.

4.2. Branching

The branching in branch-and-price algorithms includes several aspects. We start by discussing several possible branching decisions. Crucial for the size of the branch-and-bound tree and, therefore, for the overall efficacy, is the selection of a branching variable. We describe new ideas for strong branching in the second subsection. Finally, we exploit the inherent symmetry in TTP instances from the literature and derive a priori branching rules to cut away symmetric branches.

4.2.1. Branching decisions

For branch-and-price algorithms, Vanderbeck (2000), Lübbecke and Desrosiers (2005) have pointed out that, if a compact formulation is known, branching can always be performed on the integer variables of the compact formulation. In our case, branching on the binary variables x^t_{ijs} can be done straightforwardly: The branch with $x^t_{ijs} = 0$ requires the removal of the corresponding arc $(i,j,s) \in A^t$ from the network $\mathcal{N}^t$ of team t. Additionally, tour variables $\lambda^t_p, p \in P^t$ are fixed to zero (or removed from the master program) if $(i,j,s) \in p$ holds. The alternative branch $x^t_{ijs} = 1$ can be implemented as $x^t_{k\ell s} = 0$ of all other arcs $(k\ell s) \in A^t_s \setminus \{(i,j,s)\}$, i.e., arcs in $\mathcal{N}^t$ that correspond to slot s not identical to (i,j,s). Moreover, incompatible tours $p' \in P^{t'}$ of other teams $t' \in T_{-t}$ have to be fixed to zero, too. A tour $p' = (i'_0, i'_1, \ldots, i'_s, \ldots, i'_{2\bar{n}+1})$ is incompatible with $x^t_{ijs} = 1$ if $t' = i$ (or $t' = j$) and $i'_s \neq i$ (or $i'_{s+1} \neq j$). This branching on arcs guarantees integrality, but has the disadvantage that branches are typically highly unbalanced: The removal of a single arc is rather weak, while the fixing is stronger.

Therefore, alternative branching rules, i.e., branching on aggregated decisions, are worth analyzing. Easton et al. (2003) suggest branching on the home-away assignment (HAA) of teams-slot pairs. For a given team t and time slot s, one branch is $\sum_{i \in T_{-t}, j \in T:(i,j,s) \in A^t} x^t_{ijs} = 0$ and the other branch is $\sum_{j \in T:(tjs) \in A^t} x^t_{tjs} = 0$, meaning that t is playing either home or away in slot s. Obviously, this branching rule can be implemented as the removal of several x^t_{ijs} variables, i.e., with the techniques described above. Solely branching on HAAs does not guarantee integer solutions. For TTP instances with $n \geqslant 8$, we have found several examples where all HAAs were binary, but tour variables were still fractional (see also Briskorn and Drexl, 2009). As a consequence, HAAs have to be supplemented by other rules in order to guarantee integrality in all cases.

Therefore, we analyze other rules in order to find rules that are balanced, strong (improving the lower bound substantially), and complete (always producing integer solutions):

Branching on nodes/games: Branching on node v_{is} (for $i \in T, s \in S$) of $\mathcal{N}^t = (V^t, A^t)$ assures visiting this node or leaving it out. For an opponent $i \in T_{-t}$, the decision refers to the game "t home against t'", which is either fixed or excluded. By removing ingoing and outgoing arcs from this node v_{is} or from all other nodes in slot s, both branches $\sum_k x^t_{ki,s-1} = \sum_j x^t_{ijs} = 0$ or $= 1$ are simple to implement. Branching on nodes is complete because the fixing of one node for each slot fully determines the tour.

Branching on rounds of games: The game "team t plays home against team t'" can take place in the first round (slots $s \leqslant \bar{n}$) or in the second ($s > \bar{n}$). The removal of nodes $v_{t's}$ from the first or second half of the network $\mathcal{N}^t$ establishes this rule. As branching on HAAs, this rule is not complete.

Branching on number of home games in first round: The number of home games in the first round, i.e., $\sum_{s \leqslant \bar{n}} \sum_{j \in T} x^t_{tjs}$, must be integer. Branching on this information is simple if the network $\mathcal{N}^t$ is expanded (see Section 4.1.1). Then, the states $(i,\bar{n},b,N)$ of slot $\bar{n}$ represents tours with exactly $\bar{n} - |N|$ home games. The removal of those states and tour variables with too few or too many home games implements this branching rule.

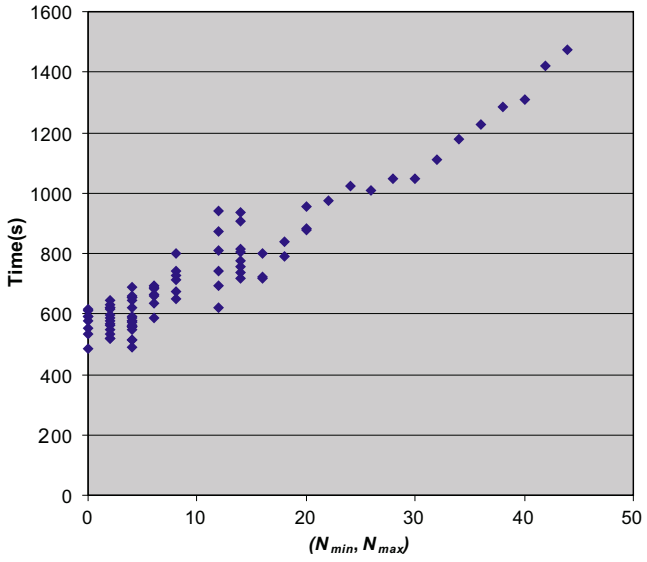

Fig. 3. Computation times for NL6 with NRCs for different parameter combinations $(N_{\min}, N_{\max})$ relative to the deviation $\Delta(N_{\min}, N_{\max})$ from $(N^*_{\min}, N^*_{\max}) = (6,20)$.

4.2.2. Strong branching

A careful selection of branching rules and variables is highly important in order to keep the number of branch-and-bound nodes and the overall computation time small. Recently, Achterberg et al. (2005) analyzed the impact of several branching rules on the performance of MIP-solvers for standard IP and MIP benchmark problems. We refer to this paper (Achterberg et al., 2005) for the associated terminology and notation.

One strategy for minimizing the number of branch-and-bound nodes is known as "strong branching". In *full strong branching*, all available candidate variables for branching are evaluated. The evaluation is performed as follows: Let lb denote the lower bound provided by the LP-relaxation of a given branch-and-bound node, and let lb_ℓ^+ and lb_ℓ^- be the (estimates of) lower bounds when branching is performed on the ℓth candidate branching variable (for ease of simplicity, we assume rules creating two branches). The quality of branching is assessed using a score function of the form

$$\sigma(\Delta_\ell^+, \Delta_\ell^-) = (1-\nu)\min\{\Delta_\ell^+, \Delta_\ell^-\} + \nu \max\{\Delta_\ell^+, \Delta_\ell^-\}, \tag{21}$$

where $\Delta_\ell^+ = lb_\ell^+ - lb$ and $\Delta_\ell^- = lb_\ell^- - lb$ is the change of lower bounds resulting from branching, and $\nu \in [0,1]$ a parameter. Thus, full strong branching chooses the ℓth variable from the full candidate set with $\sigma(\Delta_\ell^+, \Delta_\ell^-) = \max_i \sigma(\Delta_i^+, \Delta_i^-)$.

It is known that full strong branching typically leads to a small number of branch-and-bound nodes, but, on the downside, is rather time consuming. In order to circumvent this, *strong branching* uses a proper subset of candidate variables. In our implementation for the TTP, a *most infeasible* (=most fractional) selection rule is used to determine the candidates, i.e., variables x_ℓ enter the candidate list with decreasing values $\min\{x_\ell - \lfloor x_\ell \rfloor, \lceil x_\ell \rceil - x_\ell\}$. The novelty of our strong branching strategy is the sizing of the candidate set according to the relative remaining optimality gap: The idea is, similar to several more complex rules tested in (Achterberg et al., 2005), that one has to be selective close to the root node of the branch-and-bound tree. Thus, a larger candidate set is taken into account there. In contrast, branch-and-bound nodes with lb_ℓ close to the upper bound ub typically have a small subtree so that only a less careful selection of a branching variable is needed. Our sizing rule uses two parameters, $N_{\min}$ and $N_{\max}$, for the minimum and maximum number of candidates. We determine the number of candidate variables to evaluate as

$$N(lb_\ell) = \left\lfloor \frac{1}{2} + \left(\frac{ub - lb_\ell}{ub - lb_{root}}\right)^{1.5} N_{\max} + \left(1 - \left(\frac{ub - lb_\ell}{ub - lb_{root}}\right)^{1.5}\right) N_{\min} \right\rfloor. \tag{22}$$

Obviously, for the root node, $N(lb_\ell) = N_{\max}$ holds, while for nodes close to bounding $N(lb_\ell) = N_{\min}$ holds. Because of the exponent 1.5, the number of candidates is biased towards the minimum number $N_{\min}$. For instance, in the middle of the branch-and-bound tree, where $(ub - lb_\ell)/(ub - lb_{root}) = 0.5$, approximately $0.65N_{\min} + 0.35N_{\max}$ candidates are evaluated.

For the determination of the lower bounds lb_ℓ^+ and lb_ℓ^-, we completely solve the two son nodes, i.e., column generation is performed until no negative reduced-cost columns exist. We also test uncomplete evaluation strategies for the son nodes such as a limit number of column-generation iterations or artificial termination when reduced costs are small. Our finding is that almost always the full evaluation pays off because of the higher accuracy of the bounds. Computational results on strong branching are presented in Section 5.1.

4.3. Symmetry reduction

For the TTP, we exploit the fact that instances from the literature have symmetric distances $d_{ij} = d_{ji}$ for all $i,j \in T$. Thus, every solution x with tours $p^t = \left(i_1^t, \ldots, i_{2\bar{n}}^t\right), t \in T$ has a corresponding reverse solution x_{rev} with tours $p_{rev}^t = \left(i_{2\bar{n}}^t, \ldots, i_1^t\right), t \in T$. Branching rules such as branching on arcs or nodes can obviously eliminate this symmetry. However, approximately half of the branch-and-bound nodes are redundant.

The best thing is, therefore, the elimination of exactly one solution from every pair of reverse solutions right from the beginning in the root node. This can be achieved easily if bidirectional labeling is used for pricing on the expanded network. First note that it suffices to exclude one from every two reverse solutions for a particular team, w.l.o.g. team $t = 1$. The pricing problem for team $t = 1$ is then modified in the following way: States at slot $\bar{n}$ are numbered with indices $\Gamma := \left\{1, 2, \ldots, \left|V_{\bar{n},ex}^1\right|\right\}$. Recall that these states represent both forward states at slot $\bar{n}$ and backward states at slot $\bar{n}+1$. Thus, the set $A_{ex,mid}^1$ can be represented as a subset of $\Gamma \times \Gamma$. Using only connections $(\gamma_1, \gamma_2) \in \Gamma \times \Gamma$ having $\gamma_1 \leqslant \gamma_2$, i.e., excluding connections with $\gamma_1 > \gamma_2$, exactly cuts off one tour out of every pair of tours for team 1.

5. Computational results

All problem-specific algorithms were coded in C++, compiled in release mode with MS-Visual C++ .NET 2003 version 7.1; all runs were performed on a standard PC (Intel x86 family 6 model 15 Stepping 11) with 2.66 GHz, 4GB main memory, on MS-Win XP using a single thread.

Michael Trick's website http://mat.tepper.cmu.edu/TOURN, see also (Easton et al., 2003), is the basic source where TTP benchmark problems are available. The test set consists of the following groups of instances: first, the national league (NL) instances ranging from $n = 4$ to 16 teams. The NL instances have a symmetric distance matrix. Second, the circular instances ranging from $n = 4$ to 20 teams, where the distance is defined by a shortest path on a circle with edge length 1. Thus, for teams $i,j \in T, i \geqslant j$ the distance is $d_{ij} = d_{ji} = \min\{i-j, n+j-i\}$. It is empirically proven that these instances are very hard to solve due to the inherent symmetry. We do not consider the larger TTP instances, e.g., those for the National Football League (NFL) because they are too large to be handled with the available exact methods ($n \geqslant 16$ teams). Constant distance instances can be solved with specialized solution approaches (see Rasmussen and Trick, 2008) and are not considered, neither.

Since the number of NRCs is cubic in n, it is computationally too costly to add all NRCs to the initial RMP. Instead, only violated NRCs are added dynamically as LP cuts whenever the violation in (12) exceeds 0.2. Moreover, in all experiments we start the branch-and-price algorithm with an initial upper bound ub that is set to the best known solution +1. By doing so, the node selection strategy (best first or depth first) does not have a significant impact on the computation time. Since proving optimality or improving lower bounds are the key issues for exact methods, it is reasonable to provide good feasible solution which are easily found using state-of-the-art metaheuristics ([see]Rasmussen and Trick, 2008).

5.1. Results on branching

Which branching rules are best suited when optimal solutions need to be computed? Section 4.2.1 has discussed several branching rules, and our findings can be summarized as follows: Whenever applicable, branching on HAAs is superior to all other branching rules. Since HAA branching is not complete, it must be supplemented with another branching rule: we branch on nodes/games only if all HAAs are already non-fractional.

In order to quantify the superiority of this rule, we compare the computation times and number of branch-and-bound nodes.

Table 5 shows the results for the benchmark problem NL6 allowing repeaters (similar results were obtained for other instances; details are left out for the sake of brevity). The first two columns describe the branching rule with a primary and secondary rule, where the latter is only used for incomplete primary rules and when all primary branching variables are integer. Branching on HAAs creates the lowest number of branch-and-bound nodes. The computation times are not strongly correlated to the number of nodes solved. Some rules, e.g., branching on the number of home games in the first round tend to make the RMP harder to solve: Approximately ten times more pricing iterations are needed and 14 times more columns are generated. It remains unclear to us which property of that branching rule is responsible for such a (bad) behavior.

We next show the impact of strong branching on the overall running time according to the strategy presented in Section 4.2.2. Recall that the three defining parameters are $v \in [0,1]$ for the score function (21), and $N_{\min}$ and $N_{\max}$ for the minimum and maximum number of candidate variables, see Formula (22). All of our experiments indicate that it is best to increase the lower bound in both son nodes simultaneously, i.e., v must be set to a small value. This is in accordance with the findings of Achterberg et al. (2005), thus, we use $v = 1/10$.

We next study parameter combinations $(N_{\min}, N_{\max})$ and apply branch-and-price to the instance NL6 with NRCs. The symmetry reduction and reduced-cost fixing techniques that are presented in the next subsection are turned on in order not to waste too much time. We tested all combinations $2 \leqslant N_{\min} \leqslant 20, 2 \leqslant N_{\max} \leqslant 50$ with $N_{\min} \leqslant N_{\max}$, where $N_{\min}$ is a multiple of 2 and $N_{\max}$ increased in steps of length 4.

All computation times (wall clock time) are between 487 and 1474 s (avg. 734 s). Compared to the approach without strong branching, taking 1494 s, all setups using strong branching are faster. The best parameter combination resulting in 487 s is obtained for $(N^*_{\min}, N^*_{\max}) = (6, 20)$. The computational results indicate that $N_{\min}$ should neither be smaller than 4 nor greater than 10, i.e., should not deviate too much from $N^*_{\min} = 6$. Moreover, $N_{\max}$ should not be set to values notably smaller than $N^*_{\max} = 20$, while exceeding 20 is typically not harmful. Therefore, we compare the computation times relative to the deviation of a parameter combination $(N_{\min}, N_{\max})$ from $(N^*_{\min}, N^*_{\max}) = (6, 20)$ by $\Delta(N_{\min}, N_{\max}) := |N^*_{\min} - N_{\min}| + (N^*_{\max} - N_{\max})^+ = |6 - N_{\min}| + (20 - N_{\max})^+$. The Fig. 3 depicts this dependence of the overall computation time from the value $\Delta(N_{\min}, N_{\max})$ for all parameter combinations tested.

Table 5
Effectiveness of branching rules for NL6 with repeaters allowed.

First branching rule	Secondary rule	Num B&B nodes	Time (s)
HAAs	Games/nodes	9207	135
Games/nodes	(none)	17,437	267
Rounds of games	Games/nodes	17,891	348
Home games in 1st round	Games/nodes	15,417	981

Similar comparisons for NL8 indicate that for $n = 8$ teams the parameter combination $(N_{\min}, N_{\max}) = (20, 120)$ works reasonably well. Summing up, even if the parameters are not tuned, strong branching almost always contributes with a speedup of at least factor 2.

5.2. Results on reduced-cost fixing and symmetric distances

For a careful analysis of the proposed acceleration techniques, the small set of available benchmark instances is insufficient. Therefore, we decided to generate additional instances similar to the NL instances of Easton et al. (2001). Note that the instances NLn with n between 4 and 14 were generated by taking the first n teams of the instance NL16. 20 new instances with $n = 6$ teams were generated by randomly selecting a subset of six teams. (One of those instances is the instance NL6.) By enforcing and relaxing NRCs, 40 different instances result.

Four different setups are compared, where reduced-cost variable elimination (*Rdc Fix*) and symmetry reduction (*Sym Red*) are either turned on or off, respectively. For all computations, strong branching was enabled with parameters $(N^*_{\min}, N^*_{\max}) = (6, 20)$.

Table 6 shows in each row the aggregation of the results for 20 instances: Computation times were limited to one hour (3600 s). For some of the instances with NRCs, branch-and-price is not able to prove optimality, which is indicated in column *Failed in 1 h*. The columns *Rdc Fix* and *Sym Red* indicate which of the four setups is analyzed. The reference setup is the one with both *Rdc Fix* and *Sym Red* turned off. For the computation time, the column *Time* shows the average, minimum, and maximum computation time over the 20 instances. The comparison with the reference setup considering average computation times is shown in column *% Time*: As expected, all acceleration techniques lower the computation times. Unexpectedly, but due to different branching decisions taken, the reference setup has only three unsolved while all other setups have four unsolved instances. *Sym Red* is more effective than *Rdc Fix* because it reduces the computation time to 65% (no NRCs) and 77% (with NRC) compared to only a small reduction to 88% and 97%, respectively. Interestingly, *Sym Red* favors *Rdc Fix*: The reason for this is that *Sym Red* slightly increases the lower bound at the root node. Hence, the integrality gap becomes smaller and *Rdc Fix* is able to eliminate more arcs. The same effect explains why the number of remaining arcs is smaller when *Rdc Fix* is supplemented with *Sym Red*. Both acceleration methods, *Sym Red* and *Rdc Fix* lead to smaller branch-and-bound trees. They seem to foster integrality of RMP solutions.

Finally we analyze the dependence of the percentage of eliminated arcs from the integrality gap. Fig. 4 depicts this dependency. Optimality gaps range from approximately 2% to 14% where instances with NRCs tend to have larger gaps. Inversely proportional is the percentage of eliminated arcs with a maximum of 66%.

5.3. Results for national league and circular instances

First we compare our branch-and-priced implementation with the one of Easton et al. (2003), where the subproblems are solved

Table 6
Symmetry reduction and reduced-cost arc elimination for 40 instances with 6 teams.

NRC	Rdc fix	Sym red	Time (s) avg/min/max	% Time avg.	Failed in 1 h	B&B nodes avg/min/max	(%) Arcs elim. avg/min/max
No = rep. allowed	No	No	274/32/543	100%	0	1553/166/2983	
	Yes	No	240/25/470	88%	0	1506/166/2725	31/5/66
	No	Yes	177/28/375	65%	0	1025/154/2128	
	Yes	Yes	162/22/366	59%	0	1029/146/2168	33/7/66
Yes = rep. forbidden	No	No	1768/260/3600	100%	3	6672/1211/14,358	
	Yes	No	1723/233/3600	97%	4	6646/1170/14,328	0/0/22
	No	Yes	1365/173/3600	77%	4	5249/830/14,373	
	Yes	Yes	1328/151/3600	57%	4	5224/793/14,152	22/26/3

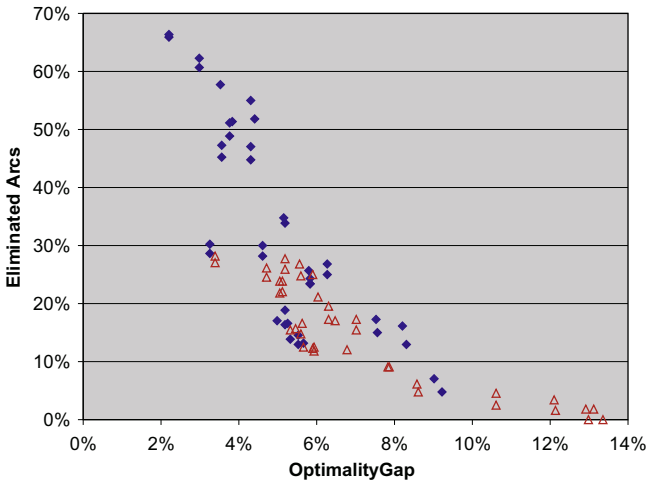

Fig. 4. Dependency of the percentage of eliminated arcs from the integrality gap; instances with $n = 6$ team without (♦) and with (△) NRCs.

with the help of a CP solver. In their approach, only instances without NRCs (allowing repeater games) are considered. Table 7 summarizes the results: A fair comparison of computing time is hardly possible, since Easton et al. (2003) use a parallel implementation on a cluster of 20 workstations (certainly producing some overhead so that parallel algorithms do not scale perfectly). However, computing times are significantly shorter and, by dynamically adding violated NRCs, we are now able to take this type of constraints into account.

The results of Table 7 also make clear that the computational effort quickly increases with the optimality gap that has to be closed via branch-and-bound. The optimality gap, i.e., $(ub - lb_{root})/lb_{root}$, increases when NRCs have to be respected. This results from the fact that we typically get the same lower bound lb_{root} at the root node of the branch-and-bound tree, but the optimum z_{ttp} differs for $n \geqslant 6$.

We briefly analyze the computation times of the pricing algorithm that also makes benefits and limitations of our approach clear: Table 8 shows the average time needed to solve a single pricing subproblem for the different NLn instances. The average is taken over all pricing problems solved in the branch-and-bound tree (for time-consuming instances with an overall time limit of one hour). For different numbers n of teams, the growth of the computation times fits in very well with the growth of the networks as discussed in Section 4.1.2. Only from $n = 6$ to $n = 8$, the time factor is around 2,7 and we suspect that the very small times for $n = 6$ and some computational overhead is responsible for this. In general, the results indicate that the computation times are reasonably small, even for $n = 12$. This suggests that also larger instances might be tractable. However, the reason why we cannot present results for $n \geqslant 14$ is that the expanded networks quickly grow so that they do not fit into the main memory of the computer.

Table 7
Comparison with results presented in Easton et al. (2003).

Instance	n	NRC	U	(Easton et al., 2003)		Now				
				Time (s)	Opt?	Time (s)	B&B nodes	lb_{root}	z_{ttp}	Opt. gap (%)
NL4	4	No	3	30	✓	0.035	5	8160	8276	1.4
NL4	4	Yes	3			0.035	5	8160	8276	1.4
NL6	6	No	3	900*	✓	61	512	22,582.6	23,552	4.3
NL6	6	Yes	3			509	2742	22,582.6	23,916	5.9
NL8	8	No	3	362,630*	✓	10,079	5097	38,670	39,479	2.1
NL8	8	Yes	3			43,321	14,890	38,670	39,721	2.7

* On a cluster of 20 workstations.

Table 8
Average computation times for pricing.

Instance	NRC	NL4	NL6	NL8	NL10	NL12
Avg. comp. time for pricing (s)	No	?*	0,00043	0.0011	0.0098	0.066
Growth by factor			?	2.6	8.9	6.7
Avg. comp. time for pricing (s)	Yes	*	0,00050	0.0014	0.0107	0.068
Growth by factor			?	2.8	7.6	6.4

* Time too small to be detected reliably.

Table 9
Results for the national league (NL) and circular instances.

Instance	n	NRC	U	Before		Now				
				lb	ub	lb	ub	Opt?	Time (s)	B&B nodes
NL8	8	Yes	3	39,479	39,721	**39,721**	39,721	✓	43,321	14,890
NL10	10	Yes	3	57,500	59,436	**58,000**			807,551	32,927
NL12	12	Yes	3	107,548	110,729	**107,600**			80,505	7135
Circ6	6	No	∞	?	54	**54**	54	✓	1358	11,668
Circ6	6	Yes	∞	?	56	**56**	56	✓	7869	41,368
Circ6	6	Yes	3	?	64	**64**	64	✓	10,789	37,109
Circ8	8	No	∞	?	100	**92**	100		398,146	74,990
Circ8	8	Yes	∞	?	102	**95**	102		914,226	552,604
Circ8	8	Yes	3	128	132	**130**	132		300,087	260,194

Note that the reduced-cost fixing requires the storing of one network per team.

Table 9 shows the new results obtained with branch-and-price for several (previously) unsolved instances. NL8 and all variants of Circ6 are solved to proven optimality for the first time. While NL8 took approximately 12 h, all variants of Circ6 were solved in less than 3 h, those without NRCs in less than 25 min.

Moreover, with the branch-and-price approach we were able to improve several of the best known lower bounds. Even though the increase of the newly computed lower bounds was significant for some instances, even multiplying the computational effort by factor 10 will probably help to close the remaining optimality gap. For those instances, additional refinements of the branch-and-price approach (e.g., using additional cutting planes) or some completely different solution approaches might be successful in the future.

6. Conclusions

In this paper we proposed a new branch-and-price-based solution approach for the practically hard-to-solve TTP sports league scheduling problem. The novelty of the algorithm is in the exploitation of the network structure introduced by a new original formulation, from which an extensive (=column-generation) formulation was derived. Pricing can be accelerated by more than one order of magnitude compared to the CP-based approach of Easton et al. (2003). The speedup first and foremost results from the reformulation of the pricing problem as an ordinary shortest-path problem over an expanded network. Additional exact network reduction and branching techniques accelerated branch-and-price even further.

According to Rasmussen and Trick (2008), "the most obvious challenge within the area and a great milestone to reach would be to prove optimality of the TTP instance NL8". NL8 is solved now, thus, the presented branch-and-price algorithm can be seen as successful. However, at the time of finishing this paper, Uthus et al. (2009) proposed a completely different approach to solve the TTP based on DFS* (depth first search with a particular upper bounding scheme). This algorithm seems to be notably faster than our approach, at least for instances with $n < 10$ teams.

Nevertheless, the new branch-and-price algorithm is useful: First, the original and extensive formulations can be used to develop alternative IP-based approaches. For example, we experimented with the addition of cutting planes (clique cuts and odd hole cuts) based on the notion of incompatible nodes and arcs of the TTP networks. Although lower bounds were increased, these experiments were not successful at the end because the additional cuts led to denser constraint matrices of the RMPs and slowed down the LP re-optimization. Second, some of the techniques proposed in this paper can and were used in the DFS* algorithm: Uthus et al. (2009) have implemented our symmetry reduction technique in order to reduce the DFS* running time. In addition, we are convinced that variable elimination techniques will further speed up DFS*. This is indispensable for solving instances with $n \geqslant 10$, since DFS* may easily take 100 h or more.

Finally, all known exact approaches are far away from being applicable to real-world sports scheduling instances. It is still an open question as to which solution paradigm, IP-based or CP-based or any combination, is best suited to tackle problems with a larger number of teams and with many types of soft and hard constraints that need to be taken into consideration.

Acknowledgment

Ulrich Schrempp implemented most parts of the network-specific routines of the branch-and-price algorithm for his Diplom/Master thesis in computer science at RWTH Aachen University. I would like to thank him for many fruitful discussions and for his support.

References

Achterberg, T., Koch, T., Martin, A., 2005. Branching rules revisited. Operations Research Letters 33 (1), 42–54.

Briskorn, D., Drexl, A., 2009. A branching scheme for finding cost-minimal round robin tournaments. European Journal of Operational Research 197 (1), 68–76.

Cheung, K., 2008. Solving mirrored traveling tournament problem benchmark instances with eight teams. Discrete Optimization 5 (1), 138–143.

Cheung, K., 2009. A Benders approach for computing lower bounds for the mirrored traveling tournament problem. Discrete Optimization 6 (2), 189–196.

Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M., Soumis, F., Villeneuve, D., 1998. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In: Crainic, T., Laporte, G. (Eds.), Fleet Management and Logistics, Chapter 3. Kluwer Academic Publisher, Boston, Dordrecht, London, pp. 57–93.

Easton, K., Nemhauser, G., Trick, M., 2001. The traveling tournament problem description and benchmarks. In: Walsh, T. (Ed.), Principles and Practice of Constraint Programming – CP 2001, Lecture Notes in Computer Science, vol. 2239. Springer Verlag, Berlin/Heidelberg, pp. 580–585.

Easton, K., Nemhauser, G., Trick, M., 2003. Solving the travelling tournament problem: a combined interger programming and constraint programming approach. In: Burke, E., De Causmaecker, P. (Eds.), Practice and Theory of Automated Timetabling IV, Lecture Notes in Computer Science, vol. 2740. Springer Verlag, Berlin/Heidelberg, pp. 100–109.

Glover, F., Punnen, A., 1994. The Traveling Salesman Problem: Linear Time Heuristics with Exponential Combinatorial Leverage. Technical Report, US West Chair in Systems Science, University of Colorado, Boulder, School of Business, Campus Box 419, Boulder, CO, 80309.

Irnich, S., 2008. Resource extension functions: properties, inversion, and generalization to segments. OR Spectrum 30 (1), 113–148.

Irnich, S., Desaulniers, G., 2005. Shortest path problems with resource constraints. In: Desaulniers, G., Desrosiers, J., Solomon, M. (Eds.), Column Generation, Chapter 2. Springer, New York, NY, pp. 33–65.

Irnich, S., Desaulniers, G., Desrosiers, J., Hadjar, A., 2007. Path-reduced costs for eliminating arcs in routing and scheduling. INFORMS Journal on Computing. doi:10.1287/ijoc.1090.0341.

Lübbecke, M., Desrosiers, J., 2005. Selected topics in column generation. Operations Research 53 (6), 1007–1023.

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

Righini, G., Salani, M., 2006. Bounded bidirectional dynamic programming for the elementary shortest path problem with resource constraints. Discrete Optimization 3 (3), 255–273.

Spoorendonk, S., 2008. Cut and Column Generation. Ph.D. Thesis, DIKU, University of Copenhagen, Denmark.

Urrutia, S., Ribeiro, C.C., 2004. Minimizing travels by maximizing breaks in round robin tournament schedules. Electronic Notes in Discrete Mathematics 18, 227–233. Latin-American Conference on Combinatorics, Graphs and Applications.

Urrutia, S., Ribeiro, C., Melo, R., 2007. A new lower bound to the traveling tournament problem. In: Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007), Honolulu, HI, pp. 15–18.

Uthus, D.C., Riddle, P.J., Guesgen, H.W., 2009. DFS* and the traveling tournament problem. In: van Hoeve, W.-J., Hooker, J.N., (Eds.), CPAIOR, LNCS 5547, Springer Verlag Berlin/Heidelberg, pp. 279–293.

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