CP AND MIP APPROACHES FOR SOCCER ANALYSIS

ROBINSON DUQUE *
AVISPA Research Group
Escuela de Ingeniería de Sistemas y Computación
Universidad del Valle, Colombia

ALEJANDRO ARBELAEZ
Riomh Research Group
Department of Computer Science
Cork Institute of Technology, Ireland

JUAN FRANCISCO DÍAZ
AVISPA Research Group
Escuela de Ingeniería de Sistemas y Computación
Universidad del Valle, Colombia

(Communicated by Jeroen Belien)

Abstract. Soccer is one of the most popular sports in the world with millions of fans that usually raise interesting questions when the competition is partially completed. One interesting question relates to the elimination problem which consists in checking at some stage of the competition if a team i still has a theoretical chance to finish first in a league or be within the first k teams in a tournament to qualify to the playoffs (e.g., become the champion if k=1). Some other interesting problems from the literature are the guaranteed qualification problem, the possible qualification problem, the score vector problem, promotion and relegation problem. These problems are NP-complete for the actual FIFA pointing rule system (0 points-loss, 1 point-tie, 3 points-win). SABIO is an online platform that helps users discover information related to soccer by letting them formulate questions in form of constraints and go beyond the classical soccer computational problems. In this paper we considerably improve the performance of an existing constraint programming (CP) model and combine the use of mixed integer programming (MIP) and CP to answer general soccer queries in a real-time application.

1. Introduction. Soccer or football is one of the most popular sports in the world with millions of fans. According to the newspapers, nearly a billion people worldwide tuned the Germany vs. Argentina football World Cup final in 2014. A soccer competition (league or tournament) consists of n teams playing against each other using a predefined format (e.g., double round-robin schedule). Depending on the

* Corresponding author: Robinson Duque.
results of the matches, every team is awarded some points under the FIFA three-point-rule (three points for a victory, one point for a draw, and zero points for a defeat).

*Tournament competitions* are usually played in two stages: a regular season played in a predefined format (e.g., single or double round-robin schedule) and a final knockout stage (aka playoffs) where typically eight teams qualify. On the other hand, *league competitions* consist of a single-stage where each team $i$ gets to play against team $j$ once or twice and at the end of the season, the first team in the standing table becomes the champion.

Common soccer computational problems (e.g., the elimination problem, possible qualification problem, the score vector problem, etc.) have been studied before using linear programming (see Section 2 for further information). SABIO (Soccer Analysis Based on Inference Outputs) is an online platform, available at [http://www.sabiofutbol.com](http://www.sabiofutbol.com), capable of answering soccer related queries by letting users formulate questions in form of constraints. To the best of our knowledge, SABIO is the first interactive platform that can be used to represent several soccer computational problems using a constraint programming approach (CP). It offers a general model that can be used to simulate different scenarios and problems where fans can combine four different kind of constrains to set up queries to analyze soccer:

- **Game Results.** These kind of queries let users create constraints about particular matches in any fixture (e.g., team A ends in a tie with team B).
- **Position in Ranking.** These kind of queries let users impose constraints about the positions of the teams at the end of a tournament (e.g., team A will be in a better position than 3).
- **Relative Position.** These kind of queries let users create constraints about the positions of two teams (e.g., team A will be in a worse position than team B).
- **Final Points.** These kind of queries let users impose constraints about the expected final points of the teams at the end of a tournament (e.g., team A scores 75 points at the end of the tournament).

At the broadest level, the former version of SABIO Web application is divided in two components, namely SABIO-Interface and SABIO-Core. The first component implements a graphical interface where users define their constraints in form of queries. SABIO-Interface also allows users to send their queries for processing through internet to the second component and also displays answers. SABIO-Core can be thought as a black box that implements the CP model described in Section 3. This black box mainly requires three files: the actual standing table of the tournament, the tournament schedule (i.e., games to play), the set of user-defined queries. The first two files are predefined in the SABIO-Core component whilst the third file is specified by the user in the SABIO-Interface component.

Based on our experience, SABIO typically gets a big load of queries in short periods of time, particularly after every fixture when a tournament is getting to an end. For instance, if 10% of the attendees to an stadium use the platform right after a match is finished (not to mention TV viewers), then we can get from 3000 to 5000 queries in short periods of time. Additionally, the load of queries is also incremented since a fixture usually has several matches that are played at the same time. Indeed, all the computational models proposed in this paper are thought to be used in the online version of SABIO. Thus, our main challenge consists in improving the amount of solved instances while finding a way to reduce computational time to offer the service to several users.
Probably the most popular problem in sports competitions is the elimination problem which consists of determining at some stage of the competition, whether a given team still has chances of winning (i.e., finishing first in a league or being within the first $k$ teams in a tournament to qualify to the playoffs). However, if a team is already eliminated, some fans might also be interested to know whether their teams have a chance to finish in another particular position like second or third place, since these positions can bring some benefits. For instance, the top teams (first, second and third) at the BBVA league qualify to the UEFA Champions League, so other interesting questions are "can team $i$ finish in position $k$?", or "can team $i$ finish in a better position than $k$?"

In 2005, Ribeiro et al. [37] proposed a mixed integer programming (MIP) formulation to tackle the guaranteed qualification problem to find the minimum number of points a given team has to win to become champion or qualify to the playoffs. However, this model was limited to single round-robin competitions. In 2006, Lucena et al. [27] extended such model to tackle single and double round robin competitions and proposed a multi-agent framework to collect, process, and publish information in the web related to the qualification and elimination problems in sports tournaments. In this paper, we use some ideas from [27, 37], and we present a MIP model with flexibility that allows us to answer more queries, i.e., game result queries, position in ranking queries, relative position queries, and final position queries. Our approach let users create different scenarios and go beyond the classical soccer computation problems.

In this paper we considerably improve our work on an existing CP model [11] [12] to improve the performance of SABIO. We propose three extensions of our CP model that include, redundant constraints, machine learning for variable/value selection, and a restart-based approach. We also explore the use of a series of mixed executions of MIP and CP models. Additionally, we extend our experiments and evaluate the performance of our models with state-of-the-art solvers: CPLEX, HaifaCSP (winner of the 2016 MiniZinc challenge), G12/FD, Gecode, and Oz. Moreover, we provide to the best of our knowledge, the first benchmark with real-life instances for soccer analysis. The benchmark comprises 18000 MiniZinc instances for two tournaments with 18 and 30 teams to evaluate the performance of the models.

The rest of the paper is structured as follows. Section 2 introduces a set of combinatorial problems in the context of sports competitions, including the elimination problem. Section 3 describes a basic CP model to tackle soccer computational problems. Section 4 extends the CP model and proposes dedicated heuristics to improve the overall performance. Section 5 extends an existing MIP model for soccer analysis to support single and double round robin competitions. Before the general conclusions in Section 7, Section 6 presents extensive experimental results.

2. Elimination problem complexity. At a particular stage of a soccer season, the elimination problem consists in "determining for a given team weather or not they are already eliminated, i.e., whether or not they can no longer become champions" [4]. A team $i$ is eliminated at some stage of a season if for all possible outcomes of the remaining games there is at least one team which has more points. The complexity of this problem depends on the way how scores are allocated according to the outcome of every match [24]. This elimination problem could be transformed into a network flow problem and would be polynomially solvable, if the old FIFA rule (0, 1, 2) was used. Nevertheless, Kern and Paulusma [24] [25] and Bernholt et
al. [4] independently proved that for the actual pointing rule (0, 1, 3) the problem is $NP$-complete.

2.1. The old FIFA scoring system. The study of the elimination problem goes back to the seminal work of Hoffman and Rivlin [22] in the 60’s for baseball competitions. Baseball competitions use the “1-point-rule” system, i.e., 1 point for a victory, 0 points for a defeat, and ties are not allowed. Interestingly in [42] Schwartz demonstrated that the elimination problem can be solved in polynomial time by modelling the problem using maximum network flows to determine whether a given team is eliminated.

Hoffman and Rivlin [22] generalized Schwartz work and gave a necessary and sufficient condition for not being eliminated from finishing in $k$-th place. Later on, McCormick [28] studied the highest final placing a team can hope for if the team is already eliminated from finishing first. That is, determining whether a team is eliminated from finishing the season in $k$-th place or better and concluded that this problem is $NP$-Complete for baseball. Adler et al. proposed in [1] an integer programming formulation to compute all the eliminated teams of a baseball tournament at certain stage. Wayne [46] proposed a structural property for the baseball elimination and ordered the teams according to their total number of possible wins and showed that if a team is eliminated, then so are all teams below it in the ordering. He showed that there is a threshold value $W^*$ for all teams $i$ with initial points $w_i$ and remaining games $g_i$, and it holds that team $i$ is eliminated if and only if $w_i + g_i < W^*$, i.e. $i$ can not win $W$ or more games. This approach brought faster algorithms since it allows to compute all eliminated teams simultaneously.

Bernholt et al. [4] pointed out that the old pointing system in soccer (two points to a win, one point to a tie and zero points to a loss), could be treated as a special instance of the baseball elimination problem where one game between teams $i$ and $j$ could be seen as two games between teams $i$ and $j$ under the 1-point-rule of baseball (without ties) and then the problem could be solved in polynomial time, using network flows as proposed by Wayne [46].

2.2. The new FIFA scoring system. Up to 1995 the traditional FIFA pointing rule consisted in assigning two points for a victory, one point for a tie, and zero points for a loss. After 1995, FIFA formally decided to increase from two to three the number of points assigned to the winning team and became standard in international tournaments, as well as most national soccer leagues, hoping that with an increase in the reward for victory, teams would become more offensive [19].

In 1999, Bernholt et al. [4] specified that for every game between $i$ and $j$ in a soccer/basketball/handball season, consisting of a set $\{1, 2, ..., N\}$ of teams, points are awarded according to some “$\alpha, \beta$-rule”. This notation means that the winner team gets $\alpha$ points and the losing team gets 0 points. The $\beta$ points are awarded for both teams if the game ends up in a tie.

Today, soccer is played under the (3,1)-rule or the “3-point-rule” and Bernholt et al. [4] showed that soccer elimination problem is hard to decide under this pointing system. They proved that the soccer elimination problem is $NP$-Complete if all teams have at most three remaining games. In the same article, they also proved that the problem can be solved in polynomial time if each team has at most two remaining games.

It is clear that soccer elimination under a (3,1)-rule is $NP$, since in [4] the authors showed that it is possible to guess the outcome of the remaining games and
compute the final ranking. To prove \( NP \)-Completeness, Bernholt et al. [4] managed to represent an instance of EFEP (European Football Elimination Problem) as an undirected multigraph and then showed that for each input formula to 3-SAT, it is possible to construct a labeled multigraph that is satisfiable only if the team for which you want to decide whether is eliminated or not, can still become champion.

As in baseball, Gusfield and Martel showed that there is a threshold value \( W^* \) for all teams \( i \) with initial points \( w_i \) and remaining games \( g_i \), and it holds that team \( i \) is eliminated if and only if \( w_i + g_i < W^* \), nevertheless, calculating such threshold for soccer might be \( NP \)-hard [20][4].

A more generalized study of complexity aspects of sports competitions (including soccer) was proposed by Kern and Paulusma [24][25], who used the triple \((\alpha, \beta, \gamma)\) to denote the points assigned to a team when participating in a match. The score is increase by \( \alpha \in \mathbb{R} \) if the team loses; by \( \beta \in \mathbb{R} \) if the game ends in a tie; and by \( \gamma \in \mathbb{R} \) if the team wins. They always assumed that \( \alpha \leq \beta \leq \gamma \). They determined the computational complexity of the sports elimination problem assuming that \((P \neq NP)\). Moreover, the authors proved that a game played under the \((\alpha, \beta, \gamma)\) rule is polynomially solvable for three cases:

1. \( \alpha = \beta \)
2. \( \beta = \gamma \)
3. \( \alpha + \gamma = 2\beta \)

in all other cases, like in soccer \((\alpha = 0, \beta = 1, \gamma = 3)\), Kern and Paulusma proved \( NP \)-Completeness by a reduction from three-dimensional matching to a particular multigraph model \( G = (V; E) \) that from a very general view, the vertices correspond to teams and edges are in 1-1 correspondence with remaining matches [24][25].

2.3. Related problems. Soccer competitions offer an excellent opportunity to model interesting problems (including the elimination problem). In this section, we briefly describe the most common computation problems related to soccer.

2.3.1. Promotion and relegation: Soccer fans are usually interested to check whether their teams can be either promoted or relegated at certain stage of a tournament. That is, the top \( l \) teams in ranking from a division of a league or tournament, are promoted to a higher division. Meanwhile, the last \( l \) teams in ranking from a higher division are relegated to a lower one. Kern and Paulusma [24] also proved \( NP \)-Completeness for questions like “is there a chance that team \( i \) ends up with the lowest final score?” or “is there a chance that team \( i \) ends up being one of the three teams that have the three lowest final scores?”, which might cause a direct relegation to a lower division. They used a version of the elimination problem to prove \( NP \)-Completeness for the rule \((\alpha = 0, \beta = 1, \gamma = 3)\) applied in soccer.

2.3.2. The elimination number: The elimination number problem relates to the minimum number of points that a team must get in order to have a chance of finishing in first place or to secure a place in the playoffs. Informally speaking, the problem for non-eliminated teams can be formulated as follows: What is the minimum number of points that a team \( i \) needs to make to at least tie for a first place or to guarantee a classification to the play-offs? Gusfield and Martel [20] showed that computing such number of points for soccer is \( NP \)-Hard and this problem is at least as hard as solving the subset sum problem [23].
2.3.3. **Guaranteed point placement**: Given a soccer tournament under the 3-point-rule system, the guaranteed point placement as defined in [8] asks if “it is the case that any assignment of outcomes to the remaining games will place team at position at least K in the final standing”. Christensen et al. [8] showed that the decision problem of whether or not a team is guaranteed a certain minimum position is coNP-Complete. They proved it by using a reduction from 3-SAT to a graph instance of the Guaranteed Point Placement problem.

2.3.4. **Guaranteed qualification problem**: This problem as defined in [37] consists of “calculating the minimum number of points any team has to win in order to be qualified, regardless of any other results”. This problem depends on the current number of points of every team in the standing table and on the missing games to be played. It was tackled by Riberio et al. [27, 37] using integer programming. They calculated a guaranteed qualification score (GQS) and their implementation can also be used to determine if a team is mathematically qualified to the playoffs. That is, if and only if its number of points is greater than or equal to its GQS. Later in 2014, Raack et al. [33] develop a general integer programming model for rankings to calculate the number of points needed to guarantee a team the ith place. The proposed model is very general and can be applied to different sports. They also studied a series of variations for problems related to the Bundesliga, Formula 1, and NHL. Interestingly, their model can be adjusted to reach different objectives. For instance, number of points needed to ensure certain achievement, such as avoiding relegation or qualifying for playoffs.

2.3.5. **Possible qualification problem**: This problem as defined in [37] consists in “computing how many points each team has to win to have any chance to be qualified”. Riberio et al. [27, 37] studied this problem using integer programming for which they calculated a possible qualification score (PQS). This score depends on the current number of points of every team in the standing table and on the missing games to be played. Analogously, to their Guaranteed Qualification Problem, this model can be used to check if a team is mathematically eliminated from the playoffs when the total number of possible points (i.e., the number of remaining games multiplied by three) plus the number of points that the team already has is less than its PQS.

2.3.6. **The score vector problem**: Kern and Paulusma [25] suspected that deciding if a score vector is reachable or not is a difficult problem. In 2009, Plvölgyi [32] denoted the problem of “deciding whether a given score vector is a possible result of a soccer-tournament or not” as the Score Vector Problem and proved that the problem is NP-complete by using a reduction from 3-SAT for the cases like the actual pointing rule ($\alpha = 0, \beta = 1, \gamma = 3$) applied in soccer.

2.3.7. **Soccer competition scheduling**: Many authors have tackled scheduling problems for different sports. For instance, [44, 48] for basketball, [10, 26] for hockey, [41, 49] for baseball, [35] for general round robin tournaments, etc.

Generating a schedule for a competition is a demanding task and has a major impact in the success of sports tournaments. A good schedule has many benefits and can make a competition more balanced, profitable, and attractive. Alarcón et al. [2] developed an integer programming approach that is currently used to schedule the Chilean soccer leagues. Their approach resulted in a positive economic impact, including reductions in television broadcaster operating costs, growth in
soccer pay-television subscriptions, increased ticket revenue, and lower travel costs for the teams. Interestingly, the authors also proposed the operations research methodology that is being used to schedule the South American qualification stage for the World Cup in Russia in 2018 at the moment of writing this article.

Interestingly, stakeholders’ wishes play an important role when creating a competition schedule. In [18] the authors proposed an approach to evaluate league formats in order to run a more attractive competition based on the importance of its games. The authors used simulation to generate the outcomes of the matches. They used historical match results of the Belgian league to estimate outcomes and evaluated four league formats. Additionally, the authors used optimization and proposed a MIP model to develop match schedules and to determine the importance of a match after each round.

In soccer, the scheduling problem has been studied for different tournaments and leagues. For instance, Noronha et al. [31] proposed an integer programming formulation and a branch-and-cut strategy to schedule a highly constrained Chilean soccer tournament. However, the approaches used in practice for Chile’s first and second division are respectively detailed in [13] and [15]. In both articles the authors define integer linear programming models that consider several operational, economic, geographical and sporting constraints.

Ribeiro and Urrutia [38] proposed a four-phase integer programming approach for scheduling the Brazilian soccer tournament under two main objectives, i.e., break minimization (fairness) and maximize the revenues from TV broadcasting. In 2012, they extended their previous work with a three-phase decomposition approach that satisfies a number of constraints demanded by the tournament organizers [39].

Related approaches include scheduling for the Belgian soccer league [17], a triple round robin tournament for the Danish soccer league [34], the professional soccer leagues of Austria and Germany [3], scheduling the South American Qualifiers to the 2018 FIFA World Cup [14], which is currently being used in the South American qualification tournament for the 2018 World Cup at the moment of writing this document.

3. Basic CP model for interactive soccer queries. Constraint programming (CP) is a powerful paradigm that can be used to solve combinatorial problems [40]. Typically, CP combines backtracking search with constraint propagation to filter inconsistent values and reduce the search space. In [11], we described a CP model for position in ranking queries combined with a machine learning classifier for value selection to perform a biased search for constraints that include the equality operator. We also managed to extend such model in [12] by introducing three new type of queries, i.e., game results, relative position, final points. Additionally, we also proposed a set of redundant constraints that help to prune the search tree for position ranking queries.

In this section, we describe our Basic CP formulation for soccer competitions. Namely, we differentiate between five categories of notations, variables, and constraints: basic formulation, game result queries, position in ranking queries, relative position queries and final point queries.

Basic Formulation: these variables capture basic information to formulate a general model for soccer competitions:

- $n$: number of teams in the competition;
- $T$: set of team indexes in the competition;
• $i, j$: team indexes, such that $(i, j \in T)$;
• $initialP ts_i$: initial points of team $i$. If $i$ has not played any games, then $initialP ts_i = 0$;
• $F$: number of fixtures left to be played in the competition. A fixture consists of one or more games between competitors$^1$;
• $k$: represents a fixture number, $(1 \leq k \leq F)$;
• $G$: set that represents the schedule of the remaining games to be played. Every game is represented as a triple $ng_e = (i, j, k)$ where $k$ is the fixture when both teams meet in a game and $1 \leq e \leq |G|$ if $|G| \geq 1$;
• $gameP ts_{ik}$: represents the points that team $i$ gets in fixture $k$, $(1 \leq k \leq F$ and $gameP ts_{ik} \in \{0, 1, 3\})$. If team $i$ is not scheduled to play fixture $k$, then $gameP ts_{i,k} = 0$.
• $totalP ts_i$: total points of team $i$ at end of the competition;
• $geq_{ij}$: Boolean variable indicating if team $j$ has greater or equal total points as $i$ at the end of the competition: if $totalP ts_j \geq totalP ts_i$ then $geq_{ij} = 1$; otherwise $geq_{ij} = 0$ $(\forall i, j \in T)$;
• $eq_{ij}$: boolean variable indicating if two different teams $i$ and $j$ tie in points at the end of the competition: if $totalP ts_j = totalP ts_i$ then $eq_{ij} = 1$; otherwise $eq_{ij} = 0$ $(\forall i, j \in T)$.
• $pos_i$: position of team $i$ at the end of the competition;
• $worstPos_i$: upper bound for $pos_i$;
• $bestPos_i$: lower bound for $pos_i$;

Constraints (1), (2), and (3) represent a valid game point assignment $(0,3), (3,0)$ or $(1,1)$ for each game $ng_e \in G$ between two teams $i$ and $j$ in a fixture $k$. Constraint (4) corresponds to the final points $totalP ts_i$ of a team $i$:

\[
(0 \leq gameP ts_{ik} \leq 3) \land (0 \leq gameP ts_{jk} \leq 3) \quad \forall ng_e \in G \land ng_e = (i, j, k) \quad (1)
\]

\[
(gameP ts_{ik} \neq 2) \land (gameP ts_{jk} \neq 2) \quad \forall ng_e \in G \land ng_e = (i, j, k) \quad (2)
\]

\[
2 \leq gameP ts_{ik} + gameP ts_{jk} \leq 3 \quad \forall ng_e \in G \land ng_e = (i, j, k) \quad (3)
\]

\[
totalP ts_i = initialP ts_i + \sum_{k=1}^{F} gameP ts_{ik} \quad \forall i \in T \quad (4)
\]

Constraints (5) to (8) are used to calculate final positions. All the final positions must be different and every position is bounded by $bestPos_i$ and $worstPos_i$:

\[
geq_{ij} = \begin{cases} 
1, & \text{if } totalP ts_j \geq totalP ts_i \\
0, & \text{otherwise}
\end{cases} \quad \forall i, j \in T \quad (5)
\]

\[
worstPos_i = \sum_{j=1}^{n} geq_{ij} \quad \forall i \in T
\]

$^1$We recall that this CP model is currently implemented in SABIO (using OZ) and the decision of using fixtures $K$ instead of a set of games, corresponds to a model that is closer to how our Web application manages to display the results of every match, when there’s a feasible solution to a soccer scenario with a set of constraints. Additionally, expressing the model either way, lead to the same number of variables.
Constraint (9) allows users to include assumptions about the outcome of remaining games to constrain the points of teams $i$ and $j$ in a fixture $k$, e.g., team A ends in a tie with team B.

- $Q$: set of game result queries for a pair of teams $(i, j)$ in a fixture $k$. Every query is defined as a tuple $nq_{a} = (ptc_{ik}, ptc_{jk})$ and $1 \leq a \leq |Q|$ if $|Q| \geq 1$;
- $ptc_{ik}$ and $ptc_{jk}$: are user suppositions about the points that a pair of teams $(i, j)$ will get in a fixture $k$, i.e., $(ptc_{ik}, ptc_{jk}) \in \{(0, 3), (3, 0), (1, 1)\}$;

\[
\text{gameP}ts_{ik} = ptc_{ik} \land \text{gameP}ts_{jk} = ptc_{jk} \quad \forall nq_{a} \in Q \land nq_{a} = (ptc_{ik}, ptc_{jk})
\]  

(9)

**Position in Ranking Queries:** we use this set of constraints to indicate whether a given team can be above, below, or at a given position $ptn_{i}$, e.g., team A will be in position 3. Constraint (10) depicts the five possibilities:

- $P$: set of possible position in ranking queries, defined as a set of triples $np_{b} = (i, opr_{i}, ptn_{i})$ and $1 \leq b \leq |P|$ if $|P| \geq 1$;
- $opr_{i}$: logical operator $(opr_{i} \in \{<, \leq, >, \geq, =\})$ to constrain team $i$;
- $ptn_{i}$: denoting the expected position for team $i$; $1 \leq ptn_{i} \leq n$;

\[
\forall np_{b} \in P \land np_{b} = (i, opr_{i}, ptn_{i}) \begin{cases} 
\text{pos}_{i} = ptn_{i}, & \text{if } opr_{i} \text{ is =} \\
\text{pos}_{i} < ptn_{i}, & \text{if } opr_{i} \text{ is <} \\
\text{pos}_{i} \leq ptn_{i}, & \text{if } opr_{i} \text{ is } \leq \\
\text{pos}_{i} > ptn_{i}, & \text{if } opr_{i} \text{ is >} \\
\text{pos}_{i} \geq ptn_{i}, & \text{if } opr_{i} \text{ is } \geq
\end{cases}
\]  

(10)

**Relative Position Queries:** these queries indicate whether a given team $i$ will be above, below, or equal to another team $j$ at the end of the tournament and constraint (11) depicts the five queries, e.g., team A will be in a better position than team B. In this particular case we use $totalPts_{i}$ and $totalPts_{j}$ instead of $pos_{i}$ and $pos_{j}$. We consider that two teams $i$ and $j$ might tie up in the same position if they have the same points at the end of the competition. We recall that we do not use $pos_{i}$ and $pos_{j}$ due to the $alldifferent$ constraint in (8) and also because the total points can prune the search tree before the positions are calculated.

- $R$: set of possible relative position queries defined as a set of triples $nr_{c} = (i, opr_{ij}, j)$ and $1 \leq c \leq |R|$ if $|R| \geq 1$;
- $opr_{ij}$: denoting a logical operator $(opr_{ij} \in \{<, \leq, >, \geq, =\})$ to constrain a pair of teams $i$ and $j$. 

\[
eq_{ij} = \begin{cases} 
1, & \text{if } totalPts_{j} = totalPts_{i} \\
0, & \text{otherwise}
\end{cases} \quad \forall i, j \in T
\]  

(6)

\[
bestPos_{i} = worstPos_{i} - \sum_{j=1, j\neq i}^{n} eq_{ij} \quad \forall i \in T \land i \neq j
\]  

(7)

\[
bestPos_{i} \leq pos_{i} \leq worstPos_{i} \quad \forall i \in T
\]  

(8)
1544 ROBINSON DUQUE, ALEJANDRO ARBELAEZ AND JUAN FRANCISCO DÍAZ

∀_{r \in R} \forall_{c \in R} \forall_{r \cdot c = (i, o_{ij}, j)}

\left\{ \begin{array}{ll}
totalPts_i = totalPts_j, & \text{if } o_{ij} \text{ is } = \\
totalPts_i < totalPts_j, & \text{if } o_{ij} \text{ is } < \\
totalPts_i \leq totalPts_j, & \text{if } o_{ij} \text{ is } \leq \\
totalPts_i > totalPts_j, & \text{if } o_{ij} \text{ is } > \\
totalPts_i \geq totalPts_j, & \text{if } o_{ij} \text{ is } \geq \\
\end{array} \right. \quad (11)

**Final Point Queries:** (also known as score queries) we use these variables for queries about the final points of the teams, e.g., team A scores 75 points at the end of the competition. Constraint (12) guarantees final point queries.

\begin{itemize}
  \item $S$: set of possible final point queries defined as a set of tuples $ns_d = (i, s_i)$ and $1 \leq d \leq |S|$ if $|S| \geq 1$;
  \item $s_i$: denoting the wanted final points of team $i$.
\end{itemize}

\[(totalPts_i = s_i) \forall ns_d \in S \wedge ns_d = (i, s_i)\] \quad (12)

4. Extended CP model.

4.1. Redundant constraints. In our previous basic CP model, the position bounds (i.e., bestPos$_i$ and worstPos$_i$) for position in ranking queries can only be computed after finding the total points ($totalPts_i$) for all the teams in the competition, then the position constraints are validated. This formulation leads to an exhaustive search with a late pruning rule based on the teams positions.

The redundant constraints proposed in this section make inferences about the teams positions based on the total points, in order to start pruning as early as possible while the search unfolds. To depict our approach, consider the following position in ranking constraint: “team A will be in the same position as 1” which can be represented as the triplet $(A, =, 1)$ or $pos_A = 1$ according to Constraint (10). In order to satisfy such constraint, it must hold that during the search, the number of teams with more points than A has to be 0, otherwise, A will never be in first position. To take this scenario into account we propose a set of redundant constraints that constantly validate the number of teams with more (resp. less) points than A. Therefore, let $L$ denote the set of constrained teams included in all the triples $nP_b \in P$, such that $nP_b = (i, opr_i, ptn_i)$ where $i \in L$ and $L \subseteq T$.

In order to extend the CP model described in Section 3 we introduce the two following variables for constrained teams $i \in L$:

- $less_{ij}$: Boolean variables denoting whether teams $j$ have less points than $i$, i.e., if $totalPts_j < totalPts_i$ then $less_{ij} = 1$; otherwise $less_{ij} = 0$ ($\forall j \in T \wedge \forall i \in L$);
- $grtr_{ij}$: Boolean variables denoting whether teams $j$ have more points than $i$, i.e., if $totalPts_j > totalPts_i$ then $grtr_{ij} = 1$; otherwise $grtr_{ij} = 0$ ($\forall j \in T \wedge \forall i \in L$).

“Greater than” redundant constraint, i.e., $nP_b = (i, >, ptn_i)$. During search, the number of teams with fewer points than team $i$ must be limited to $(n - ptn_i)$:

$$\sum_{j=1, j \neq i}^n less_{ij} < (n - ptn_i) \quad \forall i \in L$$ \quad (13)

Similarly, we use $\sum_{j=1, j \neq i}^n less_{ij} \leq (n - ptn_i)$ for constraints $nP_b = (i, \geq, ptn_i)$.
“Less than” redundant constraint, i.e., \( np_b = (i, <, ptn_i) \). During search, the number of teams with more points than team \( i \) must be limited to \((ptn_i - 1)\):

\[
\sum_{j=1, j\neq i}^{n} grtr_{ij} < (ptn_i - 1) \quad \forall i \in L
\]  \hspace{1cm} (14)

Similarly, we use \( \sum_{j=1, j\neq i}^{n} grtr_{ij} \leq (ptn_i - 1) \) for constraints \( np_b = (i, \leq, ptn_i) \).

“Equal to” redundant constraint, i.e., \( np_b = (i, =, ptn_i) \). During search, we constrain the number of teams above (resp. below) of a team \( i \) to:

\[
\sum_{j=1, j\neq i}^{n} grtr_{ij} < (ptn_i) \quad \forall i \in L
\]  \hspace{1cm} (15)

\[
\sum_{j=1, j\neq i}^{n} less_{ij} < (n - ptn_i + 1) \quad \forall i \in L
\]  \hspace{1cm} (16)

4.2. Variable/Value selection. Variable and value selection heuristics are critical to solve real-life problems. Classical value selection heuristics to select a value from the remaining domain of a given variable can be summarized as follows: min-value selects the minimum value; max-value selects the maximum value; mid-value selects the median value; and random-value selects a random value.

On the other hand, variable selection heuristics comprehend more sophisticated algorithms. lexico is one of the simplest heuristics for variable selection, selecting the first unassigned variable in the list of decision variables. random selects an unassigned variable with a uniform distribution. mindom \cite{21} is a well established CP heuristic based on the “First-Fail Principle: try first where you are more likely to fail”, this strategy chooses the variable with minimum size domain. mindom is usually used to complement more sophisticated heuristics such as dom-deg, which selects the variable that minimizes the ration \( \text{dom/degree} \), where \( \text{dom} \) and degree denote the size of the domain of a given variable and its respectively dynamic degree. wdeg \cite{6} gives more priority to variables frequently involved in failed constraints and \cite{36} proposed the impact dynamic variable-value selection heuristic to maximize the reduction of the remaining search space.

In this section we propose some heuristics for variable/value selection. First, we introduce a set of required variables to describe a priority mechanism to select the team variables constrained in queries \( P \):

- \( \text{spos}_i \): starting position of team \( i \) before any branching strategy is applied;
- \( \text{pri}_i \): denoting the priority of team \( i \) to be selected during branching. If team \( i \) does not appear in any query, then \( \text{pri}_i = 0 \);
- \( \text{str}_i \): denoting the global branching strategy for the variables \( \text{gamePts}_{ik} \) of a particular team \( i \) in every fixture \( k \). \( \text{str}_i \) starts with “tie” as a default value.

4.2.1. Heuristics for position in ranking queries (\( P \)). Recall from (10) that we use the position \( \text{pos}_i \) to constrain a team to a wanted position \( ptn_i \). Suppose we have the query \( (\text{pos}_i < 8) \). It is straightforward to try a winning policy (i.e., \( \text{str}_i = \text{win} \)) and assign a priority using the position \( ptn_i \) from the query. We depict the general rules for variable/value selection in formulation (17):
\[
\forall n_b \in P \ni \begin{cases}
    \text{if } \text{opr}_i < \text{or} \leq, & \text{pri}_i = n - ptn_i \land \text{str}_i = \text{win} \\
    \text{if } \text{opr}_i > \text{or} \geq, & \text{pri}_i = ptn \land \text{str}_i = \text{lose} \\
    \text{if } \text{opr}_i = =, & \text{pri}_i = ptn \land \begin{cases}
    \text{str}_i = \text{lose}, & \text{if } ptn_i > n/2 \\
    \text{str}_i = \text{win}, & \text{otherwise}
    \end{cases}
\end{cases}
\]

(17)

Empirically, we have observed that the priority mechanism for variable/value heuristics proposed formulation (17) for queries with the “=” operator seem to perform poorly (see Section 6). Intuitively, suppose a scenario with a query \((pos_i = 7)\) where \(spos_i = 9\) with \(F = 8\) fixtures to play. Given that the starting position is 9 and we have to reach position 7, the global branching strategy \(\text{str}_i = \text{win}\) causes that \(pos_i\) overshoots position 7 and would require many backtracks of the search algorithm in order to reach such position. Therefore, it might be useful to perform a bias search and in the following section we tackle this problem by using machine learning.

4.2.2. Machine learning for value selection. Supervised machine learning consists of inferring a model (or hypothesis) \(f : \Omega \to S\) that predicts the value of an output variable \(S_i \in S\), given the values of the example description (e.g., a vector of features, \(\Omega = \mathbb{R}^d\)). The output can be numerical (i.e., regression) or categorical (i.e., classification).

For teams constrained with the “=” operator, we decided to assign a high priority \((\text{pri}_i = |spos_i - ptn_i| \cdot n)\) for variable selection. To avoid position overshooting, we trained a machine learning model that selects among 9 branching strategies:
\[
S_1 = [1,0,0], \quad S_2 = [0,1,0], \quad S_3 = [0,0,1], \quad S_4 = [0.5,0.5,0], \quad S_5 = [0.5,0,0.5], \quad S_6 = [0,0.5,0.5], \\
S_7 = [0.5,0.25,0.25], \quad S_8 = [0.25,0.5,0.25], \quad S_9 = [0.25,0.25,0.5].
\]

Each strategy defines probabilities to select among \([\text{win, tie, lose}]\) respectively, e.g, \(S_7\) means that for a team \(i\), every variable \(\text{gamePts}_{i,k}\) will be assigned \(\text{win}\) with a probability of 0.5, \(\text{tie}\) and \(\text{lose}\) with a probability of 0.25 each. We use the selected strategy with a restart-based search; therefore we restart the algorithm when some cutoff in the execution time is met. (3 secs in this paper). Notice that that we excluded a strategy \([1/3, 1/3, 1/3]\) as preliminarily experiments showed a poor performance for this alternative.

**Feature vector:** in this paper, we use a vector of five features for queries with the “=” operator to identify the most suitable branching strategy.

- **Starting Position** (\(spos_i\)): This feature was selected because every team has an initial position depending on its initial points (\(\text{initialPts}_i\)) and also because there is a relation between the starting position and the wanted position in order to choose a heuristic for value selection.
- **Wanted Position** (\(ptn_i\)): This feature was selected because there is a relation between the starting position and the wanted position in the constraint.
- **Direction and Distance** (\(spos_i - ptn_i\)): This feature determines the direction where a team has to move in the standing table (i.e., up or down) and also represents the distance or the number of positions that the team has to move. This feature has the following characteristics:
  1. If the value of \((spos_i - ptn_i)\) is positive, it indicates that the team has to go up in the standing table and the greater the distance is, the more games the team has to win.
2. If the value of \((s_{pos} - p_{tn})\) is negative, it indicates that the team has to go down in the standing table and the smaller the distance is, the more games the team has to lose.

- **Fixtures to Play** \((F)\): This feature might determine how fast a team has to reach certain position. For instance, if a team has few fixtures to play, it has to reach the wanted position very quickly.

- **Range Rate** \(\left|s_{pos} - p_{tn}\right|/n\): This feature determines the rate of the distance from the starting position to the wanted position in relation to the number of teams.

**Machine Learning model:** in order to train a classification model, we created a total of 500 \(P\) queries with the equality operator at different stages of the Colombian tournament (fixture 7, 9, 11, 14 and 16) with 18 teams, scheduled in a single round robin.

During the training phase, we identified the best strategy and created a dataset with the vector of features and the best branching strategy \(S_i \in \{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9\}\). In particular, in this paper we use J48 (the Weka v3.6.12 implementation of C4.5 [47]) to evaluate the performance of the algorithms. Figure 1 shows a training file example, each line indicates the best branching strategy for the training instance, i.e., a query \(P\) with the equal operator “=”.

4.3. **Sequential and parallel restart-based search.** Inspired by the quickest first principle [5], we execute the strategies in a predefined order and we use a restart-based search with a fixed time cutoff. For teams involved in at least one
query we use the above mentioned variable/value selection heuristics in all restarts. For the remaining teams, we use a different strategy for each restart, starting with S1 for the first restart and using S9 for the ninth restart, in the tenth restart we use a random strategy for unconstrained teams. The last restart is executed until a solution is observed or a time limit is reached.

These restart strategies can be executed either sequentially or in parallel for a fixed cutoff time (i.e., one restart after another in a single core, or one restart per core in a multi-core machine). In the parallel version with four cores, we execute in parallel (for unconstrained teams) \( \{S_1, \ldots, S_4\} \) followed by \( \{S_5, \ldots, S_9\} \). We finish the execution with the random strategy for all cores.

5. **MIP model for soccer queries.** In the following we describe the variables and notations used in our model, we differentiate between five categories of variables: basic formulation, game result queries, position in ranking queries, relative position queries, and final point queries. While a few of these variables had already been used in the literature \[28, 46, 37, 27\], we recall that many variables had to be added to formulate certain features of our model, e.g., flexibility of single and double round-robin competition and flexibility to represent different soccer scenarios.

### Basic Formulation Variables

- **\( n \)**: number of teams in the competition;
- **\( T \)**: set of team indexes in the competition;
- **\( i, j \)**: team indexes, such that \((i, j) \in T\);
- **\( \text{initialPts}_i \)**: initial points of team \( i \). If \( i \) has not played any games, then \( \text{initialPts}_i = 0 \);
- **\( g_{ij} \)**: number of remaining games between a pair of teams \( i \) and \( j \);
- **\( w_{ij} \)**: number of games that team \( i \) wins over team \( j \);
- **\( t_{ij} \)**: number of games that team \( i \) ties with team \( j \);
- **\( l_{ij} \)**: number of games that team \( i \) loses over team \( j \);
- **\( \text{totalPts}_i \)**: total points of team \( i \) at end of the competition;
- **\( \text{pos}_i \)**: position of team \( i \) at the end of the competition;
- **\( \text{worstPos}_i \)**: upper bound for \( \text{pos}_i \);
- **\( \text{bestPos}_i \)**: lower bound for \( \text{pos}_i \);

Depending on the competition and the current state of the tournament, \( g_{ij} \) can be set to 0 (no games left between the teams), 1 or 2. We recall that \( g_{ij} \) must be equal to \( g_{ji} \), the games a team \( i \) wins over team \( j \) (i.e., \( w_{ij} \)) must be also equal to the games team \( j \) loses over team \( i \) (i.e., \( l_{ji} \)). Respectively, \( l_{ij} = w_{ji} \), and \( t_{ij} = t_{ji} \) to represent tied games \[27, 37\]. In this scenario, a team can get in every game 3 points - win, 1 point - tie, 0 points - loss. Thus, the total number of points of team \( i \) can be calculated by adding its initial points \( \text{initialPts}_i \) and the points obtained against every other team.

Unlike previous work in \[37, 27\] which is limited to tackle elimination and classification problems, we added flexibility (i.e., more variables) to let our model represent different scenarios. For instance, users can create constraints about the possible outcomes of particular games (i.e., decide which team wins, ties, or loses specific games). Constraints \( (18) \), \( (19) \), and \( (20) \) describe a basic soccer model with a valid (win, tie, lose) assignment for every game between a pair of teams \((i, j)\) with \( g_{ij} \).
games left to play.

\[ w_{ij} + t_{ij} + l_{ij} = g_{ij} \quad \forall i,j \in T \wedge g_{ij} \geq 0 \quad (18) \]

\[ w_{ij} = l_{ji} \wedge t_{ij} = t_{ji} \wedge l_{ij} = w_{ji} \quad \forall i,j \in T \quad (19) \]

\[ \text{totalPts}_i = \text{initialPts}_i + \sum_{j=1,j \neq i}^{n} 3 \cdot w_{ij} + t_{ij} \quad \forall i \in T \quad (20) \]

**Game Result Queries:** we use this set of variables to represent user defined assumptions about the remaining games, e.g., team A ends in a tie with team B. Taking this into account we extend our basic model with the linear equations in (21).

- \( Q \): set of game result queries for pairs of teams \((i, j)\). \( Q \) is defined as a set of triples \( nq_a = (w_{cij}, lc_{ij}, tc_{ij}) \) and \( 1 \leq a \leq |Q| \) if \(|Q| \geq 1 \);
- \( wc_{ij} \): minimum number of games that team \( i \) wins over team \( j \);
- \( tc_{ij} \): minimum number of games that team \( i \) ties with team \( j \);
- \( lc_{ij} \): minimum number of games that team \( i \) loses over team \( j \);

\[
(w_{cij} + tc_{ij} + lc_{ij} \leq g_{ij}) \quad \forall nq_a \in Q \wedge nq_a = (wc_{ij}, tc_{ij}, lc_{ij})
\]

\[
(w_{ij} \geq wc_{ij}) \wedge (t_{ij} \geq tc_{ij}) \wedge (l_{ij} \geq lc_{ij}) \quad \forall nq_a \in Q \wedge nq_a = (wc_{ij}, tc_{ij}, lc_{ij}) \quad (21)
\]

\[ wc_{ij}, tc_{ij}, lc_{ij} \geq 0 \]

**Position in Ranking Queries:** we use this set of variables to represent queries of teams at the end of the competition. A position in ranking query involves a set of constrained teams \( L \subseteq T \) and indicates whether a given team can be above, below, or at a given position \( ptn_i \). Constraint (22) depicts the five possibilities:

- \( P \): set of possible position in ranking queries, defined as a set of triples \( np_b = (i, opr_i, ptn_i) \) and \( 1 \leq b \leq |P| \) if \(|P| \geq 1 \);
- \( opr_i \): logical operator \( opr_i \in \{\lt, \leq, \gt, \geq, =\} \) to constrain team \( i \);
- \( ptn_i \): denoting the expected position for team \( i \); \( 1 \leq ptn_i \leq n \);
- \( L \): denoting the set of team indexes included in all the triples \( np_b \in P \) such that \( np_b = (i, opr_i, ptn_i) \) and \( i \in L \) and \( L \subseteq T \);
- \( geq_{ij} \): boolean variable indicating if team \( j \) has greater or equal total points as \( i \): if \( \text{totalPts}_j \geq \text{totalPts}_i \) then \( geq_{ij} = 1 \); otherwise \( geq_{ij} = 0 \) \( \forall i \in L, \forall j \in T \);
- \( eq_{ij} \): boolean variable indicating if two different teams \( i \) and \( j \) tie in points at the end of the competition: if \( \text{totalPts}_j = \text{totalPts}_i \) then \( eq_{ij} = 1 \); otherwise \( eq_{ij} = 0 \) \( \forall i \in L, \forall j \in T \).

\[
\forall np_b \in P \wedge np_b = (i, opr_i, ptn_i) \quad \begin{cases} 
\text{pos}_i = ptn_i, & \text{if } opr_i \text{ is } = \\
\text{pos}_i \leq ptn_i - 1, & \text{if } opr_i \text{ is } < \\
\text{pos}_i \leq ptn_i, & \text{if } opr_i \text{ is } < \\
\text{pos}_i \geq ptn_i + 1, & \text{if } opr_i \text{ is } > \\
\text{pos}_i \geq ptn_i, & \text{if } opr_i \text{ is } \geq 
\end{cases} \quad (22)
\]

Constraint (23) indicates the number of teams \( j \) that finish the competition with better or equal points as team \( i \). This number (i.e., \( \text{worstPos}_i \)) is the upper bound
for pos as expressed in constraint (25).

\[
geq_{ij} = \begin{cases} 
1, & \text{if } totalPts_j \geq totalPts_i \\
0, & \text{otherwise}
\end{cases} \quad \forall j \in T \land \forall i \in L
\]

\[
worstPos_i = \sum_{j=1}^{n} geq_{ij} \quad \forall i \in L
\]  

(23)

Constraint (24) indicates whether teams \(i\) and \(j\) \((i \neq j)\) end up with the same points at the end of the competition. The lower bound for pos \((i.e., \text{bestPos}_i)\) can be computed by subtracting from the upper bound, the total teams in the same position as team \(i\).

\[
eq_{ij} = \begin{cases} 
1, & \text{if } totalPts_j = totalPts_i \\
0, & \text{otherwise}
\end{cases} \quad \forall j \in T \land \forall i \in L
\]

\[
\text{bestPos}_i = \text{worstPos}_i - \sum_{j=1,j \neq i}^{n} eq_{ij} \quad \forall i \in L \land i \neq j
\]  

(24)

Finally, constraint (25) sets the bounds for the position of team \(i\) and constraint (26) indicates that the positions for two teams must be different.

\[
\text{bestPos}_i \leq \text{pos}_i \leq \text{worstPos}_i \quad \forall i \in L
\]  

(25)

\[
(\text{pos}_i \neq \text{pos}_j) \quad \forall i, j \in L \land i \neq j
\]  

(26)

Relative Position Queries: we use these variables to represent queries about relative positions between two teams, e.g., team A will be in a better position than team B. Constraint (27) depicts the five queries. In this particular case we use totalPts\(_i\) and totalPts\(_j\). We consider that two teams \(i\) and \(j\) might tie up in the same position if they have the same points at the end of the competition. We recall that we don’t use pos\(_i\) and pos\(_j\) due to constraint (26) which indicates that two teams must have different positions at the end of the competition.

- \(R\): set of possible relative position queries defined as a set of triples \(nr_c = (i, op_{ij}, j)\) and \(1 \leq c \leq |R|\) if \(|R| \geq 1\);
- \(op_{ij}\): denoting a logical operator \((op_{ij} \in \{<, \leq, >, \geq, =\})\) to constrain a pair of teams \(i\) and \(j\).

\[
\forall nr_c \in R \land nr_c = (i, op_{ij}, j)
\begin{cases} 
totalPts_i = totalPts_j, & \text{if } op_{ij} = = \\
totalPts_i \leq totalPts_j - 1, & \text{if } op_{ij} = < \\
totalPts_i \leq totalPts_j, & \text{if } op_{ij} = \leq \\
totalPts_i \geq totalPts_j - 1, & \text{if } op_{ij} = > \\
totalPts_i \geq totalPts_j, & \text{if } op_{ij} = \geq
\end{cases}
\]  

(27)

Final Point Queries: (also known as score queries) we use these variables for queries about the final points of the teams, e.g., team A scores 75 points at the end of the competition. To guarantee a set \(S\) of final point queries, the model should include the linear constraints from (28). Notice that the score \(s_i\) should have a lower bound of initialPts\(_i\), and an upper bound of all the possible points if team \(i\) wins all its the remaining games \(g_{ij}\):

- \(S\): set of possible final point queries defined as a set of tuples \(ns_d = (i, s_i)\) and \(1 \leq d \leq |S|\) if \(|S| \geq 1\);
\(s_i\): denoting the wanted final points of team \(i\).

\[
(totalP_{ts_i} = s_i) \quad \forall ns_d \in S \land ns_d = (i, s_i)
\]

\[
initialP_{ts_i} \leq s_i \leq initialP_{ts_i} + 3 \cdot \sum_{j=1, j \neq i}^{n} g_{ij} \quad \forall ns_d \in S \land ns_d = (i, s_i)
\] (28)

**Objective Function:** Users might be interested in either minimizing or maximizing the total points for a given team as depicted in (29):

\[
\text{maximize: } totalP_{ts_i} \quad (i \in T)
\] (29)

6. **Empirical evaluation.** In this section we evaluate our CP model from section 3 and the impact of the redundant constraints (Section 4.1), the variable/value selection heuristics (Section 4.2), the heuristics for position in ranking queries (Section 4.2.1), the classifier for value selection (Section 4.2.2), and the restart-based search (Section 4.3). We also evaluate our MIP model from Section 5 and perform empirical comparisons against the CP model.

6.1. **Tests configuration.** We evaluated our models using five reference solvers. Three of them distributed with MiniZinc (V 2.0.14). Namely, Gecode, G12/FD, and HaifaCSP. We also used Mozart-Oz (V 1.4.0) and CPLEX (V12.6.2). MiniZinc is a free and open-source modeling language for constraint satisfaction and optimization problems [30]. Both Gecode and G12/FD solvers are included in the distribution 2.0.14 of MiniZinc. Gecode is a C++ constraint solving library [16] and G12/FD is a Mercury FD solver developed by the G12 project, concerned with developing a software platform for solving large-scale industrial combinatorial optimisation problems [43]. HaifaCSP is a CSP solver with non-clausal learning that won two gold medals (free search and parallel search categories), and a silver medal (Open class category) in the 2016 MiniZinc challenge [45]. Mozart implements the Oz language and the version 1.4.0 provides constraint programming support [9]. CPLEX is an optimization engine developed by IBM for solving problems expressed as mathematical programming model.

In particular we focus our attention in two leagues:

- **Colombian league** (liga Postobón 2014-I) with 18 teams and 18 fixtures to play in a single round-robin schedule (17 fixtures + 1 extra fixture for the derbies)
- **Argentine league** (2015) with 30 teams and 30 fixtures to play in a single round-robin schedule (29 fixtures + 1 extra fixture for the derbies)

For both leagues, we provided five experimental scenarios to explore different stages of the competitions. That is, fixtures 7, 9, 11, 14, and 16 for the Colombian league (resp. fixtures 5, 10, 15, 20, and 25 for the Argentine league). We randomly created 9,000 instances for both leagues with 3,000 position in ranking queries \((P)\), 3,000 relative position queries \((R)\), and 3,000 final point queries \((S)\). We excluded game results queries \((Q)\) from our experiments as they can be trivially solved with our models. Moreover, we explored combinations of queries with 2, 3, 4, 5, 7, and 9 suppositions.

The 9,000 instances were the result of creating 100 instances for every combination of query type \((P, R, S)\), fixture \((7, 9, 11, 14, 16)\) for the Colombian league. Resp. \((5, 10, 15, 20, 25)\) for the Argentine league), and number of suppositions \((2, 3, 4, 5, 7, 9)\). We would like to remark that these instances are to the best of our knowledge the first soccer computational benchmark to be released. Both league
benchmarks and our models can be downloaded from https://sites.google.com/site/robinsonunivalle/.

We use the default search configuration for all the solvers. All the experiments were performed in a 4-core machine featuring an Intel Core i5 processor at 2.3 Ghz and 4GB of RAM. For each instance we used an execution time limit of 30 seconds. We would like to highlight that the models described in this paper solved nearly all instances within 30 seconds; Section 6.2.1 provides experimental results varying the time limit from 30 seconds to 10 minutes.

We recall that our Web-based version of SABIO implements the constraint programming model presented in Section 3. Our main challenge consists in maximizing solved instances and reducing computational time in order to be able to offer the service to tens of thousands of users. Furthermore, SABIO is an interactive tool, and as such, the literature [29][7] suggests that 1 second is the time limit for a user to feel that s/he is freely interacting with an application. Additionally, 10 seconds is the time limit to keep the attention of the users.

Tables 1 and 2 summarize the solvers and the model configurations of our tests. Namely, we evaluated 5 model configurations: CP which consists of the basic constraint programming model from Section 3; CP-R which extends the basic CP model with the redundant constraints from Section 4.1; CP-ML which extends the basic CP model using the variable/value selection heuristics from Section 4.2, including our ML approach; E-CP which extends the basic CP model using all the improvements from Section 4; Finally, the MIP model which presents our basic mixed integer programming formulation from Section 5. We recall that our models including heuristics for variable selection and our machine learning approach for value selection (i.e., CP-ML and E-CP) were only implemented using the Mozart-Oz solver as depicted in Table 2.

In order to analyze the performance of our models, we reported sequential executions using a single core for all the solvers and also parallel executions using 4 cores for Oz, HaifaCSP, Gecode, and CPLEX.

6.2. Tests results. We start by studying the impact of our variable/value selection heuristics and the redundant constraints using three of our models (i.e., CP, CP-ML,
In this section we mainly focus our attention in analysing two variables that should be observed as a whole. Namely, amount of solved instances and average execution time.

| Constraint Type | 1 Queries | 2 Queries | 3 Queries |
|-----------------|-----------|-----------|-----------|
| Fixtures 7      |           |           |           |
| Fixtures 9      |           |           |           |
| Fixtures 11     |           |           |           |
| Fixtures 14     |           |           |           |
| Fixtures 16     |           |           |           |
| Unsolvable      | 1000      | 0         | 1         |

| Constraint Type | 1 Queries | 2 Queries | 3 Queries |
|-----------------|-----------|-----------|-----------|
| Fixtures 7      |           |           |           |
| Fixtures 9      |           |           |           |
| Fixtures 11     |           |           |           |
| Fixtures 14     |           |           |           |
| Fixtures 16     |           |           |           |
| Unsolvable      | 1000      | 0         | 1         |

| Constraint Type | 1 Queries | 2 Queries | 3 Queries |
|-----------------|-----------|-----------|-----------|
| Fixtures 7      |           |           |           |
| Fixtures 9      |           |           |           |
| Fixtures 11     |           |           |           |
| Fixtures 14     |           |           |           |
| Fixtures 16     |           |           |           |
| Unsolvable      | 1000      | 0         | 1         |

**Table 3.** Unsolvable instances and average running times for the Colombian league (18 teams) evaluated sequentially and in parallel with the CP, CP-ML, and E-CP models using Mozart-Oz.
time of those solved instances. Our models provide significant improvements for SABIO. In particular, when the load of queries grow as a tournament goes. The improvements can be observed in the number of solved instances and the runtime to solve individual instances.

6.2.1. Oz solver results. The overall results of the Oz implementations for the Colombian league using sequential and parallel executions are shown in Table 3.

| Constraint Type | 1 Core Experiments | 4 Core Experiments |
|-----------------|--------------------|--------------------|
|                 | T Queries          | S Queries          | T Queries | S Queries |
| Fixture 5       | Oz-CP              | Oz-CP-ML           | Oz-CP     | Oz-CP-ML  |
| Unsolved Instance | 1310              | 782                | 0         | 1         |
| Oz-CP           | 0                  | 0                  | 0         | 0         |
| Oz-CP-ML        | 0                  | 0                  | 0         | 0         |
| Fixtures 10     | Oz-CP              | Oz-CP-ML           | Oz-CP     | Oz-CP-ML  |
| Unsolved Instance | 1099              | 0                  | 0         | 1         |
| Oz-CP           | 0                  | 0                  | 0         | 0         |
| Oz-CP-ML        | 0                  | 0                  | 0         | 0         |
| Fixtures 15     | Oz-CP              | Oz-CP-ML           | Oz-CP     | Oz-CP-ML  |
| Unsolved Instance | 1154              | ROBINSON DUQUE, ALEJANDRO ARBELAEZ AND JUAN FRANCISCO DÍAZ | 0         | 1         |
| Oz-CP           | 0                  | 0                  | 0         | 0         |
| Oz-CP-ML        | 0                  | 0                  | 0         | 0         |
| Fixtures 20     | Oz-CP              | Oz-CP-ML           | Oz-CP     | Oz-CP-ML  |
| Unsolved Instance | 127               | 0                  | 0         | 1         |
| Oz-CP           | 0                  | 0                  | 0         | 0         |
| Oz-CP-ML        | 0                  | 0                  | 0         | 0         |
| Fixtures 25     | Oz-CP              | Oz-CP-ML           | Oz-CP     | Oz-CP-ML  |
| Unsolved Instance | 198               | 0                  | 0         | 1         |
| Oz-CP           | 0                  | 0                  | 0         | 0         |
| Oz-CP-ML        | 0                  | 0                  | 0         | 0         |

Table 4. Unsolved instances and average running times for the Argentine league (30 teams) evaluated sequentially and in parallel with the CP, CP-ML, and E-CP models using Mozart-Oz.
Both tables show the number of unsolved instances and the average runtimes of the solved ones using 1 and 4 cores respectively. It can be observed in both leagues that $R$ and $S$ queries are the easiest ones and nearly all instances can be solved within the time limit (i.e., 30 sec). Alternatively, $P$ queries are the hardest ones, particularly for our basic CP approach.

For the Colombian league (Table 3), we observe 1069 unsolved instances using the CP model in a single core, that is about 36% unsolved $P$ instances. We attribute this to 2 main reasons: first, the position bounds (i.e., $bestPos_i$ and $worstPos_i$ from the basic CP model in section 3) can only be computed after finding the total points ($totalPts_i$) for all the teams in the competition. As a result, position in ranking constraints are validated only when the search algorithm performs a complete game points assignment for all teams. Second, we observed that our variable/value selection heuristics struggle with queries related to the “=” operator and the lack of a biased search causes position overshooting.

In Table 3 it can be observed that our CP-ML provides a good trade-off between the number of unsolved instances and the runtime. Generally speaking, the machine learning approach improves the effectiveness of the algorithm by reducing the number of unsolved instances from 1069 to 629, at a cost of increasing the average runtime of solved instances from 0.51s to 1.61s. Additionally, our extended model $E$-CP managed to reduce the number of unsolved instances to 170, that is 899 and 456 more than CP and CP-ML respectively. These results show that the combination of our redundant constraints and our machine learning approach has a positive impact in our experiments by reducing the number of unsolved instances up to 6.2 times compared to the basic CP model.

We also point out that the model with redundant constraints helps to reduce the runtime of the machine learning approach. We observe a small improvement in the amount of solved instances and average runtimes of the parallel execution with respect to the sequential execution.

As an academic exercise, we evaluated the performance of our models with longer time limits, i.e., increasing the time limit from 30 seconds to 10 minutes. However, the improvement with the additional time is negligible and the number of solved instances remains the same as with the 30 seconds time limit. We attribute this behaviour to the fact that after a 30 second execution, most of the unsolved instances are typically UNSAT (i.e., there is no feasible solution). Therefore, solving these instances would require an exhaustive search that leads to no improvements due to the NP-completeness nature of the problem.

We now move our attention to the Argentine league in Table 4 which includes 30 teams in the competition (12 teams more than the Colombian league). Despite the fact that both tests use different instances, we observe that the increment in the number of teams also has an impact in the number of unsolved instances. It went from 1069 unsolved instances in the Colombian league to 1419 in the Argentine league. Moreover, in the best case scenario using the $E$-CP model, we can observe that it went from 170 to 396 unsolved instances.

Despite the increment in the number of unsolved instances, the models tested with the Argentine league present a similar behaviour as in the Colombian league. That is, we observe the highest number of unsolved instances using the CP model in a single core for $P$ queries. On the other hand, the CP-ML model improves the amount of solved instances while displaying an average time trade-off. Finally, the $E$-CP model represents the best approach by reducing up to 3.5x the amount of
unsolved instances with respect to the CP model, while reducing up to 1.4x the time trade-off introduced by the machine learning approach in the CP-ML.

6.2.2. MiniZinc solver results. In this section we present empirical results for three models (CP, CP-R, and MIP) tested using MiniZinc solvers and CPLEX. Tables 5 and 6 summarise the results for the Colombian and Argentine leagues with sequential and parallel executions. In these tables we indicate that an instance is satisfiable (SAT) if there is found a feasible solution for the instance, and unsatisfiable (UNSAT) otherwise.

For both leagues we observe that the overall best solver using the CP and the CP-R models is HaifaCSP while CPLEX is the best solver using the MIP model. In the case of the Colombian league with a single core (Table 5), HaifaCSP solves 8949 instances with the basic CP model and outperforms G12/FD and Gecode by 357 and 649 solved instances, as well as in their average runtime. We also observe that

### Table 5. Test results using MiniZinc solvers and CPLEX for the Colombian League with 18 teams

| Model | Solver | Solved Instances | Unsolved Instances | Avg Solved Instances | Std Solved Instances | Solved SAT Instances | Solved UNSAT Instances |
|-------|--------|------------------|--------------------|----------------------|----------------------|----------------------|------------------------|
| CP    | G12/FD | 8592             | 408                | 0.85                 | 2.44                 | 5851                 | 2741                   |
|       | Gecode | 8300             | 700                | 0.67                 | 2.21                 | 5584                 | 2716                   |
|       | HaifaCSP | 8949 | 51                 | 0.36                 | 1.79                 | 6178                 | 2771                   |
| CP-R  | G12/FD | 8473             | 525                | 0.79                 | 2.25                 | 5701                 | 2774                   |
|       | Gecode | 8199             | 801                | 0.82                 | 2.56                 | 5664                 | 2535                   |
|       | HaifaCSP | 8967 | 3                  | 0.15                 | 0.75                 | 6220                 | 2777                   |
| MIP   | CPLEX  | 8955             | 5                  | 0.45                 | 0.29                 | 6246                 | 2779                   |
|       | G12/FD | 8468             | 532                | 1.36                 | 3.4                  | 5769                 | 2699                   |
|       | Gecode | 8198             | 802                | 0.77                 | 1.93                 | 5499                 | 2699                   |
|       | HaifaCSP | 8916 | 84                  | 0.61                | 2.07                 | 6190                 | 2766                   |

### Table 6. Test results using MiniZinc solvers and CPLEX for the Argentine League with 30 teams

| Model | Solver | Solved Instances | Unsolved Instances | Avg Solved Instances | Std Solved Instances | Solved SAT Instances | Solved UNSAT Instances |
|-------|--------|------------------|--------------------|----------------------|----------------------|----------------------|------------------------|
| CP    | G12/FD | 7635             | 1365               | 2.14                 | 2.35                 | 6166                 | 1469                   |
|       | Gecode | 7406             | 1594               | 1.5                  | 1.72                 | 5973                 | 1433                   |
|       | HaifaCSP | 8303 | 637                | 1.07                 | 3.23                 | 6821                 | 1542                   |
| CP-R  | G12/FD | 7389             | 1611               | 1.3                  | 2.14                 | 5839                 | 1550                   |
|       | Gecode | 7752             | 1248               | 1.48                 | 2.12                 | 6197                 | 1555                   |
|       | HaifaCSP | 8858 | 142                | 0.94                 | 2.63                 | 7280                 | 1578                   |
| MIP   | CPLEX  | 8979             | 21                 | 1.42                 | 1.63                 | 7411                 | 1568                   |
|       | G12/FD | 6401             | 2599               | 3.66                 | 3.58                 | 5176                 | 1225                   |
|       | Gecode | 6745             | 2255               | 1.56                 | 2.54                 | 5428                 | 1317                   |
|       | HaifaCSP | 8263 | 737                | 1.8                  | 3.58                 | 6742                 | 1521                   |
our redundant constraints improve the search performance of HaifaCSP by reducing the number of unsolved instances from 51 to 3. This improvement also has an impact in the average runtime which improves from 0.36s to 0.15s. Interestingly, the same behaviour can be observed in the Argentine league (Table 5).

For the Argentine league experimentations (Table 6), we can observe that the parallel executions slightly improve the sequential search performance for HaifaCSP. It improves from 637 to 459 and from 142 to 88 unsolved instances using the CP and CP-R models respectively. However, this improvement is only present for the CP model in the parallel Colombian league (Table 5) where the solver improves from 51 to 39 unsolved instances, but the CP-R model goes from 3 to 6 unsolved instances.

In our experimentations, CPLEX outperforms the rest of the solvers with the MIP model. For instance, in the Argentine league using a sequential execution, CPLEX solves 121 more instances than the best execution of HaifaCSP which was possible due to the CP-R model. CPLEX is the solver with the smallest number of unsolved instances and present the overall best effectiveness, particularly with the parallel executions (i.e., 1 and 79 unsolved instances for the Colombian and Argentine leagues respectively).

In this section we compare our best solver-model configurations and propose mixed execution approaches in order to improve performance (i.e., solved instances and average execution time). In general, the best performance in our experiments
Figure 3. Argentine league runtime comparison (time in seconds for parallel executions)

were displayed by HaifaCSP using the \textit{CP-R} model, CPLEX with the MIP model, and Oz with the \textit{E-CP} model.

We start with Figures 2 and 3 where we compare the performance of our best solver-model configurations for both leagues. In these figures, the \textit{x} and \textit{y}-axis represent the runtime of two models in log scale, blue (resp. red) points indicate \textit{SAT} (resp. \textit{UNSAT}). Points below the diagonal indicate that the model in the \textit{y}-axis is faster, respectively for the models in the \textit{x}-axis with points above the diagonal.

Figure 2(a) evaluates HaifaCSP and CPLEX for the Colombian league. In this experiment HaifaCSP is typically faster. Additionally HaifaCSP seems to be more robust than CPLEX for \textit{UNSAT} instances in both competitions as depicted in Figures 2(a) and 3(a).

Figures 2(b) and 2(c) show the results for Oz, HaifaCSP, and CPLEX for the Colombian league (resp. Figures 3(b) and 3(c) for the Argentine league). Interestingly Oz is considerably faster than the other solvers, but HaifaCSP and CPLEX are typically better with respect to capacity solving (total number of solved instances). This behaviour can be observed in the cumulative solved instances below the diagonal for Oz, as well as the accumulations of timeouts at the top of the before mentioned figures.

Tables 7 and 8 analyse the previous behaviour in more detail. Each table contains a comparison of common solved instances for two solver-model configurations. We
can observe in Table 7 that Oz is faster than CPLEX in 8745 and 8320 common solved instances for the Colombian and the Argentine leagues respectively. CPLEX is faster than Oz in only 89 common solved instances. This behaviour shows that Oz is faster than CPLEX about 97% and 92% of the time in the Colombian and Argentine experiments. However, CPLEX is more effective than Oz and leaves 1 and 79 unsolved instances (Tables 5 and 6) while Oz reports 165 and 393 unsolved instances for both leagues in the parallel executions (Tables 3 and 4).

Additionally, in Table 8 we can see that Oz is also faster than HaifaCSP in 7223 and 7927 instances. That is, 80% and 88% faster than HaifaCSP for the Colombian and the Argentine tournaments. We also observe that HaifaCSP is more effective than Oz since it reports 6 and 88 unsolved instances (Tables 5 and 6) compared to the 165 and 393 unsolved instances for both leagues in the parallel executions with the Oz solver (Tables 3 and 4).

6.2.3. Solver-model comparisons and mixed approaches. Interestingly, these solvers report a complementary behaviour. That is, CPLEX and HaifaCSP are typically more effective than Oz whilst Oz is typically faster. Figures 2 and 3 show this behaviour for the Colombian and Argentinean competitions, instances below the diagonal indicate the Oz is faster. Therefore, to exploit the complementary behaviour of the solvers, we experimented with two mixed executions:

- Oz(E-CP) & CPLEX(MIP)
- Oz(E-CP) & HaifaCSP(CP-R)

In the mixed execution we decided to run first our Oz(E-CP) solver for 1 second to solve as many instances as possible, then we run our second solver (i.e., CPLEX or HaifaCSP) for the remaining 29 seconds.

In Figures 4(a) and 5(a) we exploit the complementary behaviour of Oz and CPLEX and the corresponding models (i.e., E-CP and MIP) using mixed executions for the Colombian and Argentine leagues. We can observe in both figures that our
mixed approach (on the $y$-axis) is considerably faster than CPLEX (on the $x$-axis) and only for a few instances it would have been better to alternate the execution of MIP and CP (i.e., instances above the diagonal). Interestingly, the same behaviour can be observed in Figures 4(b) and 5(b) with respect to Oz and HaifaCSP.

For the Colombian league, our mixed approach using Oz(E-CP) and CPLEX (MIP) is able to solve 8999 instances with an average time of 0.11s and a standard deviation of 0.3 as shown in Table 9. This approach outperforms the parallel Oz(E-CP) execution depicted in Table 3 by 164 solved instances. In this experiment we observe an improvement in the average runtime of 0.36s (from 0.47s to 0.11s). We recall that these results should be observed as a whole, i.e., number of solved instances and average runtime. For instance, in our previous experiment we improved the number of unsolved instances from 165 to only 1. Furthermore, we also would like to point out that our models are meant to be used in SABIO, thus, runtime improvements can become significant when the number of queries grow, particularly in short periods of time (e.g., after “important” soccer matches).
Our experimental results show that the impact of our mixed approaches seems to become more important as the tournament grows. For instance, in the Argentine league the Oz(E-CP) and CPLEX(MIP) mixed execution is able to solve 8958 instances with an average time of 0.56s and a standard deviation of 1.74 as shown in Table 10. That is, 351 and 37 more instances than the parallel Oz(E-CP) execution depicted in Table 4 and the parallel execution of CPLEX(MIP) in Table 6. Additionally, our mixed approach reduces the average time for solved instances from 1.12s to 0.56s and the standard deviation from 2.19 to 1.74.

We recall that a similar behaviour is presented by the mixed execution of Oz(E-CP) and HaifaCSP(CP-R). However, we only focused on the mixed execution of Oz(E-CP) and CPLEX(MIP) since it represents the overall best solving approach with respect to runtime and capacity solving.

We now introduce the results from Table 11. This table let us observe the behaviour of our best model-solver configurations with respect to the suggestions about time limits of applications in general \[29\][7]. i.e., 1 second to freely interact with an application and 10 seconds as the time limit to keep the attention of the users. We can observe for the Colombian league that our CP-R approach running with HaifaCSP solves the highest amount of instances under a second. Additionally, our mixed approach using OZ & CPLEX gets to the highest amount of solved instances within 10 seconds.

The differences in the number of solved instances is more evident in the Argentine league. For instance, our E-CP model running with OZ, solves 918 and 1533 more instances than MIP/Cplex and CP-R/HaifaCSP within a second. Additionally, our mixed approach solves more instances than the best individual model/solver configurations after a second of execution (e.g., 10, 20, and 30 seconds).

7. Conclusions. SABIO is an online platform that implements a Constraint Programming (CP) model that allows soccer fans formulate questions about their teams. However, the former CP computational model struggled to solve position in ranking queries. In this paper we presented and evaluated five computational models to solve general soccer fan queries at different stages of the competition. Namely, the models are based on a Constraint Programming and a Mixed Integer Programming (MIP) formulation. We presented a series of improvements for the CP model that included: redundant constraints, dedicated variable/value selection
heuristics, and a restart-based approach. These improvements led us to define three extensions of the CP model, i.e., CP-R, CP-ML, and E-CP.

We also defined and released what is to the best of our knowledge, the first soccer computational problems benchmark, including instances for two soccer competition leagues. We evaluated and compared our 5 models using five reference solvers including Mozart Oz, CPLEX, and 3 MiniZinc solvers (i.e., Gecode, G12/FD, and HaifaCSP). Our experiments showed that the basic CP model is highly improved by the respective extensions and the top performance in solving speed was obtained mainly by using Oz with the E-CP model, whilst the top capacity solving was obtained by using HaifaCSP with the CP-R model. On the other hand, our experiments showed that CPLEX performs better than the MiniZinc solvers using our MIP model and outperforms the CP approaches in solving capacity (i.e., number of solved instances).

We compared our CP formulations against the MIP models and observed a complementary behaviour between the two approaches. Particularly, the E-CP approach is faster than the MIP model. However, the MIP model is able to solve more instances than the CP one. As a result, we proposed a series of mixed model executions where the combination of both E-CP and MIP lead to a very fast and robust solution. Finally, we recall that we expect our SABIO Web application to interact with thousands of users at the same time, therefore both speed and capacity solving of the two models are expected to play an important role.

**Acknowledgments.** SABIO is a platform created with the academic support of the AVISPA research group from the Universidad del Valle. We would like to thank Luis Felipe Vargas, María Andrea Cruz and Carlos Martínez for working in early versions of the CP model implemented for SABIO and the development of the online platform under the supervision of Juan Francisco Díaz. Robinson Duque is supported by the Universidad del Valle and also by Colciencias, the Colombian Administrative Department of Science, Technology and Innovation under the PhD scholarship program. Additionally, we would like to thank the anonymous reviewers for their suggestions and comments that helped to improve this paper.

**REFERENCES**

[1] I. Adler, A. L. Erera, D. S. Hochbaum and E. V. Olinick, Baseball, optimization, and the world wide web, Interfaces, 32 (2002), 12–22.
[2] F. Alarcón, G. Durán, M. Guajardo, J. Miranda, H. Muñoz, L. Ramírez, M. Ramírez, D. Sauré, M. Siebert, S. Souyris, A. Weintraub, R. Wolf-Yadlin and G. Zamorano, Operations research transforms the scheduling of Chilean soccer leagues and South American World Cup qualifiers, *Interfaces*, 47 (2017), 52–69.

[3] T. Bartsch, A. Drexl and S. Kröger, Scheduling the professional soccer leagues of Austria and Germany, *Computers & Operations Research*, 33 (2006), 1907–1937.

[4] T. Bernholt, A. Gälich, T. Hofmeister and N. Schmitt, Football elimination is hard to decide under the 3-point-rule, *International Symposium on Mathematical Foundations of Computer Science*, (1999), 410–418.

[5] J. Borrett, E. P. Tsang and N. R. Walsh, Adaptive constraint satisfaction: The quickest first principle, in *European Conference on Artificial Intelligence*, (1996).

[6] F. Boussemart, F. Hemery, C. Lecoutre and L. Sais, Boosting systematic search by weighting constraints, in *ECAI*, (2004), 146–150.

[7] S. K. Card, G. G. Robertson and J. D. Mackinlay, The information visualizer, an information workspace, in *SIGCHI Conference on Human Factors in Computing Systems (ACM)*, (1991), 181–186.

[8] J. Christensen, A. N. Knudsen and K. S. Larsen, Soccer is harder than football, *International Journal of Foundations of Computer Science*, 26 (2015), 477–486.

[9] M. Consortium, The mozart programming system, 2013. Available from *http://mozart.github.io/*.

[10] D. Goossens and F. Spieksma, Scheduling the Belgian soccer league, *Interfaces*, 39 (2009), 109–118.

[11] D. R. Goossens, J. Beliën and F. C. Spieksma, Comparing league formats with respect to match importance in Belgian football, *Annals of Operations Research*, 194 (2012), 223–240.

[12] J. C. Guedes and F. S. Machado, Changing rewards in contests: Has the three-point rule brought more offense to soccer?, *Empirical Economics*, 27 (2002), 607–630.

[13] W. Kern and D. Paulusma, The new FIFA rules are hard: Complexity aspects of sports competitions, *Discrete Applied Mathematics*, 108 (2001), 317–323.

[14] W. Kern and D. Paulusma, The computational complexity of the elimination problem in generalized sports competitions, *Discrete Optimization*, 1 (2004), 205–214.

[15] J. Kyngas and K. Nurmi, The subset sum problem in *Knapsack Problems*, (1967), 391–401.

[16] H. Kellerer, U. Pferschy and D. Pisinger, *The Subset Sum Problem*, (2004), 73–86.
[28] S. T. McCormick, Fast algorithms for parametric scheduling come from extensions to parametric maximum flow, *Operations Research*, **47** (1999), 744–756.

[29] R. B. Miller, Response time in man-computer conversational transactions, in *Joint Computer Conference, part 1*, (1968), 267–277.

[30] Monash University, Data61 Decision Sciences and University of Melbourne. Minizinc, 2014, Available from [http://www.minizinc.org/](http://www.minizinc.org/).

[31] T. F. Noronha, C. C. Ribeiro, G. Duran, S. Souyris and A. Weintraub. A branch-and-cut algorithm for scheduling the highly-constrained Chilean soccer tournament, in *International Conference on the Practice and Theory of Automated Timetabling*, (2006), 174–186.

[32] D. Pálvölgyi, Deciding soccer scores and partial orientations of graphs, *Acta Univ. Sapientiae, Math.*, **1** (2009), 51–71.

[33] C. Raack, A. Raymond, T. Schlechte and A. Werner, Standings in sports competitions using integer programming, *Journal of Quantitative Analysis in Sports*, **10** (2014), 131–137.

[34] R. V. Rasmussen, Scheduling a triple round robin tournament for the best danish soccer league. *European Journal of Operational Research*, **185** (2008), 790–810.

[35] R. V. Rasmussen and M. A. Trick, Round robin scheduling-a survey, *European Journal of Operational Research*, **188** (2008), 617–636.

[36] P. Refalo, Impact-based search strategies for constraint programming, in *International Conference on Principles and Practice of Constraint Programming*, (2004), 557–571.

[37] C. C. Ribeiro and S. Urrutia, An application of integer programming to playoff elimination in football championships, *International Transactions in Operational Research*, **12** (2005), 375–386.

[38] C. C. Ribeiro and S. Urrutia, Scheduling the Brazilian soccer tournament with fairness and broadcast objectives, in *International Conference on the Practice and Theory of Automated Timetabling*, (2006), 147–157.

[39] C. C. Ribeiro and S. Urrutia, Scheduling the Brazilian soccer tournament: Solution approach and practice, *Interfaces*, **42** (2012), 260–272.

[40] F. Rossi, P. van Beek and T. Walsh, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, Elsevier, 2006.

[41] R. A. Russell and J. M. Leung, Devising a cost effective schedule for a baseball league. *Operations Research*, **42** (1994), 614–625.

[42] B. L. Schwartz, Possible winners in partially completed tournaments, *SIAM Review*, **8** (1966), 302–308.

[43] P. J. Stuckey, M. G. de la Banda, M. Maher, K. Marriott, J. Slaney, Z. Somogyi, M. Wallace and T. Walsh, The G12 project: Mapping solver independent models to efficient solutions, in *International Conference on Logic Programming*, (2005), 9–13.

[44] T. Van Voorhis, Highly constrained college basketball scheduling, *Journal of the Operational Research Society*, **53** (2002), 603–609.

[45] M. Veksler, HaifaCSP - constraints-satisfaction problem (CSP) solver, 2016. Available from [http://strichman.net.technion.ac.il/haifacsp/](http://strichman.net.technion.ac.il/haifacsp/).

[46] K. D. Wayne, A new property and a faster algorithm for baseball elimination, *SIAM Journal on Discrete Mathematics*, **14** (2001), 223–229.

[47] I. H. Witten and E. Frank, *Data mining: Practical machine learning tools and techniques, ACM SIGMOD Record*, **31** (2002), 76–77.

[48] M. B. Wright, Scheduling fixtures for basketball New Zealand, *Computers & Operations Research*, **33** (2006), 1875–1893.

[49] J. T. Yang, H.-D. Huang and J.-T. Horng, Devising a cost effective baseball scheduling by evolutionary algorithms, in *Evolutionary Computation*, (2002), 1660–1665.

Received December 2016; revised August 2017.

E-mail address: robinson.duque@correounivalle.edu.co
E-mail address: alejandro.arbelaez@cit.ie
E-mail address: juanfco.diaz@correounivalle.edu.co