This is a post-peer-review, pre-copyedit version of JIMO, 2019, Vol: 15
doi: 10.3934/jimo.2018109

Manuscript submitted to
AIMS' Journals
Volume **X**, Number **0X**, XX **200X**

doi:10.3934/xx.xx.xx.xx

pp. **X–XX**

CP AND MIP APPROACHES FOR SOLVING SOCCER COMPUTATIONAL PROBLEMS

Robinson Duque*

AVISPA Research Group
Universidad del Valle, Colombia

Alejandro Arbelaez

Insight Centre for Data Analtyics
University College Cork, Ireland

Juan Francisco Díaz

AVISPA Research Group
Universidad del Valle, Colombia

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 become the champion, (i.e., finish first in a league or be within the first k teams in a tournament to qualify to the playoffs). Some other interesting problems from literature are the *guaranteed qualification problem,* the *possible qualification problem,* the *the score vector problem, promotion and relegation.* 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 the paper we considerably improve the performance of an existing CP model and combine the use of 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 in a single or double round-robin schedule. Depending on the 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 single or double round-robin schedule for the regular season 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.

2010 *Mathematics Subject Classification.* Primary: 58F15, 58F17; Secondary: 53C35.
Key words and phrases. Dimension theory, Poincaré recurrences, multifractal analysis.
* Corresponding author: Robinson Duque.

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 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. 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 impose constraints about particular matches in any fixture. (e.g., Barcelona ends in a tie with R.Madrid).
- *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., R.Madrid will be in a better position than 3).
- *Relative Position.* These kind of queries let users impose constraints about the positions of two teams (e.g., R.Madrid will be in a worse position than Barcelona).
- *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., Barcelona score at the end of the tournament is 75).

Probably the most studied 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? can team i finish in a better position than k?"

Recently, [18] proposed a 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 is limited to single round-robin competitions. In this paper, we use some ideas from [18], and we extend it for single and double round-robin competitions. Additionally, we would like to point out that the flexibility of our model allows us to answer more queries, i.e., *game result queries*, *position in ranking queries*, *relative position queries*, and *final position queries*, that let users create different scenarios and go beyond the classical soccer computation problems.

In this paper we considerably improve an existing CP model [7], and we combine the use of MIP and CP to improve the performance of SABIO.

2. **Elimination Problem Complexity.**

At a particular stage of a baseball / soccer / basketball / handball season, the elimination problem consists of determining if a team i is already eliminated and cannot become champion or tied for first place for all the possible outcomes of the remaining games, given that team i has w_i wins and g_{ij} games left to play against team j. The complexity of this problem depends on the way how scores are allocated according to the outcome of every match [14].

This elimination problem could be transform into a network flow problem and would be polynomially solvable, if the old FIFA rule (0, 1, 2) was used, nevertheless, Kern and Paulusma [14] [15] and Bernholt et al. [3] independently proved that for the actual pointing rule (0, 1, 3) the problem is NP-complete.

2.1. **The Old FIFA Scoring System.** The elimination problem is a classical computational problem that has already been studied since the 60's and first popularized by Alan Hoffman, particularly in baseball where there are no ties and the games are played under the *"1-point-rule"*, i.e., the winner gets 1 point and no ties are allowed. There have been different studies about the elimination problem and most of them are related to baseball. For instance, Schwartz [19] proposed a method using maximum network flows in order to determine if a single team is eliminated. *These early solutions to the elimination problem in baseball using network flows resulted in polynomial time algorithms.*

Hoffman and Rivlin [13] generalized Schwartz proposal and gave a necessary and sufficient condition for not being eliminated from finishing in k-th place. Later on, McCormick [16] showed that *determining whether a team is eliminated from finishing the season in k-th place or better 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 [22] 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. [3] 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 [22].

2.2. **The new FIFA Scoring System.** In soccer and before 1995, the traditional pointing rule was to assign two points to a win, one point to a tie and zero points to 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 [10].

In 1999, Bernholt et al. [3] 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 "(α, β)-rule". This notation means that the winner team gets α points and the losing team gets 0 points. The β 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. [3] 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 [3] 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. [3] 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 [11][3].

A more generalized study of complexity aspects of sports competitions (including soccer) was proposed by Kern and Paulusma [14][15], who used the triple (α, β, γ) to denote the points assigned to a team when participating in a match. The score is increase by $\alpha \in R$ if the team loses, by $\beta \in R$ if the game ends in a tie and by $\gamma \in R$ if the team wins and they always assumed that $\alpha \leq \beta \leq \gamma$. They determined the computational complexity of the sports elimination problem assuming that ($P \neq NP$) and proved that a game played under the (α, β, γ) rule is polynomially solvable for three cases:

1. $\alpha = \beta$
2. $\beta = \gamma$
3. $\alpha + \gamma = 2\beta$

in all other cases, like in the case of soccer competitions with score allocations $(\alpha = 0, \beta = 1, \gamma = 3)$ 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 [14][15].

2.3. **Related Problems.** Soccer competitions offer an excellent opportunity to model interesting problems besides the elimination problem. Below we offer a brief explanation of the most common computational 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, while the last l teams in ranking from a higher division are relegated to a lower one. Kern and Paulusma [14] also showed that 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 for a particular team i, turn out to be NP-Complete and 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. In other words, it can be simplified as the following question: if team i is not eliminated, what is the minimum

number of points that i can win and still at least tie for first place or qualify to the playoffs? Gusfield and Martel showed that computing such number of points for soccer is NP-Hard and that it can be as hard as a subset sum problem [11].

2.3.3. *Guaranteed Point Placement:* Given a soccer tournament under the 3-point-rule system, the guaranteed point placement determines if it is possible that any assignment of outcomes to the remaining games will place team t at position at least K in the final standing [5]. Christensen et al. [5] showed that the decision problem of whether or not a team is guaranteed a certain minimum position is coNP-Complete 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 consists of calculating the minimum number of points any team has to win in order to be qualified, given any results of the rest of the teams. This problem depends on the current number of points of every team in the standing table and on the missing games to be played. This problem was tackled by Riberio and Urrutia [18] 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, if and only if its number of points is greater than or equal to its GQS.

2.3.5. *Possible Qualification Problem:* This problem consists of calculating the amount of points each team has to win in order to have any chance to be qualified. This problem was studied by Riberio and Urrutia [18] using integer programming for which they calculated a possible qualification score (PQS) which also 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 [15] suspected that deciding if a score vector is reachable or not is a difficult problem. In 2009, Plvölgyi 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 [17] by using a reduction from 3-SAT for the cases like the actual pointing rule ($\alpha = 0, \beta = 1, \gamma = 3$) applied in soccer.

3. **Basic CP Model for Interactive Soccer Queries.** Constraint programming (CP) is a powerful paradigm that can be used to solve combinatorial problems. Typically, CP combines backtracking search with constraint propagation to filter inconsistent values and reduce the search space. In [7], 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 [8] 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 $(i, j \in T)$;
- p_i: initial points of team i. If i has not played any games, then $p_i = 0$;
- F: number of fixtures left to be played in the competition. A fixture consists of one or more games between competitors;
- 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 $ng_e = (i, j, k) \wedge 0 \leq e \leq |G|$, where k is the fixture when both teams meet in a game;
- pt_{ik}: represents the points that team i gets in fixture k, $(1 \leq k \leq F$ and $pt_{ik} \in \{0, 1, 3\})$. If team i is not scheduled to play fixture k, then $pt_{i,k} = 0$.
- tp_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: if $tp_j \geq tp_i$ then $geq_{ij} = 1$; otherwise $geq_{ij} = 0 \quad (\forall i, j \in T)$;
- eq_{ij}: boolean variable indicating if two different teams i and j tie in points at the end of the competition: if $tp_j = tp_i$ and $i \neq j$ then $eq_{ij} = 1$; otherwise $eq_{ij} = 0 \quad (\forall i, j \in T)$.
- 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;

Constraint (1) represents 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 and constraint (2) corresponds to the final points tp_i of a team i:

$$2 \leq pt_{ik} + pt_{jk} \leq 3 \quad \forall ng_e \in G \wedge ng_e = (i, j, k) \tag{1}$$

$$tp_i = p_i + \sum_{k=1}^{F} pt_{ik} \quad \forall i \in T \tag{2}$$

Constraints (3) to (6) are used to calculate final positions. All the final positions must be different and every position is bounded by $bestPos_i$ and $worstPos_i$:

$$worstPos_i = \sum_{j=1}^{n} geq_{ij} \quad \forall i, j \in T \tag{3}$$

$$bestPos_i = worstPos_i - \sum_{j=1, j\neq i}^{n} eq_{ij} \quad \forall i, j \in T \wedge i \neq j \tag{4}$$

$$bestPos_i \leq pos_i \leq worstPos_i \quad \forall i \in T \tag{5}$$

$$alldifferent(pos_1, ..., pos_n) \tag{6}$$

Game result queries: Constraint (7) 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., Barcelona ends in a tie with R. Madrid:

- 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 $0 \leq a \leq |Q|$;
- 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.,$(ptc_{ik}, ptc_{jk}) \in \{(0, 3), (3, 0), (1, 1)\}$);

$$(pt_{ik} = ptc_{ik} \wedge pt_{jk} = ptc_{jk}) \quad \forall nq_a \in Q \wedge nq_a = (ptc_{ik}, ptc_{jk}) \tag{7}$$

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., R.Madrid will be in position 3. Constraint (8) 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 $0 \leq b \leq |P|$;
- opr_i: logical operator ($opr_i \in \{<, \leq, >, \geq, =\}$) to constrain team i;
- ptn_i: denoting the expected position for team i; $1 \leq ptn_i \leq n$;

$$\forall np_b \in P \wedge np_b = (i, opr_i, ptn_i) \begin{cases} pos_i = ptn_i, & \text{if } opr_i \text{ is } = \\ pos_i < ptn_i, & \text{if } opr_i \text{ is } < \\ pos_i \leq ptn_i, & \text{if } opr_i \text{ is } \leq \\ pos_i > ptn_i, & \text{if } opr_i \text{ is } > \\ pos_i \geq ptn_i, & \text{if } opr_i \text{ is } \geq \end{cases} \tag{8}$$

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 (9) depicts the five queries, e.g., Barcelona will be in a better position than R. Madrid. In this particular case we use tp_i and tp_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 (6).

- R: set of possible relative position queries defined as a set of triples $nr_c = (i, op_{ij}, j)$ and $0 \leq c \leq |R|$;
- op_{ij}: denoting a logical operator ($op_{ij} \in \{<, \leq, >, \geq, =\}$) to constrain a pair of teams i and j.

$$\forall nr_c \in R \wedge nr_c = (i, op_{ij}, j) \begin{cases} tp_i = tp_j, & \text{if } op_{ij} \text{ is } = \\ tp_i < tp_j, & \text{if } op_{ij} \text{ is } < \\ tp_i \leq tp_j, & \text{if } op_{ij} \text{ is } \leq \\ tp_i > tp_j, & \text{if } op_{ij} \text{ is } > \\ tp_i \geq tp_j, & \text{if } op_{ij} \text{ is } \geq \end{cases} \tag{9}$$

Final Point Queries: (also known as score queries) we use these variables for queries about the final points of the teams, e.g., Barcelona scores at the end of the competition 75 points. Constraint (10) guarantees *final point queries*.

- S: set of possible final point queries defined as a set of tuples $ns_d = (i, s_i)$ and $0 \leq d \leq |S|$;
- s_i: denoting the wanted final points of team i.

$$(tp_i = s_i) \quad \forall ns_d \in S \wedge ns_d = (i, s_i) \tag{10}$$

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 (tp_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: *"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 (8). 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 kind of 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 $L \subseteq T$. Now, let us start by introducing a set of variables for constrained teams $i \in L$:

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

"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):

$$\sum_{j=1, j\neq i}^{n} less_{ij} < (n - ptn_i) \quad \forall j \in T \wedge \forall i \in L \tag{11}$$

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):

$$\sum_{j=1, j\neq i}^{n} grtr_{ij} < (ptn_i - 1) \quad \forall j \in T \wedge \forall i \in L \tag{12}$$

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:

$$\sum_{j=1, j\neq i}^{n} grtr_{ij} < (ptn_i) \quad \forall j \in T \wedge \forall i \in L \tag{13}$$

$$\sum_{j=1, j\neq i}^{n} less_{ij} < (n - ptn_i + 1) \quad \forall j \in T \wedge \forall i \in L \tag{14}$$

4.2. **Heuristics and Machine Learning for Variable/Value Selection.** Generic heuristics (e.g., [2, 12]) typically do not perform well for real-life problems as these heuristics do not exploit the structure of the problem. Therefore, in this paper we propose some query based heuristics for variable/value selection. First we introduce a set of required variables in order to describe a priority mechanism to select the team variables constrained in queries P:

- $spos_i$: starting position of team i before any branching strategy is applied;
- pri_i: denoting the priority of team i to be selected during branching, If team i does not appear in any query, then $pri_i = 0$;
- str_i: denoting the global branching strategy for the variables pt_{ik} of a particular team i in every fixture k. str_i starts with "tie" as a default value.

4.2.1. *Heuristics for Position in Ranking Queries (P).* Recall from (8) that we use the position pos_i to constrain a team to a wanted position ptn_i. Suppose we have the query $(pos_i < 8)$. It's natural to try $str_i = win$ and assign a priority using the position ptn_i from the query. We depict in (15) some general rules for variable value selection:

$$\forall np_b \in P \wedge np_b = (i, opr_i, ptn_i) \begin{cases} \text{if } opr_i \text{ is } < \text{ or } \leq, & pri_i = n - ptn_i \wedge str_i = win \\ \text{if } opr_i \text{ is } > \text{ or } \geq, & pri_i = ptn \wedge str_i = lose \\ \text{if } opr_i \text{ is } =, & pri_i = ptn \wedge \begin{cases} str_i = lose, \text{ if } ptn_i > n/2 \\ str_i = win, \text{ otherwise} \end{cases} \end{cases} \tag{15}$$

Interestingly the defined heuristics in (15) for queries with the "=" operator seem to fail quite often (see section **??**). 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 $str_i = win$ 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.* For teams constrained with the "=" operator, we decided to assign a high priority $(pri_i = |spos_i - ptn_i| \cdot n)$ for variable selection and to avoid position overshooting, we trained a classifier that selects among 9 branching strategies: S1= [1,0,0], S2= [0,1,0], S3= [0,0,1], S4= [0.5,0.5,0], S5= [0.5,0,0.5], S6= [0,0.5,0.5], S7= [0.5,0.25,0.25], S8= [0.25,0.5,0.25], S9= [0.25,0.25,0.5]. Each strategy defines probabilities to select among [win, tie, lose] respectively, e.g, S7 means that for a team i, every variable pt_{ik} will be assigned *win* with a probability of 0.5, *tie* and *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 tests showed a poor performance for this alternative.

Identifying an Array of Features: Calculating the features of a given instance shouldn't be a computationally expensive process, therefore, we managed to identify an array of 5 features for teams constrained to a position with the equals operator that might be useful for the classifier in order to decide which strategy should be applied:

- *Starting Position* ($spos_i$): This feature was selected because every team has an initial position depending on its initial points (p_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, in order to choose a heuristic for value selection.
- *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 ($spos_i - ptn_i$) 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* ($|spos_i - ptn_i|/n$): This feature determines the rate of the distance from the starting position to the wanted position in relation to the number of teams.

Training the classifier: In order to train this classifier, 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. We ran every query with each of the 9 branching strategies in order to get the strategy that solved the instance in the shortest time and created a data set with the array of features mentioned before and the best executing strategy $\in \{S1, S2, S3, S4, S5, S6, S7, S8, S9\}$ as depicted in Table 1.

In this paper we use J48 (the Weka v3.6.12 implementation of C4.5 [23]) to evaluate the performance of the algorithms. The objective is that for each query P with the equality operator "=", J48 assigns one of the nine branching strategies to the constrained team. The model was trained using Cross-validation and it correctly classified 81.8% of the instances and had a relative absolute error of 27.3%.

4.3. **Sequential and Parallel Restart-Based Search.** Inspired by the quickest first principle [4], 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) {S1, ..., $S4$} followed by {S5, ..., $S9$}. We finish the execution with the random strategy for all cores.

```
@RELATION SoccerComputationalProblem

@ATTRIBUTE Direction     NUMERIC
@ATTRIBUTE StartingPos   NUMERIC
@ATTRIBUTE WantedPos     NUMERIC
@ATTRIBUTE RangeRate     NUMERIC
@ATTRIBUTE Fixtures      NUMERIC
@ATTRIBUTE Strategy      {S1,S2,S3,S4,S5,S6,S7,S8,S9}

@DATA
-1,4,5,0.0555555556,11,S8
-5,11,16,0.2777777778,11,S2
3,14,11,0.1666666667,11,S2
-13,1,14,0.7222222222,11,S3
15,18,3,0.8333333333,11,S7
5,16,11,0.2777777778,11,S8
5,7,2,0.2777777778,11,S8
-1,11,12,0.0555555556,11,S2
-6,5,11,0.3333333333,11,S6
6,10,4,0.3333333333,11,S5
...
```

TABLE 1. Data set sample with 10 records to train a classifier for value selection

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 [16, 22, 18], 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: these variables capture basic information to formulate a 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 $(i, j \in T)$;
- p_i: initial points of team i. If i has not played any games, then $p_i = 0$;
- 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;
- tp_i: total points of team i at end of the competition;
- 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 (16), (17), and (18) 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. 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. In this scenario, a team can get (3 points win, 1 point tie, 0 points loss) in every game, then the total number of points of team i can be calculated by adding its initial points p_i and the points obtained against every other team. Depending on the competition and the current state of the tournament g_{ij} is set to 0 (no games left between the teams), 1 or 2. Unlike previous work in [18] which is limited to single round-robin competitions, this representation is flexible to represent both single and double round-robin competitions.

$$w_{ij} + t_{ij} + l_{ij} = g_{ij} \quad \forall_{i,j \in T} \wedge g_{ij} \geq 0 \tag{16}$$

$$w_{ij} = l_{ji} \wedge t_{ij} = t_{ji} \wedge l_{ij} = w_{ji} \quad \forall_{i,j \in T} \tag{17}$$

$$tp_i = p_i + \sum_{j=1, j \neq i}^{n} 3 \cdot w_{ij} + t_{ij} \quad \forall_{i,j \in T} \tag{18}$$

Game Result Queries: we use this set of variables to represent user defined assumptions about the remaining games, e.g., Barcelona ends in a tie with R. Madrid. Taking this into account we extend our basic model with the linear equations in (19).

- Q: set of game result queries for pairs of teams (i, j). Q is defined as a set of triples $nq_a = (wc_{ij}, tc_{ij}, lc_{ij})$ and $0 \leq a \leq |Q|$;
- 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;

$$\begin{aligned} (wc_{ij} + tc_{ij} + lc_{ij} \leq g_{ij}) \;\; & \forall nq_a \in Q \wedge nq_a = (wc_{ij}, tc_{ij}, lc_{ij}) \\ (w_{ij} \geq wc_{ij}) \wedge (t_{ij} \geq tc_{ij}) \wedge (l_{ij} \geq lc_{ij}) \;\; & \forall nq_a \in Q \wedge nq_a = (wc_{ij}, tc_{ij}, lc_{ij}) \\ & wc_{ij}, tc_{ij}, lc_{ij} \geq 0 \end{aligned} \tag{19}$$

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 (20) 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 $0 \leq b \leq |P|$;
- opr_i: logical operator ($opr_i \in \{<, \leq, >, \geq, =\}$) to constrain team i;
- ptn_i: denoting the expected position for team i; $1 \leq ptn_i \leq n$;
- 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 $tp_j \geq tp_i$ then $geq_{ij} = 1$; otherwise $geq_{ij} = 0 \quad (\forall i \in L, \forall j \in T)$;
- eq_{ij}: boolean variable indicating if two different teams i and j tie in points at the end of the competition: if $tp_j = tp_i$ and $i \neq j$ then $eq_{ij} = 1$; otherwise $eq_{ij} = 0 \quad (\forall i \in L, \forall j \in T)$.

$$\forall np_b \in P \wedge np_b = (i, opr_i, ptn_i) \begin{cases} pos_i = ptn_i, & \text{if } opr_i \text{ is } = \\ pos_i \leq ptn_i - 1, & \text{if } opr_i \text{ is } < \\ pos_i \leq ptn_i, & \text{if } opr_i \text{ is } \leq \\ pos_i \geq ptn_i + 1, & \text{if } opr_i \text{ is } > \\ pos_i \geq ptn_i, & \text{if } opr_i \text{ is } \geq \end{cases} \tag{20}$$

Constraint (21) indicates the number of teams j that finish the competition with better or equal points as team i. This number (i.e., $worstPos_i$) is the upper bound for pos_i as expressed in constraint (23).

$$\begin{aligned} geq_{ij} &= \begin{cases} 1, & \text{if } tp_j \geq tp_i \\ 0, & otherwise \end{cases} \quad \forall j \in T \wedge \forall i \in L \\ worstPos_i &= \sum_{j=1}^{n} geq_{ij} \quad \forall j \in T \wedge \forall i \in L \end{aligned} \tag{21}$$

Constraint (22) 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 (i.e., $bestPos_i$) can be computed by subtracting from the upper bound, the total teams in the same position as team i.

$$\begin{aligned} eq_{ij} &= \begin{cases} 1, & \text{if } tp_j = tp_i \text{ and } i \neq j \\ 0, & otherwise \end{cases} \quad \forall j \in T \wedge \forall i \in L \\ bestPos_i &= worstPos_i - \sum_{j=1, j\neq i}^{n} eq_{ij} \quad \forall j \in T \wedge \forall i \in L \end{aligned} \tag{22}$$

Finally, constraint (23) sets the bounds for the position of team i and constraint (24) indicates that the positions for two teams must be different.

$$bestPos_i \leq pos_i \leq worstPos_i \quad \forall i \in L \tag{23}$$

$$(pos_i \neq pos_j) \quad \forall i, j \in L \wedge i \neq j \tag{24}$$

Relative Position Queries: we use these variables to represent queries about relative positions between two teams, e.g., Barcelona will be in a better position than R. Madrid. Constraint (25) depicts the five queries. In this particular case we use tp_i and tp_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 (24) 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 $0 \leq c \leq |R|$;
- op_{ij}: denoting a logical operator ($op_{ij} \in \{<, \leq, >, \geq, =\}$) to constrain a pair of teams i and j.

$$\forall nr_c \in R \wedge nr_c = (i, op_{ij}, j) \begin{cases} tp_i = tp_j, & \text{if } op_{ij} \text{ is } = \\ tp_i \leq tp_j - 1, & \text{if } op_{ij} \text{ is } < \\ tp_i \leq tp_j, & \text{if } op_{ij} \text{ is } \leq \\ tp_i \geq tp_j + 1, & \text{if } op_{ij} \text{ is } > \\ tp_i \geq tp_j, & \text{if } op_{ij} \text{ is } \geq \end{cases} \tag{25}$$

Final Point Queries: (also known as score queries) we use these variables for queries about the final points of the teams, e.g., Barcelona scores at the end of the competition 75 points. To guarantee a set S of *final point queries*, the model should include the linear constraints from (26). Notice that the score s_i should have a lower bound of p_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 $0 \leq d \leq |S|$;
- s_i: denoting the wanted final points of team i.

$$\begin{aligned} (tp_i = s_i) \quad & \forall ns_d \in S \wedge ns_d = (i, s_i) \\ p_i \leq s_i \leq p_i + 3 \cdot \sum_{j=1, j\neq i}^{n} g_{ij} \quad & \forall ns_d \in S \wedge ns_d = (i, s_i) \end{aligned} \tag{26}$$

Objective Function: Users might be interested in either minimizing or maximizing the total points for a given team as depicted in (27):

$$\text{maximize: } tp_i \quad (i \in T) \tag{27}$$

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). Both Gecode and G12/FD solvers are included in the distribution 2.0.14 of MiniZinc. Gecode is a C++ constraint solving library [9] 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 [20]. 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 [21]. Mozart implements the Oz language and the version 1.4.0 provides constraint programming support [6]. 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)

Model	Short Name	OZ	CPLEX	Haifa	G12FD	Gecode
CP model	CP	✓		✓	✓	✓
CP model + Redundant Constraints	CP-R			✓	✓	✓
MIP model	MIP		✓	✓	✓	✓

TABLE 2. Solvers and model configurations used in the empirical tests

Model	Short Name	OZ
CP model + Var/Val Heuristics including Machine Learning	CP-ML	✓
Extended CP (CP model + Redundant Constraints + Var/Val Heuristics including Machine Learning)	E-CP	✓

TABLE 3. OZ Solvers and model configurations using Var/Val heuristics

- 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 used the default search configuration for all the solvers and we assumed that MiniZinc models have been converted to FlatZinc. 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 recall that our models are part of SABIO, a Web based application and long answer times are not desirable.

Tables 2 and 3 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 3.

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.

1 Core Experiments																			
Constraint Type		*P* **Queries**						*R* **Queries**						*S* **Queries**					
Fixture 7	OZ CP	24	28	38	46	50	52	0	0	0	0	0	0	0	2	0	0	6	3
	OZ CP-ML	5	9	14	21	27	35	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	1	3	4	5	11	15	0	0	0	0	0	0	0	0	0	0	0	0
Fixture 9	OZ CP	23	28	35	46	49	44	0	0	0	0	0	0	0	0	0	0	1	1
	OZ CP-ML	1	4	9	18	27	31	0	0	0	0	0	0	0	0	0	0	0	1
	OZ E-CP	2	1	4	6	11	15	0	0	0	0	0	0	0	0	0	0	0	1
Fixture 11	OZ CP	25	27	31	44	49	47	0	0	0	0	0	0	0	0	0	0	0	0
	OZ CP-ML	2	7	11	18	31	34	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	2	3	4	8	17	16	0	0	0	0	0	0	0	0	0	0	0	0
Fixture 14	OZ CP	23	33	38	41	51	45	0	0	0	0	0	0	0	0	0	1	0	0
	OZ CP-ML	8	22	25	34	44	42	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	2	2	7	9	14	8	0	0	0	0	0	0	0	0	0	0	0	0
Fixture 16	OZ CP	23	31	29	31	25	13	0	0	0	0	0	0	0	0	0	0	0	0
	OZ CP-ML	19	25	25	33	28	17	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unsolved Instances	OZ CP	1069						0						14					
	OZ CP-ML	626						0						1					
	OZ E-CP	170						0						1					
Fixture 7	OZ CP	0.22	0.48	0.32	0.41	0.77	1.1	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.06	0.07	0.23	0.22
	OZ CP-ML	1.21	1.35	1.84	2.41	3.29	3.09	0.06	0.06	0.06	0.06	0.06	0.05	0.05	0.11	0.05	0.05	0.26	0.2
	OZ E-CP	0.89	0.94	1.05	1.06	0.95	1.19	0.06	0.06	0.07	0.07	0.06	0.06	0.07	0.12	0.06	0.06	0.27	0.22
Fixture 9	OZ CP	0.09	0.42	0.23	0.66	0.32	0.73	0.07	0.07	0.07	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.22
	OZ CP-ML	1.69	1.48	2.26	2.81	2.79	1.67	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.07	0.07
	OZ E-CP	0.81	0.56	1.16	1.16	1.9	0.46	0.07	0.06	0.06	0.06	0.07	0.06	0.06	0.06	0.05	0.05	0.08	0.08
Fixture 11	OZ CP	0.56	0.1	0.76	0.69	0.87	0.45	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.08	0.05	0.05	0.05
	OZ CP-ML	1.53	1.09	2.02	2.68	2.57	2.47	0.06	0.06	0.05	0.06	0.05	0.05	0.05	0.06	0.08	0.06	0.05	0.04
	OZ E-CP	0.4	0.4	0.75	0.64	0.78	0.23	0.06	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.07	0.05	0.05	0.04
Fixture 14	OZ CP	0.42	0.14	0.49	1.02	1.54	0.74	0.05	0.05	0.05	0.05	0.05	0.04	0.05	0.05	0.05	0.05	0.05	0.04
	OZ CP-ML	1.17	1.3	1.46	1.49	1.08	1.48	0.04	0.03	0.03	0.03	0.03	0.03	0.04	0.03	0.03	0.06	0.03	0.03
	OZ E-CP	0.07	0.37	0.11	0.16	0.44	0.54	0.05	0.05	0.04	0.04	0.05	0.04	0.05	0.04	0.04	0.07	0.04	0.04
Fixture 16	OZ CP	0.12	0.05	0.9	0.56	0.46	0.66	0.04	0.04	0.04	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.04
	OZ CP-ML	0.56	0.5	0.93	0.05	0.12	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
	OZ E-CP	0.05	0.04	0.03	0.07	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
Avg. Time	OZ CP	0.51						0.06						0.07					
	OZ CP-ML	1.61						0.05						0.06					
	OZ E-CP	0.57						0.05						0.07					
4 Core Experiments																			
Constraint Type		*P* **Queries**						*R* **Queries**						*S* **Queries**					
Fixture 7	OZ CP-ML	4	8	15	18	26	37	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	1	3	4	5	11	18	0	0	0	0	0	0	0	0	0	0	0	0
Fixture 9	OZ CP-ML	2	4	9	14	23	28	0	0	0	0	0	0	0	0	0	0	0	1
	OZ E-CP	0	1	3	5	10	13	0	0	0	0	0	0	0	0	0	0	0	1
Fixture 11	OZ CP-ML	2	6	11	17	29	32	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	2	3	4	7	16	16	0	0	0	0	0	0	0	0	0	0	0	0
Fixture 14	OZ CP-ML	8	21	24	34	42	44	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	2	2	7	9	13	9	0	0	0	0	0	0	0	0	0	0	0	0
Fixture 16	OZ CP-ML	19	25	26	33	28	17	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unsolved Instances	OZ CP-ML	606						0						1					
	OZ E-CP	165						0						1					
Fixture 7	OZ CP-ML	0.63	0.77	0.55	1.37	2.28	1.35	0.08	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.06	0.07	0.06	0.09
	OZ E-CP	0.35	0.44	0.45	0.28	0.49	0.69	0.08	0.07	0.08	0.07	0.07	0.07	0.08	0.08	0.07	0.07	0.07	0.09
Fixture 9	OZ CP-ML	0.53	0.38	0.54	1.24	2.92	1.9	0.07	0.07	0.07	0.07	0.07	0.06	0.07	0.06	0.06	0.06	0.05	0.05
	OZ E-CP	0.43	0.25	0.55	0.34	1.03	0.82	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.06	0.06	0.06	0.06	0.06
Fixture 11	OZ CP-ML	0.38	0.45	0.92	1.0	1.46	1.6	0.07	0.06	0.06	0.06	0.06	0.05	0.06	0.06	0.05	0.05	0.05	0.05
	OZ E-CP	0.16	0.15	0.31	0.35	0.69	0.32	0.07	0.06	0.06	0.06	0.07	0.06	0.06	0.06	0.06	0.05	0.05	0.05
Fixture 14	OZ CP-ML	0.38	0.88	0.78	0.68	0.54	1.23	0.05	0.04	0.05	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.04
	OZ E-CP	0.05	0.21	0.06	0.08	0.35	0.51	0.05	0.05	0.04	0.04	0.04	0.04	0.05	0.05	0.04	0.04	0.04	0.04
Fixture 16	OZ CP-ML	0.19	0.18	0.28	0.05	0.22	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
	OZ E-CP	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
Avg. Time	OZ CP-ML	0.83						0.06						0.05					
	OZ E-CP	0.31						0.06						0.05					

TABLE 4. Unsolved 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

6.2. **Tests Results.** Given that there are several ways to group and compare our models, in numeral 6.2.1 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, and E-CP) implemented under Mozart-OZ. Later, in numeral 6.2.2, we study three models tested with MiniZinc solvers and CPLEX (i.e., CP, CP-R, MIP). Finally, in numeral 6.2.3 we compare our best solver-model configurations

1 Core Experiments																			
Constraint Type		*P* **Queries**						*R* **Queries**						*S* **Queries**					
Fixture 5	OZ CP	31	37	44	52	68	67	0	0	0	0	0	0	0	0	0	0	0	7
	OZ CP-ML	11	16	24	28	47	49	0	0	0	0	0	0	0	0	0	0	0	1
	OZ E-CP	9	14	15	21	36	33	0	0	0	0	0	0	0	0	0	0	0	1
Fixture 10	OZ CP	28	37	45	56	69	67	0	0	0	0	0	0	0	0	0	1	4	3
	OZ CP-ML	6	12	18	27	42	46	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	5	9	10	19	27	30	0	0	0	0	0	0	0	0	0	0	0	0
Fixture 15	OZ CP	29	35	46	57	65	73	0	0	0	0	0	0	0	0	1	0	2	5
	OZ CP-ML	5	11	14	24	31	40	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	3	3	6	12	20	24	0	0	0	0	0	0	0	0	0	0	0	0
Fixture 20	OZ CP	33	36	43	65	67	63	0	0	0	0	0	0	0	0	0	0	1	3
	OZ CP-ML	8	14	17	30	41	47	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	3	1	9	10	30	33	0	0	0	0	0	0	0	0	0	0	0	0
Fixture 25	OZ CP	29	39	35	38	37	28	0	0	0	0	0	0	0	0	0	0	1	0
	OZ CP-ML	23	27	28	36	34	26	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	3	1	1	2	4	3	0	0	0	0	0	0	0	0	0	0	0	0
Unsolved Instances	OZ CP	1419						0						28					
	OZ CP-ML	782						0						1					
	OZ E-CP	396						0						1					
Fixture 5	OZ CP	0.23	0.46	0.21	0.23	0.31	0.89	0.56	0.53	0.51	0.5	0.5	0.49	0.43	0.44	0.4	0.39	0.38	0.34
	OZ CP-ML	2.1	2.98	2.72	3.38	3.98	3.82	0.28	0.27	0.27	0.28	0.3	0.34	0.32	0.31	0.3	0.28	0.26	0.44
	OZ E-CP	1.82	1.88	2.54	2.26	2.79	3.84	0.49	0.49	0.49	0.49	0.47	0.46	0.44	0.41	0.41	0.38	0.37	0.51
Fixture 10	OZ CP	0.33	0.2	0.24	0.35	0.18	0.41	0.37	0.39	0.39	0.39	0.38	0.37	0.34	0.34	0.32	0.32	0.28	0.25
	OZ CP-ML	1.69	2.35	2.55	3.3	3.4	3.85	0.27	0.28	0.29	0.29	0.28	0.27	0.25	0.24	0.23	0.26	0.45	0.29
	OZ E-CP	1.35	1.51	2.26	2.17	2.69	2.4	0.38	0.37	0.38	0.36	0.37	0.34	0.32	0.32	0.3	0.32	0.51	0.33
Fixture 15	OZ CP	0.48	0.2	0.14	0.56	0.43	0.14	0.29	0.29	0.33	0.32	0.29	0.31	0.27	0.28	0.26	0.23	0.19	0.21
	OZ CP-ML	1.59	1.79	2.85	3.38	4.36	5.03	0.21	0.21	0.22	0.22	0.21	0.2	0.19	0.19	0.23	0.17	0.23	0.4
	OZ E-CP	1.27	1.26	2.04	2.39	2.9	3.12	0.28	0.28	0.28	0.28	0.27	0.25	0.23	0.2	0.19	0.15	0.19	0.38
Fixture 20	OZ CP	0.12	0.71	0.26	0.7	0.12	0.24	0.19	0.21	0.22	0.22	0.2	0.18	0.19	0.18	0.18	0.16	0.14	0.12
	OZ CP-ML	1.49	1.7	2.43	4.33	3.05	2.81	0.15	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.14	0.13	0.14	0.19
	OZ E-CP	0.8	1.28	1.41	2.33	1.95	1.49	0.14	0.14	0.14	0.13	0.12	0.12	0.11	0.11	0.12	0.11	0.13	0.18
Fixture 25	OZ CP	0.07	0.22	0.42	0.13	0.12	0.07	0.09	0.08	0.08	0.07	0.06	0.06	0.1	0.1	0.1	0.09	0.07	0.08
	OZ CP-ML	0.59	1.3	0.68	0.24	0.28	0.18	0.06	0.05	0.05	0.05	0.04	0.04	0.07	0.07	0.06	0.06	0.08	0.05
	OZ E-CP	0.17	0.06	0.21	0.21	0.44	0.09	0.09	0.08	0.08	0.07	0.06	0.06	0.09	0.09	0.09	0.08	0.11	0.07
Avg. Time	OZ CP	0.29						0.3						0.24					
	OZ CP-ML	2.38						0.19						0.21					
	OZ E-CP	1.6						0.26						0.24					
4 Core Experiments																			
Constraint Type		*P* **Queries**						*R* **Queries**						*S* **Queries**					
Fixture 5	OZ CP-ML	10	17	21	26	48	49	0	0	0	0	0	0	0	0	0	0	0	1
	OZ E-CP	9	15	16	18	37	32	0	0	0	0	0	0	0	0	0	0	0	1
Fixture 10	OZ CP-ML	6	12	17	26	38	47	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	4	8	11	19	25	28	0	0	0	0	0	0	0	0	0	0	0	0
Fixture 15	OZ CP-ML	5	10	12	19	33	38	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	4	3	5	11	21	26	0	0	0	0	0	0	0	0	0	0	0	0
Fixture 20	OZ CP-ML	8	16	17	30	40	44	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	3	3	9	10	30	32	0	0	0	0	0	0	0	0	0	0	0	0
Fixture 25	OZ CP-ML	22	27	28	36	34	26	0	0	0	0	0	0	0	0	0	0	0	0
	OZ E-CP	3	1	1	2	4	3	0	0	0	0	0	0	0	0	0	0	0	0
Unsolved Instances	OZ CP-ML	762						0						1					
	OZ E-CP	393						0						1					
Fixture 5	OZ CP-ML	1.02	1.22	1.35	2.15	1.72	1.91	0.46	0.45	0.47	0.45	0.45	0.44	0.4	0.4	0.39	0.37	0.36	0.32
	OZ E-CP	0.78	0.74	1.41	2.01	1.63	2.13	0.45	0.44	0.45	0.46	0.44	0.43	0.39	0.39	0.39	0.36	0.35	0.32
Fixture 10	OZ CP-ML	0.86	0.88	0.91	1.72	2.37	1.69	0.36	0.36	0.36	0.35	0.34	0.33	0.32	0.31	0.3	0.29	0.33	0.25
	OZ E-CP	0.95	0.62	0.77	0.69	0.92	1.2	0.3	0.32	0.34	0.34	0.34	0.33	0.33	0.31	0.29	0.29	0.31	0.25
Fixture 15	OZ CP-ML	0.57	0.7	1.42	2.18	1.58	3.19	0.28	0.27	0.27	0.27	0.26	0.25	0.23	0.23	0.23	0.21	0.2	0.23
	OZ E-CP	0.32	0.42	1.07	0.9	1.23	1.32	0.27	0.27	0.27	0.27	0.26	0.25	0.23	0.24	0.23	0.21	0.2	0.22
Fixture 20	OZ CP-ML	0.45	0.46	0.81	2.21	0.69	2.43	0.19	0.18	0.18	0.18	0.17	0.15	0.17	0.16	0.16	0.16	0.14	0.11
	OZ E-CP	0.25	0.62	0.54	1.21	0.82	1.59	0.18	0.17	0.17	0.17	0.15	0.15	0.15	0.14	0.15	0.14	0.12	0.11
Fixture 25	OZ CP-ML	0.49	0.51	0.25	0.16	0.32	0.13	0.08	0.07	0.07	0.07	0.05	0.05	0.08	0.09	0.08	0.08	0.08	0.07
	OZ E-CP	0.08	0.07	0.19	0.07	0.08	0.08	0.07	0.06	0.07	0.06	0.05	0.05	0.08	0.08	0.08	0.07	0.07	0.06
Avg. Time	OZ CP-ML	1.15						0.26						0.22					
	OZ E-CP	0.77						0.25						0.22					

TABLE 5. 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

according to the number of solved instances and average execution time in order to propose mixed strategies to improve execution performance.

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 4 (resp. Table 5 for the Argentine league). 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., 30s). Alternatively, P queries are the hardest ones, particularly for our basic CP approach.

For the Colombian league (Table 4), 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 (tp_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.

We also observe in Table 4 that our *CP-ML* implementation seems to perform better and the classifier improves the effectiveness of the algorithm by reducing the number of unsolved instances from 1069 to 626 while displaying a small trade-off in running time (it goes from 0.51s to 1.61s). Additionally, our extended model *E-CP* managed to reduced 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 redundant constraints introduced in this extended model reduce the time trade-off introduced by the use of the machine learning approach. Finally, we also observe a small improvement in the amount of solved instances and average runtimes by using the parallel execution with respect to the sequential execution.

We now move our attention to the Argentine league in Table 5 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 numeral we present evaluation results for three models (CP, CP-R, and MIP) tested using MiniZinc solvers and CPLEX. Our results are summarized in Tables 6 and 7 for the Colombian and the Argentine leagues using sequential and parallel executions.

Colombian League - 1 Core Test							
Model	**Solver**	**Solved Instances**	**Unsolved Instances**	**Avg T. Solved Instances**	**Std T. Solved Instances**	**Solved SAT Instances**	**Solved UNSAT Instances**
CP	G12fd	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	G12fd	8475	525	0.79	2.25	5701	2774
	Gecode	8199	801	0.82	2.56	5664	2535
	HaifaCSP	8997	3	0.15	0.75	6220	2777
MIP	CPLEX	8995	5	0.45	0.29	6216	2779
	G12fd	8468	532	1.36	3.4	5769	2699
	Gecode	8198	802	0.77	1.93	5499	2699
	HaifaCSP	8916	84	0.61	2.07	6150	2766
Colombian League - 4 Core Test							
CP	Gecode	8427	573	0.49	1.99	5704	2723
	HaifaCSP	8961	39	0.45	1.7	6202	2759
CP-R	HaifaCSP	8994	6	0.16	0.34	6220	2774
MIP	CPLEX	8999	1	0.47	0.28	6220	2779
	HaifaCSP	8960	40	0.54	1.82	6207	2753

TABLE 6. Test results using MiniZinc solvers and CPLEX for the Colombian League with 18 teams

Argentine League - 1 Core Test							
Model	**Solver**	**Solved Instances**	**Unsolved Instances**	**Avg T. Solved Instances**	**Std T. Solved Instances**	**Solved SAT Instances**	**Solved UNSAT Instances**
CP	G12fd	7635	1365	2.14	2.35	6166	1469
	Gecode	7406	1594	1.5	1.72	5973	1433
	HaifaCSP	8363	637	1.07	3.23	6821	1542
CP-R	G12fd	7389	1611	1.8	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
	G12fd	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
Argentine League - 4 Core Test							
CP	Gecode	7537	1463	0.93	1.77	6101	1436
	HaifaCSP	8541	459	1.69	3.96	7017	1524
CP-R	HaifaCSP	8912	88	1.36	2.98	7333	1579
MIP	CPLEX	8921	79	1.12	2.19	7338	1583
	HaifaCSP	8491	509	2.56	4.38	6988	1503

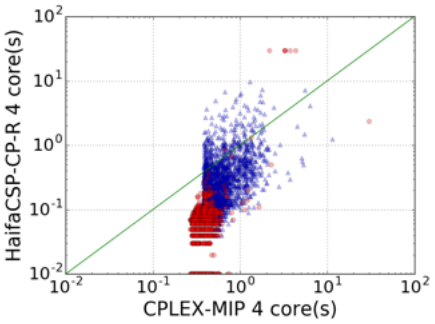
TABLE 7. Test results using MiniZinc solvers and CPLEX for the Argentine League with 30 teams

For both leagues we can 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 using a single core (Table 6 top section), HaifaCSP solves 8949 instances with the basic *CP* model and outperforms G12fd and Gecode by 357 and 649 solved instances, as well as in their average runtime. We also observe that our redundant constraints improve the search performance of HaifaCSP by reducing the number of unsolved instances from 51 to 3. This

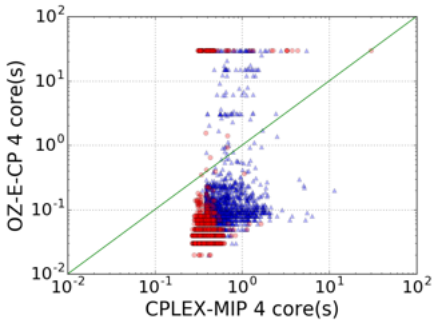
improvement also has an impact in the average runtime which improves from 0.36s to 0.15s. Interestingly, the same behaviour can be observed in the Argentine league (Table 7 top section).

For the Argentine league experimentations (Table 7 bottom section), we can observe that the parallel executions slightly improve the sequential search performance for HaifaCSP and go 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 6 bottom section) where the solver improves from 51 to 39 unsolved instances, but the CP-R model does not show an improvement and goes from 3 to 6 unsolved instances .

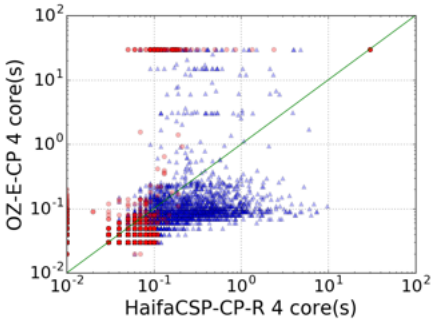
In our experimentations, CPLEX outperforms the rest of the solvers using the MIP model. For instance, in the Argentine league using a sequential execution, CPLEX solves 121 more instances than the best execution of HaifaCSP which was possible due to the CP-R model. CPLEX is the solver with the smallest 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).



(a) HaifaCSP(CP-R) vs. CPLEX(MIP)



(b) OZ(E-CP) vs. CPLEX(MIP)

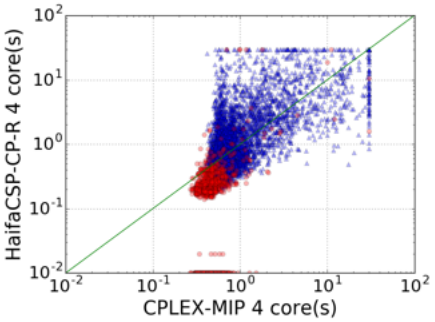


(c) OZ(E-CP) vs. HaifaCSP(CP-R)

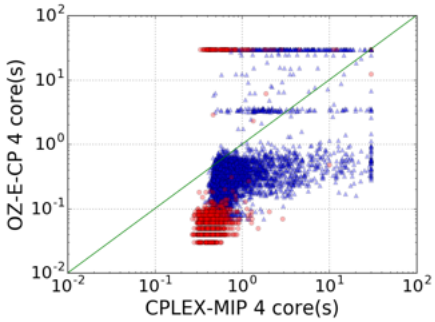
FIGURE 1. Colombian league runtime comparison (time in seconds for parallel executions)

6.2.3. *Solver-Model Comparisons and Mixed Approaches.* In this numeral we compare our best solver-model configurations and propose mixed execution approaches

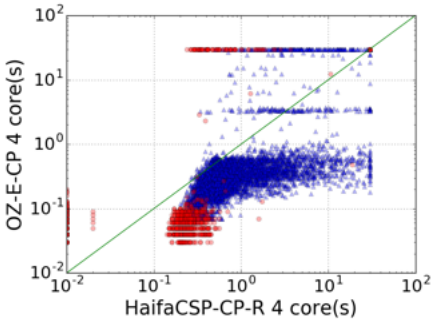
in order to improve performance (i.e., solved instances and average execution time). In general, the best performance in our experiments were displayed by HaifaCSP using the *CP-R* model, CPLEX with the *MIP* model, and OZ with the *E-CP* model.



(a) HaifaCSP(CP-R) vs. CPLEX(MIP)



(b) OZ(E-CP) vs. CPLEX(MIP)



(c) OZ(E-CP) vs. HaifaCSP(CP-R)

FIGURE 2. Argentine league runtime comparison (time in seconds for parallel executions)

We start with Figures 1 and 2 where we compare the performance of our best solver-model configurations for both leagues. For every figure, the x and y axis represent the runtime of two models in log scale and blue (resp. red) points indicate SAT (resp. UNSAT). Points below the diagonal indicate that the model in the y axis is faster, respectively for the models in the x axis with points above the diagonal.

When comparing HaifaCSP and CPLEX for the Colombian league in Figure 1(a), we observe that HaifaCSP is typically faster than CPLEX. Moreover, HaifaCSP seems to be more efficient than CPLEX for UNSAT instances in both competitions as depicted in Figures 1(a) and 2(a).

Interestingly, when comparing OZ with both HaifaCSP and CPLEX in Figures 1(b) and 1(c) for the Colombian league (resp. 2(b) and 2(c) for the Argentine league), it can be observed that OZ is considerably faster than both solvers despite the fact that HaifaCSP and CPLEX are typically better with respect to capacity solving. Such characteristic can be observed in the big accumulation of solved instances below the diagonal for OZ, as well as the accumulations of timeouts at the top of the before mentioned figures.

We created Tables 8 and 9 in order to analyse the previous behaviour in more detail. Each table contains the comparison of common solved instances for two solver-model configurations. We can observe in Table 8 that OZ is faster than CPLEX in 8745 and 8320 common solved instances for the Colombian and the Argentine leagues respectively, while 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 tests. However, CPLEX is more effective than OZ and leaves 1 and 79 unsolved instances (Tables 6 and 7) while OZ displays 165 and 393 unsolved instances for both leagues in the parallel executions (Tables 4 and 5).

Additionally, in Table 9 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 tests. We also observe that HaifaCSP is more effective than OZ since it displays 6 and 88 unsolved instances (Tables 6 and 7) compared to the 165 and 393 unsolved instances for both leagues in the parallel executions with the OZ solver (Tables 4 and 5).

Colombian League CPLEX(MIP) 4 cores vs OZ(E-CP) 4 cores				
Model	**Solver**	**Faster in**	**SAT**	**UNSAT**
MIP	CPLEX	89	83	6
E-CP	OZ	8745	6091	2654
Argentine League CPLEX(MIP) 4 cores vs OZ(E-CP) 4 cores				
Model	**Solver**	**Faster in**	**SAT**	**UNSAT**
MIP	CPLEX	230	226	3
E-CP	OZ	8320	6837	1481

TABLE 8. CPLEX(MIP) vs. OZ(E-CP) common solved instances comparison

Colombian League HaifaCSP(CP-R) 4 cores vs OZ(E-CP) 4 cores				
Model	**Solver**	**Faster in**	**SAT**	**UNSAT**
CP-R	HaifaCSP	1171	399	772
E-CP	OZ	7223	5500	1723
Argentine League HaifaCSP(CP-R) 4 cores vs OZ(E-CP) 4 cores				
Model	**Solver**	**Faster in**	**SAT**	**UNSAT**
CP-R	HaifaCSP	599	231	368
E-CP	OZ	7927	6809	1116

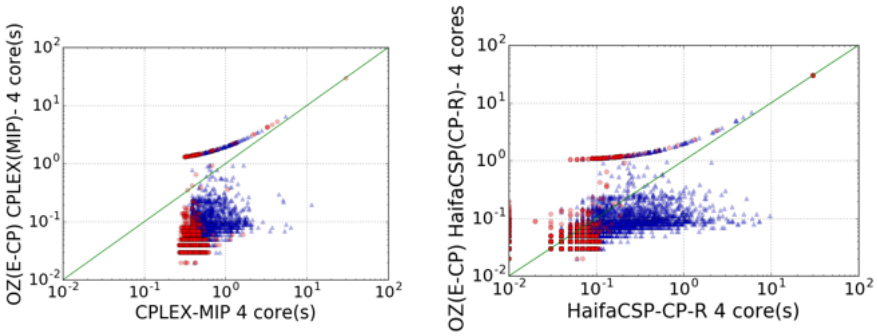
TABLE 9. HaifaCSP(CP-R) vs. OZ(E-CP) common solved instances comparison

We then managed to observe that these solvers present a complementary behaviour. That is, CPLEX and HaifaCSP are typically more effective than OZ whilst OZ is typically faster. Therefore, to exploit the best features of the solvers, we created two mixed executions:

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

As depicted in Figures 1 and 2, most of the instances where OZ is faster than HaifaCSP and CPLEX are solved within the first second. Therefore, we decided

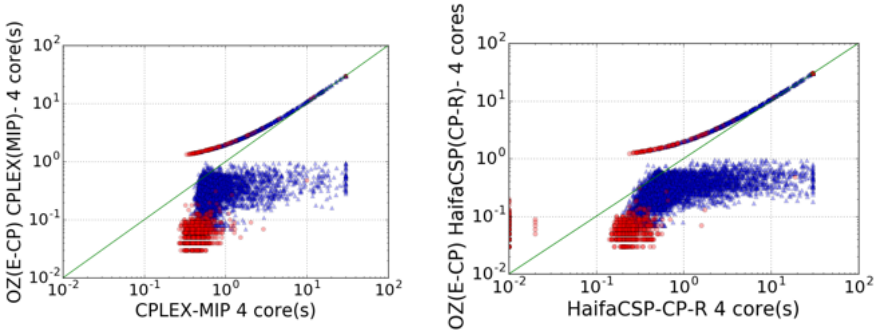
to run first our OZ(E-CP) solver during 1 second to solve as many instances as possible, then run our second solver-model implementation for the remaining 29 seconds.



(a) OZ(E-CP) and CPLEX(MIP) mixed execution vs. CPLEX(MIP)

(b) OZ(E-CP) and HaifaCSP(CP-R) mixed execution vs. HaifaCSP(CP-R)

FIGURE 3. Colombian league runtime comparison for mixed solver executions (time in seconds for parallel executions)



(a) OZ(E-CP) and CPLEX(MIP) mixed execution vs. CPLEX(MIP)

(b) OZ(E-CP) and HaifaCSP(CP-R) mixed execution vs. HaifaCSP(CP-R)

FIGURE 4. Argentine league runtime comparison for mixed solver executions (time in seconds for parallel executions)

In Figures 3(a) and 4(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 3(b) and 4(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 10. This approach outperforms the parallel OZ(E-CP)

execution depicted in Table 4 by 164 solved instances. Despite the fact that our approach does not improve the number of solved instances presented by the parallel execution of CPLEX(MIP) in Table 6, it does reduce the average time for solved instances from 0.47 to 0.11.

Colombian League - 4 Core Mixed Solvers Tests							
Model	**Solver**	**Solved Instances**	**Unsolved Instances**	**Avg T. Solved Instances**	**Std T. Solved Instances**	**Solved SAT Instances**	**Solved UNSAT Instances**
E-CP & MIP	OZ & CPLEX	8999	1	0.11	0.3	6220	2779
E-CP & CP-R	OZ & HaifaCSP	8994	6	0.1	0.25	6220	2774

TABLE 10. Test results using MiniZinc solvers and CPLEX for the Colombian League with 18 teams

Argentine League - 4 Core Mixed Solvers Tests							
Model	**Solver**	**Solved Instances**	**Unsolved Instances**	**Avg T. Solved Instances**	**Std T. Solved Instances**	**Solved SAT Instances**	**Solved UNSAT Instances**
E-CP& MIP	OZ& CPLEX	8958	42	0.56	1.74	7376	1582
E-CP& CP-R	OZ& HaifaCSP	8940	60	0.62	2.05	7361	1579

TABLE 11. Test results using MiniZinc solvers and CPLEX for the Argentine League with 30 teams

Our experiment 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 11. That is, 351 and 37 more instances than the parallel OZ(E-CP) execution depicted in Table 5 and the parallel execution of CPLEX(MIP) in Table 7. 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 speed and capacity solving.

7. **Conclusions.** 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 (CP) and a Mixed Integer Programming (MIP) formulation. We presented a series of improvements for the CP model that included redundant constraints and heuristics for variable/-value selection that implemented a machine learning and a restart approach. These improvements let us create three extensions of the CP model (i.e., CP-R, CP-ML, and E-CP).

We also created 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 using five reference solves including Mozart OZ, CPLEX, and 3 MiniZinc solvers (i.e., Gecode, G12fd, 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 considerably 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.

Acknowledgements. 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 Colciencias, the Colombian Administrative Department of Science, Technology and Innovation under the PhD scholarship program. Alejandro Arbelaez is supported by the DISCUS (FP7 Grant Agreement 318137), and Science Foundation Ireland (SF) Grant No. 10/CE/I1853. The Insight Centre for Data Analytics is also supported by SFI under Grant Number SFI/12/RC/2289.

References

[1] I. Adler, A. L. Erera, D. S. Hochbaum, and E. V. Olinick. Baseball, optimization, and the world wide web. *Interfaces*, 32(2):12–22, 2002.

[2] A. Arbelaez and Y. Hamadi. Exploiting weak dependencies in tree-based search. In *SAC'09*, pages 1385–1391, 2009.

[3] T. Bernholt, A. Gälich, T. Hofmeister, and N. Schmitt. Football elimination is hard to decide under the 3-point-rule. In *MFCS*, pages 410–418, 1999.

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

[5] J. Christensen, A. N. Knudsen, and K. S. Larsen. Soccer is harder than football. *International Journal of Foundations of Computer Science*, 26(04):477–486, 2015.

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

[7] R. Duque, J. F. Díaz, and A. Arbelaez. Constraint programming and machine learning for interactive soccer analysis. In *Learning and Intelligent Optimization*. Springer International Publishing AG, 2016.

[8] R. Duque, J. F. Díaz, and A. Arbelaez. Sabio: An implementation of mip and cp for interactive soccer queries, lncs 9892. In M. Rueher, editor, *Principles and Practice of Constraint Programming: 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, pages 575–583, Cham, 2016. Springer International Publishing.

[9] Gecode Team. Gecode: Generic constraint development environment, 2016. Available from `http://www.gecode.org`.

[10] J. C. Guedes and F. S. Machado. Changing rewards in contests: Has the three-point rule brought more offense to soccer? *Empirical Economics*, 27(4):607–630, 2002.

[11] D. Gusfield and C. Martel. The structure and complexity of sports elimination numbers. *Algorithmica*, 32(1):73–86, 2002.

[12] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. IJCAI'79, pages 356–364, San Francisco, CA, USA, 1979.

[13] A. Hoffman and T. Rivlin. When is a team mathematically eliminated? pages 391–401, Princeton, NJ, 1967. Princeton Symposium on Mathematical Programming.
[14] W. Kern and D. Paulusma. The new fifa rules are hard: complexity aspects of sports competitions. *Discrete Applied Mathematics*, 108(3):317–323, 2001.
[15] W. Kern and D. Paulusma. The computational complexity of the elimination problem in generalized sports competitions. *Discrete Optimization*, 1(2):205–214, 2004.
[16] S. T. McCormick. Fast algorithms for parametric scheduling come from extensions to parametric maximum flow. *Operations Research*, 47(5):744–756, 1999.
[17] D. Pálvölgyi. Deciding soccer scores and partial orientations of graphs. *Acta Univ. Sapientiae, Math*, 1(1):35–42, 2009.
[18] C. C. Ribeiro and S. Urrutia. An application of integer programming to playoff elimination in football championships. *ITOR*, 12(4):375–386, 2005.
[19] B. L. Schwartz. Possible winners in partially completed tournaments. *SIAM Review*, 8(3):302–308, 1966.
[20] 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*, pages 9–13. Springer, 2005.
[21] M. Veksler. Haifacsp - constraints-satisfaction problem (csp) solver, 2016. Available from `http://strichman.net.technion.ac.il/haifacsp/`.
[22] K. D. Wayne. A new property and a faster algorithm for baseball elimination. *SIAM Journal on Discrete Mathematics*, 14(2):223–229, 2001.
[23] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, third edition, 2005.

Received xxxx 20xx; revised xxxx 20xx.

E-mail address: robinson.duque@correounivalle.edu.co
E-mail address: alejandro.arbelaez@insight-centre.org
E-mail address: juanfco.diaz@correounivalle.edu.co