This is a post-peer-review, pre-copyedit version
CP 2016: Principles and Practice of Constraint Programming
doi: 10.1007/978-3-319-44953-1_36

SABIO: An Implementation of MIP and CP for Interactive Soccer Queries

Robinson Duque[1], Juan Francisco Díaz[1], and Alejandro Arbelaez[2]

[1] Universidad del Valle, Cali, Colombia
{robinson.duque, juanfco.diaz}@correounivalle.edu.co
[2] Insight Centre for Data Analytics, University College Cork, Ireland
{alejandro.arbelaez}@insight-centre.org

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. 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

A soccer competition (league or tournament) consists of n teams playing against each other in a single or double round-robin schedule. 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 the first team in the standing table becomes the champion.

The elimination problem is well-known in sports competitions [1,2] and consists in determining whether at some stage of the competition a given team still has the opportunity to be within the top teams to qualify for playoffs or become the champion. This problem was proved NP-Complete for the current FiFA score system (0 points-loss, 1 point-tie, 3 points-win) [3,4]. However, in [4] the authors pointed out that with the old FIFA score system (0, 1, 2) from the 90's, the elimination problem could be solved in polynomial time using network flow algorithms as first proposed in [5]. Kern and Paulusma [3,6] generalized NP-completeness depending on the sports' score system and showed that questions

like "is there a chance that team i ends up being one of the three teams that have the three lowest final scores?", is actually an NP-complete problem. Later, Pálvölgyi showed that deciding whether a given score vector is a possible outcome of a soccer-tournament (i.e., *score vector problem*) is also NP-complete [7].

Recently, [8] 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 [8], 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.

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. In this paper we considerably improve an existing CP model [9], and we combine the use of MIP and CP to improve the performance of SABIO.

2 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 [9], we described a CP model for position in ranking queries. Here we extend such model by introducing three new type of queries, i.e., *game results*, *relative position*, *final points*. Additionally, we also propose a set of redundant constraints that help to prune the search tree for position ranking queries.
Basic Soccer Model: 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 $p_i = 0$;
- 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 three of 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; $1 \leq ptn_i \leq n$;

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

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 three of 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.

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

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.

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

3 Extended CP Model

In our 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:

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

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

Variable/Value Selection: In SABIO the variable/value selection strategies involve identifying the outcome of a game for a selected team. Generic heuristics (e.g., [10,11]) typically do not perform well in this domain as they do not exploit the structure of the problem. Therefore, in [9] we proposed a variable selection system of priorities for *position ranking* queries. We use nine strategies $\{S1 \ldots \text{S9}\}$ for the outcome of the game in the value selection process. Each strategy represents the probability of a win, tie, or lose. For instance, $S9 = \langle 0.25, 0.25, 0.5 \rangle$, indicates that the selected team wins or ties the game with a probability of 0.25 each and loses with a probability of 0.5. We use decision trees to select the most suitable strategy for teams constrained with the equal operator.

For *relative position queries* we use an alternative approach. Let us assume we want to answer a *greater than* query for two teams (i.e., $tp_i > tp_j$). In this

case it is natural to get team i to *win* and team j to *lose* the remaining games. Therefore, we increase the priority of both teams to be selected during branching (similarly for less than queries). Alternatively, for queries indicating ($tp_i = tp_j$) we found that both teams have to be assigned a high priority and also that a (win, lose) strategy may generate final points overshooting, therefore, we decided to assign both teams a *tie* strategy.

For *final point queries* we assign a high priority to teams involved in at least one query. Finally, *game result queries* can be trivially solved with our models by only using constraint propagation.

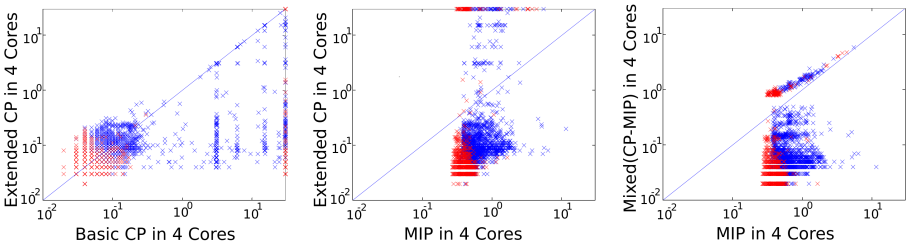
Sequential and Parallel Restart-Based Search: Inspired by the quickest first principle [12], we execute the strategies in a predefined order and we use a restart-based search with a fixed time cutoff. For teams involved in at least one query we use the above mentioned variable/value selection heuristics 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.

4 Empirical Evaluation

Tests Configuration: We evaluated our models using CPLEX (V12.6.2) as our MIP reference solver and Mozart-Oz (V 1.4.0) as our CP reference solver. All the experiments were performed in a 4-core machine featuring an Intel Core i5 processor at 2.3 Ghz and 4GB of RAM. In particular we focus our attention in the 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). We provided five experimental scenarios by exploring different stages of the competition (i.e., fixtures 7, 9, 11, 14, and 16). For each fixture we created instances with position in ranking queries (P), relative position queries (R), and final point queries (S). We excluded game results queries (Q) from our experiments as they can be trivially solved with our models.

The scenarios for every query type (P, R, S) and fixture (7, 9, 11, 14, 16) included 100 instances with 2 suppositions, 100 with 3 suppositions, and the same for 4, 5, 7, and 9 suppositions. For each instance (9000 in total) we used a time limit of 30 seconds. We recall that our models are implemented in SABIO, a Web based application and long answer times are not desirable. In order to analyze the performance of the models, we reported executions with 1 and 4 cores and experimented four scenarios separately: the *basic CP model* proposed in [9]; the *extended CP model* with redundant constraints proposed in this paper; an extended version of the *MIP model* of [8] able to deal with the same queries as our CP model; and a *mixed* execution of our CP and MIP models (i.e., we



(a) Ext. CP vs. Basic CP (b) Ext. CP vs. MIP (c) Mixed CP-MIP vs. MIP

Fig. 1. Runtime (in secs) comparison of the different approaches

observed in our experiments that MIP is slightly better for a small number of instances (mainly unsat). Therefore, to exploit the best features of both CP and MIP, we run first our CP model during 0.5 secs (average time in 1 core) to solve as many instances as possible, then run our MIP implementation for 29.5 secs).[3]

Tests Results: We start with Figure 1 where we compare the performance of extended model with the redundant constraints against the basic model from [9], the MIP model, and the mixed approach CP-MIP. The x-axis gives the runtime of the extended model and the y-axis gives the runtime of the basic model (resp. MIP model). Blue (resp. red) points indicate SAT (resp. UNSAT). Points below the diagonal indicate that the extend model is faster. In Figure 1(a) it can be observed that the extended version is typically better (w.r.t. speed and capacity solving) than the basic model, except for a few instances where the runtime for both approaches is less than 0.1 secs. Figure 1(b) shows that the extended model is considerably faster than the MIP model on 8744 instances. Interestingly MIP is particularly better than CP for UNSAT instances, in fact, 72% of the unsolved instances for the CP model are UNSAT. We attribute this to the fact that for UNSAT instances it is necessary to explore the complete search tree. In Figure 1(c) we exploit the complementary behaviour of MIP and our extended CP model, here we observe that only for 243 (out of 9000) instances it would have been better to alternate the execution of MIP and CP.

We now move our attention to Table 1 where we present complete statistics of the four approaches. This table shows the number of unsolved instances (top), the standard deviation and the average run-times (bottom) for solved instances. Certainly, R and S queries are the easiest ones and nearly all instances can be solved within the time limit. However, we observed that for these instances the extended CP model is faster than MIP. Alternatively, P queries are the hardest ones, particularly for our CP approaches.

We would like to highlight the importance of the redundant constraints in the extended CP model. The new constraints help to solve 456 more instances than

[3] **Electronic Supplementary Materials** The online version of this chapter contains supplementary material of the MIP model, which is available to authorized users.

Num. Cores	1 Core			4 Cores		
Query Type	P **Queries**	R **Queries**	S **Queries**	P **Queries**	R **Queries**	S **Queries**
Unsolved B. CP	627	-	1	607	-	1
Unsolved E. CP	171	-	1	166	-	1
Unsolved MIP	5	-	-	1	-	-
Uns. mixed(CP-MIP)	2	-	-	1	-	-
Avg/std (B. CP)	1.60/3.97	0.05/0.02	0.06/0.24	0.83/2.97	0.05/0.02	0.05/0.04
Avg/std (E. CP)	0.55/2.08	0.05/0.02	0.06/0.24	0.31/1.66	0.05/0.02	0.05/0.05
Avg/std (MIP)	0.54/0.47	0.41/0.09	0.40/0.09	0.58/0.46	0.40/0.06	0.41/0.07
Avg/std mixed(CP-MIP)	**0.19**/0.58	0.05/0.01	0.06/0.03	**0.16**/0.39	**0.04**/0.02	**0.04**/0.03

Table 1. Unsolved instances and run-times (avg, std) in seconds

the basic model from [9] and it also considerably improves the average runtime from 0.83 to 0.31 secs for 4 cores.

Additionally, we observe that the MIP and the extended CP approach are complementary. The MIP model in a 4-core execution, is able to solve more instances, i.e., 607 more than the basic CP approach and 166 more than the extended CP with redundant constraints. The extended CP approach is about 46%, 87%, and 88% faster than the MIP approach for P, R, and S queries using 4 cores. Interestingly, for P queries we observe that extended CP in 1 and 4 cores present a less uniform behaviour (std: 2.08 & 1.66) compared to MIP (std: 0.47 & 0.46). We attribute this to answers found in later restarts. Such behaviour can also be observed in Figure 1(b), where most of MIP instances range between 0.1 and 10 secs with observed min. (resp. max.) runtime values of 0.28 (resp. 1.46) secs, while extended CP ranges between 0.01 and 100 with min. (resp. max.) values of 0.03 (resp. 25.38) secs.

Finally, the overall best approach is the mixed solution between CP and MIP, this solution exploits the best features of both alternatives. We would like to remark that SABIO has been highlighted as an outstanding platform for soccer fans in Colombian's main news paper such as *El Tiempo* and *ADN Cali*. We are planning a new release of SABIO with the models described in this paper.

5 Conclusions

In this paper we have improved an existing CP model to solve general soccer fan queries at different stages of the computation. We compared our improved CP approach against a MIP formulation and observed a complementary behaviour between the two approaches. The CP approach is considerably faster than the MIP model. However, the MIP model is able to solve more instances than the CP one. 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 for a fast and robust solution.

We would like to thank Luis F. Vargas, María A. Cruz and Carlos Martínez for developing early versions of the CP model under the supervision of Juan F. Díaz. Robinson Duque is supported by Colciencias under the PhD scholarship program. Alejandro Arbelaez is supported by SFI Grant No. 10/CE/I1853.

References

1. Schwartz, B.L.: Possible winners in partially completed tournaments. SIAM Review **8**(3) (1966) 302–308
2. Hoffman, A., Rivlin, T.: When is a team "mathematically" eliminated?, Princeton, NJ, Princeton Symposium on Mathematical Programming (1967) 391–401
3. Kern, W., Paulusma, D.: The new fifa rules are hard: complexity aspects of sports competitions. Discrete Applied Mathematics **108**(3) (2001) 317–323
4. Bernholt, T., Gälich, A., Hofmeister, T., Schmitt, N.: Football elimination is hard to decide under the 3-point-rule. In: MFCS. (1999) 410–418
5. Wayne, K.D.: A new property and a faster algorithm for baseball elimination. SIAM Journal on Discrete Mathematics **14**(2) (2001) 223–229
6. Kern, W., Paulusma, D.: The computational complexity of the elimination problem in generalized sports competitions. Discrete Optimization **1**(2) (2004) 205–214
7. Pálvölgyi, D.: Deciding soccer scores and partial orientations of graphs. Acta Univ. Sapientiae, Math **1**(1) (2009) 35–42
8. Ribeiro, C.C., Urrutia, S.: An application of integer programming to playoff elimination in football championships. International Transactions in Operational Research **12**(4) (2005) 375–386
9. Duque, R., Díaz, J.F., Arbelaez, A.: Constraint programming and machine learning for interactive soccer analysis. In: To Appear in LION 10. (2016)
10. Arbelaez, A., Hamadi, Y.: Exploiting weak dependencies in tree-based search. In: SAC'09. (2009) 1385–1391
11. Haralick, R.M., Elliott, G.L.: Increasing tree search efficiency for constraint satisfaction problems. IJCAI'79, San Francisco, CA, USA (1979) 356–364
12. Borrett, J., Tsang, E.P., Walsh, N.R.: Adaptive constraint satisfaction: The quickest first principle. In: European Conference on Artificial Intelligence. (1996)