Basketball Games Scheduling using Ant Colony System

Pietro Fronza. University of Trento, Italy

***Abstract*—The job scheduling problem aims to sort a sequence of jobs in order for the execution to proceed smoothly and end in the shortest time possible. This concept is taken up in many areas, such as sports. In fact, the sports leagues aim to distribute the matches during the year conclude the championships in the best way possible, in order to possibly prepare new tournaments for the final part of the sport's season. The purpose of this paper is to evaluate the distribution of the games of some basketball championships and try to create an algorithm that allows an efficient distribution of matches to minimize the risks of overlap and allow some room of movement in case of unforeseen events.**

I. Introduction

TIME slots allocation related to the games of a championship is crucial for the success of a sports season. The italian sports federations, after collecting the participation of several sports clubs to the championships, must draw calendars up for matches regarding those teams. In most cases, a team participates only in one championship, normally of the same category of the team components, in order to have an adequate number of commitments and the chance to let the players enjoy the game. However, there are exceptions: the same team can register for a higher category competition: e.g. a team of fifth graders can register for both the championship of their category and for the championship with the sixth graders.

In this case the federations do not know the several links between the teams involved, so overlaps between matches problems may arise, which usually need to be solved by the teams involved.

The aim of this project is to help the federations, in particular the basketball federation of Trento, in assigning compatible time slots over the whole calendar in order to provide the teams with the best calendar possible. In this way, there will be only a little problem related to the gym availability and unforeseen commitments of the teams (such as participation in a tournament hosted by another team not organized by the federation).

I chose to work in on this specific problem since I am a minibasketball coach for Aquila Basket Trento. The groups that are trained by us are usually able to emerge rapidly within the regional championship: for this reason the club has chosen to register each team both in the championship of the reference category and in the upper category whenever it is possible. In recent years I found myself in the situation where we had to move many games as there were overlaps in the calendar, making it difficult to agree with other teams to play against each other. This solution aims to simplify the scheduling of matches, trying to optimize the distribution of the latter, which can be played simultaneously to facilitate the task of solving unforeseen events to the coaches of the involved teams.

Since this is a scheduling problem, the problem can be addressed using *Ant Colony System* (ACS). I have therefore chosen to address this problem through ACS to assess the efficiency of the method towards this slightly more complex type of problem, as it has some more complicated constraints to solve. The code used can be found at the public Github repository [3].

II. Problem Statement

As discussed in the previous section, the problem in question is the scheduling of basketball league matches. According to the definition of the job scheduling problem of Zhang et. al. [1], our problem can be represented with the tuple *(J,M,G,C_max)*: in this specific case, *J* represents the number of the games to be scheduled, *M* is the number of machines participating in the process, *G* is the graph of the dependencies between the different batches, i.e. games, and C_{max} is the minimum makespan, that is the shortest possible completion time.

In the problem we are considering, however, we are only interested in one possible ordering. For this reason we define *M=1*. In addition, the duration of the games is the same for each element. Consequently the value of C_{max} will always be the same. These factors have led me to better redefine the typology of the problem: since we require only one possible scheduling, we can redefine the problem as an instance of Graph Coloring.

In this way the calendar is represented, in accordance with the representation of the dependencies of Job Scheduling, as a graph G. As for the colors to be used, instead of generating random colors identified with numbers up to n, we define a list of usable colors, i.e. time slots. This way, the only check to be performed is to verify that the number of time slots is sufficient to solve the problem. Since there is a study related to the resolution of Graph Coloring using Ant Colony System (Bessedick et. al. [2]), I chose to continue on this path and turn the problem into an abstraction of Graph Coloring. Let's now take a closer look at the composition of the graph representing the calendar and the list of available slots.

The graph representing the matches' calendar is a weighted graph with the following properties: V is the set of games' identifiers predetermined by the basketball federation, $E = \{(i,j)|w(i,j) > 0\}$, i.e. all the connections between two games predetermined by the federation.

The weight function is defined in the following way:

$$w(i,j) = \begin{cases} 0 & \text{if } i \text{ and } j \text{ are not connected,} \\ 1 & \text{if } i \text{ and } j \text{ share at least one team,} \\ 2 & \text{if } i \text{ and } j \text{ share the same gym but no team.} \end{cases}$$

The last 2 cases can be troublesome to understand: if we have two team that share the same home gym, their matches should never happen in the same slot. In order to differentiate this case from the one where two games share at least one team, I decided to use a different weight. In this way, the matches that occurs in the same gym but that are not connected by the same team will be treated differently.

For what regards the colors to be used, as already mentioned, I have chosen to represent them as a list of available slots. These slots are represented by a class built ad hoc called *TimeSlot*. This class contains the date of the slot for making calculations about constraints and a boolean value that identifies whether the slot is located in the morning or in the afternoon.

In this way, having already available the list of slots to be allocated, it is already possible to verify the existence of a possible solution to the Graph Coloring problem.

III. Methodology

The main contribution of this work consists in the creation of the *benchmark function* for the resolution of the problem through Ant Colony System. In order to ensure the correctness of the data, the first approach to be applied is to check whether a solution to the Graph Coloring problem exists. Having the list of nodes, the adjacency matrix that represents the graph and the list of available slots, it is possible to verify the existence of a solution for the problem by taking the maximum degree of the graph, i.e. the largest number of neighbors for any node in the graph, and verify that this value is less than or equal to the number of available slots.

If a solution exists, the next step is to generate said solution. The solution to our problem is represented by a list of length *n*, i.e. the number of nodes in the graph. $\forall i \in [0, n)$, $sol[i]$ contains the index of the corresponding slot. As the ids of the matches chosen by the federation are not continuous, e.g. a championship may start with identifier 1300 while the second championship may start with identifier 1 as belonging to different categories, I chose to use a mapping related to the list of nodes to represent them within the solution. The same applies to the representation of the time slot list.

To build this solution we need to create an assignment of a slot for each node. Until each node has a valid slot, we select a node from the unvisited ones, i.e. that do not have a slot assigned, and check which slots are available to be allocated.

First we search for the neighbors of the considered node and check if they have been assigned slots. These slots cannot be assigned to the current node as they will be in conflict since the games are connected by an edge. Of all the other slots available, we have to pay attention to overlap constraints: we have to compare each available slot with those already assigned to neighbors.

If the connection with the neighbour is only related to the gym (*w(i,j)=2*), the available slots are all usable. If the connection with the neighbour is related to one of the involved teams (*w(i,j)=1*), we must pay attention to further constraints: a team cannot play, except in exceptional cases, more than one match per week for the same championship. Having only weekend slots available, this implies that the distance between two games belonging to the same championship must be at least of 6 days. In addition, a team may play a maximum of one match per day considering all the championships it take part in, i.e. the distance between the two matches belonging to different championships must be of at least one day.

To highlight matches belonging to a given championship, we just need to know the total number of matches per league. The championships will be saved in a list where each element is characterized by the corresponding partition of the list of nodes (e.g. [56,54] means that championship[0] contains the first 56 nodes of the node list, while championship[1] contains the latter 54).

I also chose to use as fitness funcion the number of different slots in the solution. The objective of the problem is in fact to spread the games reducing as much as possible the number of slots used.

I also added a penalty function to the fitness evaluation: if, in the solution, overlaps are present, the fitness value will become equal to *n+1*, i.e. a value greater than the number of nodes in the graph. In this way, unfeasible solutions will be discarded because they have a higher value.

Regarding the parameters of Ant Colony System I have used the values reported in the following table.

TABLE I: Ant Colony System parameter

Parameter	**Value**
Population size	50
Learning rate	0.1
Evaporation rate	0.1

IV. Experimental Results

In order to verify the efficiency of the model created in Section III, some study cases are required. Since there are no instances to test, I created them by hand using csv files, where I saved the adjacency matrix (*instanceGraph.csv*), the available slots (*instanceSlots.csv*) and the ground truth provided by the federation (*instanceGT.csv*).

I decided to create four different instances:

- one that contains only matches related to a team for a single championship;
- one that contains all the matches of a single championship;
- one that contains all the matches of a single league with the addition of the matches of a single team from a second league;
- one that contains all the matches of two separate championships.

As it can be imagined, the instance one has a trivial solution: having n matches all connected to each other (i.e. the graph represented is a clique) the solution can only contain *n* different slots.

As for the other instances, more interesting results can be noted. For each instance in fact I display the fitness trend of the execution (Figures 1, 4, 7), in addition to a graphic display of the distribution of the games in the time slots made by Ant Colony System (Figures 2, 5, 8) and the same representation of ground truth (Figures 3, 6, 9).

A. *Study Case (Two championship scenario)*

The main study case is the one with the higher number of games, meaning the instance with 2 championship. In fact, we can infer many strenghs of the solution if compared with instances with lower number of games.

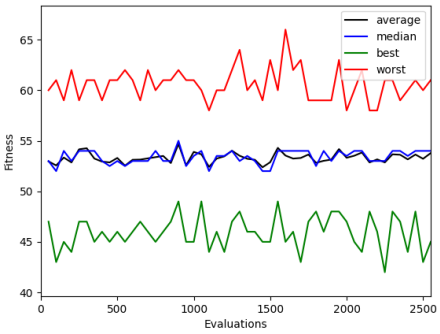

Fig. 1: Fitness trend for two championship instance

As we can see from the graphic, the fitness trend is a bit unstable. There is no clear downward trend, probably due to the fact that I chose to view evaluations instead of generations. Nevertheless, if we look at the results, we can assume that, although the representation through generations could be more linear, there would be no real downward trend. This might be related to the small size of the data: This could be related to the reduced data size: since the problem considered is not too big (for the larger instance we have a total of 56 matches per championship), the number of different solutions generated is very small. In addition, different solutions can have the same fitness in case they have the same number of unique slots assigned.

The initial population, as defined above in Section III, is fixed at 50 individuals, as it is the number of generations used in this evaluation. We can also note that, during the evolutionary process, there were no solutions with conflicts in the allocation of slots, as there are no evaluations with a value equal to 113, i.e. the number of nodes increased by 1.

In Figure 2 and Figure 3 we can have a comparison between the result of the experiment and the ground truth provided by the federation.

These graphic representations are constructed as a calendar where in the x axis we find the selected day as a slot, while in the y axis we find the subdivision between morning and afternoon.

We can observe that the matches within the federation calendar (Figure 3) are much more scattered within the time slots. This means a better distribution of matches in terms of the commitments of the federation, but it also means a

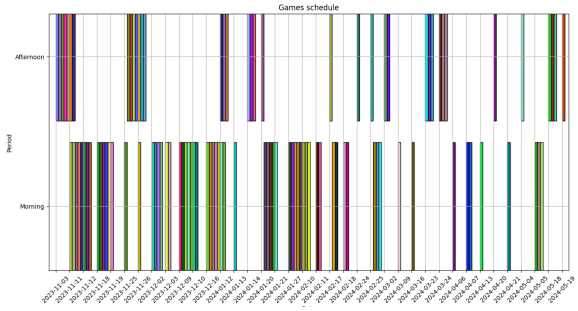

Fig. 2: Results of experiment representation

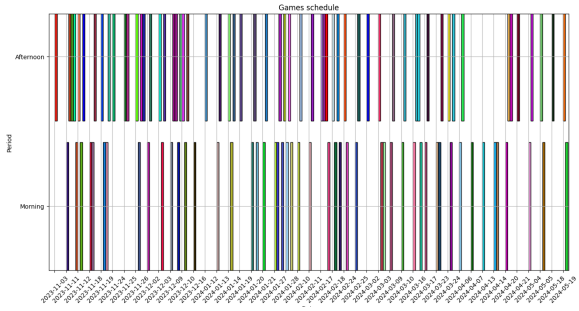

Fig. 3: Ground truth representation

further issue for the participating teams, since, in case of an unforeseen events, it becomes complicated to move the game into a free slot to play it.

Looking instead at the solution deriving from Ant Colony System (Figure 2), we can see that there are many simultaneous games in the same slot. This allows on one hand to minimize the number of time slots used, and provides more flexibility to teams in the event of an unexpected event.

B. *Other scenarios*

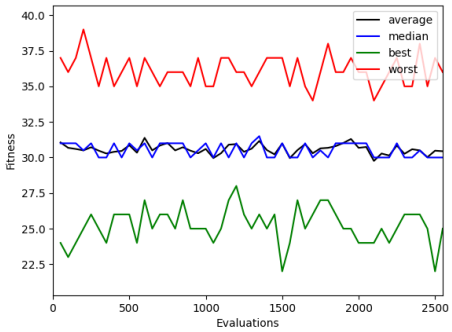

Fig. 4: Fitness trend for one championship instance

The same observations made for the main study case can also be found for the other two non-trivial instances.

We can see that in these cases there are very few matches that are scheduled on the same day between morning and afternoon. This is certainly due to the composition of the

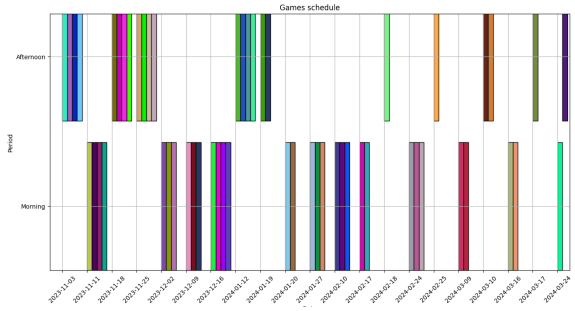

Fig. 5: Results of experiment representation for one championship instance

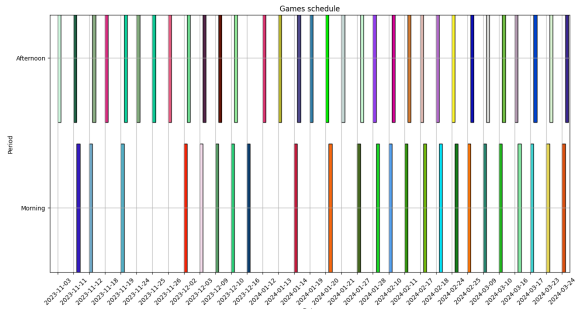

Fig. 6: Ground truth representation for one championship instance

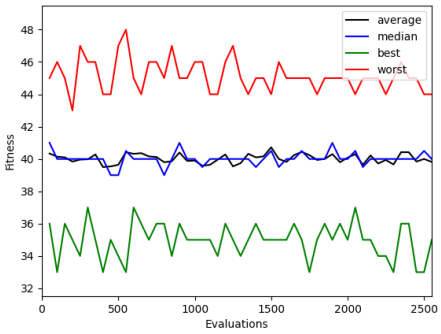

Fig. 7: Fitness trend for one championship with the addition of the games of another championship of one team

data, as, by assumption, the matches played in the same gym are normally related to the same team, and are subject to the constraint of 6 days delay. For the instance where only some games of the second championship are introduced, we can notice instead that some games are arranged on close days, but the algorithm tends to assign similar slots to games that can be played simultaneously.

V. Conclusion

In this job I have seen with my hand the realization of a specific benchmark in order to resolve the selected problem using Ant Colony System.

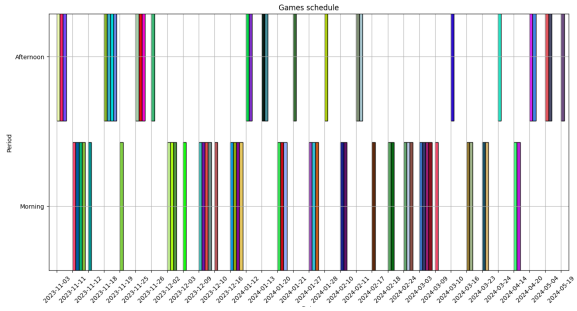

Fig. 8: Results of experiment representation for one championship with the addition of the games of another championship of one team

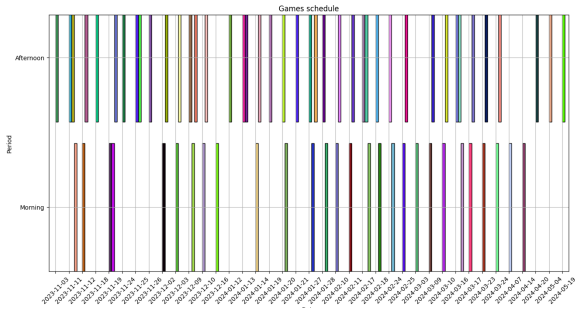

Fig. 9: Ground truth representation for one championship with the addition of the games of another championship of one team

The results show how ACS helps in game scheduling, trying to optimize the assignment of time slot to solve the problem using as few as possible.

In particular, we can see that this approach is optimal with an increasing number of championships, providing a scheduling able to fit the several matches taking also into account the availability of the gyms.

A possible future work could be to identify more slots by identifying time slots of 3 hours to allow a better distribution of matches in case of many championships.

Another possible approach can be to evaluate different methodologies of selection of the next node to visit, to fully exploit the potential of Ant Colony System.

References

[1] Zhang, Jun & Hu, Xiaomin & Tan, X & Zhong, Jinghui & Huang, Q. (2006). Implementation of an Ant Colony Optimization technique for job shop scheduling problem. Transactions of The Institute of Measurement and Control - TRANS INST MEASURE CONTROL. 28. 10.1191/0142331206tm165oa.

[2] Bessedik, Malika & Laib, R. & Boulmerka, Aissa & Drias, Habiba. (2005). Ant Colony System for Graph Coloring Problem. 786- 791. 10.1109/CIMCA.2005.1631360.

[3] Pietro Fronza. GitHub repository. URL: https://github.com/PietroFronzaUniTn/calendar_scheduling_ea. 2024.