The Carry Over Effect in Round Robin Tournaments : An Exploration of Optimization Strategies

Auteur : Heine, Marion
Promoteur(s) : Crama, Yves
Faculté : HEC-Ecole de gestion de l'Université de Liège
Diplôme : Master en ingénieur de gestion, à finalité spécialisée en Supply Chain Management and Business Analytics
Année académique : 2021-2022
URI/URL : http://hdl.handle.net/2268.2/15443

THE CARRY OVER EFFECT IN ROUND ROBIN TOURNAMENTS : AN EXPLORATION OF OPTIMIZATION STRATEGIES

JURY :
Supervisor :
Yves CRAMA
Reader :
Marie BARATTO

Master thesis by
Marion HEINE
For a Master's degree in Business Engineering with a specialization in Supply Chain Management and Business Analytics
Academic year 2021/2022

I. Acknowledgements

I would like to thank a few people who have been of great help and support for me during the realization of this thesis.

First off all, I am very grateful to my supervisor, Yves CRAMA, for suggesting this topic to me and for his help and guidance. I enjoyed working on this subject and I have learned a lot thanks to him. He has always been very patient and available to answer all my questions. I would not have been able to write this thesis without his support.

Second, I would like to thank my friend and my sister who have accepted to read my work and gave me their feedback.

Finally, I thank my parents, family and friends for their encouragements.

II. Table of content

III. Lists of figures and graphs

Figures

Graphs

IV. List of tables

V. List of abbreviations

COE : ..... Carry-Over Effect
GR : ..... Game Rotation
HAP : ..... Home-Away Pattern
ILP : ..... Integer Linear Programming
ILS : ..... Iterated Local Search
OR : ..... Operations Research
PRS : ..... Partial Round Swap
PTS : ..... Partial Team Swap
QIP : ..... Quadratic Integer Programming
RRT : ..... Round Robin Tournament
RS : ..... Round Swap
SA : ..... Simulated Annealing
SRRT : ..... Single Round Robin Tournament
TARS : ..... Teams And Round Swap
TS : ..... Team Swap
VNS : ..... Variable Neighborhood Search

1 Introduction

1.1 Presentation of the topic

The subject of this thesis is the optimization of sport timetables. The main focus is on the carry-over effect (COE) value which is a fairness criterion that assesses the sequence of the matches in a tournament. The objective is to build a feasible schedule that minimizes the COE value. This is thus a decision problem. The COE is a measure of how teams efforts are balanced throughout the tournament. The idea is that, for a given team, playing against a strong (resp. weak) opponent has a certain effect on the team's overall performance for the next game, and that an unbalanced sequence of games could penalize (resp. advantage) that team while advantaging (resp. disadvantaging) the other teams. Therefore, it is interesting to have a look at how the tournament is scheduled and try to keep away from reiterating the same sequence of opponents when designing a schedule. The goal is to avoid the repeating occurrence in the tournament of a given team *a* playing against the last opponent of another strong (or weak) team *b* because it could advantage (or disadvantage) team *a* through the effect of the last match on that common opponent. We say that team *b* gives a COE to team *a* through their common opponent, from one round to the next one. The concept of COE will be defined more precisely later on, in the second chapter.

The problem of minimizing the COE in sport scheduling is interesting in many ways. In mathematical programming it represents a difficult combinatorial optimization problem which makes it a challenging subject for academics and researchers. Indeed, the COE minimization problem falls into the complexity class NP (nondeterministic polynomial time). The set of NP problems are those for which we can efficiently (i.e. in polynomial time, as opposed to exponential time) verify a given solution but it is much more complex to solve them efficiently. In other words, there is no efficient (polynomial) algorithm that solves the problem, at least not yet. In contrast, the complexity class P refers to problems that can be efficiently (in polynomial time) solved and verified as well. In fact, the probably most famous open question in complexity theory is to know whether P = NP or not. For some problems initially classified as NP, this equality has been proved but for others, despite many efforts, no efficient algorithm has been found yet (Arora and Barak, 2009).

Moreover, fairness and optimality in the design of sport schedules are relevant for business, security and logistical matters. Sports events nowadays represent huge business opportunities and challenges. The most obvious and popular example would be soccer, but many other disciplines are being more and more mediatized and monetized. Recently, the UEFA has organized a new competition, the UEFA Europa Conference League. In addition to that, sport betting has become a flourishing business in the last few years thanks to social medias and online betting platforms. Based on that, it definitely seems that the interest in sports competitions is not going to disappear so soon. Many stakeholders, such as the police and broadcasting companies, depend on the organization of those events. The world is also impatiently waiting for the next Soccer World Cup or the next Olympic Games. For all these reasons, the entertainment and excitement offered by sport competitions through media coverage are a precious opportunity to do business. The tournaments and the competitions need to remain attractive for the public, the teams participating, and the various stakeholders involved.

Finally, taking into account the impact of the COE is especially relevant in body-contact sports where players might end up injured or where suspensions for the next game or competition might happen.

1.2 Research objectives

The starting point of this thesis is an article of Guedes and Ribeiro (2011). They present a Quadratic Integer Programming (QIP) formulation of the weighted COE value minimization problem and then develop a heuristic algorithm to find a solution to the problem more efficiently and improve their results. They use weights to represent the relative strength between the participants/teams.

The first objective of this thesis is to linearize and simplify their basic QIP formulation through different techniques and try to tighten the formulation. Then, it is interesting to see how a present-day solver performs on such a model and how much it can improve the resolution times reported by Guedes and Ribeiro in 2011 on the smallest instances (at most 8 teams) on their own version of the linearized model (for which they don't provide any algorithmic details in the paper).

As mentioned above, the COE minimization problem is a complex one. Finding an exact solution becomes more and more complex as the size of the instances increases. Therefore, in order to obtain better solutions (not necessarily optimal) in a reasonable time, the use of heuristic methods becomes handy. That is what Guedes and Ribeiro did in their paper, they used an Iterated Local Search (ILS) metaheuristic combined with a multistart strategy. The second objective of this thesis is thus to create my own version of a heuristic model in order to solve the problem efficiently and hopefully find good and quick solutions to the smallest but especially to some of the larger instances (at least 10 teams). As generating initial fixtures is not an easy task and very few alternatives are presented in the literature, the main challenge will be to find a way to produce, within a reasonable time, initial solutions that are as varied as possible, without resorting to the method that is classically used.

1.3 Overview

This document is organized in six chapters. This section is the end of the first one, dedicated to the introduction of the topic and research objectives statement. The second chapter is an introduction to the theories, methods and concepts that are going to be discussed in chapter 3, 4 and 5. These three chapters make up the development part of this work. They will respectively deal with the linearization of the QIP formulation of the problem, with the generation of initial fixtures, and the implementation of a Simulated Annealing (SA) metaheuristic to optimize these initial solutions. Each of them has a part dedicated to numerical results. Finally, chapter 6 contains the conclusion of this thesis. Limitations are discussed along with ideas for future research.

2 Literature review

2.1 Description of the COE Minimization problem:

Unless specifically stated, the explanations and definitions in this section are based on the work of the following authors : Guedes and Ribeiro (2011), Kendall et al. (2010), Goossens, Yi and Van Bulck (2020).

In this thesis, the COE minimization problem in a Single Round Robin Tournament (SRRT) is considered. That is a competition in which each team plays exactly once against every other team. The basic formulation of the problem has integer variables and is a quadratic minimization problem.

I consider instances with an even number "n" of teams (or players). A possible solution to this problem is a schedule composed of n-1 rounds and n/2 matches per round. This schedule is called a compact schedule as every team plays in every round, i.e. on every match day. It is said that there is no bye. On the contrary, a so-called time relaxed schedule is a schedule that has more slots for games than needed, which means there are some teams that are not playing in each round. This happens especially when the number of team is uneven, at least one team has to have a bye in each round.

A schedule is also characterized by its Home-Away Pattern (HAP). A game is usually modelized as a binary variable depending on various indices including two, let's say *i* and *j*, representing the two teams that oppose each other. The first team *i* is considered as the host for the game, so the home team, and the second team *j* as the away team. From this variable, the HAP of the tournament can be drawn. If team *i* is the home (resp. away) team twice in a raw, that is to say team *i* plays at home (resp. away) in rounds *t*-1 and *t*, we say that it has a break in round *t*. However, HAP is not the focus of this work and won't be dealt with.

Finally, the COE is defined by Lambrechts et al. (2018) as follows:

> "*Given a feasible schedule, we say that team $i \in N$ gives a carry-over effect (coe) to team $j \in N$ in round r, if there exists a team $k \in N$ that plays team i in round $r-1$, and plays team j in round r, $1 \leq r \leq n-1$. We also say that team j receives a coe from team i in round r*."(p.280)

When assessing the COE value of a tournament, the rounds are considered cyclically. The teams in round 1 receive a COE from the last round. In a tournament we can observe the number of COE given by team *i* to team *j*, a value represented by c_{ij}. The expression $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^2$ is used to assess the COE value of a tournament. We can represent all c_{ij} in a matrix C of dimensions $n \times n$ in which each entry is the number of COE given by *i* to *j*.

2.2 Prior work

2.2.1 Carry Over Effect

Russell (1980) introduced the concept COE and balanced schedule in a paper where he claims that balanced schedules exist if and only if n is a power of 2. Later on this conjecture will be proven to be false as balanced schedules have been constructed for instances with n = 20 and n = 22. Nevertheless, Russell provides a first definition of the COE : "*Each team is considered to have an effect on its opponents which carries over to the next match. If team A meets team B in one match and team C in the next, then it is reasonable that team A's performance against team C will have been affected by team B*." (1980, p.127).

Then, Miyashiro and Matsui (2006) proposed a fast heuristic algorithm that generated better solutions than the best ones known at that time. They considered a simple version of the COE minimization problem on instances with an even number n of teams, n-1 match days, so no bye and every team plays every other once. They used the classical circle method or polygon method (explained later on in this work) to construct schedules of high COE value. In parallel, they claimed that this method actually produces solutions with maximum COE value. Then they permuted the rounds of the obtained schedules, in order to minimize the COE value and obtained their results in less than 1 second.

In the following years, Guedes and Ribeiro (2011) issued a paper in which they solve a weighted version of the COE minimization problem. The idea is to take into account the different strength levels of the teams and to create balanced schedules accordingly. As mentioned in chapter one, they developed a heuristic algorithm that they applied to both weighted and unweighted instances. For their computational experiments, they generated four classes of weighted instances : random, linear, perturbed linear and real-life inspired instances. They managed to improve the best known value for n = 12.

A few years later, Lambrechts et al. (2018) proved what Miyashiro and Matsui had conjectured. They even showed that any schedule that displays maximal COE value can be generated with the circle method.

Before going deeper into the development of this thesis, there are some values, theory concepts and methods that must be defined and introduced to the reader.

2.2.2 Integer Programming and related topics

i. N-3

An interesting value in the COE minimization problem is the one I refer to as M in the *Model 2* in section 3.2.2 This value, explained in Lambrechts et al. (2018), is an upper bound on the maximum number of COE a team can receive or give in any feasible schedule. It is rigorously proven by the authors but one can also understand it with a simple reasoning. Consider two arbitrary teams *a* and *b*. Since a SRRT is made of n-1 rounds, *a* has at most n-1 opportunities to give a COE to *b*. Assume now that team *a* plays

team *b* in round 2 (this can be assumed without loss of generality, since the schedule is cyclic). Then, team *a* does not give any COE to *b* from round 1 to 2, nor from round 2 to 3 (by definition of a COE). Thus, *a* can give at most n-3 COEs to *b*. To resume, a team *a* cannot give a COE to team *b* from round *r-1* (e.g. round 1) to *r* (e.g. round 2) nor from round *r* (e.g. round 2) to *r+1* (e.g. round 3) knowing that they play each other in round *r*. Therefore, the value of M is set to n-3.

ii. Lower bounds

Based on the definition and computation of the COE value, it can be seen that an ideal schedule with respect to the COE is a schedule that displays unit values for all elements except the ones in the diagonal (because a team cannot give herself a COE, by definition). Indeed, as the c_{ij} are squared in the function that computes the COE value of a tournament, the best case scenario is when each c_{ij} is of value 1. This is the case when each team gives only one COE to any of the other teams. Then, the lower bound on the problem of minimizing the COE value is n(n-1), when each team gives one COE in each round.

A schedule is qualified as balanced with respect to the COE when it is evenly spread over the teams and when the COE value of the tournament reaches its smallest bound value : n(n-1). On the contrary, if the COE value is not spread evenly, the schedule is said to be unbalanced.

Here are the theoretical lower bounds on the instances that will be considered at some point in this work :

Table 1 : Lower bounds on the COE minimization problem in SRRT.

Instance size n	n.(n-1)
4	12
6	30
8	56
10	90
12	132
14	182
16	240
20	380
24	552

Table 1 - Data from own calculations.

iii. Standard Linearization

In chapter 3, the Standard Linearization Method is introduced. What I refer to is the standard approach, described in the paper of Glover (1975) as a method used to linearize polynomials function. To do so, the polynomial function is expressed as 0-1 variables function and then new 0-1 variables are introduced to take the place of the cross-product terms. Additional constraints are concurrently introduced to ensure that the new variables will take the right values. The reader is invited to read the ***Appendix A*** for more details on this method.

iv. Big O notation

The big O notation, for example "$O(n^3)$", is used to assess the complexity of an algorithm, so the complexity of the operations we ask the computer to realize through the code we have written. In polynomial functions, it generally refers to the term of higher degree which is the dominant term. Imagine the input of a function is of positive size n and the function has, let's imagine, a constant, a quadratic and a cubic term. Then, the constant term, or any other term of lower degree in the function, won't ever grow faster than the dominant term, the cubic one, as the value of n increases. The big o notation is a way to assess the worst case scenario for algorithmic performance (in both execution time and memory space occupation). The idea is to know how and what will happen as n increases towards infinity (Bae, 2019).

Here is a formal definition from the book "Cryptography" (Rubinstein-Salzedo, 2018) :

> "*Let f(x) and g(x) be two functions, we say that* $f(x) = O(g(x))$ *if there is some constant* $C > 0$*, which does not depend on x, so that*
>
> $$|f(x)| \leq Cg(x)$$
>
> *for all x*" (p.75).

Note : "x" is considered as sufficiently large, that is to say there exists some value A such that

$|f(x)| \leq Cg(x)$ for all $x \geq$ A.

2.2.3 Heuristic methods

i. Complexity of the solution space in round robin scheduling problems

In a quite recent paper, Januario and Urrutia (2016) explained that existing neighborhood structures commonly used with local search methods do not fully connect the solution space of round robin tournament (RRT) problems. That means there is no path that connects all the solutions. It is very difficult if not impossible to link all possible solutions in the solutions space, going from one neighbor to another, using the available structures. In reaction to this, they propose a neighborhood structure that provides a higher connectivity of the solution space. They call it TARS, for Teams And Rounds Swap. It is a combination of the existing moves PTS (Partial Team Swap[1]) and PRS (Partial Round Swap[2]). They provide proof of the increased connectivity and assess their neighborhood structure using two case studies.

ii. Circle or polygon method

The circle method is easy to set up and provides a fast and practical way to generate one feasible schedule for SRRT with an even number n of teams. The result is a compact schedule of n-1 rounds

[1] A definition of the PTS move can be found in ***Appendix B***.
[2] A definition of the PRS move can be found in ***Appendix B***.

each composed of n/2 games. It can also be referred to as a Kirkman tournament, for the name of the person that published the method in the mid-nineteenth century. Most of the time, the circle method is used to generate the very initial solution that will be the starting point to (iterated) local search procedures. However, as stated earlier in section 2.2.1 by Miyashiro and Matsui (2006), RRT obtained with the circle method have maximum COE value, moreover this method produces only one calendar per instance size.

A representation based on graph theory is often used to illustrate this technique. An edge linking two nodes in a given graph represents a game in a given round. As you can see, ***Figure 1*** below is showing the three first rounds of a schedule built with this method. The first round opposes team n with team 1, team 2 with team 7, and so on. Note the edges that cross perpendicularly. Then, for the second round, the edges rotate so that n is facing team 2, team 1 is facing team 3, etc. Nodes are not moving, only the edges are rotating. An algorithmic description of how to implement the circle method is provided in ***Appendix C***.

Figure 1 : Visual representation of a part of a schedule constructed with the circle method.

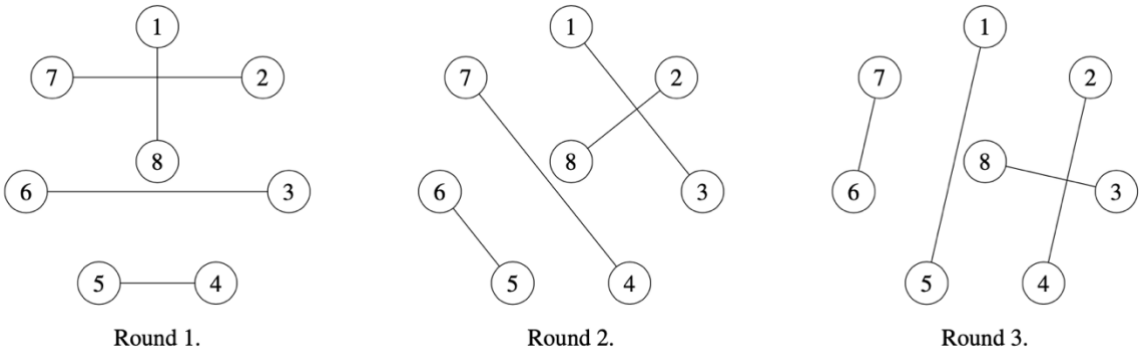


Figure 1 – Visual representation of a part of a schedule constructed with the circle method. Reproduced from "Round-robin tournaments generated by the circle method have maximum carry-over effect", by Lambrechts et al., 2018, FEB Research Report KBI_1603, 9682, pp. 178–189.

iii. Simulated annealing

In chapter five, I describe the metaheuristic I used to find the numerical results to the COE optimization problem. SA is an alternative to exact procedures that have shown good and rapid results in many problem situations.

This method is inspired from real-life metallurgy technique called annealing. It is commonly used to tackle complex combinatorial problems. SA is a heuristic technique used to find feasible and hopefully good solutions to such problems. The SA procedure is an analogy of the slow cooling of solid metal technique used to bring the material to an optimal state. The comparison stands in the idea of arranging elements to make the whole product stable. Similarly, we want to arrange elements of a solution so that it gives the best result, that is to say it optimizes the value of interest. Concretely in this thesis, the challenge is to determine the best way to arrange teams in a compact schedule in order to minimize its COE value.

The reader can have a look at ***Appendix D*** for more details on the SA procedure.

iv. Ejection chains

Here, the concept of ejection chains is very briefly introduced to the reader so that he acknowledges this is part of prior work theory that I am using in this work. Not many details will be given on the theory behind ejection chains as it is quite heavy and abstract. It will rather be illustrated later on.

In Glover (1996), ejection chains are "*procedures based on the notion of generating compound sequences of moves, leading from one solution to another, by linked steps in which changes in selected elements cause other elements to be « ejected from » their current state, position or value assignment*"(p.223). This idea of elements state being changed and therefore other elements being ejected from their current state will be very useful in order to understand the Game Rotation (GR) neighborhood definition in section 5.2. In that section I will use a concrete example to explain and illustrate both the concept of ejection chains and the GR move.

2.3 Methodology

In this section, I want to explain to the reader how I will proceed and why I will follow that methodology. But first, it is important to understand what Operation Research (OR) is, how it is performed and what kind of research it constitutes.

OR is an analytical method aiming to help organizations making decisions in complex situations. To better understand the role of OR, one might first have a look at the analytics continuum in data science (see ***Figure 2***). It is a graphical representation that illustrates the value that can be brought to the organization as the maturity of data analysis increases. During the first stage, data is collected and analyzed to see what happened, this is the descriptive analytics. Then, to get more insight, reasons for what happened are identified and predictions are made based on those reasons. These are the diagnostic and predictive analytics. The last stage is the prescriptive analytics, based on simulations and automation, it consists of prescribing a course of actions to reach a certain result.

Figure 2 : Analytics continuum.

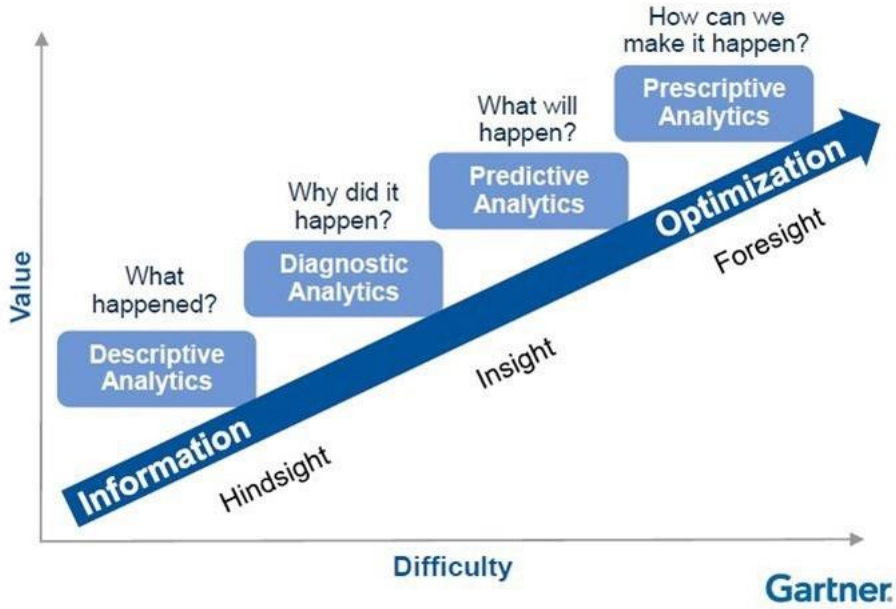


Figure 2 – Analytics continuum. Image : Gartner (2012).

Operations research is a complementary discipline to data science which tries to evaluate and optimize decisions using modelling. Moreover, research in OR can be seen as design research, which, as suggested by Manson (2006) "*is a process of using knowledge to design and create useful artefacts, and then using various rigorous methods to analyse why, or why not, a particular artefact is effective*"(p.161). In other words, design research is part of the science of the artificial. According to that vision of research in OR, a distinction can be made between the practical applications of OR, and the actual research for new methods, models, algorithms, theories and tools to improve the practice of OR.

In the framework of this thesis, a specific problem is studied, the minimization of the COE in SRRT. The methodology intended is as follows : different models and algorithms will be built and/or modified with the aim of improvement. Then, they will be repeatedly evaluated and further modified and tested until the obtention of satisfactory results and/or until conclusions can be drawn. This process can also be seen through the lens of Deming's PDCA (Plan Do Check Act) tool. The use of theory and personal thoughts to design a model or algorithm would be the "Plan". Actually designing it using commercial solver interface, the coding phase, would be the "Do". Testing the model or algorithm and its efficiency would be the "Check". And, further modifications in the search of improvement would be the "Act".
The evaluation of an algorithm or model is often made through the assessment of utility, efficiency and quality of the resolution. How fast can we get solutions and how good, how useful, are they? How well is the model or algorithm performing?

I believe this method is appropriate for that type of research. As mentioned above, OR follows a process of designing artefacts and assessing their effectiveness which means it is important to stop regularly to test and decide if what has been created is useful, relevant and if it is aligned with the objective. There is a bit of mystery in how the solver deals with the problem it has been given. It is not always easy to understand the relationship between the model structure and the performance of the solver, nor between the structure of a solution and its value. That means surprising and unexpected

results might, and probably will, occur sometimes. Hence the importance of systematically review the model and the strategy used to solve the problem.

3 Integer Programming

3.1 Context

In this third chapter different strategies to simplify and linearize a model are successively tried. As suggested by the precedent chapter, the methodology consist in regularly stop, assess and, based on the observations, adapt the strategy. Therefore, this chapter is organized into logical steps leading to the final model and results. The different models proposed are all provided in ".jl" files and can be tested with the Julia Programming application.

3.2 Linearization of the initial quadratic model

In their paper, Guedes and Ribeiro (2011) introduced a QIP formulation, with $O(n^3)$ variables, of the weighted COE minimization problem in a SRRT (see *Model 1* below). Then, they linearized it and reported the results obtained with this reformulation for n = 4, 6, and 8, on the unweighted version, where $w_{i,j} = 1$, for i, j = 1,…, n : i ≠ j, in which each team is considered to have the same relative strength as any other team. The commercial solver they used (CPLEX 11.0) was not able to solve larger instances in a reasonable time. At the time they applied the solver, about ten years ago, computing performances available were lower than the current ones, as suggested by Moore's law and by algorithmic improvements in the implementation of integer programming solvers (Koch, Berthold, Pederson & Vanaret, 2021). Consequently, the first practical step of this thesis was to see how a present-day solver (Gurobi Optimizer 9.1.1) could perform on the linearized model. However, in the paper, the Integer Linear Programming (ILP) model is not provided, the only information given is that the reformulation has $O(n^4)$ variables. So, before everything else, the QIP model had to be linearized. The targeted result is a linearized model with $O(n^4)$ variables. In the next two sections, 3.2.1. and 3.2.2., two linearization methods are detailed and tested for their efficiency. The basic quadratic formulation presented in the paper is the starting model (*Model 0*) and is gradually modified with the different linearization techniques until obtaining the final version of the linearized model (*Model 2*).

The following model corresponds to the Julia Programming file Model0.jl.

Model 0

Parameters :
Number of teams: n
(Weights : $w_{i,j} = 1$ for i, j = 1, …, n : i ≠ j)

Variables :
$Y_{i,j,k}$ = 1 if team i plays team j in round k, for i, j, k = 1, …,n.
0 otherwise.

$Z_{i,j}$ = number of COE given by team i to team j, for i, j = 1, …, n : i ≠ j

Objective function :

Min $\sum_{i=1}^{n}\sum_{j=1}^{n}(w_{i,j} *) Z_{i,j}^{2}$ (1)

Constraints :

For i, j, l = 1,…, n ; k=1, …, n-1 :

If i ≠ j : $Z_{i,j} = \sum_{l=1}^{n}\sum_{k=1}^{n-1} Y_{i,l,k} * Y_{j,l,k+1}$ (2)

If i = j : $Z_{i,j} = 0$

$Y_{i,j,k} = Y_{j,i,k}$, for i, j, k = 1, …, n. (3)

$\sum_{i=1}^{n} Y_{i,j,k} = 1$, for j, k = 1, …, n. (4)

$\sum_{k=1}^{n-1} Y_{i,j,k} = 1$, for i, j = 1, …, n : i ≠ j. (5)

$Y_{i,i,k} = 0$, for i, k = 1, …, n. (6)

$Y_{i,j,n} = Y_{i,j,1}$, for i, j = 1, …, n. (7)

$Y_{i,j,k} \in \{0,1\}$, for i, j, k = 1, …, n. (8a)

In this initial model, the objective function (1) minimizes the sum of all the (weighted) squared COEs. Note that $w_{i,j}$ could be omitted in this unweighted formulation as its value is 1 for each pair of teams (i, j) : i ≠ j. The first constraint (2) defines the COE value as the number of COEs given by i to j from one game to the next, over the tournament. The next constraint (3) makes sure that if team i plays team j in round k, then conversely team j plays team i in that same round. Constraints (4) and (5) insure respectively that one team plays exactly one other team in each round and that two teams only meet once over the tournament. Constraint (6) guarantees that a same team cannot play against itself. Constraint (7) states that rounds are considered cyclically. Finally, constraint (8a) insures that variable $Y_{i,j,k}$ is binary.

3.2.1 First linearization method : standard linearization

Attempt number one to linearize the model was to apply the standard linearization method, mentioned in section 2.2. As a reminder, the objective function considered here is $\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j} * Z_{i,j}^{2}$, where $w_{i,j}$ can be omitted in the case of an unweighted COE, as explained above. The outcome of this first try is the *Model 1* below.

To begin with, let's consider $Z_{i,j}$, which has been defined as a sum of products of two other variables, see constraint (2) of Model 0 above, and is therefore not linear. Applying the standard linearization method to this product of variables results in the introduction of O(n^4) new binary variables $U_{i,j,l,k}$ representing the products $Y_{i,l,k} * Y_{j,l,k+1}$, and new constraints : (10) and (8b). Note that $U_{i,j,l,k} = 1$ exactly when team i gives a COE to team j through team l from round k to round k+1.
To help the reader, from one model to another, the number of each new constraint is in bold type.

Next, once the product defining $Z_{i,j}$ has been replaced by a unique variable $U_{i,j,l,k}$, the squared operation from the initial objective function must be taken care of. Indeed, the objective function is still quadratic after the first standard linearization : $\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\sum_{l=1}^{n}\sum_{k=1}^{n-1} U_{i,j,l,k}\right)^{2}$. This expression

defines the square of a summation which, once it is developed, contains the product of two variables that must also be linearized using the standard method. This operation leads to the introduction of $O(n^6)$ new binary variables $V_{i,j,l,k,r,s}$ representing the products $U_{i,j,l,k}$ $U_{i,j,r,s}$ and new constraints : (11) and (8c). The resulting new objective function (9) has $O(n^6)$ variables.
Note that constraints (3) to (8a) remain from the initial *Model 0*.

The following model corresponds to the Julia Programming file Model1.jl.

Model 1

Parameters :
Number of teams: n

Variables :
$Y_{i,j,k} = 1$ if team i plays team j in round k, for i, j, k = 1, …, n.
0 otherwise.

$U_{i,j,l,k}$ = 1 if team i gives a COE to team j through team l from round k to round k+1, for i, j, l, k = 1, …, n.
0 otherwise

$V_{i,j,l,k,r,s}$ = 1 if i gives a COE to j through l from round k to k+1 and another COE through r from round s to s+1, for i, j, l, k, r, s = 1, …, n.
0 otherwise

Objective function :
Min $\sum_{i=1}^{n}\sum_{j=1}^{n}\left[\sum_{l=1}^{n}\sum_{k=1}^{n-1} U_{i,j,l,k} + \sum_{l=1}^{n}\sum_{k=1}^{n-1}\sum_{r=1}^{n}\sum_{s=1}^{n-1} V_{i,j,l,k,r,s}\right]$ **(9)**

Constraints :
For i, j, l = 1, …, n ; k = 1, …, n-1 :
If i ≠ j ≠ l ≠ i : $U_{i,j,l,k} \geq Y_{i,l,k} + Y_{j,l,k+1} - 1$ **(10)**
Otherwise, $U_{i,j,l,k} = 0$,

For i, j, l, r = 1, …, n ; k, s = 1, …, n-1:
If l = r and k = s : $V_{i,j,l,k,r,s} = 0$ **(11)**
Otherwise, $V_{i,j,l,k,r,s} \geq U_{i,j,l,k} + U_{i,j,r,s} - 1$

$Y_{i,j,k} = Y_{j,i,k}$, for i, j, k = 1, …, n. (3)

$\sum_{i=1}^{n} Y_{i,j,k} = 1$, for j, k = 1, …, n. (4)
$\sum_{k=1}^{n-1} Y_{i,j,k} = 1$, for i, j = 1, …, n : i ≠ j. (5)

$Y_{i,i,k} = 0$, for i, k = 1, …, n. (6)
$Y_{i,j,n} = Y_{i,j,1}$, for i, j = 1, …, n. (7)

$Y_{i,j,k}$, $U_{i,j,l,k}$, $V_{i,j,l,kr,s} \in \{0,1\}$, for i, j, l, k = 1, …, n. (8a, **8b**, **8c**)

Limitations of the model:

Because of the complexity of the variable $V_{i,j,l,k,r,s}$, running this model for instances with n > 6 teams is too demanding for the solver, at least on the computer used for the trials. The program becomes almost overloaded and a message of error appears after a period of freeze of the interface. Even though with great patience, and ignoring this error message, it is eventually possible to obtain a solution for n = 6 and n = 8 teams, this is not practical nor efficient. It seems that the model is overly complex in terms of memory space occupation. Larger instances would be impossible to solve using this model.

Moreover, this $O(n^6)$ variables model is not compatible with our targeted problem size. Remember that the ILP model formulated by Guedes and Ribeiro has allegedly $O(n^4)$ variables. Therefore, in the next section, an improved and more efficient linearization technique is implemented.

3.2.2 Second linearization method : improved and computationally efficient

As explained before, the model obtained from the standard linearization method applied twice is not satisfying because it has $O(n^6)$ variables. Consequently, a second method is considered in this section.

Glover (1975) proposed an improved formulation for nonlinear quadratic integer problems that produces smaller size problems. Glover's technique results in the introduction of m new variables and 4m new constraints (m being the number of variables introduced). Later, Oral and Kettani (1990) published a paper where they propose a more compact formulation technique leading to the introduction of m new variables and only 2m new constraints.

Based on these two articles, an improved and more efficient formulation was elaborated. The resulting *Model 2* is given below and explained right after.

The following model corresponds to the Julia Programming file Model2.jl.

Model 2

Parameters :
Number of teams : n
M = n-3

Variables :
$Y_{i,j,k}$ = 1 if team i plays team j in round k, for i, j, k = 1, …, n.
0 otherwise.

$U_{i,j,l,k}$ = 1 if team i gives a COE to team j through team l from round k to round k+1, for i, j, l, k = 1, …, n.
0 otherwise

$W_{i,j,l,k}$= number of COEs given by i to j if $U_{i,j,l,k}$= 1, for i, j, l, k = 1, …, n.
0 otherwise.

$$W_{i,j,l,k} = U_{i,j,l,k} * (\sum_{r=1}^{n} \sum_{s=1}^{n-1} U_{i,j,r,s})$$

Objective function :

Min $\sum_{i=1}^{n} \sum_{j=1}^{n} [\sum_{l=1}^{n} \sum_{k=1}^{n-1} U_{i,j,l,k} + \sum_{l=1}^{n} \sum_{k=1}^{n-1} W_{i,j,l,k}]$ **(13)**

Constraints :

For i, j, l = 1, …, n ; k = 1, …, n-1 :

If i ≠ j ≠ l, $U_{i,j,l,k} \geq Y_{i,l,k} + Y_{j,l,k+1} - 1$ (10)

Otherwise, $U_{i,j,l,k} = 0$.

$W_{i,j,l,k} \geq [\sum_{r=1}^{n} \sum_{s=1}^{n-1} U_{i,j,r,s}] - [M * (1 - U_{i,j,l,k})]$,
for i, j, l = 1, …, n ; k = 1, …, n-1 : r ≠ l or k ≠ s. **(12)**

$Y_{i,j,k} = Y_{j,i,k}$, for i, j, k = 1, …, n. (3)

$\sum_{i=1}^{n} Y_{i,j,k} = 1$, for j, k = 1, …, n. (4)

$\sum_{k=1}^{n-1} Y_{i,j,k} = 1$, for i, j = 1, …, n : i ≠ j. (5)

$Y_{i,i,k} = 0$, for i, k = 1, …, n. (6)

$Y_{i,j,n} = Y_{i,j,1}$, for i, j = 1, …, n. (7)

$Y_{i,j,k}$, $U_{i,j,l,k} \in \{0,1\}$, $W_{i,j,l,k}$ integer for i, j, k, l = 1, …, n. (8a, 8b, **8d**)

First and foremost, in this model a new parameter M is introduced. The value of M is n-3, corresponding for any feasible schedule to the maximum number of COEs a team can give to another. This observation is proven by Lambrechts et al. (2018) and has been explained earlier in section 2.2.

Furthermore, variable $U_{i,j,l,k}$ present in *Model 1* is conserved in this new model. However, this time a new integer O(n^4) variable is introduced, $W_{i,j,l,k}$, which represents the number of COE team i gives to team j if $U_{i,j,l,k}$ is equal to one. The introduction of this variable brings 2 new constraints, respectively (12) and (8d), in accordance with Oral and Kettani's method. The resulting objective function is (13) and has O(n^4) variables. Constraint (12) can be interpreted as the additional COE(s) given by i to j, other than the minimal and optimal value of 1 COE over the whole tournament. $W_{i,j,l,k}$ is the variable that captures the possible second or more COE(s) given by i to j through any other team r ≠ l in any round s = 1, …, n-1, or in any other round s ≠ k through any team r = 1, …, n. When i is not giving a COE to j through l in round k, the second part of the constraint, the expression $-[M * (1 - U_{i,j,l,k})]$, ensures that $W_{i,j,l,k}$ can take the lowest value possible which is zero by definition of $W_{i,j,l,k}$. When i is giving a COE to j through l in round k, that second part is taking the value of 0 and $W_{i,j,l,k}$ must take at least the value of $[\sum_{r=1}^{n} \sum_{s=1}^{n-1} U_{i,j,r,s}]$ which, as explained before, represents the other COE(s) given by i to j (except the first one, through l in round k) over the tournament.

Finally, note that constraints (3) to (8b) from *Model 1* are still present in this last version as they are linked to variables $Y_{i,j,k}$ and $U_{i,j,l,k}$.

3.3 Introduction of additional constraints

This last version of the model (*Model* 2) still had some room for improvement. Indeed, from the output of the solver during the resolution, we can see that once it found the optimal solution, it needs an extra time to close the gap between this solution and the best bound found.

Next, additional valid constraints are included to the model to try to reduce the time of resolution of the problem. Computational results are in section 3.3 below.

3.3.1 Valid inequality

Valid inequalities, also called cutting planes or cuts, are used to tighten the formulation without compromising optimality. Those constraints are satisfied by all solutions or at least by some optimal solutions (Crama, 2020). They can be stated in order to help the solver in the resolution. Those inequalities do not always directly improve the resolution time but they often help improve the quality of the bounds. They are a good strategy for improvement and are worth trying out.

In *Model 2*, multiplying constraint (4) by $Y_{j,l,k+1}$, for an arbitrary team j different from i, we obtain:

$\sum_{i=1}^{n} Y_{i,l,k}\, Y_{j,l,k+1} = Y_{j,l,k+1}$ for l = 1, …, n : k = 1, …, n-1.

The following inequality can be imposed given the definition of variable $U_{i,j,l,k}$. In fact, this is exactly what is wanted in optimal solutions :

$Y_{i,l,k} Y_{j,l,k+1} = U_{i,j,l,k}$ for l = 1, …, n : k = 1, …, n-1.

Thus, the following constraint that strengthens the formulation can be imposed :

$\sum_{i=1}^{n} U_{i,j,l,k} = Y_{j,l,k+1}$ for j, l = 1, …, n, j ≠ l; k = 1, …, n-1 (14)

The introduction of this constraint leads to another simplified formulation of the objective function. Indeed, constraint (14) insures that if team j is playing team i in round k+1, then if we consider every possible game between i and l in round k, it is sure there is a COE from i to j through l from round k to k+1, thus $\sum_{l=1}^{n} U_{i,j,l,k}$ = 1. From there, in the first term of the objective function (13), the expression $\sum_{l=1}^{n} \sum_{k=1}^{n-1} U_{i,j,l,k}$ is the sum of all the COEs given by team i to team j, for an arbitrary team i different from j, through any of the other teams from round k to round k+1 over the whole tournament, so for k=1, …, n-1. This term takes exactly the value of 1 when the variable $Y_{j,l,k+1}$ takes the value 1, that is

to say when i is giving a COE to j through l, so when l is the common opponent of i and j from round k to k+1. Knowing that and knowing the value of n, we have the following:

$$\begin{aligned} \sum_{i=1}^{n}\sum_{j=1}^{n}\left[\sum_{l=1}^{n}\sum_{k=1}^{n-1}U_{i,j,l,k}\right] &= \sum_{j=1}^{n}\sum_{l=1}^{n}\left[\sum_{k=1}^{n-1}Y_{j,l,k+1}\right] && (14)\\ &= \sum_{j=1}^{n}\sum_{l=1,l\neq j}^{n}1 && (5)\\ &= n*(n-1) \end{aligned}$$

This value is the lower bound I mentioned in section 2.2.2. – ii. and is obtained as soon as the solver starts the resolution, thanks to constraint (14) that reinforces the model.

3.3.2 Symmetry breaking

Symmetry in ILP arises in problems where different solutions of equivalent value can be obtained as a result of permuting the model's variables with no effect on its structure (Margot, 2010). When in a problem there are many solutions symmetric to others, visiting them all during the resolution process is a waste of time ; the search space is needlessly large and its size could be reduced without compromising the ability of finding (an) optimal solution(s). Therefore, symmetry breaking constraints are a good strategy to optimize a model and thus the resolution time of the problem.

In the COE minimization problem, for a given optimal solution, if we decide to rename every team with another team's name, that is to say if we permute the teams, the result remains the same as long as each different team is renamed consistently (e.g. *team a* is always renamed *team b*). For instance, for problem with {*team a*, *team b*, *team c*, *team d}*, another solution in which *team a* is renamed *team b*, *team b* renamed *team c*, *team c* renamed *team d* and *team d* renamed *team a*, has the same value as the initial one. This move is one of the n! = 24 possible permutations of the set of n = 4 teams. The problem will have more potential symmetric solutions as the value of n increases. Without symmetry breaking constraints, the solver will evaluate the set of every possible solution, including symmetric ones, resulting in an unnecessary long running time.

To speed up the resolution time, symmetry breaking constraints are added to the model and their utility is assessed. For this scheduling problem, a first option considered is to fix arbitrarily all the matches of the first round and the first match of the second round, see constraint (16), this way, (n/2)+1 variables are fixed. A second option, see constraint (17), is to fix all the matches of one team (e.g. team 1) over the n-1 rounds, in this case n-1 variables are fixed. And the last option is to not fix any match. Each of the two first options will initially generate one or more COE(s). Constraint (16) will set only one COE from round 1 to round 2 and constraint (17) will set n-1 COEs, one between each round. However, in both cases, the COEs produced are from and to different teams. Thus, there is no team that receives or gives more than 1 COE, which is what we are looking for in optimal solutions. Optimality is not compromised by those additional constraints.

Now that these constraints have been considered, the final ILP model (see the Julia Programming file FinalModel.jl) is as follows.

Final Model

Parameters :
Number of teams: n
M = n-3

Variables :
$Y_{i,j,k}$ = 1 if team i plays team j in round k, for i, j, k = 1, …, n.
0 otherwise.

$U_{i,j,l,k}$ = 1 if team i gives a COE to team j through team l from round k to round k+1, for i, j, l, k = 1, …, n.
0 otherwise

$W_{i,j,l,k}$= number of COEs given by i to j if $U_{i,j,l,k}$= 1, for i, j, l, k = 1, …, n.
0 otherwise.

Objective function :
Min $\mathbf{n} * (\mathbf{n}-\mathbf{1}) + \sum_{i=1}^{n}\sum_{j=1}^{n}\left[\sum_{l=1}^{n}\sum_{k=1}^{n-1} W_{i,j,l,k}\right]$ **(15)**

Constraints :
For i, j, l = 1, …, n ; k = 1, …, n-1 :
If i ≠ j ≠ l, $U_{i,j,l,k} \geq Y_{i,l,k} + Y_{j,l,k+1} - 1$ (10)
Otherwise, $U_{i,j,l,k} = 0$.

$W_{i,j,l,k} \geq \left[\sum_{r=1}^{n}\sum_{s=1}^{n-1} U_{i,j,r,s}\right] - \left[M * \left(1 - U_{i,j,l,k}\right)\right]$,
for i, j, l = 1, …, n ; k = 1, …, n-1 : r ≠ l or k ≠ s. (12)

$Y_{i,j,k} = Y_{j,i,k}$, for i, j, k = 1, …, n. (3)

$\sum_{i=1}^{n} Y_{i,j,k} = 1$, for j, k = 1, …, n. (4)

$\sum_{k=1}^{n-1} Y_{i,j,k} = 1$, for i, j = 1, …, n : i ≠ j. (5)

$Y_{i,i,k} = 0$, for i, k = 1, …, n. (6)
$Y_{i,j,n} = Y_{i,j,1}$, for i, j = 1, …, n. (7)

$Y_{i,j,k}$, $U_{i,j,l,k} \in \{0,1\}$, $W_{i,j,l,k}$ integer for i, j, k, l = 1, …, n. (8a, 8b, 8d)

$\sum_{i=1}^{n} U_{i,j,l,k} = Y_{j,l,k+1}$ for j, l = 1, …, n : k = 1, …, n-1. **(14)**

Symmetry breaking constraints:

$Y_{2k-1,2k,1} = 1$ for k = 1, …, n/2. **(16)**

$Y_{1,3,2} = 1$

OR

$Y_{1,k+1,k} = 1$ for k = 1, …, n-1. **(17)**

3.4 Computational results

Table 2 shows the results obtained by Guedes and Ribeiro (2011) using the commercial solver CPLEX 11.0 on their own version of the linearized model, and it also shows the results obtained with the initial QIP formulation (*Model* 0) using the solver Gurobi Optimizer version 9.1.1. A limit of 7200 seconds (2 hours) was imposed on the running time for the QIP model. For instance size n = 10, there was no improvement of the solution after 4024 seconds and until the time limit was reached. Values in the "Best to date" column are the best COE values ever found, all methods considered.

For the Guedes and Ribeiro's results, the computational experiments were performed on an AMD Athlon 64 X2 machine with 2.3 GHz and one GB of RAM memory. The code was implemented in C++ and compiled with the GNU C/C++ compiler (GCC) version 4.2.4 under Ubuntu Linux 8.04. For all the other results, the computational experiments were performed on a MacBook (2015) with 1,1 GHz Intel Core M dual and 8 Go 1600 MHz DDR3. The code was implemented using the Julia programming language in Atom editor under MacOS Big Sur version 11.6.5.

Table 2 : Results obtained by Guedes and Ribeiro (2011) with their own ILP model compared to the results recently obtained (with the MacBook) on the initial QIP model.

Teams (n)	Lower bound	Best to date	Guedes & Ribeiro		QIP-Gurobi	
			Result	Running time	Result	Running time in seconds
4	12	12	12	Couple of seconds	12	0.02
6	30	60	60	Few minutes	60	3.14
8	56	56	56	3 days	56	35.41
10	90	108	/	/	126	4024

Table 2 - Data from own calculations using the commercial solver Gurobi in Julia Programming (Bezanson et al., 2017)

Table 3 : COE values obtained with the three final versions of the ILP model : one with constraint (16), one with constraint (17) and one without any of these two constraints.

Teams (n)	Lower bound	Best to date	Final Model C(16)		Final Model C(17)		Final Model no fix	
			COE	Running time in seconds	COE	Running time in seconds	COE	Running time in seconds
4	12	12	12	0.01	12	0.01	12	0.01
6	30	60	60	0.17	60	0.12	60	0.84
8	56	56	56	2.90	56	3.35	56	6.94
10	90	108	138 130 126	92 1372 1543	138 130 126	51 751 3852	138 130 126	55 3045 3045

Table 3 - Data from own calculations using the commercial solver Gurobi in Julia Programming (Bezanson et al., 2017)

As can be seen in ***Table 3***, the use of symmetry breaking constraints (16) and (17) leads to a reduction in the running time when comparing the time to reach a COE value of 126 (which is the best value I could obtain for n=10 during my trials). However it is not clear whether the use of (16) or the use of (17) is more efficient. The model with constraint (16) is the fastest to reach 126 but for most of the resolution process, the model with constraint (17) is performing better as it reaches 138 and 130 before any of the other two models. So if I had decided to stop the resolution before 1543 seconds, I would have concluded that the model with constraint (17) is performing much better than the other two.

As anecdote, a few trials were realized on a PC x64 Intel® Core™ i5-8250U 8th generation CPU 1,6 GHZ, 1800 MHz, 4 hearts, 8 processors under Microsoft Windows 11. For n = 10, the solver reached 124 in 19 938 seconds which is about 5 hours and 31 minutes. This result could not be achieved on the MacBook even after more than 9 hours of running time. This is an example of what I mentioned at the end of section 2.3. One cannot really explain the difference in the results that can be obtained with the same solver on the same model from one computer to another.

3.5 Conclusions on results

From this first part on ILP I came to the conclusion that progress in computer development and algorithmic developments in commercial solvers in ten years are mostly responsible for the improvement in the time of resolution. Indeed, the solver is faster on the original QIP model than on the ILP model of Guedes and Ribeiro. Then, if we compare the results obtained from the ILP model I developed, my *Final Model*, and from the one of Guedes and Ribeiro, it is difficult to determine if the improvements are due to the model structure proposed to the solver or to the improved performance of the solver itself. As I don't have the details on the ILP model of Guedes and Ribeiro, I cannot further investigate that part. In any case, the room for improvement in ILP appears to be limited. Modifications of the model that affect positively the resolution of small instances quickly fades away as instance size increases. Therefore, the use of heuristic methods seems to be the logical next step of this work.

Additionally, from one computer to another, results might differ but it seems clear that no significant difference in the resolution time, or in the result achieved, exists between the two machines. This small

difference is related to technical features of the computers that are quite complex to measure or explain.

Lastly, any significant progress regarding the optimal resolution of the problem is more likely to be the result of an improvement on the algorithmic side than purely on the computer development side.

4 Generate initial solutions

4.1 Context

For the second part of this thesis the objective was to tackle the problem of COE minimization using heuristic techniques. These methods are likely to be faster and more efficient than the exact methods developed in chapter 3.

Heuristic methods are based on empirical insights and aim to rapidly find good, but not necessarily optimal, solutions to a problem. They are non-exact techniques, meaning they are not purely mathematical but rather based on observed natural processes, empirical experience and even randomness, whereas ILP is an exact technique based on mathematical principles and backed up by theory. Optimality of the solution found can be proven if exact procedures are employed but this is not true for heuristic methods, although they often find better solutions in a reasonable time (Gendreau & Potvin, 2019, p. xii). Beyond heuristic are metaheuristic methods. These procedures are more sophisticated. They are designed to escape from local optimality by combining local search with different strategies to move away from the areas of the solution space where the procedure might end up stuck. The development of such procedures is often more challenging in terms of coding skills.

In the literature on COE optimization, a few examples of the techniques I encountered the most are tabu search, SA and Variable Neighborhood Search (VNS). All of them are metaheuristic procedures that have proven their worth in combinatorial optimization. In particular, in their paper, Guedes and Ribeiro (2011) used a metaheuristic procedure that consists in two main phases. In the first one, many initial solutions are generated by modifying a unique starting fixture built either with the circle method or with a second method applicable when n is a multiple of four. Then, right after, the initial fixture undergoes a local search procedure called VND (Variable Neighborhood Descent). In a predefined order, the calendar is improved with respect to four different neighborhoods : Team Swap[3] (TS), Round Swap[4] (RS), PTS and PRS. These are the classic local neighborhoods structures often used in RRT optimization. In the second phase, an ILS based on different perturbations of the hundred initial solutions they got from the previous stage is performed. These perturbations include the exploration of the GR and two other destructive neighborhoods, Rows Destruction[5] and Columns Destruction[6].

What I could observe about the way Guedes and Ribeiro developed their heuristic procedure is that the starting point, the very initial solution, is the one obtained with the circle method (or the other method only applicable for instance size which are multiples of four). The thing is that these methods only provide one unique calendar per instance size. Building different starting fixtures is not easy at all. For small instance sizes it might be achievable but as the number of teams grows, it becomes trickier. Especially since the calendar must be compact. Also, remember that instances with less than 10 teams were solved rapidly with the ILP algorithm in the first part of this work. Therefore, the need for heuristics methods arose especially for instances greater than or equal to 10 teams.

[3] A definition of the TS move can be found in ***Appendix B***.
[4] A definition of the RS move can be found in ***Appendix B***.
[5] A definition of the Rows Destruction move can be found in ***Appendix B***.
[6] A definition of the Columns Destruction move can be found in ***Appendix B***.

Another interesting information from the literature, is the observation Januario and Urrutia (2016) present in their paper. As mentioned in section 2.2.3. - i., the solution space of RRT problems is complex and the classic neighborhoods structures used to explore it do not fully connect the solution space.

From these two observations, one question came out :
Is there another way to generate several and varied initial solutions from scratch?
And what is also implied by this question is trying to skip over the circle method as a starting point to the generation of initial solutions.

4.2 Making the most out of the solver

Given the difficulty of connecting the different solutions from the solution space, ideally, initial solutions should be of different structures. This way, they will hopefully come from more sparse parts of the solution space. Then, we can reasonably think of being able to cover more of it when applying metaheuristics to optimize initial schedules. To resume the idea, if we cannot start from a unique point and spread out to reach any point of the solution space, then we can try to start from various, distinct, and hopefully scattered points, and search around these starting fixtures.

In chapter 3, the solver is used to find an optimized solution to the integer linear formulation of the problem. Results show that, even for the most improved version of the model, the resolution time is too long and the problem cannot be solved to optimality for instances of 10 teams and more.

Nevertheless, the solver becomes interesting at this point as it can be stopped at any time and the incumbent solution can be retrieved easily. Which means we can use it to generate various solutions and use them as a starting point for future optimization.

Thus, the following idea emerged. It would be interesting to use the solver and weights to produce a set of initial solutions and then recompute these solutions COE value with weights equal to one. The goal is to produce the most varied collection of initial solutions. In principle, the weighting will influence the construction of the calendar toward certain configuration. If the weights are different from one iteration to another, the solver should be able to issue various calendars. The next section is about how to generate these initial fixtures with the solver.

4.2.1 Playing with weights matrices

To overcome the long running time of the solver on large instance sizes, the objective function from *Model 0* has been simplified by removing the square on variable $Z_{i,j}^2$ and then linearized using the standard linearization method. This new function forces a small total COE value but the missing square on $Z_{i,j}$ removes the focus on sharing evenly the total value of the COE among the teams. The goal was not to obtain very good solutions but rather to generate varied “random” schedules. To do so, in addition, the COEs are weighted in the model. Note that “random” is not exactly the appropriate word as the objective function still orients the solver toward schedules of quite small COE value. Let’s say that weights are there to give the random ingredient to the schedules but the simplified objective function helps the solver to stay on track with respect to the final goal of minimizing the COE value.

The model also includes symmetry breaking constraints to fasten the resolution. The ILP model proposed to the solver is as follows :

Simplified ILP Model

Parameters :
Number of teams: n
Weights : $Wei_{i,j}$ = random value* for i, j = 1, …, n : i ≠ j

Variables :
$Y_{i,j,k}$ = 1 if team i plays team j in round k, for i, j, k = 1, …,n.
0 otherwise.

$U_{i,j,l,k}$ = 1 if team i gives a COE to team j through team l from round k to round k+1, for i, j, l, k = 1, …, n.
0 otherwise

Objective function :
Min $\sum_{i=1}^{n}\sum_{j=1}^{n}\left[Wei_{i,j} * (\sum_{l=1}^{n}\sum_{k=1}^{n-1} U_{i,j,l,k})\right]$

Constraints :
For i, j, l = 1, …, n ; k = 1, …, n-1:
If i ≠ j ≠ l ≠ i : $U_{i,j,l,k} \geq Y_{i,l,k} + Y_{j,l,k+1} - 1$ (10)
Otherwise, $U_{i,j,l,k} = 0$,

$Y_{i,j,k} = Y_{j,i,k}$, for i, j, k = 1, …, n. (3)

$\sum_{i=1}^{n} Y_{i,l,k} = 1$, for l, k = 1, …, n. (4)
$\sum_{k=1}^{n-1} Y_{i,l,k} = 1$, for i, l= 1, …, n : i ≠ l. (5)

$Y_{i,i,k} = 0$, for i, k = 1, …, n. (6)
$Y_{i,j,n} = Y_{i,j,1}$, for i, j = 1, …, n. (7)

$Y_{i,j,k} \in \{0,1\}$, for i, j, k = 1, …, n. (8a)

$Y_{1,k+1,k} = 1$ for k = 1, …, n-1. (17)

In this simplified version of the problem, weights, represented by the parameters in $Wei_{i,j}$, are randomly generated beforehand. $Wei_{i,j}$ is a $n \times n$ matrix that can be symmetric or not. Its entries are either integer or non-integer values which can be in the interval $[1\ ;\ 2n]$ for the integer ones or $[1\ ;\ n]$ for the others. Let's consider the following 3 options of weights matrix :

1. Symmetrical $Wei[i,j]$:

for i in $1:n$

for j in $i+1:n$

$Wei[i,j]$ = random value in $[1;n]$

$Wei[j,i] = Wei[i,j]$

end

end

2. Unsymmetrical $Wei[i,j]$:

for i in $1:n$

for j in $1:n$

$Wei[i,j]$ = random unit value in $[1;2n]$

end

end

3. Inverse weights

for i in $1:n$

for j in $i+1:n$

$Wei[i,j]$ = random value in $[1;n]$

$Wei[j,i] = 1/Wei[i,j]$

end

end

In this third option, there is a relation between $Wei[i,j]$ and $Wei[j,i]$, they are the inverse of each other. There is some kind of logic behind : if weights are used to represent the relative strength between i and j and if $Wei[i,j] \neq 1$, then it means that i has a certain positive (or negative) effect on j and the reverse situation is assumed to be true for the effect j has on i. This should influence the solver toward fixtures where COEs between pairs of teams showing the larger values are avoided. For example, if $Wei[i,j] > Wei[j,i]$, then COEs from j to i are more likely than COEs from i to j.

In order to create more than one schedule, the random generation of weights and the solver are incorporated into a loop and a time limit is imposed on the solver. When the solver finishes its run, each solution fixture produced with the weights matrix is registered. Then, the COE value is computed with unit weights values.

For example, if the time limit is set to 5 seconds and the loop is set to perform five iterations, five schedules are built in 25 seconds. It can reasonably be assumed that those are varied given that the weights matrix is renewed at every iteration. In ***Appendix E,*** the reader can have a look at an example of five schedules, and their weighted and unweighted COE values, generated in 25 seconds, for n = 10.

Julia Programming files used for the next numerical results are : GenerateIS_SolverOption1.jl, GenerateIS_SolverOption2.jl, GenerateIS_SolverOption3.jl.

4.2.2 Numerical results

In this new section I tried to evaluate the quality of the initial solutions produced with the solver with respect to each of the three different ways of generating the weights matrix. To this end, the solver was run for 20, 10, or 5 iterations with each of the three options and on pair instance sizes (n = 10, 12, 14, 16, 20, 24). A limit on the resolution time of the solver that guarantees the generation of at least one, two or a few solutions has been arbitrarily decided for each instance size. The larger n, the longer the time limit on the solver. The number of iterations has also been chosen arbitrarily so that the total running time would not be excessively long (about 1 hour and 15 minutes for the largest instance).

As mentioned before, we are looking for varied solutions. The easiest way to check the variety of schedules is simply by evaluating their COE value. It would be complicated and time consuming to analyse and compare different schedules configurations. Plus, I could not know if two different schedules might display the same COE value or not. However, I can be pretty sure that two different COE values cannot correspond to the exact same schedule. Therefore, for each set of solutions generated, every unique COE value was counted. The goal was to see if the options generate varied sets of initial solutions and to choose which one(s) to keep for the trials in the next chapter.

In ***Table 4*** below, **n** corresponds to the instance size. **Weights option** specifies which of the three options presented above is used to generate the matrix of the weights for the run. **Time limit (in seconds)** is the running time limit of the solver for each iteration. **Iterations** is the number of initial solutions produced during the run concerned. **Total time** indicates the total running time. **Unique count** is the number of unique COE values out of the number of solutions produced at the end of the run concerned. **Unique count among 3 options** shows the number of unique values that came out only with the option used among all the options. And finally, **Total unique count in 3 options** is the total number of unique solutions that came out of the 3 runs of the tested options combined.

Table 4 : Count of unique solutions in different sets of initial solutions obtained with each of the three weights matrix options, per instance size n.

n	Weights option	Time limit (in seconds)	Iterations	Total time	Unique count	Unique count among 3 options	Total unique count in 3 options
10	1				16/20	7	
10	2	5	20	1 min 40	16/20	7	32/60
10	3				14/20	6	
12	1				19/20	14	
12	2	5	20	1 min 40	18/20	12	39/60
12	3				13/20	5	
14	1				17/20	9	
14	2	30	20	10 min	19/20	13	41/60
14	3				16/20	9	
16	1				13/20	10	
16	2	30	20	10 min	17/20	14	37/60
16	3				16/20	8	
20	1				10/10	7	
20	2	300	10	50 min	10/10	9	27/30
20	3				10/10	8	
24	1				4/5	4	
24	2	900	5	1h15	5/5	5	14/15
24	3				5/5	5	

Table 4 - Data from own calculations using the commercial solver Gurobi in Julia Programming (Bezanson et al., 2017)

Considering the instance sizes between 10 and 16, the numbers from the table above show us that each of the three options can produce between 65% and 95% of different initial fixtures per run. Additionally, the three options all bring unique COE values that the two others do not. It appears that options 1 and 2 often produce more unique solutions than the third option. For the instances of sizes 20 and 24, less iterations where performed because the solver needs much more time to generate solutions. They are almost all different, which is not so surprising given the low number of iterations and the fact that possible arrangements of the teams in the calendar matrix exponentially increase with n. In general, we can see that option 2 gives always quite a good number of unique COE values but results do not clearly indicate that one of the three options is much worse or much better than the others.

Limitations and conclusion on the results

Given the random nature of the weights matrices, the arbitrary choice of iterations, weights matrix generation and running times, and the unknowns about the solver, those results are to be considered carefully. Also, after these trials, I have noticed that with weights matrix option 2, the time limit of 900 seconds on the biggest instance size (n = 24) does not always guarantee sufficient time for the solver to reach a solution. Depending on the weights generated, the solver sometimes needs a lot more time. It seems that with weights that are in a large interval (i.e. [1 ; 2n] for option 2 compared to [1 ; n] for the other two options), it makes it more difficult to find a feasible solution.

The solver is a good alternative concerning the issue of how to build solutions. It overcomes the difficulty associated with the organization of the elements in the calendar matrix. But, as far as time is concerned, it clearly becomes much less efficient as the instance size grows.

My conclusion on these results is that the solver allows to generate various unique solutions. Each option brings unique solutions that the other two do not. Therefore, it might be interesting to keep them all for the next tests in order to increase the probability of finding a good solution at the end.

5 Simulated Annealing metaheuristic

5.1 Description of the SA metaheuristic

An original way to generate initial solutions using the solver has been considered in the last section. In this fifth chapter, a SA metaheuristic procedure has been used to improve the COE value of initial fixtures. As mentioned before, SA is often a successful method when it comes to finding good solutions to discrete problems with complex, large configuration spaces.

In the SA procedures, “moves” are performed to explore the neighborhood of an initial solution. This means an initial solution is slightly, randomly modified to generate a new one, that is close to it in the solution space, his neighbor. The move can be defined as the modification applied to that initial solution. During the SA procedure, many neighbors are generated one after the other. Depending on the solution value a neighbor results in, it is either kept as incumbent solution or not. The next move is then applied to that solution. This is similar to the process of local improvement except SA allows deteriorations of the current solution, in a controlled way, in order to escape from local optima. It goes further than simple local search strategies, the evolution of the incumbent solution is guided by a set of rules.

As a reminder to the reader, more practical details on the implementation of the SA method are presented in ***Appendix D***.

5.2 Game Rotation neighborhood

Apart from existing moves such as switching rounds or teams and their derivatives, it is difficult to find more sophisticated neighborhood structures to explore the solution space of the COE in SRRT problem. The recovery of a feasible fixture, after a modification, is complex to visualize and to translate into programming language. The slightest change in one match of a calendar inevitably induces other changes, possibly in any part of the matrix. As was judiciously pointed out to me, it is similar to a sudoku game. Once a few numbers are placed in the boxes, there is only one possible feasible solution. You cannot produce a different solution unless you fundamentally change the structure of the grid, which requires more than one or two modifications. In the same way here, it is all about finding a move that can change the structure of the solution. In the literature, the Game Rotation move appears to be capable of fundamentally changing the structure of a fixture as it goes further than simply switching rounds or all the opponents of two teams. It randomly choses one game to be played in another round and then recovers a feasible solution by adapting the schedule around that change.

The definition of this neighborhood relies on the principle of ejection chain already introduced in section 2.2.3. - iv. The whole move is a logical sequence of changes performed on an initial solution. It starts with a (randomly selected) game that is moved to another round (randomly selected too) than initially scheduled, ejecting the game that was initially planned in that selected round of destination.

From there, adjustments to recover a feasible solution are successively triggered. The final result is a different solution configuration.

Hereafter is an explanation of the game rotation move and the code created to implement it (see the file GameRotationMove.jl). The explanation is based on a concrete example (see ***Figure 3*** to ***Figure 11***), modeled using graph theory, and adapted from (Ribeiro, March 2019).

Game Rotation Move

To start with, different matrices and vectors are created to store variables and parameters throughout the process.

- IS is a $(n \times n-1)$ matrix that stores a copy the initial solution.
- NS is a $(n \times n-1)$ matrix that will store the new solution obtained with the GR move. It is initially set with zero values.
- $rounds$ is a vector of undetermined length that will record every round from the initial solution that will be modified throughout the process, in a chronological order.

The idea with NS and IS is that IS will never be modified and NS will only register the changes compared to IS. At the end of the move, every missing elements in NS will be copied from IS. This way the initial solution remains untouched and can be used to refer to the teams when building the new solution.

Two "true/false" parameters are used to control the flow of the loops :

- GR is set to "true" ("GR" stands for "Game Rotation") ;
- s is set to "false" ("s" stands for "stop").

Before launching the loops that will perform the GR move, the game and the round in which it will be rescheduled are randomly chosen:

- $t1$ is a random unit value between 1 and n, this is the first team composing game that will be rescheduled in another round ;
- r is a random unit value between 1 and $n-1$, this is the round from which the game between $t1$ and his opponent will be removed ;
- $t2 = IS[t1, r]$, that is to say $t2$ is the opponent of $t1$ in round r ;
- dr is a random unit value $\in \{1\ ;\ n-1\} \setminus r$, this is the round in which the game $t1\ -\ t2$ will be rescheduled ("dr" stands for "destination round") ;
- dr is set as the first element of $rounds$;
- index i is set to 1.

Then,

1) Enforce the game $t1 - t2$ (i.e. game $1 - 3$), initially scheduled in r (i.e. round 5), to be played in round dr (i.e. round 2). Note that this leaves $t1$ and $t2$ with no opponent in round r (i.e. round 5).
   → Update the matrix NS accordingly.

Figure 3 : Game rotation move illustration (1).

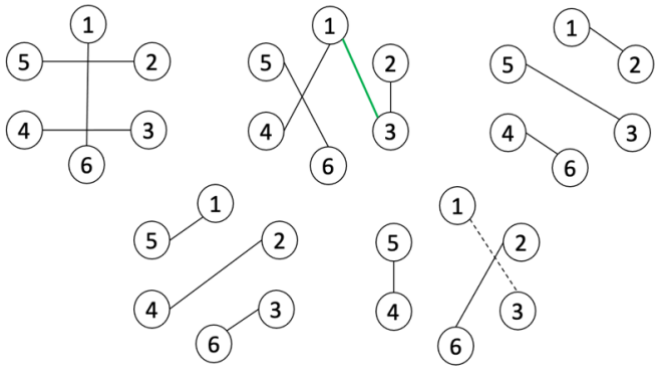


Figure X.1 - Game rotation move illustration (1). Adapted from "Applications of metaheuristics to optimization problems in sports" [Conference presentation], by Ribeiro, C. C. (2019, March), 8th International workshop on operations research, La Havana, Cuba.

2) In round dr (i.e. 2), the respective initial opponents of $t1$ and $t2$ (i.e. teams 4 and 2) will be forced to play each other to maintain a complete round in dr (i.e. round 2).
→ Update the matrix NS accordingly.

Figure 4 : Game rotation move illustration (2).

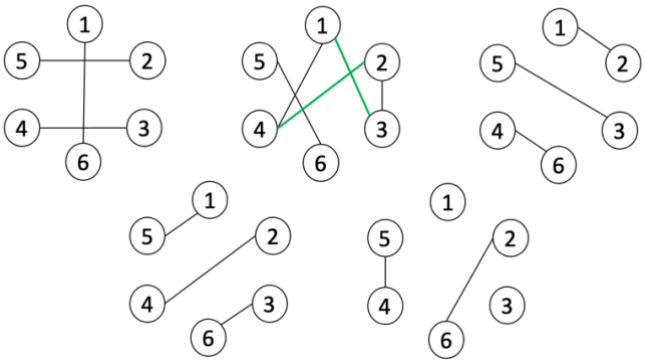


Figure X.2 - Game rotation move illustration (2). Adapted from "Applications of metaheuristics to optimization problems in sports" [Conference presentation], by Ribeiro, C. C. (2019, March), 8th International workshop on operations research, La Havana, Cuba.

3) In round dr (i.e. round 2), the two games initially scheduled between $t1$ and his opponent and $t2$ and his opponent are ejected as a consequence of the previous step. They will have to be rescheduled sometimes somewhere in the tournament.

Figure 5 : Game rotation move illustration (3).

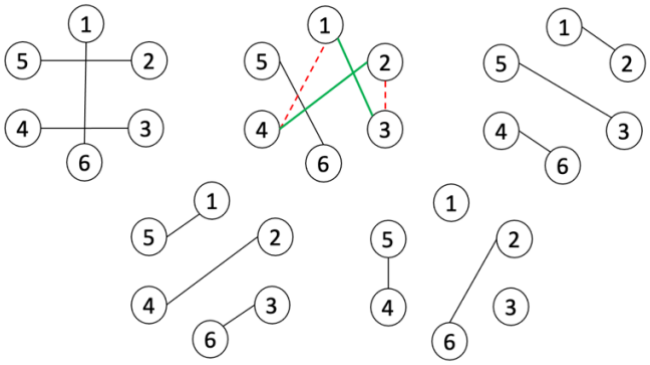


Figure X.3 - Game rotation move illustration (3). Adapted from "Applications of metaheuristics to optimization problems in sports" [Conference presentation], by Ribeiro, C. C. (2019, March), 8th International workshop on operations research, La Havana, Cuba.

4) As a result of 2), in the initial solution, the game between the opponents of $t1$ and $t2$ (i.e. teams 4 and 2) has to be ejected (the same game cannot be scheduled twice). To do so, the round in which it was scheduled in the initial solution has to be found, let's call it round R (i.e. round 4). Then, the game is ejected and the teams (4 and 2) have no opponent anymore in this round.

Figure 6 : Game rotation move illustration (4).

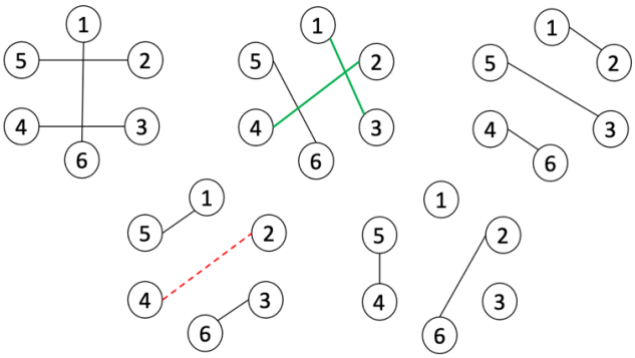


Figure X.4 - Game rotation move illustration (4). Adapted from "Applications of metaheuristics to optimization problems in sports" [Conference presentation], by Ribeiro, C. C. (2019, March), 8th International workshop on operations research, La Havana, Cuba.

5) The game initially scheduled in round dr (i.e. round 2) between $t1$ and his opponent is a pending game. It has to be rescheduled somewhere. As his opponent (i.e. team 4) has just been "freed" in round R (i.e. round 4), the game (i.e. game $4-1$) is rescheduled there. This game is no longer pending and the initial opponent of t1 (i.e. team 4) is no longer free in round R (i.e. round 4).
→ Update the matrix NS accordingly.

Figure 7 : Game rotation move illustration (5).

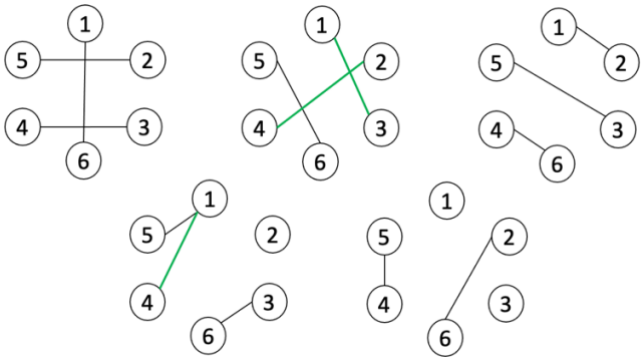


Figure X.5 - Game rotation move illustration (5). Adapted from "Applications of metaheuristics to optimization problems in sports" [Conference presentation], by Ribeiro, C. C. (2019, March), 8th International workshop on operations research, La Havana, Cuba.

6) Again, the initial opponents of the teams composing the game rescheduled will be forced to play each other (i.e. game $5-2$) in order to maintain a complete round in R (i.e. round 4). There is no more free team in this round.
→ Update the matrices NS accordingly.

Figure 8 : Game rotation move illustration (6).

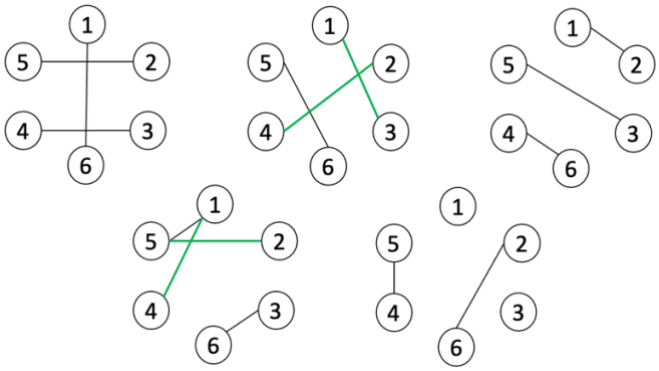


Figure X.6 - Game rotation move illustration (6). Adapted from "Applications of metaheuristics to optimization problems in sports" [Conference presentation], by Ribeiro, C. C. (2019, March), 8th International workshop on operations research, La Havana, Cuba.

7) At this point, two cases may occur :
   a. $R = r$:
      Then it means the move is over as the last games that had to be rescheduled have all been reprogrammed in that round. The loop is closed.
   b. $R \neq r$: thus, in round R (i.e. round 4), as a result of step 6), one game (i.e. game 5-1) initially scheduled is ejected and will need to be rescheduled sometimes somewhere else.

Figure 9 : Game rotation move illustration (7).

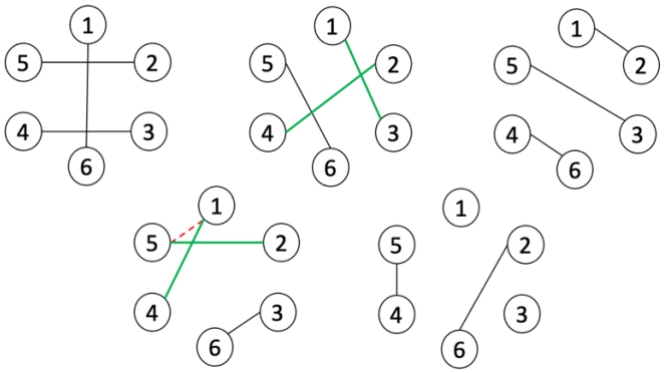


Figure X.7 - Game rotation move illustration (7). Adapted from "Applications of metaheuristics to optimization problems in sports" [Conference presentation], by Ribeiro, C. C. (2019, March), 8th International workshop on operations research, La Havana, Cuba.

8) Also as a result of step 6), the game that has just been forced (i.e. game $5-2$) has to be ejected from the initial solution (it cannot be scheduled twice). To do so, find the round R (i.e. round 1) in which it was initially scheduled. The two teams composing that game are now free in that round.

Figure 10 : Game rotation move illustration (8).

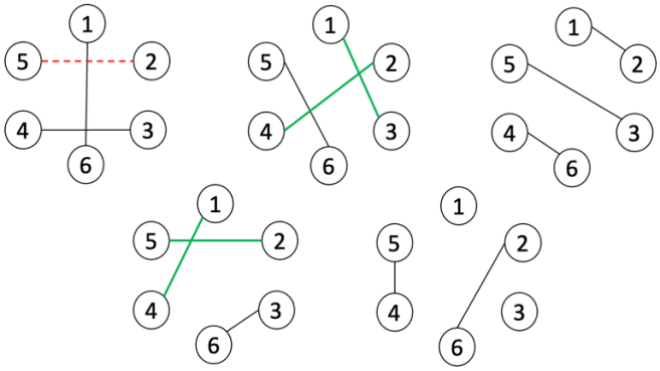


Figure X.8 - Game rotation move illustration (8). Adapted from "Applications of metaheuristics to optimization problems in sports" [Conference presentation], by Ribeiro, C. C. (2019, March), 8th International workshop on operations research, La Havana, Cuba.

From here, the loop starts over. The last ejected game (i.e. game $5-1$) is rescheduled in the round (i.e. round 1) where one of the two opponents (i.e. team 5) composing that game is free.
→ Update the matrix NS accordingly.

Figure 11 : Game rotation move illustration (9).

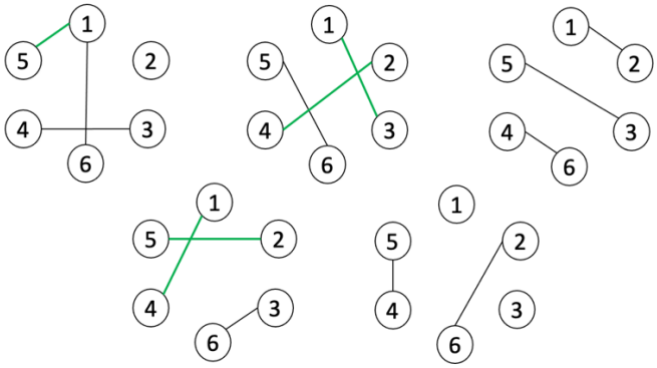


Figure X.9 - Game rotation move illustration (9). Adapted from "Applications of metaheuristics to optimization problems in sports" [Conference presentation], by Ribeiro, C. C. (2019, March), 8th International workshop on operations research, La Havana, Cuba.

The process keeps going through steps 6 to 8 until it reaches step 7) - a. and stops

5.3 Setting the initial temperature

As it can be understood from ***Appendix D***, in the SA procedure, the temperature plays an important role in the function that determines the probability of accepting a deteriorating move. This temperature will decrease along with the progression of the procedure. This implies that the initial temperature must be set at some point in the beginning. It must be high enough to go through the process without getting too fast too close to zero.

The initial temperature can be defined by the following expression :

$$T_0 = (-\frac{\Delta F_1}{log\ 0.5}),$$

Where $\Delta F_1 = F(x) - F(x^1)$, $F(x)$ being the value of the incumbent solution, which is the initial solution at the beginning of the procedure, and $F(x^1)$ being the value of the first neighbor produced.

In this case, the probability of accepting a deteriorating move during the first L (plateau length) iterations will be of 0.5.

However, depending on the first move generated, ΔF_1 can be of many different values, small, large or even zero. A small value would result in a particularly low temperature as the value $-log(0.5) \cong 0.3$ is constant.

Since the value of the solution resulting from the first GR move can't be controlled, two tricks were implemented :

The first trick consists in starting over the move on the initial solution until the solution value obtained is at least strictly lower than the value of the initial solution. This way we ensure that the first move is

going to be accepted later on in the procedure and that it will improve the incumbent solution. When the move has been performed, the COE value of the neighbor solution is assessed and if it is strictly lower than the COE value of the initial solution, the procedure moves on to the next step. If not, we are back to the initial solution and the move is performed again.

In the next step, the initial temperature will be set. In order to fix a temperature that is neither too high, neither too low, the second trick comes into play. The value of ΔF_1 has been calculated based on a sample of ΔF_n (see file DeltaFComputation.jl). The GR move was performed many times on three different initial fixtures. At each iteration, when $F(x) - F(x^z) \neq 0$, the absolute value of ΔF_z was calculated. When a thousand ΔF_z are obtained, the mean ΔF_z is computed. Note that during the process, each move solution becomes the incumbent solution, there is no rules of acceptation. The mean of the three mean delta values, rounded up, is the value that will be used as ΔF_1 in the trials of section 5.5. This operation was carried out for each instance size. ***Table 5*** below displays the values found for each of them.

To be consistent, for each instance size, the initial fixture is generated by the solver with weights matrix option 1 (arbitrary choice) and the same time limit as the one specified in

Table 4 is applied. The initial solution from the solver is run through the code performing the thousand moves.

Table 5 : Mean of ΔF_n over a thousand iterations per instance size n.

Instance size n	Initial solution COE value	Mean ΔF_n over 1000 successive moves	Mean of the means
10	164 166 180	18.1 17.4 14.5	16.6 → 17
12	262 278 310	17 21.5 48.5	29
14	384 406 422	26.6 46.7 58.1	43.8 → 44
16	502 564 566	32.3 87.6 87.8	69.23 → 70
20	796 800 888	53.2 42.7 126	73.96 → 74
24	1342 1344 1382	235.7 246.8 276.6	252.97 → 253

Table 5 - Data from own calculations using Julia Programming (Bezanson et al., 2017)

From this table, we can observe two things. The first one is that the mean variation in COE value from an initial solution to another of the same instance size can largely vary. For example, it more than doubles for n = 12, 14, 16 and 20 from the first to the third mean ΔF_n registered. Secondly, the delta COE value might seem to be somewhat evolving proportionally to the initial solution value, as suggested by instances 12, 14, 16 and 24. Values lead us to think that the higher the initial COE value, the larger the mean ΔF_n. But, instances 10 and 20, especially, do not follow that pattern.

It is difficult to draw conclusions on the mean ΔF_n value from these results. However it gives us the information that applying the GR can really lead to very different solutions values. This is quite reassuring for the next trials and reinforces the idea that starting from many different solutions and applying a move that changes the solution structure could quickly lead to good results if some of these starting solutions happen to be "good candidates". By this I mean they can be successively transformed by the SA procedure to eventually become a good solution.

5.4 Tuning the parameters of the SA

The course of the SA procedure relies on the choices made regarding the value of its parameters. The quality of the solution found at the end will largely depend on those settings. If the SA is not well

configured, it might stop too early and miss on a better solution, or too late resulting in an unnecessary long running time.

Here is the list of parameters that needs to be set before launching the SA procedure (see the file SA_GRmove.jl) :

- *a* is the maximum deterioration percentage. The incumbent solution can be deteriorated by at most *a*% of its value by the next move ;
- *L* is the plateau length. This is the number of iterations between each temperature diminution. This parameter is set to be equal to $n \times (n-1)$;
- *cf* stands for cooling factor and $\in \{0; 1\}$. The temperature is multiplied by *cf* after a number of iterations determined by the plateau length *L* ;
- *E* is the moves acceptation coefficient. This coefficient sets the minimum percentage of moves that have to be accepted during a plateau length of iterations ;
- *stop* refers to the stopping parameter. Maximum number of plateaus during which the solution is not improved and less than *E*% of moves are accepted ;
- *TimeLimit* is the maximum running time allowed for the SA procedure, in seconds ;
- *Delta* is the mean ΔF_1 that will define the initial temperature, as explained above.

Before running any tests, these settings have to be refined. In order to decide on the value of these parameters, I tested different combinations of their respective values on the same initial solution for each instance size. The combination of values that appeared to give the best results was chosen for the trials in the next section. I also had a look at the graphical representation of the course of the procedure to get a better idea on how to adjust the values.

I won't show many details on how I tested the different parameters but I will show an example : For n = 10, starting from the same initial solution, we can try the following two different settings configurations where only one parameter is different :

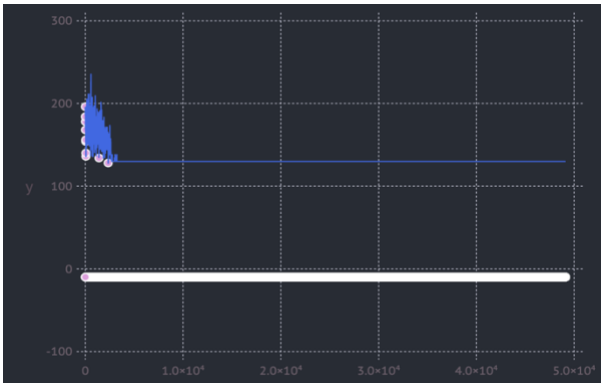
A. a = 0.5 ; L = 90 ; cf = 0.95 ; stop = 3 ; E = **0.2** ; TimeLimit = 8 ; Delta = 17.
B. a = 0.5 ; L = 90 ; cf = 0.95 ; stop = 3 ; E = **0.5** ; TimeLimit = 8 ; Delta = 17.

Results and graphs associated :

A.

Initial solution : 196
Running time : 8 sec.
Final solution : 128

Graph 1 : Evolution of incumbent solution with parameters configuration A.

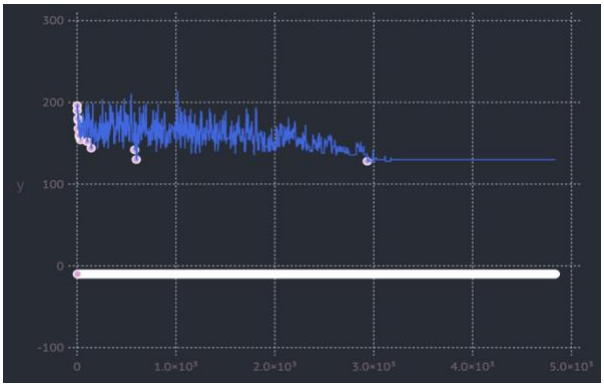


Graph 1 – Data from own calculations, graphs are from Julia Programming (Bezanson et al., 2017).

B.

Initial solution : 196
Running time : 1.63 sec.
Final solution : 128

Graph 2 : Evolution of incumbent solution with parameters configuration B.



Graph 2 – Data from own calculations, graphs are from Julia Programming (Bezanson et al., 2017).

In these two graphs, the blue line corresponds to the evolution of the incumbent solution, it tracks every accepted moves. The purple points correspond to the first time a best solution value appears during the process. When there is no improvement of the best solution value, the point is set to -10 in order not to overload the graph and to stay in line with the evolution of the incumbent solution.

For the same final solution, parameters configuration B is faster, performs less iterations and displays a graph that is more pleasant and representative of the SA procedure. The process slows down progressively and after a few iterations with no improvement and no accepted move, it stops. Whereas

graph A shows unnecessary long running time, it is concentrated on a small portion of the horizontal axis. We can observe that the solution is found after about 3 x 10^3 iterations in both graphs.

All the other parameters where adjusted following the same methodology. The final configurations per instances size are as follows :

- n = 10 : a = 0.5 ; L = 90 ; cf = 0.95 ; stop = 3 ; E = 0.5 ; TimeLimit = 8 ; Delta = 17.
- n = 12 : a = 0.5 ; L = 132 ; cf = 0.95 ; stop = 3 ; E = 0.5 ; TimeLimit = 8 ; Delta = 29.
- n = 14 : a = 0.5 ; L = 182 ; cf = 0.95 ; stop = 3 ; E = 0.5 ; TimeLimit = 15 ; Delta = 44.
- n = 16 : a = 0.5 ; L = 240 ; cf = 0.95 ; stop = 3 ; E = 0.5 ; TimeLimit = 15 ; Delta = 70.
- n = 20 : a = 0.5 ; L = 380 ; cf = 0.95 ; stop = 3 ; E = 0.5 ; TimeLimit = 25 ; Delta = 74.
- n = 24 : a = 0.5 ; L = 552 ; cf = 0.95 ; stop = 3 ; E = 0.5 ; TimeLimit = 25 ; Delta = 253.

5.5 Numerical results

The following final trials were realized with the file SA_GRmoveLoop.jl. Tests were run on three sets of five initial solutions each, for every instance size. Each set was generated by the solver with a weights matrix corresponding to one of the three options tested in the previous chapter. Time limits on the generation of the initial solutions are consistent with

Table 4. The SA is performed three times, in a loop, on each initial solutions. The best result is kept as final solution for the run. Given the stochastic nature of the SA, from one run to another, the result might change, for better or for worse. Based on the different tests realized, applying the SA three times on a same initial solution gives a better chance to reach a better result than just running the SA once for a longer period of time. Doing it more than three times rarely gave a better result. The time limit on the SA is quite short, between 8 and 25 seconds depending on the instance size, which makes it possible to repeat the procedure three times without it to be too time consuming. The max running time until obtaining the final result from one initial solution will thus be between 24 and 75 seconds.

Results are presented in ***Table 6*** each line of the table corresponds to the best result out of one set (i.e. the best result out of five initial solutions). For each line, the instance size, the weights matrix option, the lower bound (L.B.), the best solution value ever found, the initial solution (I.S.) COE value, the time limit in seconds, the running time in seconds, and the final solution (F.S.) COE value, are provided. The line corresponding to the best result found for each instance size is in bold type. The final total time (including the generation of the set of initial solutions) is given. The final solution configuration and a graph showing the evolution of the procedure are provided in ***Appendix F***.

Table 6 : Results of the SA procedure, for different instance sizes, on initial solutions built with the solver according to the three weights matrix options considered.

n	Weights	L.B.	Best	I.S. COE	Time limit (in sec.)	Time (in sec.)	F.S. COE	Total time	Graph N°
10	**1**	90	108	**216**	3x8 = 24	**2.15**	**108**	**27.15 sec.**	**3**
10	2			172		2.36	124	/	/
10	3			174		2.49	114	/	/
12	**1**	132	160	**288**		**5.46**	**180**	**30.46 sec.**	**4**
12	2			442		4.12	184	/	/
12	3			312		3.94	182	/	/
14	1	182	234	398	3x15 = 45	7.47	256	/	/
14	2			414		8.00	256	/	/
14	**3**			**396**		**6.62**	**254**	**10 min. 6.62 sec.**	**5**
16	1	240	240	546		11.2	338	/	/
16	2			682		11.8	338	/	/
16	**3**			**1136**		**12.06**	**334**	**10 min. 12.06 sec.**	**6**
20	1	380	380	890	3x25 = 75	28.9	552	/	/
20	**2**			**944**		**26.7**	**542**	**50 min. 26.7 sec.**	**7**
20	3			852		20.1	546	/	/
24	1	552	664	1296		72.8	802	/	/
24	2			1250		60.7	810	/	/
24	**3**			**1348**		**75**	**792**	**1 hour 15 min. 75 sec.**	**8**

Table 6 - Data from own calculations using Julia Programming (Bezanson et al., 2017), except the values in column "Best" which are reproduced from "A tabu-search for minimising the carry-over effects value of a round-robin tournament", by Kidd, M. (2010), Orion (Johannesburg, South Africa), 26(2), 125–141.

Results confirm that keeping the three types of weights matrices to generate the initial solutions was not a bad decision. The best solution found for n = 10, 108, corresponds to the best mentioned so far in the literature. The other solutions are not so good but they are still close to the ones obtained by Guedes and Ribeiro (2011), for n = 10,…,16. In particular, for n = 10 and 14, the best solutions found match theirs on the unweighted instances. Major part of the total running time is because of the solver. Especially for n = 20 and 24, for which the running time is way longer than for $n < 20$.

5.6 Limitations and conclusion on the final results

Various arbitrary choices have been made regarding the generation of initial fixtures and these last tests. The results obtained probably depend on those choices. Moreover, the random nature of the move and the fact that only one neighborhood structure is implemented are certainly two other limiting factors that influence the final results. However, despite that, some good results were achieved without the need of a sophisticated metaheuristic procedure such as the one used in (Guedes and Ribeiro, 2011). Only one neighborhood has been implemented within a simple SA procedure.

The conclusion on those final results is that the approach of the solver coupled with a simple SA procedure exploring one neighborhood structure is working. Results are satisfying when comparing to

those obtained by Guedes and Ribeiro. The real drawback is clearly the time spent on the solver with the generation of initial solutions, which increases drastically with the instance size.
However, on the positive side, the solver is an interesting option when trying to overcome the issue of small connectivity of the solution space. We have no proof it produces unique solutions that are unattainable with the traditional local search and metaheuristic procedures, but neither do we have the proof it does not. What we know is that the solver is an exact method so the procedure of resolution is different from heuristic methods. This leads to think that, as a result, the solutions produced are probably different too.

Finally, as a substantial part of the process relies on randomness (the game rotation move is performed randomly), a few trials on each of the various initial solutions were necessary in order to reach good final solutions. A few fast runs turned out to do better than a slower run. As explained before, this is a very complicated problem with a complex solution structure. It is very difficult to gain insight into how to approach the problem. There is, yet, no indication on how to achieve optimality, which is after all a proper feature of NP problems.

6 Conclusion

The overall objective of this thesis was to explore optimization strategies to solve the problem of COE in round robin tournaments. The paper of Guedes and Ribeiro (2011) has been used as a benchmark and as a source of inspiration all along this work. Methodology and theoretical background on the subject have been introduced to the reader before entering into the practical part of this thesis.

To start with, in chapter 3, I focused on exact methods with the production of an ILP formulation of the problem. One of the objectives was to see how the present-day solver, Gurobi Optimizer 9.1.1, would perform compared to the commercial solver used by the authors about ten years ago. Results showed a considerable improvement in the running times, more likely due to progress in computer development than to the ILP formulation given to the solver. For instance sizes greater than eight teams, clearly, the solver has its limitations. Nevertheless, this chapter brings and explains a detailed ILP formulation of the problem that was not provided by Guedes and Ribeiro in their article.

For the second part of this thesis, in chapter 4 and 5, heuristic methods were envisaged to try to find better solutions and address larger instances of the problem. The goal was to see if there was an alternative to the traditional circle method that could produce various schedules to be used as starting solutions to metaheuristic optimization procedures. The outcome is that using the solver and weights matrices is definitely a way to produce more than one, and varied, solutions structures but it ends up being a time consuming alternative as the instance size increases. From such initial fixtures, good results were achieved with a simple SA procedure exploring only the GR neighborhood. This means the combination of the solver generated solutions and the SA showed some real potential, especially for instance size of 10 teams. The total running time to reach a final solution is mostly made up of the solver's running time, which is the major drawback of this strategy. Additionally, to increase the chances of reaching good solutions, the SA procedure has been applied more than once to the same initial solution.

The novelty brought through this work is the use of the solver and weights especially and only to generate initial solutions. The SA is applied to these same, but then unweighted, initial solutions. It came from the focus on diversifying them through a different strategy than (local) perturbations of the fixture produced by the circle method.

To conclude, future work on that matter could be valuable. The use of the solver as a source of initial solutions could be further investigated by trying to find a simplified objective function that allows a faster generation of solutions. As mentioned at the end of section 4.2.2, decisions regarding the generation of weights matrices also appear to be responsible for the time the solver needs to generate a feasible calendar. The quality of the solutions produced is impacted. Consequently, more in-depth research on weights matrices and their effect on the fixture generated might be another lead for future research.

VI. Appendices

A. Standard Linearization Method

The following theory is inspired from Asghari et al. (2022) and Glover (1975).

In a nonlinear model, let x_i and x_j be two binary variables. The term $x_i * x_j$ can be replaced by a new binary variable u_{ij} if the following constraints are added to the model:

$$u_{ij} \leq x_i \quad \text{(i)}$$
$$u_{ij} \leq x_j \quad \text{(ii)}$$
$$u_{ij} \geq x_i + x_j - 1 \quad \text{(iii)}$$
$$u_{ij} \geq 0 \quad \text{(iv)}$$
$$u_{ij} \in \{0,1\} \quad \text{(v)}$$

The idea is to replace the cross product of the two binary variables by a unique binary variable satisfying new constraints. These ensure the new variable will take the right value for the product it replaces.

In the specific case of the COE minimization problem considered in this thesis, the theory from above can be applied as follows (example for the first linearization applied, see *Model 1* in section 3.2.1.) :

The two binary variables are $Y_{i,l,k}$ and $Y_{j,l,k+1}$, for $i, l, j = 1, \dots, n\,;\ k = 1, \dots, n-1$, the nonlinear term is $Y_{i,l,k} * Y_{j,l,k+1}$ and we can introduce a new binary variable $U_{i,j,l,k}$, for $i, j, l = 1, \dots, n\,;\ k = 1, \dots, n-1$, to take the place of the cross products. The following new constraints are added to the model :

$$U_{i,j,l,k} \geq Y_{i,l,k} + Y_{j,l,k+1} - 1$$
$$U_{i,j,l,k} \in \{0,1\}$$

For $i, j, l = 1, \dots, n\,;\ k = 1, \dots, n-1$.

As we are in a minimization problem, constraints (i) and (ii) are not necessary. Constraint (iv) is already implied by the binary nature of variable $U_{i,j,l,k}$.

B. Moves definitions

Some of the definitions below are entirely or partially reproduced from (Guedes and Ribeiro, 2011). Some are inspired from the same article.

- Columns Destruction (CD) :
  In the matrix used to depict a schedule in this thesis, some of the columns, representing the tournament slots, are erased and then reconstructed differently.

The solver is used to perform the reconstruction and constraints to ensure the schedule is rebuilt differently are added to the model (i.e. the ILP model in the paper of Guedes and Ribeiro (2011), which would correspond to my *Final Model* in section 3.3.2.)

- Partial Team Swap (PTS) :
  "*For any round r and for any two teams t1 and t2, let S be a minimum cardinality subset of rounds including round r in which the opponents of teams t1 and t2 are the same. A move in this neighborhood corresponds to swapping the opponents of teams t1 and t2 over all rounds in S.*"(p.659)

- Partial Round Swap (PRS) :
  "*for any team t and for any two rounds r1 and r2, let U be a minimum cardinality subset of teams including team t in which the opponents of the teams in U in rounds r1 and r2 are the same. A move in this neighborhood consists of swapping the opponents of all teams in U in rounds r1 and r2.*"(p.659)

- Round Swap (RS) :
  "*a move in this neighborhood consists of swapping all games of a given pair of rounds*"(p.659), which is the same as swapping two columns in the matrix used to represent the schedules in this thesis.

- Rows Destruction :
  In the matrix used to depict a schedule in this thesis, some of the rows, representing sequence of games played by a team, are erased and then reconstructed differently. The solver is used to perform the reconstruction and constraints to ensure the schedule is rebuilt differently are added to the model (i.e. the ILP model in the paper of Guedes and Ribeiro (2011), which would correspond to my *Final Model* in section 3.3.2.)

- Team Swap (TS) :
  "*a move in this neighborhood corresponds to swapping all opponents of a given pair of teams over all rounds*"(p.659), which is the same as swapping two rows in the matrix used to represent the schedules in this thesis.

C. Circle or polygon method

For each round $r \in \langle 1, \dots, n-1 \rangle$,

- team n plays team r,
- for $i, j \in N \setminus \{r, n\}$: team i plays team j if $i + j \equiv 2r \ mod \ (n-1)$.

The following pseudo code can be used to build schedules according to the circle method :

```
For i in 1 : n − 1
    Set the game between i and n in round i
    For j in 1 : (n/2) − 1
        if i + j < n
            if i − j > 0
                Set the game between teams i + j and i − j in round i
            end
            if i − j ≤ 0
                Set the game between team i + j and i − j + n − 1 in round i
            end
        end
        if i + j ≥ n
            if i − j > 0
                Set the game between i + j − n + 1 and i − j in round i
            end
        end
    end
end
```

D. Simulated Annealing heuristic method

Here is a description of the SA basic principle, strongly inspired and resumed from the work of Crama & Schyns (2003) :

For a problem of the form :

$$\min F(x)\ s.t.\ x \in X,$$

1. Choose an initial solution $x^{(0)}$ and compute the value of $F(x^{(0)})$. Initialize the incumbent solution : $(x^*, F^*) \leftarrow (x^{(0)}, F(x^{(0)}))$.

2. Until a stopping criterion is fulfilled and for z initialized at 0 :
   a. Draw a solution x at random in the neighborhood $V(x^{(z)})$ of $x^{(z)}$.
   b. If $F(x) \leq F(x^{(z)})$ then $x^{(z+1)} \leftarrow x$ and if $F(x) < F^*$ then $(x^*, F^*) \leftarrow (x, F(x))$.
      If $F(x) > F(x^{(z)})$ then draw a number p at random in [0 ; 1] and :
         If $p \leq p(n, x, x^z)$ then $x^{(z+1)} \leftarrow x$
         Else, $x^{(z+1)} \leftarrow x^{(z)}$.

SA is often implemented with the non-increasing function $p(n, x, x^z) = \exp(-\frac{1}{T_z}\Delta F_z)$, where $\Delta F_z = F(x) - F(x^{(z)})$ and T_z is the temperature at step z. Temperature is unchanged until a constant number L of consecutive iterations (plateau length) has passed. Then the temperature is multiplied by a constant value $\alpha \in [0 ; 1]$.

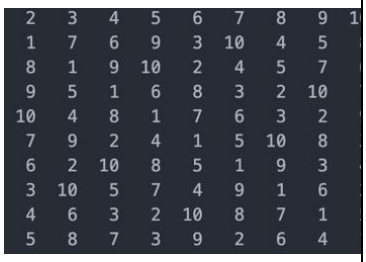
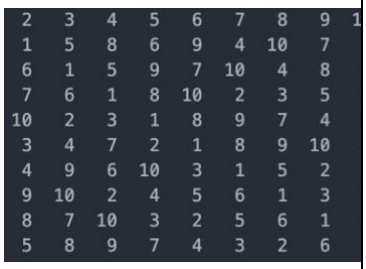
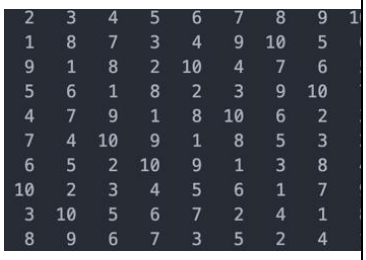
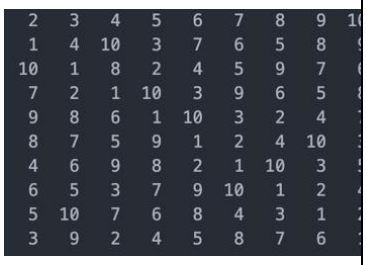
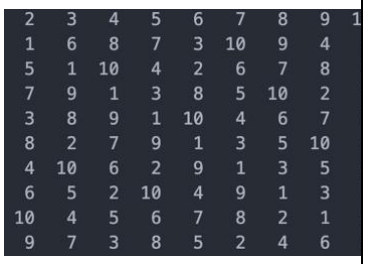
The stopping criterion is a value K that sets the limit of number of consecutive plateaus with no improvement of the incumbent solution F^* (condition 1), and

also sets the max number of consecutive plateaus during which less than $\varepsilon\%$ of the iterations are accepted moves, that means the process is no longer making progress, it freezes (condition 2). If both conditions are met, the process stops.

E. Example : 5 schedules generated with the solver in 25 seconds

The elements in the first row and the first element of the second row are the same in each calendar because of the symmetry breaking constraint in the ILP model.

As a reminder, element in row i and column j is the team playing team i in round j.

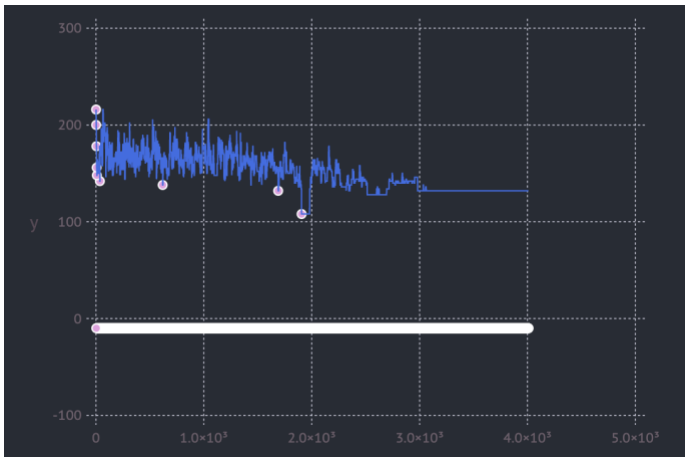
1		Weighted COE value = 799 COE value = 182	2		Weighted COE value = 833 COE value = 210
3		Weighted COE value = 748 COE value = 240	4		Weighted COE value = 747 COE value = 206
5		Weighted COE value = 667 COE value = 224			

F. Results of the SA procedure : best solutions and graphs

- Best Solution n = 10 :

2	3	4	8	10	5	7	9	6
1	8	9	10	6	4	5	3	7
5	1	6	7	4	10	9	2	8
6	10	1	9	3	2	8	7	5
3	7	8	6	9	1	2	10	4
4	9	3	5	2	7	10	8	1
9	5	10	3	8	6	1	4	2
10	2	5	1	7	9	4	6	3
7	6	2	4	5	8	3	1	10
8	4	7	2	1	3	6	5	9

Graph 3 : Evolution of the incumbent solution, best fixture found for n = 10.

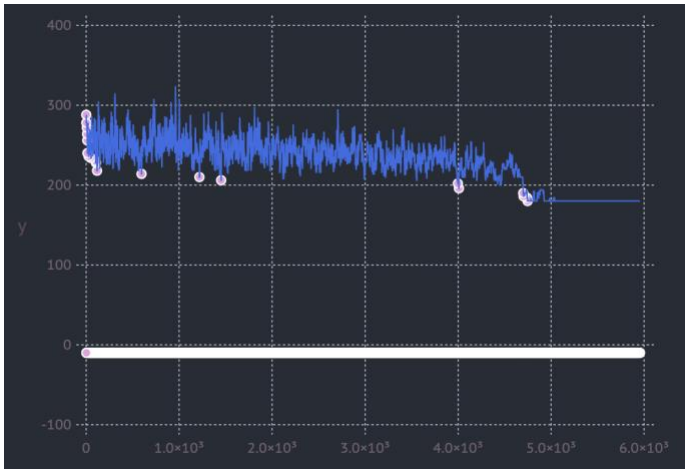


Graph 3 – Data from own calculations, the graph is from Julia Programming (Bezanson et al., 2017).

- Best Solution n = 12 :

7	8	10	9	11	6	4	3	5	2	12
4	5	11	8	7	3	6	12	9	1	10
12	6	9	7	8	2	11	1	4	10	5
2	7	5	6	10	8	1	9	3	12	11
6	2	4	11	9	12	10	7	1	8	3
5	3	8	4	12	1	2	10	11	9	7
1	4	12	3	2	9	8	5	10	11	6
10	1	6	2	3	4	7	11	12	5	9
11	10	3	1	5	7	12	4	2	6	8
8	9	1	12	4	11	5	6	7	3	2
9	12	2	5	1	10	3	8	6	7	4
3	11	7	10	6	5	9	2	8	4	1

Graph 4 : Evolution of the incumbent solution, best fixture found for n = 12.

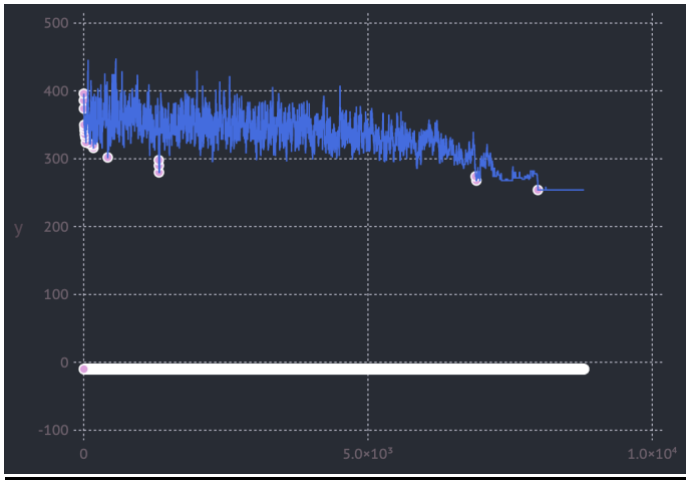


Graph 4 – Data from own calculations, the graph is from Julia Programming (Bezanson et al., 2017).

- <u>Best solution n = 14 :</u>

9	3	13	7	10	12	14	8	4	5	11	2	6
12	10	8	11	13	9	6	4	7	14	3	1	5
5	1	12	4	6	14	7	10	11	9	2	8	13
8	12	7	3	9	10	13	2	1	11	6	5	14
3	14	10	9	7	6	8	11	12	1	13	4	2
7	13	11	10	3	5	2	14	9	8	4	12	1
6	8	4	1	5	13	3	12	2	10	14	9	11
4	7	2	13	14	11	5	1	10	6	12	3	9
1	11	14	5	4	2	12	13	6	3	10	7	8
14	2	5	6	1	4	11	3	8	7	9	13	12
13	9	6	2	12	8	10	5	3	4	1	14	7
2	4	3	14	11	1	9	7	5	13	8	6	10
11	6	1	8	2	7	4	9	14	12	5	10	3
10	5	9	12	8	3	1	6	13	2	7	11	4

Graph 5 : Evolution of the incumbent solution, best fixture found for n = 14.

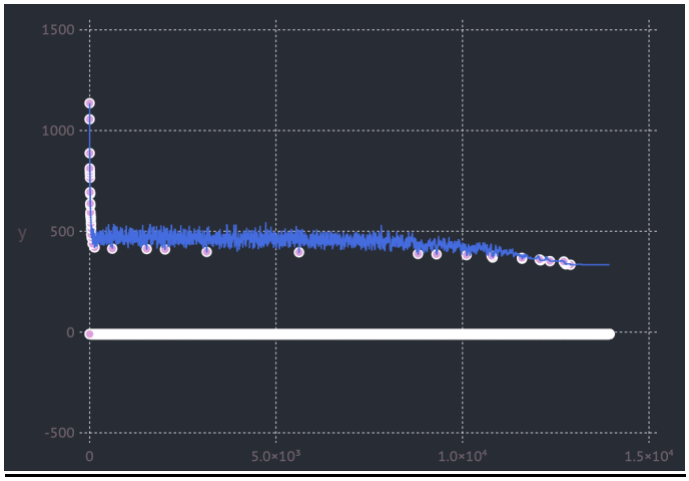


Graph 5 – Data from own calculations, the graph is from Julia Programming (Bezanson et al., 2017).

- <u>Best solution for n = 16 :</u>

3	15	13	8	5	10	11	14	6	12	4	9	16	7	2
8	4	3	5	16	11	9	12	10	15	7	6	13	14	1
1	14	2	10	6	15	12	8	7	4	5	11	9	13	16
15	2	6	13	10	7	5	16	9	3	1	8	14	12	11
10	9	7	2	1	12	4	6	11	8	3	13	15	16	14
9	16	4	11	3	14	13	5	1	7	10	2	12	15	8
14	11	5	9	8	4	15	13	3	6	2	16	10	1	12
2	12	9	1	7	13	14	3	16	5	15	4	11	10	6
6	5	8	7	14	16	2	15	4	10	12	1	3	11	13
5	13	12	3	4	1	16	11	2	9	6	14	7	8	15
13	7	15	6	12	2	1	10	5	14	16	3	8	9	4
16	8	10	14	11	5	3	2	13	1	9	15	6	4	7
11	10	1	4	15	8	6	7	12	16	14	5	2	3	9
7	3	16	12	9	6	8	1	15	11	13	10	4	2	5
4	1	11	16	13	3	7	9	14	2	8	12	5	6	10
12	6	14	15	2	9	10	4	8	13	11	7	1	5	3

Graph 6 : Evolution of the incumbent solution, best fixture found for n = 16.

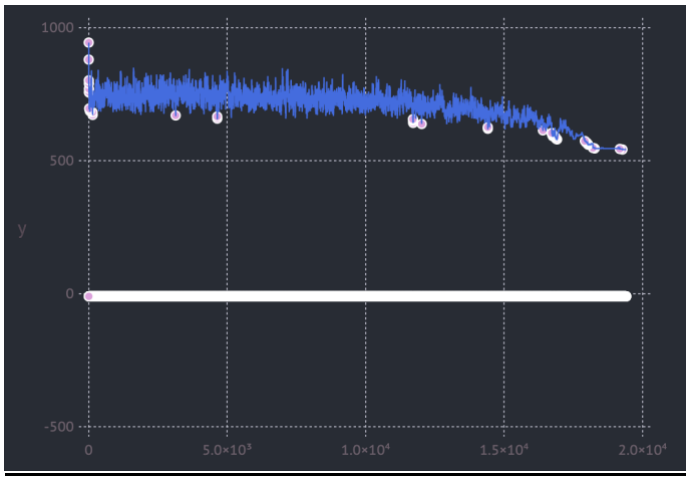


Graph 6 – Data from own calculations, the graph is from Julia Programming (Bezanson et al., 2017).

- Best solution for n = 20 :

2	11	9	10	16	13	7	14	19	6	20	8	3	12	17	5	4	15	18	
1	14	12	17	8	9	13	16	7	5	15	3	6	4	18	19	11	10	20	
11	7	15	13	6	14	18	4	17	19	5	2	1	10	8	20	12	16	9	
18	13	14	11	17	15	5	3	16	20	9	6	7	2	10	12	1	8	19	
17	10	7	19	18	16	4	6	20	2	3	9	12	8	15	1	13	11	14	
13	12	18	15	3	7	20	5	10	1	17	4	2	9	19	16	8	14	11	
12	3	5	20	14	6	1	9	2	16	10	15	4	11	13	17	18	19	8	
10	9	19	14	2	20	16	12	15	17	11	1	13	5	3	18	6	4	7	
19	8	1	18	15	2	12	7	13	10	4	5	14	6	16	11	20	17	3	
8	5	17	1	20	19	15	11	6	9	7	12	18	3	4	14	16	2	13	
3	1	20	4	13	18	17	10	14	15	8	16	19	7	12	9	2	5	6	
7	6	2	16	19	17	9	8	18	13	14	10	5	1	11	4	3	20	15	
6	4	16	3	11	1	2	20	9	12	19	14	8	17	7	15	5	18	10	
16	2	4	8	7	3	19	1	11	18	12	13	9	15	20	10	17	6	5	
20	17	3	6	9	4	10	18	8	11	2	7	16	14	5	13	19	1	12	
14	20	13	12	1	5	8	2	4	7	18	11	15	19	9	6	10	3	17	
5	15	10	2	4	12	11	19	3	8	6	18	20	13	1	7	14	9	16	
4	19	6	9	5	11	3	15	12	14	16	17	10	20	2	8	7	13	1	
9	18	8	5	12	10	14	17	1	3	13	20	11	16	6	2	15	7	4	
15	16	11	7	10	8	6	13	5	4	1	19	17	18	14	3	9	12	2	

Graph 7 : Evolution of the incumbent solution, best fixture found for n = 20.

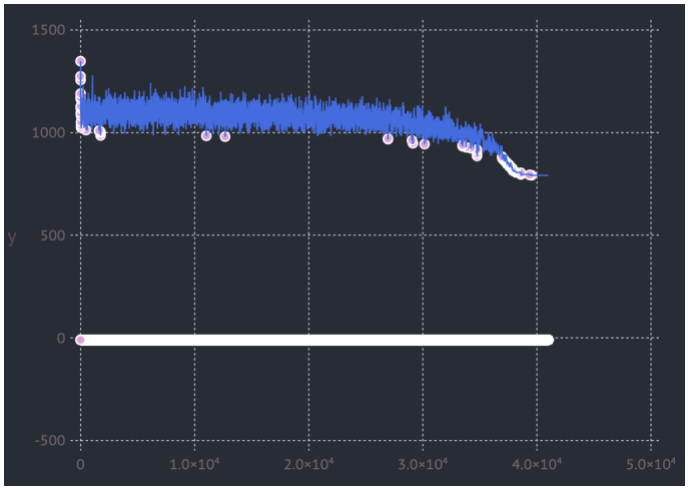


Graph 7 – Data from own calculations, the graph is from Julia Programming (Bezanson et al., 2017).

- Best solution for n = 24 :

10	3	23	2	24	17	11	12	19	16	20	22	18	6	15	4	8	7	14	13	5	9	21
4	13	17	1	16	10	8	20	7	6	22	11	9	14	12	24	15	21	3	23	18	5	19
14	1	4	24	19	22	20	15	5	11	8	6	23	9	13	12	17	10	2	18	21	7	16
2	17	3	9	18	23	12	16	11	13	7	10	6	21	5	1	19	24	22	20	8	14	15
19	10	9	11	15	7	13	6	3	12	16	17	14	24	4	21	20	18	23	22	1	2	8
13	8	22	14	21	19	17	5	20	2	9	3	4	1	18	10	16	23	24	11	7	15	12
15	24	18	20	14	5	9	21	2	23	4	19	12	22	8	11	13	1	17	16	6	3	10
23	6	21	16	10	11	2	9	15	14	3	18	22	19	7	17	1	13	20	24	4	12	5
18	11	5	4	17	21	7	8	24	19	6	12	2	3	10	22	14	16	13	15	20	1	23
1	5	13	19	8	2	24	23	21	20	12	4	15	11	9	6	18	3	16	14	17	22	7
17	9	20	5	23	8	1	22	4	3	18	2	13	10	24	7	12	15	19	6	16	21	14
24	21	15	23	20	13	4	1	17	5	10	9	7	16	2	3	11	14	18	19	22	8	6
6	2	10	21	22	12	5	24	18	4	15	16	11	20	3	23	7	8	9	1	14	19	17
3	19	16	6	7	18	22	17	23	8	21	24	5	2	20	15	9	12	1	10	13	4	11
7	20	12	22	5	16	19	3	8	17	13	23	10	18	1	14	2	11	21	9	24	6	4
20	23	14	8	2	15	21	4	22	1	5	13	24	12	17	19	6	9	10	7	11	18	3
11	4	2	18	9	1	6	14	12	15	19	5	20	23	16	8	3	22	7	21	10	24	13
9	22	7	17	4	14	23	19	13	21	11	8	1	15	6	20	10	5	12	3	2	16	24
5	14	24	10	3	6	15	18	1	9	17	7	21	8	22	16	4	20	11	12	23	13	2
16	15	11	7	12	24	3	2	6	10	1	21	17	13	14	18	5	19	8	4	9	23	22
22	12	8	13	6	9	16	7	10	18	14	20	19	4	23	5	24	2	15	17	3	11	1
21	18	6	15	13	3	14	11	16	24	2	1	8	7	19	9	23	17	4	5	12	10	20
8	16	1	12	11	4	18	10	14	7	24	15	3	17	21	13	22	6	5	2	19	20	9
12	7	19	3	1	20	10	13	9	22	23	14	16	5	11	2	21	4	6	8	15	17	18

Graph 8 : Evolution of the incumbent solution, best fixture found for n = 24.



Graph 8 – Data from own calculations, the graph is from Julia Programming (Bezanson et al., 2017).

VII. Bibliography and references

Arora, S., & Barak, B. (2009). NP and NP completeness. In Computational Complexity : A Modern Approach (pp. 38-67). Cambridge : Cambridge University Press. Doi : 10.1017/CBO9780511804090.005

Asghari, M., Fathollahi-Fard, A. M., Mirzapour Al-e-hashem, S. M. J., & Dulebenets, M. A. (2022). Transformation and Linearization Techniques in Optimization : A State-of-the-Art Survey. Mathematics, 10(2), 283. https://doi.org/10.3390/math10020283

Bae, S. (2019). Big-O notation. JavaScript data structures and algorithms, 1–11. https://doi.org/10.1007/978-1-4842-3988-9_1

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia : A fresh approach to numerical computing. SIAM Review, 59(1), 65–98. https://doi.org/10.1137/141000671

Crama, Y. (2020). Integer linear programming. [Class handouts]. HEC Liège University, MQGE0002-3.

Crama, Y., & Schyns, M. (01 November 2003). Simulated annealing for complex portfolio selection problems. European Journal of Operational Research, 150 (3), 546-571. doi:10.1016/S0377-2217(02)00784-1

Gendreau, M., & Potvin, J.-Y. (2019). Handbook of Metaheuristics (3rd ed. 2019.; M. Gendreau & J.-Y. Potvin, Eds.). https://doi.org/10.1007/978-3-319-91086-4

Glover. (1975). Improved linear integer programming formulations of nonlinear integer problems. Management Science, 22(4), 455–460. https://doi.org/10.1287/mnsc.22.4.455

Glover, F. (1996). Ejection chains, reference structures and alternating path methods for traveling salesman problems. Discrete Applied Mathematics, 65(1), 223–253. https://doi.org/10.1016/0166-218X(94)00037-E

Goossens, D., Yi, X., & Van Bulck, D. (2020). Fairness trade-offs in sports timetabling. In C. Ley & Y. Dominicy (Eds.), Science meets sports : when statistics are more than numbers (pp. 213–244). Cambridge Scholars.

Guedes, A. C. B., & Ribeiro, C. C. (2011). A heuristic for minimizing weighted carry-over effects in round robin tournaments. *Journal of Scheduling*, *14*(6), 655–667. https://doi.org/10.1007/s10951-011-0244-y

Januario, T., & Urrutia, S. (2016). A new neighborhood structure for round robin scheduling problems. Computers & Operations Research, 70, 127–139. https://doi.org/10.1016/j.cor.2015.12.016

Kendall, G., Knust, S., Ribeiro, C. C., & Urrutia, S. (2010). Scheduling in sports: An annotated bibliography. Computers & Operations Research, 37(1), 1–19. https://doi.org/10.1016/j.cor.2009.05.013

Kettani, O., & Oral, M. (1990). Equivalent formulations of nonlinear integer problems for efficient optimization. Management Science, 36(1), 115–119. http://www.jstor.org/stable/2632098

Kidd, M. (2010). A tabu-search for minimising the carry-over effects value of a round-robin tournament. Orion (Johannesburg, South Africa), 26(2), 125–141. https://doi.org/10.5784/26-2-91

Koch, T., T. Berthold, J. Pedersen, and C. Vanaret (2021). Progress in Mathematical Programming Solvers from 2001 to 2020. [Working Paper]. Zuse Institute Berlin. https://opus4.kobv.de/opus4-zib/files/8277/zr_21-20.pdf

Lambrechts, Erik, Ficker, Annette, Goossens, Dries, & Spieksma, Frits. (2016). Round-robin tournaments generated by the circle method have maximum carry-over. FEB Research Report KBI_1603, 9682, 178–189. https://doi.org/10.1007/978-3-319-33461-5_15

Manson, N. (2006). Is operations research really research? ORiON, 22(2). https://doi.org/10.5784/22-2-40

Margot F. (2010) Symmetry in integer linear programming. In : Jünger M. et al. (eds) 50 Years of Integer Programming 1958-2008. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-68279-0_17

Ribeiro, C. C. (2019, March). Applications of metaheuristics to optimization problems in sports [Conference presentation]. 8th International workshop on operations

research, La Havana, Cuba. https://slidetodoc.com/applications-of-metaheuristics-to-optimization-problems-in-sports/

Round-robin. (n.d.). In Collins. HarperCollins. Retrieved January 28, 2022, from https://www.collinsdictionary.com/dictionary/english/round-robin

Rubinstein-Salzedo, S. (2018). Big O notation and algorithm efficiency. Springer Undergraduate Mathematics Series, 75–83. https://doi.org/10.1007/978-3-319-94818-8_8

Russell, K. G. (1980). Balancing carry-over effects in round robin tournaments. Biometrika, 67(1), 127–131. https://www.jstor.org/stable/2335325

Rutenbar, R.A. (1989). Simulated annealing algorithms : an overview. IEEE Circuits and Devices Magazine, 5(1), 19–26. https://doi.org/10.1109/101.17235

Executive Summary

Fairness and optimal design of sport schedules are relevant for business, security and logistical matters. Sports events nowadays represent huge business opportunities and challenges. Many stakeholders, such as the police and broadcasting companies, depend on the organization of those events. The problem of minimizing the Carry Over Effect (COE) in Single Round Robin Tournaments (SRRT) represents a difficult combinatorial optimization problem which makes it a challenging subject for academics and researchers. In sport scheduling, the COE is a measure of how teams efforts are balanced throughout the tournament.

The article of Guedes and Ribeiro (2011) is the starting point of this thesis. The first objective was to linearize and simplify their basic Quadratic Integer Programming (QIP) formulation of the COE problem and see how a present-day solver would perform compared to the results of Guedes and Ribeiro. An Integer Linear Programming (ILP) formulation is provided and results show a reduced running time, on small instances.

In the second part, a Simulated Annealing (SA) algorithm exploring the Game Rotation (GR) neighborhood was implemented to solve larger instances. This is a stochastic metaheuristic procedure that needs to start from an initial schedule in order to modify it according to a set of rules. These structures are constrained by many requirements and the COE is a complex measure because it implies every single element of the fixture. We have no precise knowledge on how to arrange these elements to reduce the COE value. Additionally, modifying a structure is a real challenge, the number of possible arrangements of the elements of a fixture is incredibly huge but moving from one to another is not always possible with the moves structures we know so far. Indeed, the solutions space is characterized by a low connectivity between its different points. Therefore, an alternative using the solver and weights is proposed with a double target. First, overcome the difficulty of generating initial schedules. Second, diversify them as much as possible thanks to random weighting to try to overcome this connectivity issue. Then, the SA is applied a few times to each initial solution. Finally, results are analyzed and conclusions are drawn.

Initial solutions produced by the solver are varied and results are satisfying, final solutions matching the results of Guedes and Ribeiro (2011) were achieved. The conclusion is that this combination of the solver and the SA procedure is working even though it has a drawback. The time required by the solver to issue a solution increases exponentially as the instance size grows. This duration depends, notably, on choices regarding the objective function and weights. Therefore, future investigation might help reduce this running time and improve the method proposed in this work.

Word count = 19,190