AMMM - Course project

Master in Research and Innovation in Informatics

Ignacio Encinas Rubio, Adrián Jiménez Gonzalez
{ignacio.encinas,adrian.jimenez.g}@estudiantat.upc.edu

Polytechnic University of Catalonia

19 de noviembre de 2022

Contents

Problem Statement

Main requirements

1. Each contestant will play exactly once against each of the other contestants.
2. Each round will consist of $\frac{n-1}{2}$ matches.
3. Players will play 50 % of their games as white, 50 % will be played as black.

Subtle requirements

- A player can only play up to 1 game per round
- A player can't play against himself

Problem Statement: Inputs & Outputs

Inputs

- Number of contestants, n. Has to be odd
- Matrix of points per day per player, $p_{n \times n}$

Outputs

- Schedule with the set of pairings $\{\{r_1, p_i, p_j\}, \ldots, \{r_n, p_k, p_h\}\}$ that maximizes total score. Ensured to be optimal if it's obtained through the ILP.

Problem Statement: Definitions

In order to specify the constraints, we need to specify the sets and variables we're going to work with:

- $M(x, y)$ matches played among x and y (1)
- $R(r)$ matches played at round r (2)
- $W(p)$ matches played by player p as white (3)
- $B(p)$ matches played by player p as black (3)
- $G(p, r)$ games played by p at round r (4)
- $F(r)$ free players at round r (5)

1. Each contestant will play exactly once against each of the other contestants.
2. Each round will consist of $\frac{n-1}{2}$ matches.
3. Players will play 50 % of their games as white, 50 % will be played as black.
4. Players can play up to 1 match per round
5. Objective function

Integer Linear Programming Model: Variables

Every set will be constructed from a boolean multidimensional array. *matches*$[w][b][r]$ will be 1 whenever player w plays player b in round r, and 0 otherwise.

Set constructions

$$
\begin{aligned}
M(x,y) &= \{\{x,y,r\} && \mid \text{matches}[x][y][r] = 1 \lor \text{matches}[y][x][r] = 1 && \forall r \in [1, Rounds]\} \\
F(r) &= \{p && \mid \text{matches}[p][o][r] = 0 \land \text{matches}[o][p][r] = 0 && \forall o \in [1, n]\} \\
W(p) &= \{\{p,b,r\} && \mid \text{matches}[p][b][r] = 1 && \forall r \in [1, Rounds], b \in [1, n]\} \\
B(p) &= \{\{w,p,r\} && \mid \text{matches}[w][p][r] = 1 && \forall r \in [1, Rounds], w \in [1, n]\} \\
R(r) &= \{\{w,b,r\} && \mid \text{matches}[w][b][r] = 1 && \forall w, b \in [1, n]\} \\
G(p,r) &= \{\{o,p,r\} && \mid \text{matches}[p][o][r] = 1 \lor \text{matches}[o][p][r] = 1 && \forall o \in [1, n]\}
\end{aligned}
$$

Integer Linear Programming Model: Constraints

$$|M(x,y)| = 1 \quad \forall x, y \in P \mid x \neq y \tag{1}$$

$$|R(r)| = \frac{n-1}{2} \quad \forall r \in [1, \text{ Rounds}] \tag{2}$$

$$|W(p)| = \frac{n-1}{2} \quad \forall r \in [1, \text{ Rounds}], \forall p \in P \tag{3}$$

$$|G(p,r)| \leq 1 \quad \forall p \in P, r \in [1, \text{ Rounds}] \tag{4}$$

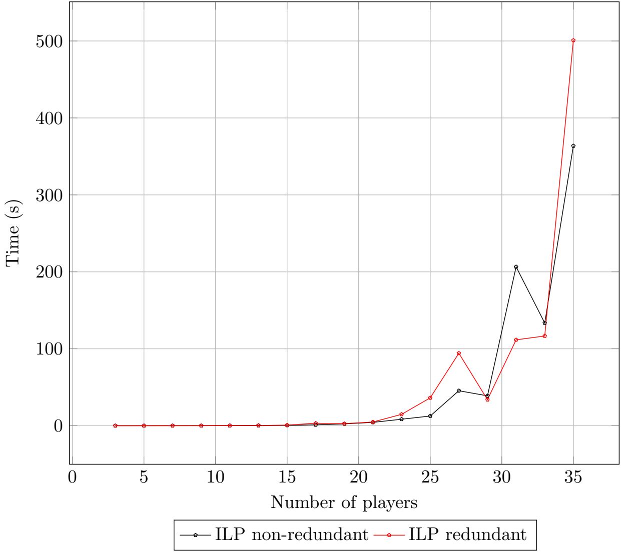
1. Each contestant will play exactly once against each of the other contestants.
2. Each round will consist of $\frac{n-1}{2}$ matches.
3. Players will play 50 % of their games as white, 50 % will be played as black.
4. Players can play up to 1 match per round

Integer Linear Programming Model: Redundant constraints

Redundant constraints might appear to make the model faster but they seem make it slower in the long run

$$|M(x,x)| = 0 \quad \forall x \in P$$

$$|B(p)| = \frac{n-1}{2} \quad \forall r \in [1, \text{ Rounds}], p \in P$$



Metaheuristics: Greedy algorithm

Greedy cost function

$$q(c, day) = c.points\_per\_day[day]$$

Algorithm Greedy algorithm

```
Players ← Set of Players
rests ← {}
for day in 0..days do
    playersToRest ← filter Players(p) | p.hasNotRested
    sortedPlayers ← sort playersToRest(p) by q(p,day) (DESC)
    select p ∈ sortedPlayers[0]
    rests[day] ← p
```

Metaheuristics: Local Search

Algorithm Local Search

```
for i in 0..days do
    best_swap_points ← 0
    best_swap ← i
    for j in 0..days do
        change = EvaluateRestSwap(i,j)
        if change > best_swap_points then
            best_swap_points ← change
            best_swap ← j
    rests[i] ↔ rests[best_swap]
```

Metaheuristics: GRASP

Algorithm constructRCL(day)

```
q_max ← sortedPlayers.first().points[d]
q_min ← sortedPlayers.last().points[d]
RCL_max ← {p ∈ sortedPlayers | p.points[d] >= q_max − α · (q_max − q_min)}
```

$q_{max} \leftarrow$ sortedPlayers.first().points[d]

$q_{min} \leftarrow$ sortedPlayers.last().points[d]

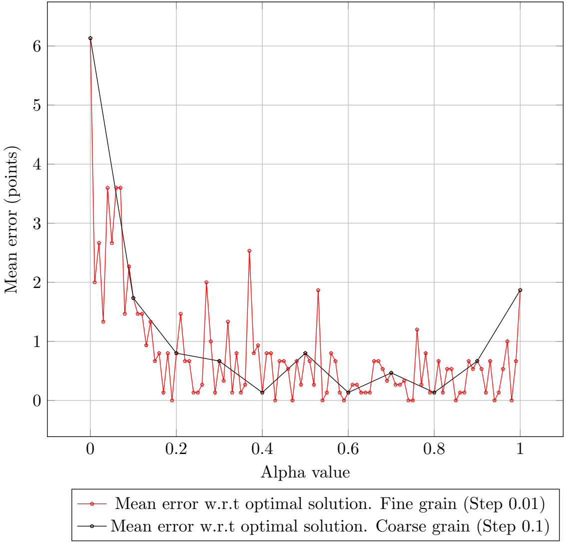
$RCL_{max} \leftarrow \{p \in sortedPlayers \mid p.points[d] >= q_{max} - \alpha \cdot (q_{max} - q_{min})\}$

Algorithm GRASP

```
rests ← {}
for day in 0..days do
    RCL ← constructRCL(day)
    select p ∈ RCL randomly
    rests[day] ← p
```

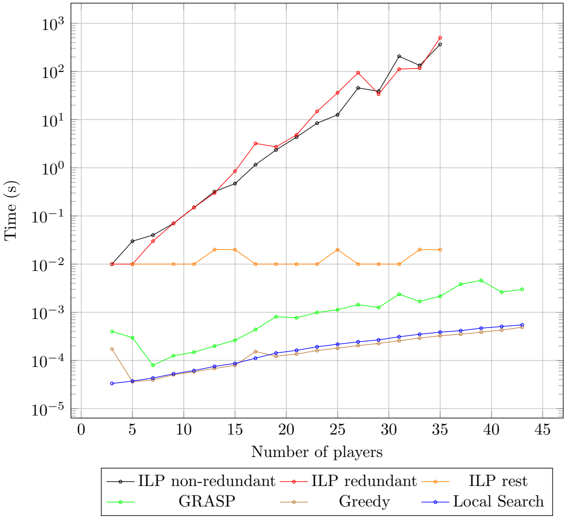
GRASP: Parameter tuning

- Hemos probao no se que no se cuantos
- Muchos alphas nos valen porque hay muchos 0 calvo de mierda



Results: Time

- ILP rest blabla
- texto



Results: Quality of solutions

- ILP rest blabla
- texto