LINEAR PROGRAMMING: LETTING COMPUTERS MAKE OUR DECISIONS

Nicolás García Aramouni & Luis Biedma

Nicolás García Aramouni

Bachelor in Business Economics @ UTDT
Master in Management & Analytics @ UTDT
Business Data Science Analyst @ Accenture
Basketball enthusiast

Email: nico.garcia.ara@gmail.com

Tw: @nicogarciaara

Luis Biedma

Lic. & Phd Student in Mathematics @ FAMAF - UNC
Assistant Professor @ FAMAF - UNC
“Chief of Research” @ Invera
Basketball enthusiast

Email: luis@invera.com.ar

Tw: @lbiedma17

What are we trying to do with analytics?

Every day in the media a lot of tech-like terms are being used: analytics, big data, artificial intelligence.

However, it may not be clear how this methods and technologies may help people in their everyday life.

Are we trying to build robots that will destroy human kind? Are we building apps that read the human mind? Will we replace all human workforce with robots?

We are using data in to take better and more effective decisions

Clarín ECONOMÍA

S CLARÍN > ECONOMÍA | POLITICA SOCIEDAD DEPORTES ESPECTÁCULOS MUNDO ECONOMÍA OPINIÓN

El color del dinero

Big data, la otra vacuna contra el coronavirus

Matías Travizano es un físico argentino con una empresa de

The New York Times

NEWS ANALYSIS

The Age of Big Data

By Steve Lohr

What can we do with analytics?

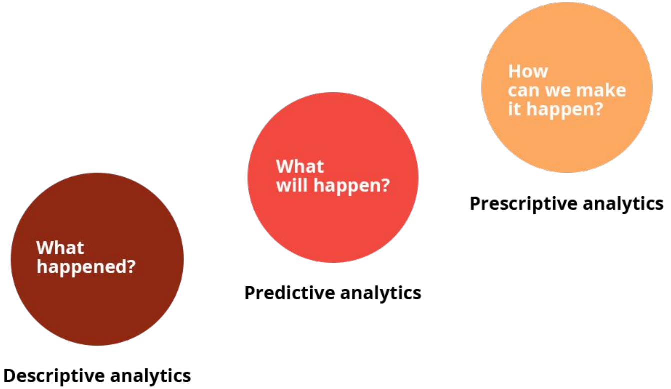
Different types of tasks can be tackled with analytics techniques. Let’s define them and give an example

Descriptive analytics □ How much did my sales grow in the last month?

Predictive analytics □ How are my sales going to be the next month?

Prescriptive analytics □ How many units should I produce in order to maximize profits?

Linear programming, Dynamic programming, etc.



LINEAR PROGRAMMING

A brief definition

“

Linear programming implies the abstraction of a problem or system that captures its behavior through mathematical relationships

Linear programming ≠ Coding

In this case programming means **planification**, not coding

We are trying to represent a problem or system with mathematical variables and relationships

We are trying to **optimize** something (e.g. minimize cost, maximize profits) making the best possible decision

This problems can be solved in a variety of interfaces:

- Complex problems are usually solved via code and specific libraries & APIs
- However, simpler problems could be solved with Excel's Solver or even pen and paper

Linear programming key concepts

Decision variables

What can be changed to optimize the objective function

E.G. Production quantities

Constraints

The conditions the solution has to meet

E.G. Don’t produce more than the factory’s capacity

Objective Function

The objective that has to be optimized

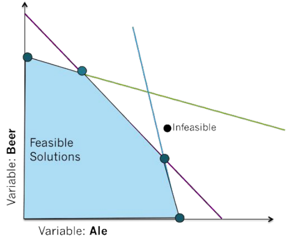
E.G. Profit formula

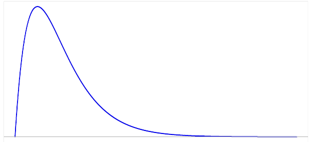
The constraints and objective función are linear □ Variables can´t be multiplied by another variable (e.g. x^2). They can be multiplied by a constant

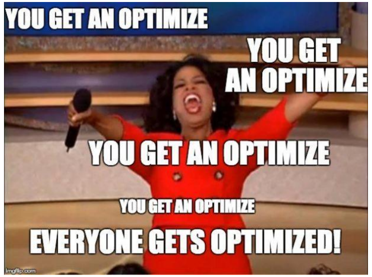
Linear programming pipeline

Define variables → Define constraints → Define obj function → Implement & Optimize! → Analyze the solution

x_i
Production of product i







What production mix maximizes profits

Cartoon equivalents to linear programming


MY MOM SAYS THIS IS TOO ABSTRACT
CAN WE SEE AN EXAMPLE?
imgflip.com

MIXED INTEGER[1] LINEAR PROGRAMMING

A practical example

[1]This is a Mixed Integer Linear Problem as it also has binary variables

The setup

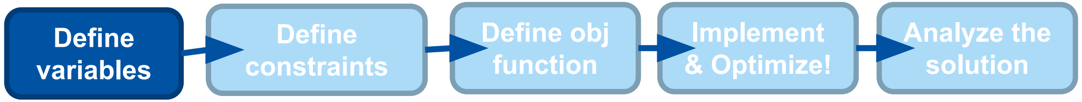
After the Coronavirus pandemic, the Argentine Football Association (AFA) is organizing a small 4-team tournament between:

- Aldosivi (ALD)
- Godoy Cruz (GCM)
- Central Córdoba (CCO)
- Talleres (TAL)

The tournament will last two weeks and, after every match, the teams don't return to their home (they travel to the next match's home stadium)

The objective is to set up a schedule that minimizes the total distance traveled

Variable definition



$$n \rightarrow Number\ of\ teams$$
$$m \rightarrow Number\ of\ matches\ per\ team = 2n - 2$$
$$r \rightarrow Number\ of\ matches\ per\ team\ in\ the\ first\ round = n - 1$$
$$N \rightarrow Set\ of\ teams. N = \{1,2, \ldots, n\}$$
$$T \rightarrow Set\ of\ matchdays. T = \{1,2, \ldots, m\}$$

$$x_{ijt} \rightarrow A\ binary\ variable\ that\ is\ equal\ to\ 1\ if\ team\ i\ plays\ at\ home\ against\ team\ j\ in\ match\ t$$
$$y_{ijkt} \rightarrow A\ binary\ variable\ that\ is\ equal\ to\ 1\ if\ team\ i\ travels\ from\ location\ j\ to\ k\ after\ match\ t$$
$$\overline{d_{jk}} \rightarrow Distance\ between\ teams\ j\ and\ k\ (This\ is\ a\ constant\ - not\ a\ decision\ variable)$$

Constraint definition

Define variables → Define constraints → Define obj function → Implement & Optimize! → Analyze the solution

$$x_{iit} = 0 \quad \forall\, i \in N, \forall\, t \in T \quad (1)$$

Teams can't play against themselves

$$\sum_{j, j \neq i} x_{ijt} + \sum_{j, j \neq i} x_{jit} = 1 \quad \forall\, i \in N, \forall\, t \in T \quad (2)$$

Teams play against one team per matchday

$$\sum_{t, t \leq r} x_{ijt} + \sum_{t, t \leq r} x_{jit} = 1 \quad \forall\, i, j \in N, i \neq j \quad (3)$$

Teams play once against each rival in the first round

$$\sum_{t} x_{ijt} + \sum_{t} x_{jit} = 2 \quad \forall\, i, j \in N, i \neq j \quad (4)$$

Teams play twice against each rival in the tournament

$$\sum_{t} x_{ijt} = 1 \quad \forall\, i, j \in N, i \neq j \quad (5)$$

Teams can't play in their stadium against the same team twice

$$y_{ijkt} \geq x_{jit} + x_{kit+1} - 1 \quad \forall\, i, j, k \in N, \forall\, t \in T, t < m \quad (6)$$
$$y_{iikt} \geq x_{ijt} + x_{kit+1} - 1 \quad \forall\, i, j, k \in N, \forall\, t \in T, t < m \quad (7)$$
$$y_{ijit} \geq x_{jit} + x_{ikt+1} - 1 \quad \forall\, i, j, k \in N, \forall\, t \in T, t < m \quad (8)$$
$$y_{iiit} \geq x_{ijt} + x_{ikt+1} - 1 \quad \forall\, i, j, k \in N\, \forall\, t \in T, t < m \quad (9)$$

These constraints relate the x and y variables

Objective function definition

Define variables → Define constraints → **Define obj function** → Implement & Optimize! → Analyze the solution

$$\min \sum_i \sum_j \sum_k \sum_t \overline{d_{jk}} * y_{ijkt} \rightarrow \text{Total distance}$$

Solution Analysis

Define variables → Define constraints → Define obj function → Implement & Optimize! → **Analyze the solution**

Generated Schedule

Matchday	ALD	GCM	CCO	TAL
1	@CCO	TAL	ALD	@GCM
2	@TAL	CCO	@GCM	ALD
3	GCM	@ALD	TAL	@CCO
4	@GCM	ALD	@TAL	CCO
5	TAL	@CCO	GCM	@ALD
6	CCO	@TAL	@ALD	GCM

Total Distance
15.604 km

Advantages of Linear Programming

Using a model that is an abstraction of reality allows the following things:

- Adapting this model for other applications (e.g vehicle routing)
- If new constraints appear or other objective function should be used, the model can be easily adapted

So, let’s change the model!

Changing the model

Instead of minimizing total distance, AFA wants to maximize the tournament's interest in the last matches of the tournament. In order to do this, they have created a metric p_{ijt} that evaluates the interest generated by a match played by teams i and j, played in matchday t, which uses the position of each team in last year's tournament (the position of team i is defined by the constant pos_i).

$$p_{ijt} = \frac{t}{1 + |pos_i - pos_j|}$$

So, how does the formulation of the problem change?

- Constraints (6) to (9) disappear as we are no longer considering distance
- The objective function is changed. The new objective function is equal to:

$$\max \sum_i \sum_j \sum_t x_{ijt} * p_{ijt}$$

Changing the model - results

New Generated Schedule

Matchday	ALD	GCM	CCO	TAL
1	@TAL	@CCO	GCM	ALD
2	@CCO	@TAL	ALD	GCM
3	GCM	@ALD	TAL	@CCO
4	TAL	CCO	@GCM	@ALD
5	CCO	TAL	@ALD	@GCM
6	@GCM	ALD	@TAL	CCO

Objective Function	Total Distance	Total Interest
Distance Minimization	15.604 km	4.93
Interest Maximization	19.392 km	5.41

CODE VERSION

Google OR-Tools

In [3]:

```python
# We create a class that has all the data and methods used in the model
class SportsSchedule:
    def __init__(self, df_dist_matrix, df_positions):
        """
        Class that has all the data and methods used in the model
        :param df_dist_matrix: dataframe that represents a distance matrix between all teams
        :param df_position: dataframe that has information of the position of each team in last year's tournament
        """
        # We setup the inputs as variables
        self.df_dist_matrix = df_dist_matrix
        self.df_positions = df_positions

        # We setup the model parameters
        self.N = list(self.df_dist_matrix['Team'])
        self.n = len(self.N)
        self.m = 2*self.n - 2
        self.r = self.n - 1

        # We create a dictionary that relates teams with its position
        self.positions = {}
        for index, row in self.df_positions.iterrows():
            self.positions[row['Team']] = row['Position']

        # We create a dictionary dist_matrix that will serve as a cost matrix. Example
        # dist_matrix[(i, j)] = distance between teams i and j
        self.dist_matrix = {}
        for index, row in self.df_dist_matrix.iterrows():
            team_one = row['Team']
            for team_two in self.N:
                self.dist_matrix[(team_one, team_two)] = row[team_two]

        # We create a dictionary called int_dict
        # int_dict[(i, j, t)] = interest of match betweens teams i and j at matchday t
        self.int_dict = {}
        for team_one in self.N:
            pos_team_one = self.positions[team_one]
            for team_two in self.N:
                pos_team_two = self.positions[team_two]
                for t in range(self.m):
                    interest = t / (1 + abs(pos_team_one - pos_team_two))
                    self.int_dict[(team_one, team_two, t)] = interest
```

```python
def model_and_variable_creation(Schedule):
    """
    A function that creates a or tools solver model and creates the decision variables used
    :param Schedule: A SportsSchedule instance
    return
    solver: an or tools solver model
    x_var_dict = a dictionary with the binary "x" variables
    y_var_dict = a dictionary with the binary "y" variables
    """
    # We create the solver
    solver = pywraplp.Solver('simple_mip_program',
                             pywraplp.Solver.CBC_MIXED_INTEGER_PROGRAMMING)

    # We create the dictionary x_var_dict
    # x_var_dict[(i,j,t)] will have the solver variable for the binary variable that will be equal to one if team i
    # hosts team j in matchday t
    # We check that the variable hasn't been already created
    created_xvariables = []
    x_var_dict = {}
    for team_one in Schedule.N:
        for team_two in Schedule.N:
            for t in range(Schedule.m):
                if str(team_one+team_two+str(t)) not in created_xvariables:
                    x_var_dict[(team_one, team_two, t)] = solver.BoolVar(str('x_'+team_one+team_two+str(t)))
                    created_xvariables.append(str(team_one+team_two+str(t)))

    # We create the dictionary y_var_dict
    # y_var_dict[(i,j,k,t)] will have the solver variable for the binary variable that will be equal to one if team i
    # travels from j to k after matchday t
    # We check that the variable hasn't been already created
    y_var_dict = {}
    created_yvariables = []
    for team_one in Schedule.N:
        for team_two in Schedule.N:
            for team_three in Schedule.N:
                for t in range(Schedule.m):
                    if str(team_one+team_two+team_three+str(t)) not in created_yvariables:
                        y_var_dict[(team_one, team_two, team_three, t)] = solver.BoolVar(
                            str('y_'+team_one+team_two+team_three+str(t)))
                        created_yvariables.append(str(team_one+team_two+team_three+str(t)))

    return solver, x_var_dict, y_var_dict
```

Constraint Creation

We setup the different constraints for the problem

Teams shouldn't play themselves

We start by creating the constraint that prohibits teams from playing themselves in a particular matchday. This can be represented by the following equation:

$$x_{iit} = 0, \forall i \in N, \forall t \in T$$

In [5]:

```
def teams_shouldnt_play_themselves_constraint(Schedule, solver, x_var_dict, y_var_dict):
    for team in Schedule.N:
        for t in range(Schedule.m):
            ct = solver.Constraint(0, 0, 'MatchesBetween'+team+'round'+str(t)+'are forbidden')
            ct.SetCoefficient(x_var_dict[(team, team, t)], 1)

    return solver
```

Teams play against one team each matchday

We create a constraint that makes each team play against one rival each matchday. This can be represented by the following equation:

$$\sum_{j, j \neq i} x_{ijt} + \sum_{j, j \neq i} x_{jit} = 1, \forall i \in N, \forall t \in T$$

In [6]:

```
def one_match_per_matchday(Schedule, solver, x_var_dict, y_var_dict):
    for team_one in Schedule.N:
        for t in range(Schedule.m):
            ct = solver.Constraint(1, 1, 'MatchesTeam'+team_one+'Round'+str(t))
            for team_two in Schedule.N:
                if team_one != team_two:
                    ct.SetCoefficient(x_var_dict[(team_one, team_two, t)], 1)
                    ct.SetCoefficient(x_var_dict[(team_two, team_one, t)], 1)
    return solver
```

Teams face once in the first round

We create a constraint that makes every pair of rivals face once in the first round of the tournament. This can be represented by the following equation:

$$\sum_{t,t \leq r} x_{ijt} + \sum_{t,t \leq r} x_{jit} = 1, \forall i, j \in N, i \neq j$$

In [7]:
```
def one_match_per_rival_first_round(Schedule, solver, x_var_dict, y_var_dict):
    for team_one in Schedule.N:
        for team_two in Schedule.N:
            if team_one != team_two:
                ct = solver.Constraint(1, 1, 'FirstRound'+team_one+team_two)
                for t in range(Schedule.r):
                    ct.SetCoefficient(x_var_dict[(team_one, team_two, t)], 1)
                    ct.SetCoefficient(x_var_dict[(team_two, team_one, t)], 1)
    return solver
```

Teams face twice in the tournament

We create a constraint that makes every pair of rivals face twice in the tournament. This can be represented by the following equation:

$$\sum_{t} x_{ijt} + \sum_{t} x_{jit} = 2, \forall i, j \in N, i \neq j$$

In [8]:
```
def two_matches_per_rival_tournament(Schedule, solver, x_var_dict, y_var_dict):
    for team_one in Schedule.N:
        for team_two in Schedule.N:
            if team_one != team_two:
                ct = solver.Constraint(2, 2, 'TwoMatchesTournament'+team_one+team_two)
                for t in range(Schedule.m):
                    ct.SetCoefficient(x_var_dict[(team_one, team_two, t)], 1)
                    ct.SetCoefficient(x_var_dict[(team_two, team_one, t)], 1)
    return solver
```

Teams can host a rival just once

We create a constraint that makes a team i host another team j just once in the tournament. This constraints exists in order to prevent a situation in which team i hosts both matches against team j. This can be represented by the following equation:

$$\sum_t x_{ijt} = 1, \forall i, j \in N, i \neq j$$

In [9]:
```
def host_rival_once(Schedule, solver, x_var_dict, y_var_dict):
    for team_one in Schedule.N:
        for team_two in Schedule.N:
            if team_one != team_two:
                ct = solver.Constraint(1, 1, team_one+'HostsOnce'+team_two)
                for t in range(Schedule.m):
                    ct.SetCoefficient(x_var_dict[(team_one, team_two, t)], 1)
    return solver
```

In [10]:
```python
def link_x_y_variables(Schedule, solver, x_var_dict, y_var_dict, obj_funct):
    if obj_funct == 'min_distance':
        constraint_names = []
        for team_one in Schedule.N:
            for team_two in Schedule.N:
                for team_three in Schedule.N:
                    for t in range(Schedule.m-1):
                        ctname = team_one+'TravelsFrom'+team_two+'To'+team_three+'AfterMatchday'+str(t)
                        if ctname not in constraint_names:
                            ct = solver.Constraint(0, 1, ctname)
                            ct.SetCoefficient(x_var_dict[(team_two, team_one, t)], 1)
                            ct.SetCoefficient(x_var_dict[(team_three, team_one, t+1)], 1)
                            ct.SetCoefficient(y_var_dict[(team_one, team_two, team_three, t)], -1)
                            constraint_names.append(ctname)

        for team_one in Schedule.N:
            for team_two in Schedule.N:
                for team_three in Schedule.N:
                    for t in range(Schedule.m-1):
                        ctname = team_one+'TravelsFrom'+team_one+'To'+team_three+'AfterMatchday'+str(t)
                        if ctname not in constraint_names:
                            ct = solver.Constraint(0, 1, ctname)
                            ct.SetCoefficient(x_var_dict[(team_one, team_two, t)], 1)
                            ct.SetCoefficient(x_var_dict[(team_three, team_one, t+1)], 1)
                            ct.SetCoefficient(y_var_dict[(team_one, team_one, team_three, t)], -1)
                            constraint_names.append(ctname)

        for team_one in Schedule.N:
            for team_two in Schedule.N:
                for team_three in Schedule.N:
                    for t in range(Schedule.m-1):
                        ctname = team_one+'TravelsFrom'+team_two+'To'+team_one+'AfterMatchday'+str(t)
                        if ctname not in constraint_names:
                            ct = solver.Constraint(0, 1, ctname)
                            ct.SetCoefficient(x_var_dict[(team_two, team_one, t)], 1)
                            ct.SetCoefficient(x_var_dict[(team_one, team_three, t+1)], 1)
                            ct.SetCoefficient(y_var_dict[(team_one, team_two, team_one, t)], -1)
                            constraint_names.append(ctname)

        for team_one in Schedule.N:
            for team_two in Schedule.N:
                for team_three in Schedule.N:
                    for t in range(Schedule.m-1):
                        ctname = team_one+'TravelsFrom'+team_one+'To'+team_one+'AfterMatchday'+str(t)
                        if ctname not in constraint_names:
                            ct = solver.Constraint(0, 1, ctname)
                            ct.SetCoefficient(x_var_dict[(team_one, team_two, t)], 1)
                            ct.SetCoefficient(x_var_dict[(team_one, team_three, t+1)], 1)
                            ct.SetCoefficient(y_var_dict[(team_one, team_one, team_one, t)], -1)
                            constraint_names.append(ctname)
    else:
        pass
    return solver
```

Objective Function Creation

We setup the different objective functions

If the objective function is equal to 'min_distance'

Then the solver will try to minimize the total distance. This can be represented by the following equation:

$$min \sum_i \sum_j \sum_k \sum_t \bar{d}_{jk} * y_{ijkt}$$

If the objective function is equal to 'max_interest'

Then the solver will try to maximize the interest in the last matchdays. This can be represented by the following equation

$$max \sum_i \sum_j \sum_t x_{ijt} * p_{ijt}$$

```
def define_objective_function(Schedule, solver, x_var_dict, y_var_dict, obj_funct):
    if obj_funct == 'min_distance':
        objective = solver.Objective()
        for team_one in Schedule.N:
            for team_two in Schedule.N:
                for team_three in Schedule.N:
                    for t in range(Schedule.m):
                        objective.SetCoefficient(
                            y_var_dict[(team_one, team_two, team_three, t)], Schedule.dist_matrix[(team_two, team_three)])
        objective.SetMinimization()

    elif obj_funct == 'max_interest':
        objective = solver.Objective()
        for team_one in Schedule.N:
            for team_two in Schedule.N:
                for t in range(Schedule.m):
                    objective.SetCoefficient(x_var_dict[(team_one, team_two, t)], Schedule.int_dict[(team_one, team_two, t)])
        objective.SetMaximization()

    else:
        pass

    solver.Solve()

    return solver, x_var_dict, y_var_dict, objective
```

```python
def get_schedule(Schedule, x_var_dict):
    # We create a list of matches for each team
    matches = {}
    for team in Schedule.N:
        matches[team] = [0]*(Schedule.m)

    # We look for each value in the x variable dictionary
    for variable in x_var_dict:
        if round(x_var_dict[variable].solution_value()) == 1:
            home = list(variable)[0]
            away = list(variable)[1]
            matchday = list(variable)[2]
            # We fill the dictionary
            matches[home][matchday] = away
            matches[away][matchday] = str("@"+home)

    # We create the dataframe
    df_schedule = pd.DataFrame()
    df_schedule['Matchday'] = list(range(1, Schedule.m+1))
    for team in matches:
        df_schedule[team] = matches[team]

    return df_schedule
```

Extract schedule's KPIs

In order to compare both solutions, we calculate for a given schedule, the total distance and the total interest generated

```
def schedule_analysis(Schedule, df_schedule):
    teams = Schedule.N

    # We create the output variables
    total_distance = 0
    total_interest = 0

    for i in range(len(df_schedule)):
        # For the distance analysis, we don't consider the first match
        if df_schedule['Matchday'][i] != 1:
            for team in teams:
                # We get the rival for this match and the previous one
                rival_this_match = df_schedule[team][i]
                rival_previous_match = df_schedule[team][i-1]

                # We get the home stadium for this match and the previous one
                # If the first item of the string is equal to '@', then the team is playing away
                if rival_this_match[0] == '@':
                    home_this_match = rival_this_match[1:]
                else:
                    home_this_match = rival_this_match

                if rival_previous_match[0] == '@':
                    home_previous_match = rival_previous_match[1:]
                else:
                    home_previous_match = rival_previous_match

                # We add up the distance covered by team
                total_distance += Schedule.dist_matrix[(home_this_match, home_previous_match)]

        # We calculate the interest of each match
        for team in teams:
            rival_this_match = df_schedule[team][i]
            # We consider just the teams that are playing home to avoid duplicating results
            if rival_this_match[0] != '@':
                total_interest += Schedule.int_dict[(team, rival_this_match, i)]

    return total_distance, total_interest
```

RESOURCES

[What is Operations Research?]

[Operations Research and Optimization: Improving Decisions from Data]

[An Application of the Traveling Tournament Problem: The Argentine Volleyball League]

[Scheduling the Chilean Soccer League by Integer Programming]

[Using Sports Scheduling to Teach Integer Programming]

[OR Tools Introduction for Python]

Thanks!

Any questions?

You can find me at:

nico.garcia.ara@gmail.com

https://www.linkedin.com/in/nicol%C3%A1s-garc%C3%ADa-aramouni-60308710a/