Miljenko Šuflaj
0036501442

Heuristic Optimization Methods

Midterm task report: Fantasy Football Draft

Zagreb, November 2020

Contents

1 Introduction

This task was solved using **Python 3.8.5**.

2 The greedy algorithm

The greedy algorithm used looks a lot like a local one.

First we determine if for a given dataset there even is a solution by greedily constructing the cheapest football team. Note, that although we use this to generate a lower bound for the algorithm and to check if there is a solution, it is possible to construct a dataset where we won't be able to find a solution.
After determining the lower bound, we could discard the solution (just memorizing the costs of the chosen players), but instead we keep it as a current solution. Then we enter an improvement loop, infinite in our case.
At the start of each iteration, we first sort the players by score in ascending order. Then we iterate through the players in that order - each time we visit a player, we try and replace him with the highest score player in the same position, if the club constraint and our budget will allow us. If we do find a replacement, we do the replacement and restart this process.
If we do not find a player through the whole inner loop, that means that we cannot find a player that wouldn't violate our constraints we have money for. Although this might not be the optimal solution (because, for an example, it doesn't care about how many players get to actually play), we return it.

Although the swapping makes it look like local search, we could've done the exact same thing without swapping, and instead using the price lower bounds as an oracle that helps us pick the players. The role of the initial solution is not to provide something to improve on, but to get an idea of when to stop (because we're just improving the worst player by replacing it with the best-in-class).

2.1 Pseudocode

The pseudocode is as follows:

```
def greedy(
    players,
    price_budget,
    position_budget
    max_players_per_club
):
    money_left := price_budget

    cheapest_players := sort players by price asc, score desc
    number_of_players := count positions in position_budget
```

```
cheapest_solution := []

for player in cheapest_players
while length(cheapest_solution) < number_of_players:
    if (
        position_budget allows player in this position
        and taking player respects club constraints
    ):
        if player.price <= money_left:
            cheapest_solution = cheapest_solution + player
            money_left = player.price
        else:
            break

if length(cheapest_solution) < number_of_players:
    return NO SOLUTION

current_solution := cheapest_solution

forever:
    sort current_solution by score ascending

    for player in current_solution:
        hypothetical_money = money_left + player.price

        for new_player in available players with the same position:
            if (
                new_player.score > player.score and (
                    player.club is new_player.club
                    or taking player respects club constraints
                )
                and new_player.price <= hypothetical_money:
                    money_left = hypothetical_money - new_player.price
                    player is now new_player

                    break

    if no swap occured:
        return current_solution
```

3 The local search algorithm

Since our greedy algorithm returns a fairly good solution, and its weakness lies in the fact that it cannot deal with team benches, the main goal of our local search is to try and fix that. Our local search will also be carried out in 2 phases.

The first phase of the local search is minimizing the cost of our bench. Because we do not intend on changing the positions of our initial choice, we will try optimizing for specific roles. As such, we look at our bench, and try to minimize its cost while trying to keep within constraints.
To do this, first we disregard the clubs and act like we've totally erased the team members.
Then we get our candidates by adding all players with a cost equal or lower to a previous bench member inside its category. This will give us a mixed list of players.
Then we sort this list by price in ascending order, so we can focus on getting the cheapest team members first.
And finally, for every player candidate we try to find the ones to insert into our bench, if our club and position constraints allow for it.

After we've found the cheapest bench given our constraints, we go and see if we can improve the main team.
We enter an infinite loop where every iteration we sort the main team by score + price efficiency in an ascending order. We want to access the members with the lowest score and price efficiency first.
Then for every player in the main team, we find replacement candidates by selecting players whose score is higher than the current one and who play the same position. If we can find a player we can afford and that doesn't violate our constraints among better players that are sorted by score descending and price ascending, we swap the two and break the inner loop.
If after the inner loop is done we have done a swap, we break again to restart the whole process. If we never swapped, then we can't find a better substitute and our current solution is optimal, so we return the current solution.

3.1 Pseudocode

```
def local(
    current_solution,
    price_budget,
    position_budget,
    max_players_per_club
):
    main_team, bench = get main team and bench such that
                       the main team maximizes the sum of
                       its members scores
```

```
bench = find cheapest bench possible
current_solution = main_team + bench

money_left = price_budget - sum of prices in current_solution

forever:
    sort main team by score + price efficiency asc

    for player in main_team:
        hypothetical_money = money_left + player.price

        better_players = x from dataset where x.score > player.score
        sort better_players by score desc, price asc

        for better_player in better_players:
            if better_player.price <= hypothetical_money and (
                player.club is better_player.club
                or taking player respects club constraints
            ):
                money_left = hypothetical_money - better_player.price
                player = better_player
                break

        if swap occured:
            break

    if no swap occured:
        break

return main_team + bench
```

4 Results

The local search improved our results significantly.

For instance 1, we drafted the following players (players with a * are substitutes):

- 10 - (GK) Henderson
- 71 - (DEF) Alexander-Arnold
- 73 - (DEF) van Dijk
- 74 - (DEF) Pereira
- 119 - (DEF) Baldock

- 130 - (DEF) Lundstram
- 280 - (MID) Salah
- 283 - (MID) De Bruyne
- 434 - (MID) Cantwell
- 542 - (FW) Vardy
- 546 - (FW) Rashford
- 70 - (GK) Button*
- 521 - (MID) Guendouzi*
- 529 - (MID) Choudhury*
- 622 - (FW) Connolly*

This gave us a score of **1419**.

For instance 2, we drafted the following players (players with a * are substitutes):

- 16 - (GK) Pope
- 72 - (DEF) Alexander-Arnold
- 73 - (DEF) Robertson
- 74 - (DEF) van Dijk
- 75 - (DEF) Pereira
- 130 - (DEF) Lundstram
- 285 - (MID) De Bruyne
- 286 - (MID) Son
- 291 - (MID) Richarlison
- 545 - (FW) Vardy
- 554 - (FW) Ings
- 70 - (GK) Button*
- 521 - (MID) Guendouzi*
- 535 - (MID) Oriol Romeu*
- 622 - (FW) Greenwood*

This gave us a score of **1520**.