NBA Schedulling Improvement using Simulated Annealing

Frederico Portugal Pinho Rocha (up201408030)
MIEIC / PRODEI
FEUP
Porto, Portugal
fred.p.p.r@gmail.com

Abstract—This project handles data from the 2018/2019 NBA regular season's schedule and improves its travelling and rest efficiency, using a simulated annealing algorithm.

Keywords—NBA, scheduling, simulated annealing, reset cycles, python

I. Introduction

Since the NBA's inception, its scheduling was made by hand, until 2015, where AI was first officially introduced to solve the league's problem. This project was developed as a way to improve the current AI-made NBA schedules, using a simulated annealing algorithm in python.

The league is composed by 30 teams, separated by half into 2 conferences, and each conference having 3 divisions, holding 5 teams each. Each team has a certain number of games it has to play throughout the season against other teams depending on which conference and division they belong to. Each team has to play exactly 82 games, totalling to 1230 games, in a matter of roughly 178 days.

When solving its scheduling problem, the NBA accounts for each team's travelling and rest goals, but also for the its own business goals, such as having popular games happen on good dates. In this project, it's only considered the teams' constraints.

The common sports schedule is represented as a Home-Away Pattern (HAP), where each team has its home-away status attributed for each round, and as an Opponent Schedule Matrix, where each team has its opponent attributed for each round. This project utilizes a single combination of the two. The simulated annealing metaheuristic was chosen as the way to improve the schedule due to the problem's heuristic-based goals and its NP-hard status. Additionally, simulated annealing benefits greatly from starting with a viable solution.

II. Design & Implementation

A. Data Handling

The extracted[1] schedule data for the 2018/2019 regular season first contained some irrelevant data to the problem, and some data which had to be slightly altered in order to become usable.

The schedule's data structure is modelled after the extracted schedule data, as a list of all 1230 games, where each game contains data regarding its day in the tournament, home team, and away team. Additionally, each team's games are stored into their own array, in order to facilitate the creation and fitness evaluation of neighbouring states.

A data structure containing a list of each team, its conference, and its division was created as well. After a schedule's first evaluation, this list will also include each team's distance score and rest score.

B. *Fitness Evaluation*

As previously mentioned, the fitness evaluation of a state will only take into consideration criteria regarding the teams' distance travelled inbetween games and the amount of resting days inbetween games. The criteria chosen were to:

- maximise total rest days inbetween games, for each team
- minimise total distance travelled, for each team
- minimise the disparity between each team's score

A team's Rest Score is calculated as such:

Σ days(n-1, n) + days(n, n+1)

days(n, m) =

- 0, if *m* - *n* > 4
- 1, if *m* - *n* = 4
- 2, if *m* - *n* = 3
- 5, if *m* - *n* = 2
- 8, if *m* - *n* = 1
- max_int, if m-n = 0

where n and m correspond to game days on a team's schedule. The values were chosen as to favour a more homogenous spacing between games, considering a team plays around 3 to 4 times every week, and to discourage back to back games and three games in five days, which is one of A max_int score is given when a schedule generates a team to play twice in the same day, in order to reject those possibilities.

A team's Travel Score is calculated as such:

Σ distance(n-1, n) + distance(n, n+1)

distance(n, m) =

- 0, if *m* and *n* are the same team
- 2, if *m* and *n* belong to the same division
- 4, if *m* and *n* belong to the same conference, but different divisions
- 10, if *m* and *n* belong to different conferences

The values were chosen to discourage solutions where 3 consecutive games for a team happen very far apart.

The schedule's fitness score is calculated as such:

Σ Rest Scores + Σ Travel Scores + [Standard Deviation(Rest Scores) + Standard Deviation(Travel Scores)] * Number of Teams

Standard deviation of the scores are added to the fitness as a way to prevent massive disparities between teams' scores.

C. *Neighbours*

Three methods to create neighbour states were considered.

Altering game dates, by choosing a random game and assigning it another available date. It has a range of possible solutions of between 12.300 and 110.700, and it affects 2 team's schedules.

Switching the dates of two games from a random team. It affects between 2 to 3 team's schedules and has a range of possible solutions of around 217.000.

Switching the home and away status for each team in a random game. To comply with league regulations, another encounter between both teams also has its home/away switched. It has a range of possible solutions of around 3.700 and affects two teams.

When evaluating the fitness of a neighbour state, a simplified fitness function, which only measures the changes in the affected teams, is called. This neighbour evaluating function ends up being between 10 times and 15 times faster than the state evaluation function.

D. Simulated Annealing

The algorithm is made to run a determined number of cycles until it stops and returns the initial and final fitness scores, as well as the improvement rate.

Each cycle it creates a neighbour state using one of the 3 possible methods. It then decides if the neighbour state is accepted depending on the movement cost, which is calculated as the difference between the neighbour's fitness and the current state's fitness. If the movement cost is negative or zero, then the neighbour state is accepted. In the case in which the cost is positive, its acceptance depends on the current temperature. The acceptance probability is calculated as such:

e^(- Cost / Temperature)

The system's initial temperature was decided to be 250, after various tests. After a cycle, if the neighbour state is accepted, then the temperature reduces to 80% of its value. Temperature resets were introduced as an improvement option, where as in case there isn't an accepted neighbour state in 200 consecutive cycles, the temperature resets to its initial value, allowing for more exploration in the algorithm.

Likewise, a best score reset method was tested as an improvement, where as in every time a state with the best fitness score is found and it is saved, along with its attributes. Every 400 cycles, the system returns to the the best state. By introducing reset cycles^2, it becomes possible to correct wrong paths when trying to exit a local minimum.

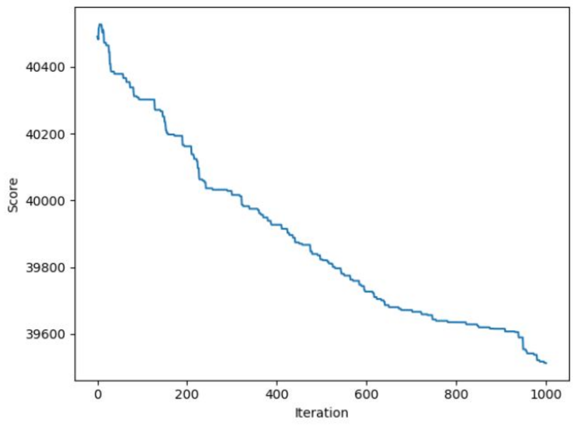
III. Testing

The algorithm was tested for the best initial temperature and temperature updates, as well as for the best combination of neighbour movements. Furthermore, the existance of reset cycles was tested to prove its usefulness. The algorithm in its final version was then tested against its default version in the Hill climbing algorithm.

A. Individual Neighbour Movements

Date Switching

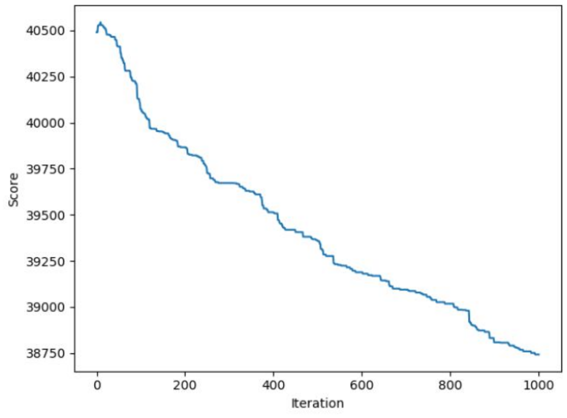
This neighbouring method brough upon good results on a first exploration phase of 1.000 cycles, and, considering its good range of possible solutions, is expected to provide good exploration with neighbour states in the long run.



1.000 cycles date switching

Home-Away Switching

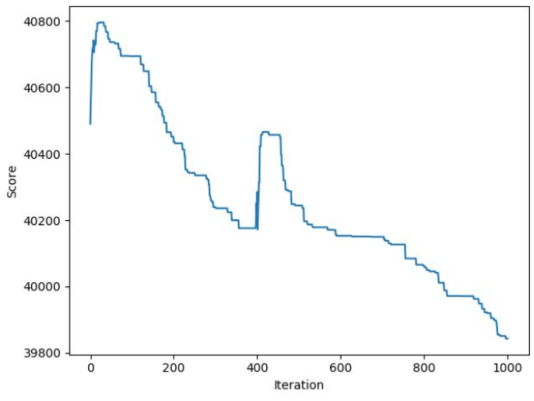
This neighbouring method brough upon the best results on a first exploration phase of 1.000 cycles, with an improvement rate of twice as much as the date switching. Considering its poor range of possible solutions, it is not expected to provide good exploration with neighbour states in the long run.



1.000 cycles home-away switching

Games Switching

This neighbouring method brough upon the worst results on a first exploration phase of 1.000 cycles, with a various instances where 200 cycles of improvements are quickly backtracked by a poor neighbour, *eg.* at around cycle 400. Considering its great range of possible solutions, however, it is expected to provide the best neighbour exploration when the algorithm improvements start stagnating in the long run.



1.000 cycles games switching

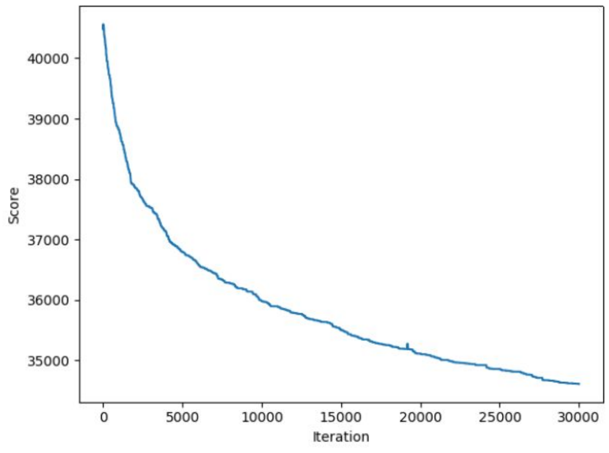
B. *Mixed Neighbour Movements*

Considering the results from the individual neighbouring methods testing, and choosing to follow a simplistic approach, the following move probabilities were assigned to the neighbouring methods:

- date switching = 30%
- home-away switching = 60%
- games switching = 10%

This mixed neighbouring approach was tested on 1.000, 10.000, 20.000, and 30.000 cycles. On the smallest cycle run, it showed somewhat similar results to the ones obtained from simply using the home-away switching method.

On the longer cycle runs, it demonstrated a two-phased pattern. On the first 3.000 cycles it had a steady improvement rate, with very few occurences bad neighbours being accepted, while on the following cycles there was an increasing acceptance rate of bad results and, consequently, an increase in best score resets. At this second phase, the algorithm starts entering stagnation, following a 1/x plotting pattern.

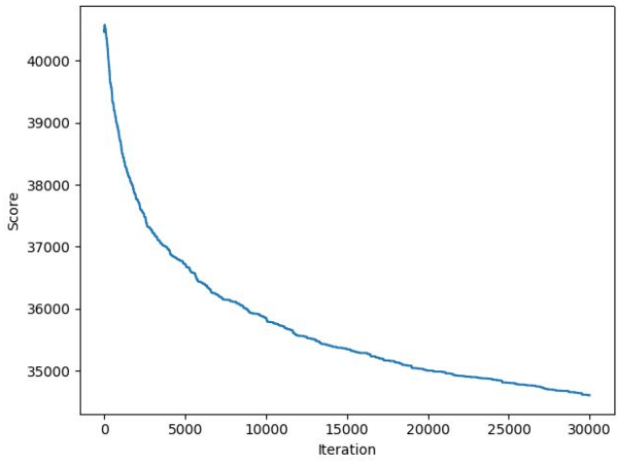


30.000 cycles mixed neighbouring

C. *Reset Cycles*

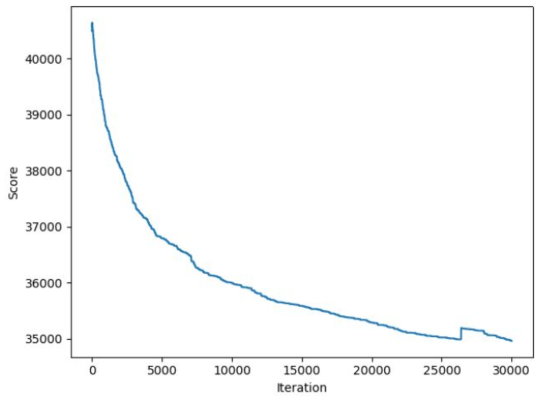
The algorithm was tested with no resets, and with only temperature resets, vs. with all resets, on a long cycle run such as 30.000 cycles, since the resets are expected to be the most useful when the algorithm starts facing stagnation.

The algorithm functioning with no resets exhibited a smoother pattern, as it very rarely considered bad options. While the results were, on average, better than with both resets, it is very liable to stopping at local minimums.



30.000 cycles no resets

The algorithm functioning with only temperature resets exhibited a rocky pattern, as it considered bad neighbours more often, providing an improvement rate consistently worse than the other options.

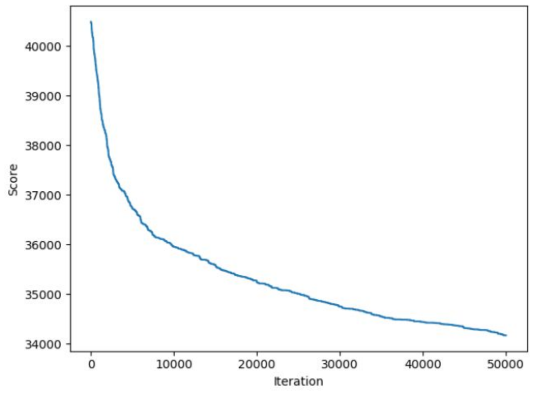


30.000 cycles temperature resets

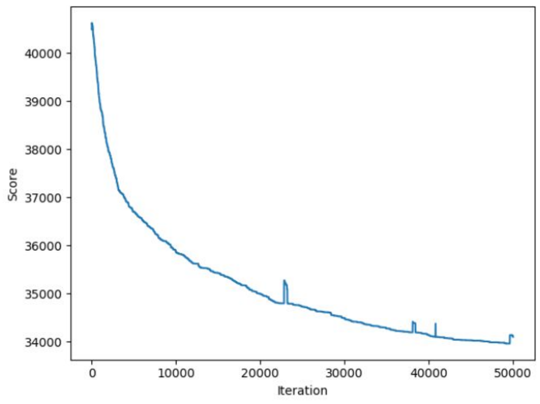
Ultimately, the algorithm with both resets fully guarantees the avoidance of local minimums, while sacrificing .

D. *Hill Climbing*

Running on a 50.000 cycle loop, the hill climbing algorithm obtained an improvement of around 15%, while on the same cycle range, the simulated annealing with resets obtained a better improvement, of 16%, even after suffering noticeable setbacks throughout its runtime.



50.000 cycles hill climbing



50.000 cycle simulated annealing w/ resets

IV. Conclusions

A. *Results*

Using an adapted version of the simulated annealing algorithm, it was possible to improve the 2018/2019 NBA schedule by focusing on team oriented scheduling goals, as proposed. The biggest improvement recorded ranged the 16th percentile, at 50.000 cycles, however, by analysing it's graph, it is believed that a better score can be achieved from running the algorithm for a longer cycle run.

The simulated annealing algorithm with temperature and best score resets proved to be an improvement over both the standard hill climbing algorithm and the simulated annealing without resets, for this case, showing its strong points when the system starts entering stagnation, at around the 20.000th cycle.

B. *Further Improvements*

Calculating the initial temperature of a simulated annealing system is understood to be a very complex endeavor[3]. As such, by experimenting with different initial temperatures and different temperature variation functions could lead to an improvement in the algorithm's performance.

Furthermore, the neighbouring method probabilities could potentially be changed to a more adaptable alternative, where each neighbouring method is given more weight in situations where its best strengths are needed the most, instead of remaining at a static value for the entire duration of the algorithm’s runtime.

V. References

[1] *https://fixturedownload.com/results/nba-2018*

[2] *https://editions-rnti.fr/render_pdf.php?p=1002455*

[3] https://www.researchgate.net/publication/227061666_Computing_the_Initial_Temperature_of_Simulated_Annealing

A. NBA Scheduling

- *http://www2.ic.uff.br/~celso/artigos/Scheduling%20in%20sports%20C&OR*
- *https://basketball-datascience.com/2017/08/21/how-the-nba-uses-analytics-to-create-an-efficient-schedule-that-features-1230-games-in-169-days*
- *https://www.nbastuffer.com/analytics101/how-the-nba-schedule-is-made*
- *https://news.cgtn.com/news/3d3d674d7749444e79457a6333566d54/share_p.html*
- *https://pdfs.semanticscholar.org/eb6d/6de22be67b1e420041e9f60b5f39bca3da29.pdf*
- *https://www.math.cmu.edu/~af1p/Teaching/OR2/Projects/P49/21-393ProjectPaper_Group1.pdf*

B. Simulated Annealing

- *https://www.math.cmu.edu/~af1p/Teaching/OR2/Projects/P49/21-393ProjectPaper_Group1.pdf*