

**Master Programme**

**Heuristic Optimization Methods**

REPORT – Lab2  **Fantasy football draft problem**

***Antoine Vieville***

**Ac. year 2023/2024**

**Contents**

1 Summary of best-found results 3

2 Tabu search (TS) 4

2.1 Pseudocode 4

2.2 Description 4

2.3 Analysis 5

3 Simulated annealing (SA) 7

3.1 Pseudocode 7

3.2 Description 7

3.3 Analysis 8

# Summary of best-found results

**Instance 1**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Score | First team lineup | Substitutions |
| Tabu Search (initial solution: greedy) | 1420 | 71,283,542,280,130,551,10,119,291,74,343 | 70,273,623,520 |
| Tabu Search (initial solution: random) | 1324 | 10,142,118,283,280,330,291,167,71,551,549 | 277,70,623,532 |
| Simulated Annealing (initial solution: greedy) | 1420 | 71,283,542,280,130,551,10,119,291,74,343 | 70,273,623,520 |
| Simulated Annealing (initial solution: random) | 1031 | 15,73,95,330,284,337,427,76,74,556,559 | 70,273,623,520 |

**Instance 2**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Score | First team lineup | Substitutions |
| Tabu Search (initial solution: greedy) | 1518 | 72,285,282,16,545,551,139,291,75,119,445 | 70,271,272,626 |
| Tabu Search (initial solution: random) | 1380 | 139,285,398,77,74,157,282,543,326,545,16 | 275,70,626,533 |
| Simulated Annealing (initial solution: greedy) | 1518 | 72,285,282,16,545,551,139,291,75,119,445 | 70,271,272,626 |
| Simulated Annealing (initial solution: random) | 1276 | 118,283,345,170,74,110,326,554,344,545,9 | 275,70,626,533 |

Programming language: Python

Note : I wasn’t able to improve best found results of the greedy search using tabu or SA search. However, when plotting the behavior of both algorithms, it seems that they accomplish what they are supposed to do in theory (as will be shown on the plots in the next parts). This is why the best results are the same for both algorithms.

# Tabu search (TS)

## Pseudocode

**Start** tabu search

**Input** : instance, solution, tabu tenure, max iter

Initial solution 🡨 solution, best solution 🡨 initial solution, tabu list 🡨 empty list

**Repeat** :

Generate neighborhood of current solution

Select the best neighbor :

If tabu : select next best neighbor

Else : compare best neighbor to current best solution

If best neighbor is better : best solution 🡨 best neighbor

current solution 🡨 best neighbor

Update the tabu list

**Until** max iter is reached

**Return** best solution

**End**

## Description

* *Specify the neighborhood structure used.*

For the neighborhood structure, I used a similar structure as for the local search in the previous lab, that is “**one player switch of the same position**”. I used a restricted neighborhood, in size, limited to half the total number of valid switches. I didn’t “orient” this neighborhood generation with criterion like points or ratio to make sure we keep the diversifying aspect of the tabu and SA search.

* *Specify the structure of the tabu list. What is the tabu tenure?*

My tabu list is a list of length 11, containing values j st. *0<= j <= tabu\_tenure*. The index of values in this list correspond to the index of the players switched j iterations ago.

If *index(j) = 0*, the player in index(j) has not been switch recently (in the last *tabu\_tenure* iterations) and therefore this switch is allowed (the **solution** obtained **is** **not tabu**)

If *index(j) >0*, then this switch is not allowed because it has already been performed recently (the **solution** that would be obtained **is tabu**).

* *Did you use any additional memory structures to intensify or diversify your search? Did you apply the aspiration criterion?*

The tabu list that I used allows for a diversified and intensified search (we force different switches throughout the iterations to avoid revisiting previously found sub optimal solution, and with a sufficient tenure we do not get stuck in local optimums). However, I didn’t use other memory structures. For the **aspiration criterion**, I chose a **criterion based on the quality of the solution** : if the solution is “much” better than the current best solution (its fitness is at least 20% higher), then we should accept it even if it is tabu.

* *What was your stopping criteria?*

For the **stopping criteria**, I have set a **maximum number of iterations** that we can change as a parameter.

## Analysis

* *Graphically portray the relationship between final solution quality and tabu tenure. Discuss your results. Show curves for both instances and in both cases when TS is initiated with a greedy solution and TS is initiated with a random solution.*

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

*Une image contenant texte, capture d’écran, ligne, diagramme

Description générée automatiquement*

Instance 1, max iter = 100

Une image contenant texte, capture d’écran, ligne, diagramme

Description générée automatiquement

Instance 2, max iter = 100

We observe that, for both random initial solutions and both instances, there is a trend : **as the tenure grows, the fitness of the best found solution decreases**. This makes sense because the higher the tenure, the more restrictive the search becomes, and might thus be less efficient. The fitness for the initial greedy solution seems unsensitive to tenure changes, maybe because the initial solution is already pretty good and potentially close to globally optimal. I couldn’t figure how, starting from the greedy solution, to modify the algorithm to make achieve this “tenure sensitivity”.

* *For a chosen tabu tenure and tabu structure, graphically portray how solution quality changes over successive iterations of your TS algorithm. Show curves for both cases when TS is initiated with a greedy solution and TS is initiated with a random solution.*

*Une image contenant capture d’écran, texte

Description générée automatiquement*

Une image contenant texte, capture d’écran, diagramme, Police

Description générée automatiquement

max iter = 300 for both cases, tenure = 5 for greedy, tenure = 2 for random

We observe that over the iterations, the tabu search explores the region and indeed espaces local optimums. With a sufficient number of iterations, we see that the search may reach a good optimum, and maybe even the global optimum. This is more obvious when starting with a random solution of poor quality : we are very likely to find improving neighbors. For the case of a initial greedy and good solution, since the search always goes forward, we are more likely to go from a better to a worse solution (because a better neighbor will be rarer), but this process allows to explore other regions of the search space.

* *Find solutions to the given instances using both the Greedy and the GRASP algorithms you implemented in Lab1. Indicate the number of iterations you ran GRASP. Compare these solutions with the best solution you found using TS (starting with a greedy initial solution). Comment on your results. Compare approximate execution time.*

Instance 1 :

Greedy search : score = **1420**

Grasp : score = 1385(ran for 10 iterations)

Tabu search : score = **420**

Instance 2 :

Greedy search : score = **1518**

Grasp : score = 1479 (ran for 10 iterations)

Tabu search : score = **1518**

We observe that, for both instances, tabu search (from greedy search) gets the best results. Grasp seems to underperform but we have to keep in mind that it was run over 10 initial randomized greedy solutions.

However, grasp runs faster than the tabu search.

* *How do you expect your TS algorithm would perform for much larger instance sizes?*

My implementation of the algorithm takes a lot of time to process theses instances, considering the neighborhood generation phase at each iteration. I haven’t found a way to make this more efficient in the given time. Then, I guess the algorithm will perform slowly in much larger instances.

# Simulated annealing (SA)

## Pseudocode

**Start** simulated annealing

**Input** : instance, solution, initial temperature, final temperature, decrement (alpha or beta)

Initial solution 🡨 solution, best solution 🡨 solution

**Repeat** :

Generate neighbor of current solution

If neighbor is improving : current solution 🡨 neighbor, best solution 🡨 neighbor

Else : accept neighbor with probability p = exp(delta\_fitness/temperature)

Update temperature wrt. Decrement (geometric or slow decrease)

**Until** T reaches final temperature

**Return** best solution

## Description

* *Specify the neighborhood structure used.*

Same neighborhood structure as for tabu search.

* *Specify the initial temperatures used.*

For both instances, I ran a “grid search” to find the combination (T0,alpha) for geometric decrement and (T0,beta) for slow decrease giving the best results.

* *Specify the temperature decrement function(s) used.*

I used geometric and slow decrease. The decrement is passed as an argument of the SA algorithm.

* *Specify the cooling schedule.*

I implemented a homogeneous cooling schedule.

* *What was your stopping criteria?*

For the stopping criteria, I set a fixed final temperature of 0.01 , close to 0, like suggested in the lecture. Changing this value (higher or lower) didn’t change the results, so I didn’t include this as a “third dimension” of the grid search.

## Analysis

* *Graphically portray the relationship between final solution quality and initial temperature. Indicate the chosen decrement function and discuss your results. Show curves for both instances and in both cases when SA is initiated with a greedy solution and SA is initiated with a random solution. Optionally, additionally analyze the impact of the decrement function.*

Une image contenant texte, capture d’écran, diagramme, ligne

Description générée automatiquement

Instance 1, alpha = 0.99

*Une image contenant texte, capture d’écran, ligne, diagramme

Description générée automatiquement*

Instance 2, alpha = 0.99

* *For a chosen initial temperature and decrement function, graphically portray how solution quality changed over successive iterations. Show curves for both cases when SA is initiated with a greedy solution and SA is initiated with a random solution.*

*Une image contenant texte, diagramme, capture d’écran, carte

Description générée automatiquement*

Instance 1, T0 = 108, alpha = 0.99

For a set cooling schedule, we observe that, for an initial random solution, the fitness of the solution increases with the iterations. Again, starting with a solution of poor fitness, we are very likely to encounter an improving neighbor and therefore accept it with a probability of 1. For the case of a greedy initial solution, we see that non improving neighbor are accepted in a probabilistic way. In this case, over the iterations, we see how the SA allows not to get stuck in a local optimum and explore other region. If the initial solution is not too bad (unlike a totally random initial solution), we can hope that the search will find better solutions in the scope of the run, by exploring other regions and intensifiying in promising regions.

Regarding the decrement function, the plot were similar and the run of the algorithm for the slow decrease decrement took some time to execute on my machine.

* *Once again consider solutions to both given instances using the greedy and GRASP algorithms you implemented in Lab1. Compare these solutions with the best solution you found using SA (starting with a greedy initial solution). Comment on your results. Compare approximate execution time.*

Same as for tabu search…

* *How do you expect your SA algorithm would perform for much larger instance sizes?*

Same answer as for the Tabu search, the algorithm takes a considerable time on these instances there fore I imagine it would perform slowly on instances of much larger size.