First report on the Tabu Search Algorithm

Fanny Kalinowski, Julien Molinier, Robin Lambert, Maxime Leras

25 Mars 2020

Sommaire

1 Introduction

Tabu Search is an algorithm created in 1986. It uses a deterministic heuristic on local search methods. The goal is to take a potential solution, that's to stay a local optimum, and to check its immediate neighbors in the hope of finding an improved solution. [1]

The objective of this report is to analyse an implementation of the Tabu Search algorithm through various configurations. Then, it will be interesting to discuss on the effects of changing the paramaters values.

2 Tests without Tabu duration on 10 cities

2.1 Run of the algorithm

First of all, the algorithm was run 10 times for 5, 10 and 100 iterations. The algorithm stops when it found the global minimum which is supposed to be 3473 km. The results are in the table below :

Run / Nb_iterations	**5**	**10**	**100**
1	-	5	4
2	-	7	4
3	-	5	4
4	-	4	6
5	-	6	4
6	3	5	7
7	-	7	4
8	-	7	4
9	-	5	7
10	-	3	5

Table 1: Number of iteration needed without tabu duration on 10 cities

From this table, the computation of the average gives 4.9 on 10 iterations and 4.9 on 100 iterations. It means the algorithm needs around 5 iterations to find the global minimum. Thus, it explains why the algorithm ended only 1 time when the Nb_iterations is equal to 5.

[1]https://en.wikipedia.org/wiki/Tabu_search

2.2 Convergence

The values computed in section 2.1 shows that the algorithm needs 5 iterations on average to converge to the global minimum. After, the algorithm computes the neighbors of this solution but as the previous solution is the global minimum the neighbors it won't be better. So the global minimum remains the same and the algorithm repeats this step again and again until the maximum number of iterations sets before launching. This is why when the maximum is 5 the algorithm can't always find the global minimum.

2.3 Number of solutions and neighbors with 2-opt

For 10 cities the number of solutions is *9!/2* and for 50 cities it is *49!/2*.

The number of neighbors with 2-opt follows the formula :

$$\binom{n}{1}\binom{n-3}{1}/2$$

For 10 cities it gives 35 and for 50 cities 1175 neighbors.

2.4 Algorithm performance

The number of solutions visited by the algorithm before the convergence is given by the formula *Nb_iterations * Nb_neighbors*. So with 10 cities if the algorithm finds the solution at the iteration 4 it means that 40 solutions were visited. Compared to the *9!/2* possible ones the algorithm tries only 1 out of 10000 solutions; this is pretty good.

3 Tests without Tabu duration on 50 cities

3.1 Run of the algorithm

Run / Nb_iterations	**10**	**100**	**1000**
1	-	44	50
2	-	44	47
3	-	48	43
4	-	48	46
5	-	40	44
6	-	42	44
7	-	44	47
8	-	43	42
9	-	42	43
10	-	44	47

Table 2: Number of iteration needed without tabu duration on 50 cities

For 10 iterations, the algorithm never found the global minimum. On average, the global minimum is found at the 43.9th for 100 iterations. For 1000 iterations, the global minimum is found at the 45.3th iteration.

3.2 Convergence

The algorithm converges during 43.9 iterations for Nb_iterations equals to 100 iterations, and during 45.3 iterations for Nb_iterations equals to 1000 iterations.

Once the algorithm has found the global minimum, it doesn't explore other paths. In the following iterations, the value of the kilometers is still blocked on the global minimum. According to the algorithm, the global minima is the best solution until yet, so there is no need to explore the neighborhood around the top. This is due to the Duration_tabou which is equal to 0. This means that the number of solution that have been explored successively are not kept in memory. Thus, the same points are explored several times. Consequently, the local minimum is not going to change.

3.3 Solutions visited before the convergence

The algorithm starts to converge once it has found a local minimum. So, in order to know the number of solutions visited before its convergence we need to count all the solutions that have been explored before the finding of the local minimum. In the first run, with Nb_iterations equals to 100, *Iteration_before_finding*1175* solutions have been explored before the finding of the local minimum. So, for each of the iterations, the best solution is displayed. It's the result of the exploration of all the 1175 neighbors.

Run / Nb_iterations	**10**	**100**	**1000**
1	-	43	49
2	-	43	46
3	-	47	42
4	-	47	45
5	-	39	43
6	-	41	43
7	-	43	46
8	-	42	41
9	-	41	42
10	-	41	46

Table 3: Number of iterations before the finding of the best solution without Duration_tabou on 50 cities

Run / Nb_iterations	**10**	**100**	**1000**
1	-	50525	57575
2	-	50525	54060
3	-	55225	49350
4	-	55225	52875
5	-	45825	50525
6	-	48175	50525
7	-	50525	54060
8	-	49350	48175
9	-	48175	49350
10	-	48175	54060

Table 4: Number of solutions visited before the convergence without Duration_tabou on 50 cities

On average, 50172.5 solutions have been explored before finding the local minima for Nb_iterations equal to 100, against 52055.5 explored solutions for

Nb_iterations equal to 1000. The two runs show that the local minima is found between the 42.7th and the 44.3th iteration.

This is logical, since the Duration_tabou is equal to 0 in the two runs. Increasing the number of iterations is interesting if the value of the Duration_tabou changes. In fact, it will allow the algorithm not to go through the same path and to have more time (thanks to the iterations) to explore. If the value of the Duration_tabou doesn't change, once the algorithm has found the local minimum, it will stop to converge to it, no matters the number of iterations. So here, the number of iterations does not influence the local minima. On 100 iterations, the local minimum is found at the 43th iteration. It results of a convergence. In the 57 following iterations, the algorithm is blocked to this local minima. That's to say that during 57 iterations there is not relevant information, as the local minimum is not going to change. On 1000 iterations, the algorithm converges to the local minimum. It's found at the 44th iteration. So, the algorithm is blocked in the following 956 iterations, meaning that during 956 iterations there is not relevant information. The neighbors are not visited since there are less interesting than the local minimum. Since the time loss is less important on 100 iterations than on 1000 iterations, the one can say that running the algorithm on 100 iterations allows to gain performance.

4 Tests with Tabu duration

4.1 Run of the algorithm

The algorithm was run 10 times for Nb_iterations equal to 1000.

Run	**Best solution in km**	**Nb_local minima visited**	**Nb_BetterSolution**
1	5691	265	43
2	5691	300	54
3	5644	294	47
4	5644	226	49
5	5771	271	48
6	5644	271	47
7	5798	242	49
8	5770	276	62
9	5644	300	48
10	5720	216	59

Table 5: Number of local minima visited with tabu duration on 50 cities

4.2 Improving the best solution

On average, the best solution is improved 50.6 times in a run. This is mostly due to the first descent to the first found local minima. If we count the number of local minimum that improves the previous local minima, we have an average of 3 improvements in a run.

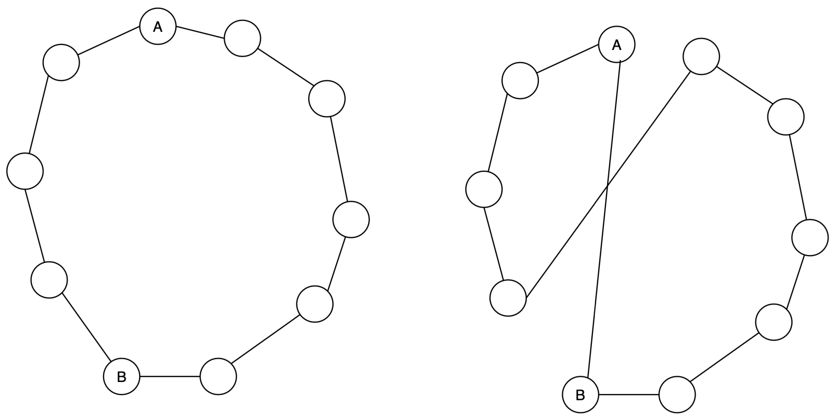
4.3 Local minima

On average, 266 local minima are visited by the algorithm. Duration_tabou is the number of solutions that have been explored successively. They are kept in memory like a path. So, the solutions belonging to the path could not be explored anymore. The one can see that the path allows the algorithm to escape from the mountains. It goes to a local optimum, escapes and then goes to another one. After having visited all the local minima, the algorithm can find the major top, so the best solution. Consequently, the algorithm doesn't converge because the solution moves from one local optima to another one. New local minima can be explored and the global minima can be updated. Introducing the tabu duration allows to use properly the algorithm. First, the Tabu search for the best neighbor up to the local optimum. After being on the top, it explores around the top in order to find the best neighbor. So, the metric Duration_tabou allows to search other neighbors by memorizing and excluding the previous paths. This is why new local optima are found. After going down this area, the algorithm jumps to another local optimum and explore new paths. Doing this way, the algorithm moves from one area to another one. At the end, the major top is the result of a complete exploration. In the section B, with the following configuration ./algo_tabou 1000 0 50 distances_entre_villes_50.txt the best solution is on average 5891.7 km. In this section, with the following configuration ./algo_tabou 1000 10 50 distances_entre_villes_50.txt the best solution is on average 5701.7 km. So, introducing the Duration_tabou allows the algorithm to gain in performance in terms of space because new mountains with new local minima are explored. At the end of the exploration, the less local minima can be considered as the major top, thus as the best solution. With the Duration_tabou equals to zero, only the local optimum of one area is explored. If we're lucky, the best solution is contained in this unique area. But in most cases, the major local minima is contained in another mountain. So what the algorithm considers as the best solution is probably the wrong one. Nevertheless, 5701.7 km may not be the best solution of all the area. In this configuration, only 10 solutions are registered and belong to the path. It's possible that the solutions who do not belong the path have been explored more than once. We have to change the value of the Duration_tabou increase, to see the effects of expanding the path.

5 Variation in Tabu duration

5.1 Explanation on the code

The Tabu list is coded so that it can be able to store, for a couple of two cities, if a switch move for all cities between them is Tabu or not. Here's an example of this move bewteen the cities A and B :



Such a move is considered as "tabu" if it has already been done before. However, the code will not remember a move after a specific number of iterations : this is the tabu duration. Added to the current iteration number, its value is stored inside the tabu list for a couple of two cities. By comparing the stored value to the current iteration, the algorithm can determine if the move is tabu or not. '-1' is stored for each couple before exploring any solution, so the first move cannot be tabu. There's also another list in the code that stores all tabu solutions, instead of the moves. This list could have been used too.

The algorithm explores all possible moves (between each cities) for a given solution : this is the neighborhood. The best solution resulting from a switch move which is not tabu is now defined as the current one.

The interesting thing is that the tabu list avoids the algorithm to be stuck in a local minimum. Because it chooses the best solution amongs the non visited neighboors, it can go for a solution which isn't necessary improving the result. If the tabu duration is long enough, it can easily get out of local minimum. However, choosing a too important value is not a good idea, because it can be completely surrounded by already visited solutions, or surrounds the best solution and be stuck again. So technically, it is possible to chose a tabu duration as long as the number of iterations, but of course it is not a good practice to do so.

5.2 Run of the algorithm

The algorithm was run 5 times for Nb_iterations equal to 1000.

Run / Duration_tabou	**20**	**50**	**80**	**100**	**1000**
1	250	209	218	220	286
2	239	213	198	208	270
3	228	219	207	243	274
4	253	200	209	235	281
5	231	229	216	223	273

Table 6: Number of local minima visited with various Duration_tabou on 50 cities

Run / Duration_tabou	**20**	**50**	**80**	**100**	**1000**
1	5649	5649	5746	5698	5867
2	5649	5775	5706	5780	5715
3	5799	5649	5649	5711	5803
4	5649	5809	5716	5692	5764
5	5649	5682	5661	5691	5716

Table 7: Best solutions with various Duration_tabou on 50 cities

5.3 Improving the best solution

Run / Duration_tabou	**20**	**50**	**80**	**100**	**1000**
Solution improve (avg) / Minima improve (avg)	50.2 / 2.8	51.4/ 3.8	51.2/ 3.4	52.2/ 4	48.8/ 1.8

Table 8: Number of improvements of the solution (Average)

The table shows the number of times a better solution than the previous one was found. It also shows the number of local minima that were found improving the solution. The results are more or less the same except when Duration_tabou is 1000 : the number of improvements is less than other results but the greatest change is in the finding of multiple better local minimum where we have only an average of 1.8 local minima. It gives an Idea for the part *Infinite Value for the Tabu Duration.*

5.4 Local minima

Duration_tabou	**20**	**50**	**80**	**100**	**1000**
local minima	120.1	107	104.8	112.7	138.4

Table 9: Average of local minima visited with various Duration_tabou on 50 cities

The number of local minima increases with the Tabu Duration but the number of local minima is limited so it converges. It's not linear as we have a lot of local minima that are visited for only 20 as Tabu duration, less for 80 as Tabu Duration and for 1000 as Tabu Duration, we visit more local minima.

5.5 Parameters settings recommanded

Duration_tabou	**20**	**50**	**80**	**100**	**1000**
best solution in km	5679	5712.8	5695.6	5714.4	5773

Table 10: Average of best solutions obtained with various Duration_tabou on 50 cities

Increasing the Duration_tabou more than 20 has no good effects on the best solution because it becomes bigger and bigger. It appears that choosing 80 as Duration_tabou is the best solution here. As said in the first section of this report it is important to choose an Nb_iterations large enough to find the global minima but it's useless to take it too large because it will just extend the algorithm running time. 100 iterations is a good compromise because we saw in section 3 that for 50 cities the average was around 44 iterations.

5.6 Infinite value for the Tabu duration

The Tabu Duration prohibits to reuse the move for a number of iterations. This is why the number must be set according to the number of variables and possible values for the variables. In fact if this number is to big all moves will be prohibited and the algorithm blocked.

6 Conclusion

The tabu algorithm is a very powerful tool that could be used to solve efficiently a complicated problem which would be impossible to solve in a decent time by exploring all solutions. However, having a good comprehension of how it works is essential to use the optimal parameters in order to find the best solution. Doing this work was a great way to understand how a specific parameter can impact performance, at different scales.