IMPLEMENTATION OF THE TABU SEARCH FOR THE MAXIMUM DIVERSITY PROBLEM

C. Arnau[†], J. Pascual[†], A. Porras[†], D. Serra[†]

[†]Universitat de València

Abstract

In this article, we present our implementation of the Tabu Search heuristic applied to the Maximum Diversity Problem (MDP), detailing the decisions made during its development. Additionally, we conduct a comparative study with the GRASP method, demonstrating how, in certain cases, our implementation outperforms GRASP in terms of solution quality. While long-term memory techniques could be incorporated, we limit our approach to the foundational principles of Tabu Search. Furthermore, we highlight the significance of the diversity problem in practical applications, extending beyond the business domain (e.g., team composition and inclusion) to fields such as education (e.g., managing multicultural classrooms), technology, and political science.

1 Introduction

Let us first examine what the *Maximum Diversity Problem* (MDP) consists of, which we aim to solve with our heuristic algorithm. It is important to note that this is an NP-hard problem, meaning that there is no known algorithm that can solve it in polynomial time. This is where heuristic techniques become relevant, as they allow us to find good solutions within a reasonable time frame. The optimization problem consists of, given a set of points, finding a subset of a given length that maximizes the sum of the distances between them. As input, we receive a set N of n points, from which we will determine the subset S with $|S| = p$ that maximizes the objective function. In mathematical notation, we define the variables as

$$x_i = \begin{cases} 1, & \text{if } e_i \in S \\ 0, & \text{if } e_i \notin S \end{cases}$$

and let $d_{ij} = d(e_i, e_j)$. The goal is to

$$\text{Maximize} \qquad z = \sum_{i=1}^{n} \sum_{j=i+1}^{n} d_{ij} x_i x_j,$$

$$\text{subject to} \qquad \sum_{i=1}^{n} x_i = p.$$

Obviously, to compute the objective function computation we will need to have defined a distance between the points of the set N. Note that this can be an arbitrary distance and we will consider different distances for different problems.

The first approach we studied to provide a solution to the MDP problem is the GRASP algorithm due to its conceptual simplicity. The GRASP (*Greedy Randomized Adaptive Search Procedures*) method is a heuristic technique designed to obtain good results in a reasonable computation time for combinatorial optimization problems where the size of the search space makes finding the optimum not possible (in terms of available time). The main idea of this method consists of dividing the process into two phases: the construction phase for the initial solution and the local search phase that aims to improve this solution.

1.1 GRASP iteration

During the construction phase, we use a Greedy approach to choose which elements we will initially introduce in the solution. We will construct the solution by adding the p elements one by one. Since it is a randomized Greedy algorithm, we restrict the list of candidates to be included in the initial solution depending on an α value. This value will regulate the randomness that we will have in this first solution. Thus, given an incomplete solution ($|S| < p$), we will evaluate the distance of each element to the so-

lution, i.e. we compute the sum of the distances to each point in the partial solution. At this point that we can construct the RCL (*Reduced Candidate List*)

$$\text{RCL} = \{e_i : l \leq d(e_i, S) \leq u\}$$

with

$$l = c^{\max} - \alpha\left(c^{\max} - c^{\min}\right),$$
$$u = c^{\max},$$

where $c^{\max}$ and $c^{\min}$ are respectively the maximum and minimum values of the distances of each candidate element to S previously calculated. The candidate that we will include in the solution will be an element of the RCL randomly chosen. We will repeat this process until we have a feasible solution, that is, that $|S| = p$.

Once the initial solution has been obtained, we proceed to improve it using local search. We evaluate the possible trade-offs by replacing the element that contributes the least to the current solution (the one that is the least distant from the other elements of the solution) by the element that contributes the most. If this results in an improvement in the objective function, we make the change and repeat the process. The search stops when no further improvements can be made, i.e., when replacing an element worsens the objective function.

In this article, we will not discuss the implementation of the GRASP algorithm. For further details, see [1].

1.2 Tabu Search approach

Our approach to the diversity problem is based on what is known as Tabu Search. Like the GRASP algorithm, we build an initial solution that we will try to improve using a local search. The idea that differentiates Tabu Search from local search is the use of short-term memory, which helps avoid cycling with local optima and increase the explored area of the solution space.

To this end, we construct a set that we will call *tabu list*, where we include the elements that have been eliminated from the initial solution. This list will have a fixed length (that we will call *tabu tenure* or τ), so that the elements we include will have a finite permanence. Thus, the elements included in the tabu list will be excluded from the list of candidates to perform the exchanges. Another important aspect is that we will perform these exchanges even if they worsen the objective function. In this way, along the implementation of the tabu list concept, we will eventually arrive at a local optimum different from the one initially obtained with a simple local search.

The key to the tabu search is to carry out this exploration of solutions by always memorizing which solution has achieved the best objective value.

2 Implementation of Tabu Search

In our implementation we have considered two different ways to construct the initial solution: a "greedy" starting point and a completely "random" solution. In the first construction, we will use a greedy randomized criterion controlled by an α parameter, while to obtain a random solution we include candidate elements randomly until a feasible solution is obtained. Note that the latter case is also a greedy algorithm with the parameter $\alpha = 1$. Later we will make a comparative study to verify that we indeed obtain better results with a completely random initial solution.

After obtaining the initial solution, we save it as `best` using the function `.copy()` (to avoid passing it by reference). Then, for the desired duration or the number of iterations we set, we call the `move` function, which is responsible for adding and removing points to modify the solution. We will keep updating `best` with the solution that has the highest objective function value.

In the `move` function, we find two important functions for our method. The first, which we will discuss in detail later, is `selectInterchange`. This function performs a local search and selects the points that will be swapped later using the `removeFromSolution` and `addToSolution` functions. It is important to note that there will always be a change, whether the solution is improved or not.

The other function is `updateTabuList`, which is responsible for updating the Tabu List by adding the points discarded during the `selectInterchange` process. Since the list size is finite, after a certain number of iterations, when new elements are added, the oldest tabu elements will be removed. However, since the size of the list is set by us, it can be adjusted as needed, making the Tabu method more flexible.

2.1 How the solution moves at each iteration

We want to highlight the `selectInterchange` function, where we tested two different methods for selecting point exchanges in the local search phase. It is worth noting that during the local search in Tabu Search, there is always going to be point swapping, whether the objective function improves or not. These methods are `tsbestimp` and `tsfirstimp`. Let's explore the differences.

As we seen in the pseudocode, in `tsbestimp` (alg. 1), until it finds the best solution in the neighborhood, it does not make the exchange. In contrast, in `tsfirstimp` (alg. 2), as soon as it finds a solution that improves the objective function, it exchanges it directly. However, if no improvement is found, both methods will keep the "least bad" solution and exchange it, as mentioned earlier.

Algorithm 1 Choice of elements to be exchanged with `tsbestimp`

```
for u ∈ S do
    for v ∈ N \ (S ∪ TabuList) do
        if bestOfVar < d(v, S \ {u}) − d(u, S \ {u}) then
            bestOfVar = ofVarUnsel - ofVarSel
            sel = u                          ▷ sel ≡ point with minimum distance to sol
            unsel = v                        ▷ unsel ≡ point with maximum distance to sol
            d(sel, S) = ofVarSel
            d(unsel, S) = ofVarUnsel
return sel, d(sel, S), unsel, d(unsel, S)
```

Algorithm 2 Choice of elements to be exchanged with `tsfirstimp`

```
for u ∈ S do
    for v ∈ N \ (S ∪ TabuList) do
        if ofVarUnsel > ofVarSel then
            return sel, d(sel, S), unsel, d(unsel, S)
        if bestOfVar < d(v, S \ {u}) − d(u, S \ {u}) then
            bestOfVar = ofVarUnsel - ofVarSel
            sel = u                          ▷ sel ≡ point with minimum distance to sol
            unsel = v                        ▷ unsel ≡ point with maximum distance to sol
            d(sel, S) = ofVarSel
            d(unsel, S) = ofVarUnsel
return sel, d(sel, S), unsel, d(unsel, S)
```

The Table 1 shows the differences between `tsfirstimp` and `tsbestimp`, in terms of iterations and the objective function value. We observe that with the same computational time, there is practically no difference in the value of the objective function between the `tsfirstimp` method and the `tsbestimp` method, so we can not draw any conclusion about this aspect. However, there is a significant difference between the number of iterations of the two methods. Since

`tsfirstimp` performs a larger number of iterations, we interpret that this algorithm explores a greater number of neighborhoods. Indeed, the `tsfirstimp` implementation adds an extra point of randomness to the way we perform the movements, and it is an efficient way to avoid cycles in Tabu Search. As a consequence, by increasing the execution time, we get better results with `tsfirstimp`. All this can be seen in more detail in the Figure 1. Therefore, we will choose it for comparing Tabu Search with the GRASP method.

	5 seconds	10 seconds	20 seconds	40 seconds
TS best improve	7753.65 (57 its)	7603.84 (119 its)	8228.71 (200 its)	7938.96 (394 its)
TS first improve	7795.03 (314 its)	7603.64 (400 its)	8212.34 (545 its)	7982.79 (731 its)

Table 1: Differences between `tsfirstimp` and `tsbestimp` in terms of iterations and objective function value of the best solution found with $\tau = 30$.

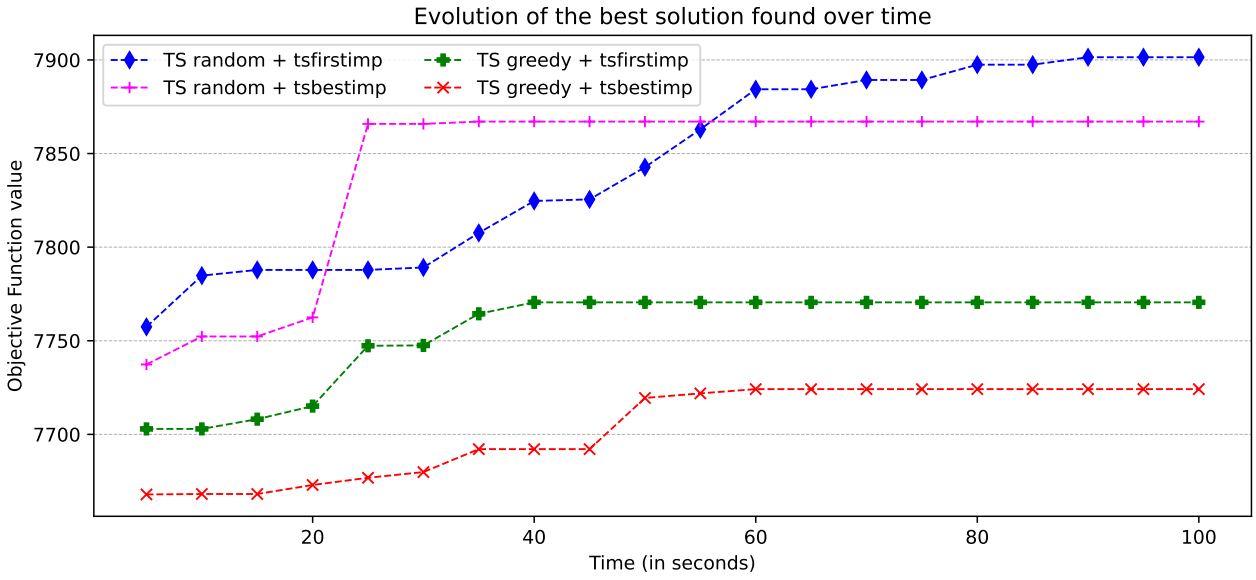

Figure 1: Comparison of the evolution of the algorithm depending on the initial solution and the `selectInterchange` function. Tabu Search with $\tau = 30$ applied to instance 13 ($n = 500$, $p = 50$).

2.2 Impact of the Initial Solution on Tabu Search performance

Another detail that stands out when we examine the Figure 1 is the significant impact that the choice of the initial solution has on the evolution of the method. As we have commented at the beginning of the section, we have taken care to provide two methods for the starting point. That is why in our implementation we have the option of constructing a "greedy" initial solution by reusing the GRASP constructive method. Alternatively, we have developed a method to construct a completely random solution:

Require: inst ▷ *Problem instance*
Ensure: sol ▷ *Feasible solution*
  sol = createEmptySolution(inst)
  n = inst['n']
  while not isFeasible(sol) **do**
    u = randint(0, $n - 1$)
    addToSolution(sol, u)
  return sol

By comparing the algorithm's performance with different initial solutions, we will see that using a random starting point significantly improves results. This is illustrated in Table 2 and Figure 2. As shown in Figures 1 and 2, we obtain better results when we use a completely random solution. We can interpret this result as an improvement in the exploration of the solution space. Therefore, we avoid entering a cycle in the early stages of the method.

We also observe in the Figure 2 that generally we obtain better results with a random solution, but the method turns out to be less consistent because it depends largely on the initial solution and we have no control over it since it is completely random.

	Instance 2	Instance 5	Instance 6	Instance 9
TS greedy 10s	7649.34	7755.23	7655.11	7674.66
TS random 10s	8687.63	8038.67	8156.89	7902.8
TS greedy 40s	7723.51	7641.99	7687.5	7709.51
TS random 40s	7985.36	8292.98	8330.56	8420.66

Table 2: Comparison of an initial greedy solution and a completely random solution. The Tabu Search is applied to different methods with $\tau = 30$.

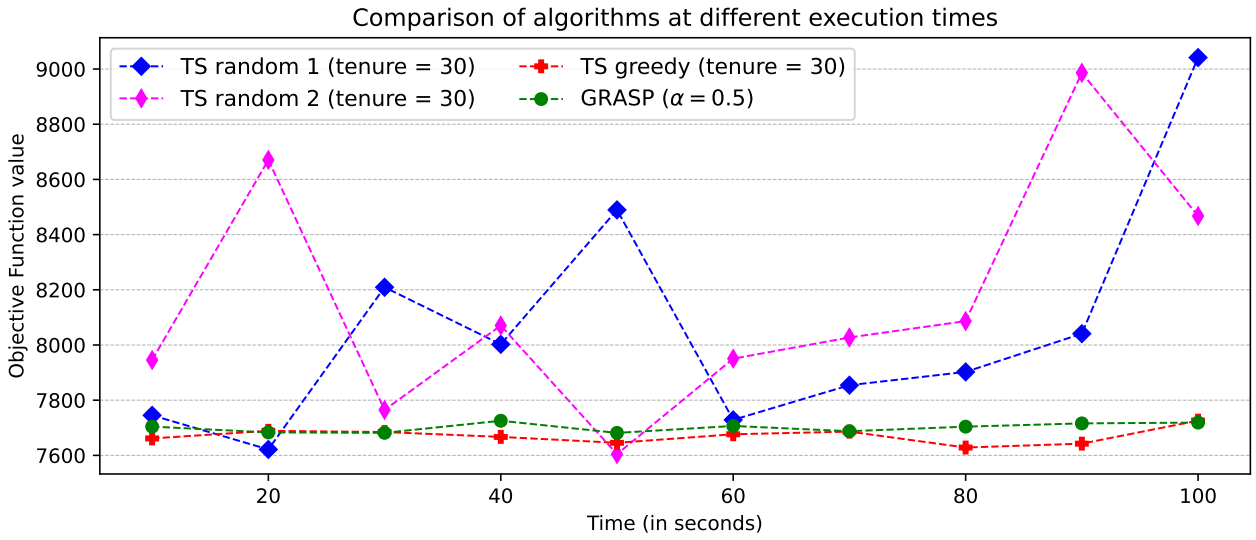

Figure 2: Best solution found by each method with different execution times. The problem to be solved is the instance 2 ($n = 500$, $p = 50$).

2.2.1 How to improve our constructive method

Another strategy that we can apply to develop our constructive method is to provide our algorithm with **long-term memory**. Although in our tabu search proposal we stick only to short term memory implementation, we find it interesting to discuss possible ways to improve the algorithm. We then consider how to implement long term memory techniques and how this would impact the performance of our code.

The first idea that comes to mind is the most natural one: to memorize the points that have led us to higher-quality solutions. We could then establish a threshold value for the objective function, above which we consider a solution to be good. Therefore, we could in the Tabu

Search constructive phase repeat several times a random choice followed by a local search in order to gather information to build an initial solution in a "smarter" way.

Another strategy is to also memorize points that have never been visited during the initial exploration of the solution space and also include them as part of the initial solution.

This approach is interesting because it would take advantage of the good results obtained with a random initial solution. It would also add consistency in the results. For more details, see [2].

2.3 Step-by-step example of the Tabu Search algorithm

Once we have seen how the Tabu Search works, let's see how it performs the movements and how the tabu list is updated. Let us now see, step by step, how the elements we remove from the solution enter the tabu list and are thus excluded as candidates for the next iteration.

In Figure 3 we show an initial solution with an objective value of 340.65. We observe that the point 40 is added to the solution while 25 is removed and added to the tabu list. The new solution has an objective value of 342.39 (it has improved which we know does not necessarily happen). Again, we replace 26 by 92 and add 26 to the tabu list.

This is how Tabu Search works, and we can see this process in more detail in Table 3. In this case, at each iteration we find an interchange of elements that improves the solution and we move to this new solution. However, this may not always happen. As explained above, in that case we choose the interchange that is least detrimental to the objective function.

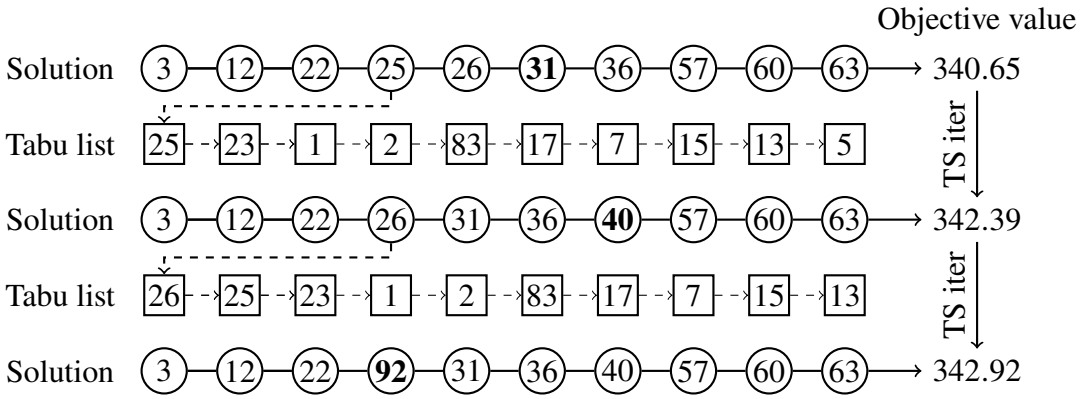

Figure 3: Graphical representation of the Tabu Search algorithm for the instance 1 ($n = 100$, $p = 10$). We are showing the iterations 17 and 18 with $\tau = 10$.

Iteration	Solution	Add	Drop	OF
3	{ 3, 72, 8, 12, 15, 17, 83, 57, 60, 63 }	12	1	294.76
4	{ 72, 8, 12, 13, 15, 17, 83, 57, 60, 63 }	13	3	299.34
5	{ 6, 8, 12, 13, 15, 17, 83, 57, 60, 63 }	6	72	303.25
6	{ 8, 12, 13, 15, 17, 83, 23, 57, 60, 63 }	23	6	304.18
7	{ 5, 12, 13, 15, 17, 83, 23, 57, 60, 63 }	5	8	306.88
8	{ 12, 13, 15, 17, 83, 22, 23, 57, 60, 63 }	22	5	308.24
9	{ 12, 15, 17, 83, 22, 23, 25, 57, 60, 63 }	25	13	312.13
10	{ 7, 12, 17, 83, 22, 23, 25, 57, 60, 63 }	7	15	316.64

Table 3: First iterations of Tabu Search algorithm for the instance 1 ($n = 100$, $p = 10$).

3 Experimental selection of parameters for GRASP and Tabu Search

In order to compare the performance of Tabu Search and GRASP in the next section, we will now try to determine the parameters we will use for these algorithms. Note that one strategy we can apply would be to adjust these parameters dynamically during execution (see [3]). In our experiments we will fix these values and compare both algorithms in both short and long executions. The first parameter to determine is the execution time for the algorithms. It is essential that for the same instance, we compare the results obtained with both algorithms under equal conditions. This is why we have to fix what should be the time available to find a good solution.

We observe in the following graphs (Fig. 4 and 5) how the algorithm evolves as a function of time. We see that in the first seconds, the initial solution provided by GRASP is generally better. This is quite intuitive because it is natural that a greedy construction gives a better result than a completely random one. However, quickly the Tabu Search solution outperforms GRASP before 10 seconds. As time progresses, each algorithm finds better solutions, but in both methods the maximum objective value stabilizes after the 40 second threshold has been passed.

We then decided to perform the experiments with 10 and 40 seconds.

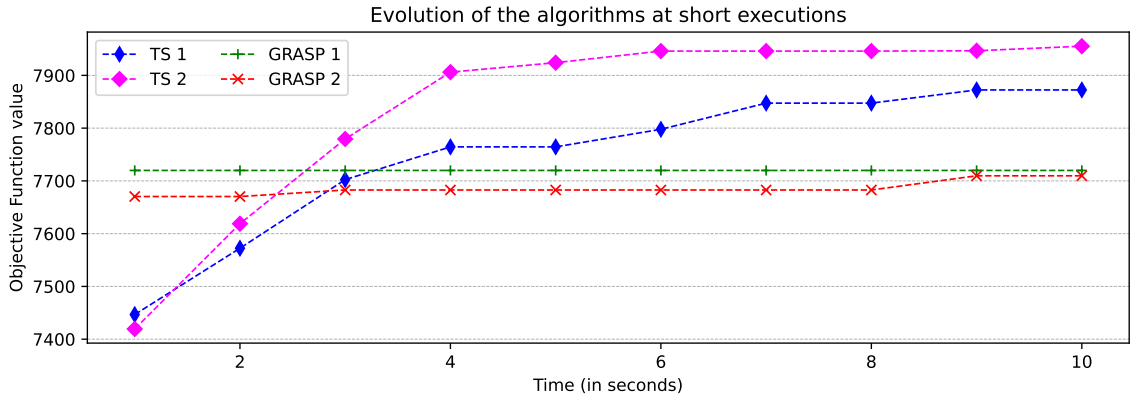

Figure 4: Evolution of the best solution found in short executions. TS with $\tau = 30$ and GRASP with $\alpha = 0.5$. Instance 13, $n = 500$, $p = 50$. Two executions of each method with different random seeds.

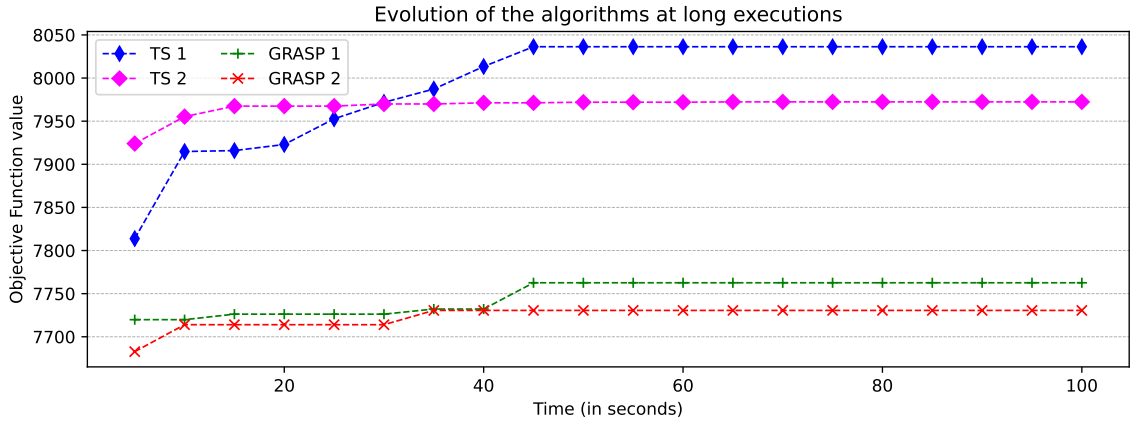

Figure 5: Evolution of the best solution found in long executions. TS with $\tau = 30$ and GRASP with $\alpha = 0.5$. Instance 13, $n = 500$, $p = 50$. Two executions of each method with different random seeds.

3.1 Determination of alpha for GRASP

Once the execution time is set, we decide the value of α. To do this, we run for 10 and 40 seconds GRASP for each value of α. We calculate for each instance the relative deviation of each solution with respect to the best solution found. Let us introduce some notation to clarify how we calculated these relative deviations. We denote

$$\bar{\alpha}_i = \underset{\alpha \in \{0.2,\ldots,0.9\}}{\arg\max} \left\{ \text{of}(S^i_\alpha) \right\}$$

for the instance i and S^i_α best solution for this intance found by GRASP with α. Thus, the relative distance of each solution to the best one is

$$\frac{\text{of}(S^i_{\bar{\alpha}_i}) - \text{of}(S^i_\alpha)}{\text{of}(S^i_{\bar{\alpha}_i})}$$

for the instance i. The result of this calculation is shown as an example in the lower part of the Table 4. In order to decide which is the best value of α, we sum the relative deviations we have obtained with that α for each instance,

$$\sum_{i \in \text{instances}} \frac{\text{of}(S^i_{\bar{\alpha}_i}) - \text{of}(S^i_\alpha)}{\text{of}(S^i_{\bar{\alpha}_i})}$$

and we take the α with a low sum of relative deviations.

In Tables 4 and 5 we see that these are $\alpha = 0.2$ and $\alpha = 0.5$ although there is not really a significant difference with respect to other values.

We proceed now to choose the value of tabu tenure for the Tabu Search algorithm.

10 seconds	**α = 0.2**	α = 0.3	α = 0.4	**α = 0.5**	α = 0.6	α = 0.7	α = 0.8	α = 0.9
MDG-a_1_100_m10	360.15	360.15	360.15	360.15	360.15	360.15	360.15	360.15
MDG-a_4_100_m10	355.72	355.72	355.72	355.72	355.72	355.72	355.72	355.72
MDG-a_10_100_m10	355.50	355.50	355.50	355.50	355.50	355.50	355.50	355.50
MDG-a_12_100_m10	354.25	354.25	354.25	354.25	354.25	354.25	354.25	354.25
MDG-a_14_100_m10	356.06	356.06	356.06	356.06	356.06	356.06	356.06	356.06
MDG-a_20_100_m10	349.31	349.31	349.31	349.31	349.31	349.31	349.31	349.31
MDG-a_2_n500_m50	7691.71	7692.78	7691.97	7755.63	7704.17	7697.08	7720.22	7692.31
MDG-a_5_n500_m50	7735.97	7675.41	7721.21	7737.42	7673.30	7699.83	7678.07	7658.52
MDG-a_6_n500_m50	7770.48	7745.09	7700.98	7708.04	7712.80	7724.76	7687.29	7715.80
MDG-a_9_n500_m50	7737.24	7730.27	7711.98	7722.94	7710.40	7712.44	7692.07	7736.72
MDG-a_13_n500_m50	7760.59	7716.15	7769.99	7793.55	7710.22	7712.85	7736.56	7747.53
MDG-a_16_n500_m50	7751.50	7743.52	7720.37	7732.86	7721.77	7754.67	7708.31	7697.36
MDG-a_17_n500_m50	7692.81	7681.85	7724.98	7688.07	7688.61	7730.70	7729.88	7762.30
MDG-a_19_n500_m50	7735.25	7731.00	7708.98	7681.40	7676.95	7693.27	7660.84	7697.00
MDG-a_20_n500_m50	7664.67	7694.55	7701.91	7696.15	7695.86	7701.41	7670.67	7657.70
#Best solution:	**9**	6	7	**9**	6	7	6	7
MDG-a_2_n500_m50	0.82%	0.81%	0.82%	0.00%	0.66%	0.75%	0.46%	0.82%
MDG-a_5_n500_m50	0.02%	0.80%	0.21%	0.00%	0.83%	0.49%	0.77%	1.02%
MDG-a_6_n500_m50	0.00%	0.33%	0.89%	0.80%	0.74%	0.59%	1.07%	0.70%
MDG-a_9_n500_m50	0.00%	0.09%	0.33%	0.18%	0.35%	0.32%	0.58%	0.01%
MDG-a_13_n500_m50	0.42%	0.99%	0.30%	0.00%	1.07%	1.04%	0.73%	0.59%
MDG-a_16_n500_m50	0.04%	0.14%	0.44%	0.28%	0.42%	0.00%	0.60%	0.74%
MDG-a_17_n500_m50	0.90%	1.04%	0.48%	0.96%	0.95%	0.41%	0.42%	0.00%
MDG-a_19_n500_m50	0.00%	0.05%	0.34%	0.70%	0.75%	0.54%	0.96%	0.49%
MDG-a_20_n500_m50	0.48%	0.10%	0.00%	0.07%	0.08%	0.01%	0.41%	0.57%
Sum of deviations:	**2.69%**	4.35%	3.82%	**3.00%**	5.86%	4.14%	5.99%	4.94%

Table 4: GRASP execution during 10 seconds. At the top of the Table, the objective values of the best solution found for each value of α. In the lower part, the relative deviations of these values with respect to the best solution found for the instance.

40 seconds	α **= 0.2**	$\alpha = 0.3$	$\alpha = 0.4$	α **= 0.5**	$\alpha = 0.6$	$\alpha = 0.7$	$\alpha = 0.8$	$\alpha = 0.9$
MDG-a_1_100_m10	360.15	360.15	360.15	360.15	360.15	360.15	360.15	360.15
MDG-a_4_100_m10	355.72	355.72	355.72	355.72	355.72	355.72	355.72	355.72
MDG-a_10_100_m10	355.50	355.50	355.50	355.50	355.50	355.50	355.50	355.50
MDG-a_12_100_m10	354.25	354.25	354.25	354.25	354.25	354.25	354.25	354.25
MDG-a_14_100_m10	356.06	356.06	356.06	356.06	356.06	356.06	356.06	356.06
MDG-a_20_100_m10	349.31	349.31	349.31	349.31	349.31	349.31	349.31	349.31
MDG-a_2_n500_m50	7721.14	7700.98	7712.27	7750.95	7724.70	7716.75	7721.72	7724.31
MDG-a_5_n500_m50	7696.98	7691.75	7693.06	7702.05	7691.54	7696.90	7720.38	7677.36
MDG-a_6_n500_m50	7735.04	7716.47	7739.39	7747.15	7734.30	7755.27	7722.56	7736.97
MDG-a_9_n500_m50	7736.06	7738.53	7720.28	7728.47	7736.18	7733.95	7740.14	7718.60
MDG-a_13_n500_m50	7778.76	7764.20	7749.29	7764.80	7760.54	7793.55	7767.56	7751.58
MDG-a_16_n500_m50	7787.41	7762.81	7724.45	7789.24	7753.80	7726.34	7727.92	7745.25
MDG-a_17_n500_m50	7768.88	7738.74	7768.69	7734.35	7753.57	7732.62	7724.23	7721.47
MDG-a_19_n500_m50	7701.99	7720.68	7736.16	7700.30	7688.93	7683.88	7719.69	7693.26
MDG-a_20_n500_m50	7697.12	7691.29	7705.85	7710.79	7691.78	7712.97	7699.39	7713.56
#Best solution:	7	6	7	**8**	6	**8**	**8**	7
Sum of deviations:	**1.87%**	3.13%	2.82%	**1.81%**	3.00%	2.78%	2.89%	3.68%

Table 5: GRASP execution during 40 seconds. Objective value of the best solution found.

3.2 Determination of tabu tenure for Tabu Search

We then proceeded to choose the τ value. We now performed the same calculation to determine the best tabu tenure. Trying with the lengths 10, 20, 30, 40, 50 and 60, we obtain the Tables 6 and 7. In this case, it is clearer that the values we should take for the length of the tabu list are 20 and 30. Moreover, these values are also the ones that have found the best solution in more cases.

10 seconds	$\tau = 10$	τ **= 20**	τ **= 30**	$\tau = 40$	$\tau = 50$	$\tau = 60$
MDG-a_1_100_m10	385.09	385.09	385.09	382.24	382.24	384.75
MDG-a_4_100_m10	376.46	375.41	375.41	372.05	366.17	367.62
MDG-a_10_100_m10	388.32	386.92	390.63	390.63	387.03	390.63
MDG-a_12_100_m10	379.10	380.79	376.83	375.07	380.79	376.75
MDG-a_14_100_m10	352.73	370.06	367.13	370.06	370.06	370.06
MDG-a_20_100_m10	384.78	386.78	387.79	386.35	387.79	387.79
MDG-a_2_n500_m50	7,762.08	7,796.99	7,821.61	7,763.23	7,746.78	7,807.61
MDG-a_5_n500_m50	7,801.31	7,830.93	7,789.86	7,756.07	7,678.65	7,738.88
MDG-a_6_n500_m50	7,862.06	7,863.25	7,870.61	7,637.61	7,762.73	7,748.03
MDG-a_9_n500_m50	7,867.50	7,857.23	7,873.19	7,868.16	7,807.08	7,746.76
MDG-a_13_n500_m50	7,849.08	7,887.08	7,784.80	7,832.09	7,804.58	7,786.53
MDG-a_16_n500_m50	7,771.18	7,827.10	7,790.20	7,732.13	7,845.92	7,748.35
MDG-a_17_n500_m50	7,860.01	7,826.25	7,760.03	7,753.87	7,773.79	7,716.06
MDG-a_19_n500_m50	7,870.44	7,817.68	7,838.96	7,801.51	7,827.18	7,766.48
MDG-a_20_n500_m50	7,821.52	7,759.98	7,745.17	7,747.94	7,768.82	7,750.65
#Best solution:	5	5	**6**	2	4	3
Sum of deviations:	9.25%	**4.23%**	**7.29%**	13.83%	12.87%	14.59%

Table 6: Tabu Search execution during 10 seconds. Objective value of the best solution found.

40 seconds	$\tau = 10$	$\boldsymbol{\tau = 20}$	$\boldsymbol{\tau = 30}$	$\tau = 40$	$\tau = 50$	$\tau = 60$
MDG-a_1_100_m10	385.09	385.09	385.09	382.24	382.24	384.75
MDG-a_4_100_m10	376.46	375.41	375.41	372.05	366.17	367.62
MDG-a_10_100_m10	388.32	386.92	390.63	390.63	387.03	390.63
MDG-a_12_100_m10	379.10	380.79	376.83	375.07	380.79	376.75
MDG-a_14_100_m10	352.73	370.06	367.13	370.06	370.06	370.06
MDG-a_20_100_m10	384.78	386.78	387.79	386.35	387.79	387.79
MDG-a_2_n500_m50	7,762.76	7,869.97	7,821.61	7,775.98	7,797.94	7,864.39
MDG-a_5_n500_m50	7,812.82	7,840.83	7,808.52	7,805.36	7,791.32	7,785.17
MDG-a_6_n500_m50	7,904.69	7,892.62	7,900.24	7,637.61	7,840.58	7,877.10
MDG-a_9_n500_m50	7,870.72	7,870.03	7,878.55	7,893.44	7,851.87	7,834.12
MDG-a_13_n500_m50	7,851.03	7,902.13	7,824.68	7,836.52	7,847.26	7,805.96
MDG-a_16_n500_m50	7,771.18	7,875.13	7,914.88	7,803.20	7,884.75	7,833.01
MDG-a_17_n500_m50	7,862.92	7,851.35	7,794.91	7,868.89	7,785.88	7,775.27
MDG-a_19_n500_m50	7,870.44	7,860.99	7,851.03	7,832.99	7,850.20	7,817.07
MDG-a_20_n500_m50	7,851.53	7,797.63	7,770.89	7,797.06	7,793.49	7,794.45
#Best solution:	5	**6**	4	4	3	3
Sum of deviations:	11.04%	**3.47%**	**6.58%**	12.22%	10.41%	10.22%

Table 7: Tabu Search execution during 40 seconds. Objective value of the best solution found.

3.3 Iterations by time

In order to add reproducibility to the comparison, we calculed the approximate the number of iterations performed by each algorithm in 10 and 40 seconds. All the tests we have performed throughout the document have been executed on a 10th generation Intel Core i7 at 1.80GHz with 16GB of ram memory. The interpreter used was Python 3.11.2 on Linux. The number of iterations obtained is shown in the Table 8. Note that this number will vary depending on the size of the problem (in cases where $n = 100$ we will get more iterations than where $n = 500$).

	GRASP $\alpha = 0.2$	GRASP $\alpha = 0.5$	TS $\tau = 20$	TS $\tau = 30$
10 sec, $n = 100$, $p = 10$	11000	8100	13000	14000
10 sec, $n = 500$, $p = 50$	200	100	380	400
40 sec, $n = 100$, $p = 10$	44000	30000	51000	55000
40 sec, $n = 500$, $p = 50$	790	390	800	820

Table 8: Iterations performed by each method dependig on the execution time.

4 Comparison between GRASP and Tabu Search performance

In this section we analyze the performance of these algorithms when solving the Maximum Diversity Problem. In our experiment, we compared the GRASP method with our Tabu Search proposal as analyzed above for the Maximum Diversity Problem described in 15 instances (6 with $n = 100$, $p = 10$ and 9 with $n = 500$, $p = 50$). The following tables show, for each procedure, the pedecentage deviation of the best solution found by each method (GRASP with $\alpha = 0.2,\ 0.5$, and TS with $\tau = 20,\ 30$). We also show the number of times that each algorithm reached the best solution found. We repeat this comparision for 10 and 40 seconds.

10 seconds	GRASP $\alpha = 0.2$	GRASP $\alpha = 0.5$	TS $\tau = 20$	TS $\tau = 30$
MDG-a_1_100_m10	360.15	360.15	359.68	360.15
MDG-a_4_100_m10	355.72	355.72	355.72	353.51
MDG-a_10_100_m10	355.50	355.50	395.00	351.90
MDG-a_12_100_m10	354.25	354.25	387.69	354.25
MDG-a_14_100_m10	356.06	356.06	356.06	414.43
MDG-a_20_100_m10	349.31	349.31	349.31	420.55
MDG-a_2_n500_m50	7,700.15	7,697.68	7,855.41	8,047.27
MDG-a_5_n500_m50	7,698.38	7,669.16	7,852.59	8,105.76
MDG-a_6_n500_m50	7,713.78	7,737.22	7,797.18	8,104.03
MDG-a_9_n500_m50	7,742.41	7,717.05	8,182.66	8,567.12
MDG-a_13_n500_m50	7,777.00	7,729.34	8,107.00	8,475.33
MDG-a_16_n500_m50	7,734.48	7,737.62	7,890.79	7,984.93
MDG-a_17_n500_m50	7,717.23	7,741.19	8,308.75	7,851.96
MDG-a_19_n500_m50	7,738.91	7,695.71	7,636.30	8,036.40
MDG-a_20_n500_m50	7,681.32	7,655.34	8,022.67	7,928.85
#Best solution:	2	2	5	10
Sum of deviations:	99.88%	101.38%	55.44%	26.83%

10 seconds	GRASP $\alpha = 0.2$	GRASP $\alpha = 0.5$	TS $\tau = 20$	TS $\tau = 30$
MDG-a_1_100_m10	0.00%	0.00%	0.13%	0.00%
MDG-a_4_100_m10	0.00%	0.00%	0.00%	0.62%
MDG-a_10_100_m10	10.00%	10.00%	0.00%	10.91%
MDG-a_12_100_m10	8.63%	8.63%	0.00%	8.63%
MDG-a_14_100_m10	14.08%	14.08%	14.08%	0.00%
MDG-a_20_100_m10	16.94%	16.94%	16.94%	0.00%
#Best solution:	2	2	3	3
Sum of deviations:	49.65%	49.65%	31.15%	20.16%

10 seconds	GRASP $\alpha = 0.2$	GRASP $\alpha = 0.5$	TS $\tau = 20$	TS $\tau = 30$
MDG-a_2_n500_m50	4.31%	4.34%	2.38%	0.00%
MDG-a_5_n500_m50	5.03%	5.39%	3.12%	0.00%
MDG-a_6_n500_m50	4.82%	4.53%	3.79%	0.00%
MDG-a_9_n500_m50	9.63%	9.92%	4.49%	0.00%
MDG-a_13_n500_m50	8.24%	8.80%	4.35%	0.00%
MDG-a_16_n500_m50	3.14%	3.10%	1.18%	0.00%
MDG-a_17_n500_m50	7.12%	6.83%	0.00%	5.50%
MDG-a_19_n500_m50	3.70%	4.24%	4.98%	0.00%
MDG-a_20_n500_m50	4.25%	4.58%	0.00%	1.17%
#Best solution:	0	0	2	7
Sum of deviations:	50.23%	51.73%	24.28%	6.67%

40 seconds	GRASP $\alpha = 0.2$	GRASP $\alpha = 0.5$	TS $\tau = 20$	TS $\tau = 30$
MDG-a_1_100_m10	360.15	360.15	360.15	355.72
MDG-a_4_100_m10	355.72	355.72	368.06	355.72
MDG-a_10_100_m10	355.50	355.50	409.21	349.25
MDG-a_12_100_m10	354.25	354.25	354.25	354.25
MDG-a_14_100_m10	356.06	356.06	367.20	356.06
MDG-a_20_100_m10	349.31	349.31	349.31	349.31
MDG-a_2_n500_m50	7,726.98	7,732.88	7,936.87	8,405.02
MDG-a_5_n500_m50	7,722.72	7,728.27	7,658.67	8,355.06
MDG-a_6_n500_m50	7,729.45	7,735.74	7,914.20	8,625.46
MDG-a_9_n500_m50	7,741.40	7,728.71	8,523.49	7,836.75
MDG-a_13_n500_m50	7,766.34	7,757.65	8,324.08	7,769.07
MDG-a_16_n500_m50	7,729.12	7,733.66	8,176.28	8,003.91
MDG-a_17_n500_m50	7,776.74	7,785.72	8,668.49	8,338.35
MDG-a_19_n500_m50	7,740.29	7,714.26	8,073.67	7,791.54
MDG-a_20_n500_m50	7,705.08	7,703.93	8,087.12	7,811.72
#Best solution:	3	3	12	5
Sum of deviations:	86.02%	86.24%	22.15%	49.81%

40 seconds	GRASP $\alpha = 0.2$	GRASP $\alpha = 0.5$	TS $\tau = 20$	TS $\tau = 30$
MDG-a_1_100_m10	0.00%	0.00%	0.00%	1.23%
MDG-a_4_100_m10	3.35%	3.35%	0.00%	3.35%
MDG-a_10_100_m10	13.13%	13.13%	0.00%	14.65%
MDG-a_12_100_m10	0.00%	0.00%	0.00%	0.00%
MDG-a_14_100_m10	3.03%	3.03%	0.00%	3.03%
MDG-a_20_100_m10	0.00%	0.00%	0.00%	0.00%
#Best solution:	3	3	6	2
Sum of deviations:	19.51%	19.51%	0.00%	22.27%

40 seconds	GRASP $\alpha = 0.2$	GRASP $\alpha = 0.5$	TS $\tau = 20$	TS $\tau = 30$
MDG-a_2_n500_m50	8.07%	8.00%	5.57%	0.00%
MDG-a_5_n500_m50	7.57%	7.50%	8.33%	0.00%
MDG-a_6_n500_m50	10.39%	10.32%	8.25%	0.00%
MDG-a_9_n500_m50	9.18%	9.32%	0.00%	8.06%
MDG-a_13_n500_m50	6.70%	6.80%	0.00%	6.67%
MDG-a_16_n500_m50	5.47%	5.41%	0.00%	2.11%
MDG-a_17_n500_m50	10.29%	10.18%	0.00%	3.81%
MDG-a_19_n500_m50	4.13%	4.45%	0.00%	3.49%
MDG-a_20_n500_m50	4.72%	4.74%	0.00%	3.41%
#Best solution:	0	0	6	3
Sum of deviations:	66.51%	66.73%	22.15%	27.54%

4.1 Analysis and interpretation of results

Let us now look at the results we have presented in the previous tables. The tables are separated by execution time (10 seconds and 40 seconds). In addition, to analyze the deviations from the best solution found we have separated the results also in two tables: one for small instances ($n = 100$ and $p = 10$) and one for large instances ($n = 500$ and $p = 50$). Below each table we find both the sum of these deviations obtained in each algorithm as well as the number of times we have obtained with this method the best known solution.

Taking a closer look we observe that, in short executions (10 seconds), the objective values of the solutions obtained by Tabu Search are similar to those obtained by GRASP. However, we already see how on large instances Tabu Search performs better in all instances and even on short runs it manages to exceed the objective function value for GRASP. Thus, we see how the best results for large instances are achieved by Tabu Search with $\tau = 30$, followed by Tabu Search with $\tau = 20$, both surpassing the two solutions obtained by GRASP. We also want to highlight that, although the difference is not so remarkable, in small instances Tabu Search has also performed better than GRASP.

Let's see the performance of these methods with longer execution time. In the table where we show the results we have in 40 seconds, we observe that the difference between the objective values this time is larger than in the 10 seconds tables. We see that if before the method that was getting the best results was TS with $\tau = 30$, now it seems that the best results are obtained by the same method but with $\tau = 20$. We note that it has obtained the best solution in 12 of the 15 instances, which is a really high percentage. Looking at the deviations tables, we see again that the difference is smaller on smaller instances, while on larger instances Tabu Search's performance far outperforms GRASP.

4.2 Testing with different initial solutions

As we have seen in the section where we discussed what the constructive phase should be, it had a great impact on the solution we arrived at. Thus, creating a completely random solution made the method inconsistent (see Figure 2). The simplest way we have to mitigate this problem (without going into long-term memory techniques) is to run several short tabu searches and take the best solution.

That is why for each instance we have run the Tabu Search algorithm 4 times (with $\tau = 30$) for 10 seconds. In the Table 9 we compare the improvement with respect to the result obteined by a single Tabu Search ($\tau = 30$) in 40 seconds.

Instance	OF	Change	Instance	OF	Change
MDG-a_1_100_m10	385.09	8.26%	MDG-a_6_n500_m50	8,435.95	−2.20%
MDG-a_4_100_m10	429.63	20.78%	MDG-a_9_n500_m50	8,255.39	5.34%
MDG-a_10_100_m10	371.57	6.39%	MDG-a_13_n500_m50	8,353.63	7.52%
MDG-a_12_100_m10	354.25	0.00%	MDG-a_16_n500_m50	8,278.21	3.43%
MDG-a_14_100_m10	358.85	0.78%	MDG-a_17_n500_m50	8,300.09	−0.46%
MDG-a_20_100_m10	388.76	11.29%	MDG-a_19_n500_m50	8,467.27	8.67%
MDG-a_2_n500_m50	8,400.51	−0.05%	MDG-a_20_n500_m50	8,327.06	6.60%
MDG-a_5_n500_m50	8,955.02	7.18%			

Table 9: Best solution of four Tabu Search ejections with $\tau = 30$ during 10 seconds each one.

We observe how we have generally improved the value of the objective function, and in the other cases it has worsened very little. We can therefore state that we have significantly improved the previous result.

In Figure 6 we run the Tabu Search four times and observe that the result stabilizes quickly. We also see that there is a significant

difference between the four solutions. So it is a good strategy to perform short runs of the algorithm many times and take the best solution. However, as we mentioned earlier, there are smarter ways to add consistency to the method. We could even apply Path Relinking to obtain new initial solutions and then apply the Tabu Search (see [4]).

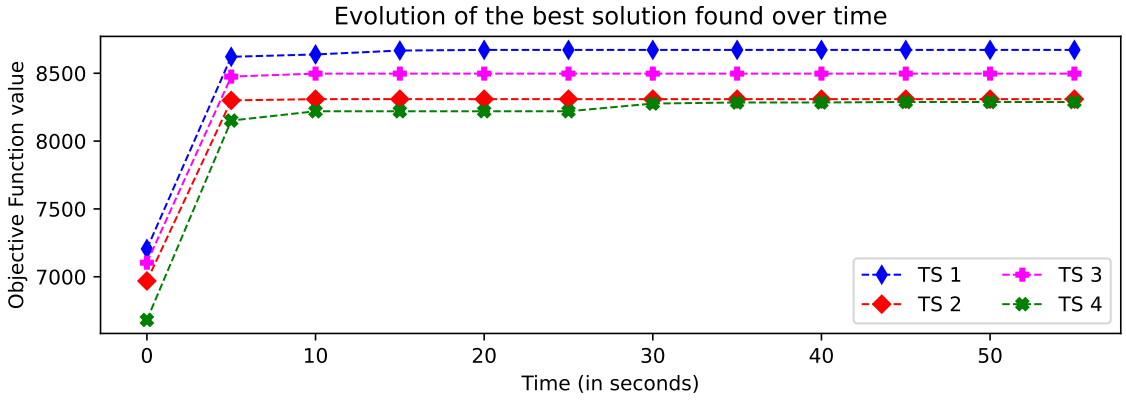

Figure 6: Instance 13 ($n = 500$, $p = 50$). Four different executions of Tabu Search with $\tau = 20$.

5 Conclusions

With the results observed in the previous chapters, we have analyzed different variations of Tabu Search and GRASP. Tabu Search performs better starting from a random solution, moving with `tsfirstimp` and with a tabu tenure between 20 and 30. For the GRASP method, the best solutions were found with $\alpha = 0.2$ and $\alpha = 0.5$. We observed that within a very short execution time, both methods generate solutions that are more or less similar, while when the execution time exceeds approximately 10 seconds, the difference between Tabu Search and GRASP becomes highly noticeable, with Tabu Search clearly outperforming GRASP. We also observe differences between the methods depending on the instances size: for larger problems, Tabu Search performs better than GRASP. Also, during the article we have suggested what may be modifications to the algorithm that would allow us to go further.

Regarding the application of this method, we observe that implementing Tabu Search is useful for solving the Maximum Diversity Problem when an instantaneous result is not required. These are actually the majority of MDP cases. For example, when we want to maximize diversity in a policy project, obtain diverse itineraries in tourism offerings, or even manage protected areas to capture maximum ecosystem diversity, we can approach it as a MDP instance. In all these cases, we do not need the solution to be immediate. Therefore, we prioritize an algorithm capable of obtaining the best results in a reasonable time.

References

[1] T.A. Feo and M.G.C. Resende. “Greedy Randomized Adaptive Search Procedures”. In: *Journal of Global Optimization* 6 (1995), pp. 109–133.

[2] F. Glover and M. Laguna. “Tabu Search”. In: *Handbook of Combinatorial Optimization* 3 (1998), pp. 621–757.

[3] A. Duarte and R. Martí. “Tabu search and GRASP for the maximum diversity problem”. In: *European Journal of Operational Research* 178 (2007), pp. 71–84.

[4] F. Glover, M. Laguna, and R. Martí. “Fundamentals of Scatter Search and Path Relinking”. In: *Control and Cybernetics* 29.3 (2000), pp. 653–684.