Tabu Search - Exercise

Claudio Metelli

December 8, 2023

1 Introduction to the problem

With this exercise we want to implement a simple Tabu Search example. The problem is a sequencing problem, with N jobs, each one with a processing time p_j, a due date d_j and a penalty for tardiness w_j, with $j = 1, \ldots, N$. In this example we have $N = 6$ and our goal is to minimize the penalty for the sequence of jobs, with the following function:

$$\min \quad \sum_{j=1}^{6} w_j \left[C_j - d_j\right]^+ = T \tag{1}$$

Where $[v]^+ = max(0, v)$ with $v \in \mathbb{R}$, and C_j is the sum of the processing time p_j of job j, and the processing times of all the jobs that have been scheduled before j. In the following table we have the values of w_j, d_j, p_j for each job.

job	w_j	d_j	p_j
1	1	6	9
2	1	4	12
3	1	8	15
4	1	2	8
5	1	10	20
6	1	3	22

The problem requests are:

1. Find a way to determine an initial feasible solution;
2. Define the move that determines the neighborhood;
3. Determine which attribute becomes tabu;
4. Define the tabu tenure;
5. Complete the first 10 iterations of the method;

2 Problem approach

Often, one of the first things to do is to create a mathematical model; in this case, we can do this, but we can also skip this part.

In fact, the problem asks to apply a Tabu Search, which is a method that does not need a model: it just needs a starting feasible solution, a way to define the neighborhood, and a way to define what is "tabu". Also, neighborhood is explored sistematically in the most cases, but if we have big neighborhoods, we can choose a way to explore it. Another think we can notice is that our instance of the problem is very small, so we won't have problems about memory or computing time. Now, we will try to go step by step and solve the problem: the first step is to get an initial feasible solution.

2.1 Initial solution

Getting an initial feasible solution means that we need a simple solution which does not violate any constraint of the problem. At this point, we don't have any constraint, because we don't have a mathematical model; but still, it is not needed. Why?

Well, our goal is to find the sequence of jobs which minimize the total penalty for tardiness; example of this sequence can be: $(1, 4, 2, 3, 6, 5)$ or $(6, 2, 1, 4, 5, 3)$, and each one of these sequences has a total value T where T is the result of the objective function (1). So, which sequences are we allowed to take and which are not?

The answer is simple: we can take every sequence we want. In fact, there is no restriction on which job need to be taken before another one or others: we can just take a simple sequence and that will be a feasible solution.

Of course there still lot of choices: we can choose a sequence with a greedy method and take always the shortest job among the remaining, or we can choose a simple random sequence, or, again, we can simply take a sequence we want, like $(6, 5, 4, 3, 2, 1)$. Of course, there are methods better than others, for example greedy method can be a good choice. In our case we have a so-small instance that we are free to choice whatever we want: our choice will be the not-so-fancy sequence $(1, 2, 3, 4, 5, 6)$, that has a global tardiness $T = 36$. This is still a good starting solution becacuse Tabu Search is able to provide the optimal solution in a really few iterations with our instance.

2.2 Neighborhood

Next, we need to decide how we define a neighborhood: again, this is a simple example, so we use one of the most intuitive ones. We can image that if we start from a generic sequence, one way to find a better one could be swap the order of some jobs, and repeat this iteratively.

So we will choose as a move the swap between two jobs (they could have been also three or four, but for now, we take two), and the neighborhood will be defined by this type of move. The move itself can be represented as tuple(i, j) where i and j are the indexes of the position that has been moved. For example, the move $(1, 2)$ on the starting solution $(1, 2, 3, 4, 5, 6)$ will produce the sequence $(2, 1, 3, 4, 5, 6)$.

Then, of course, as we said in section (2.1) each sequence as value T associated. As we sistematically explore our neighborhood, we will take the sequence with the minimum T value associated, since we are solving a minimization problem. As a starting example, let's see what is the neighborhood of our starting solution, and the value of each solution.

move	neighbor	value
(1, 2)	2, 1, 3, 4, 5, 6	37
(1, 3)	3, 2, 1, 4, 5, 6	42
(1, 4)	4, 2, 3, 1, 5, 6	32
(1, 5)	5, 2, 3, 4, 1, 6	57
(1, 6)	6, 2, 3, 4, 5, 1	40
(2, 3)	1, 3, 2, 4, 5, 6	39
(2, 4)	1, 4, 3, 2, 5, 6	30
(2, 5)	1, 5, 3, 4, 2, 6	56
(2, 6)	1, 6, 3, 4, 5, 2	43
(3, 4)	1, 2, 4, 3, 5, 6	30
(3, 5)	1, 2, 5, 4, 3, 6	40
(3, 6)	1, 2, 6, 4, 5, 3	30
(4, 5)	1, 2, 3, 5, 4, 6	44
(4, 6)	1, 2, 3, 6, 5, 4	39
(5, 6)	1, 2, 3, 4, 6, 5	29

We notice some things:

1. The yellow move $(5, 6)$ produces the next sequence $(1, 2, 3, 4, 6, 5)$ because has the best value T among all the solutions (29).

2. The neighborhood does not contain the previous sequence, which is one of the basis of Tabu Search.

3. A move has no direction: swapping 5 and 6 is the same as swapping 6 and 5, because they produce the same sequence, so the tuple $(5, 6)$ is the equivalent of $(6, 5)$, and so in general we will consider just the tuple (i, j) with $i < j$.

4. We have 15 possible moves, because we have 6 possible jobs. In general, with n jobs and a 2-swap move, we will have $\frac{1}{2}(n^2 - n)$ possible moves, so we are in a quadratical order of complexity ($O(n^2)$). So this operation could be a lot more expansive with a large n, or if we use a generic k-swap move instead of a 2-swap. This need to be considered for a problem of larger size.

2.3 Tabu Move

At this point, we have the ingredients to start a basic local search: we could run a greedy algorithm that always finds the best solution in the neighborhood until it reaches a minimum. Then, there two possibilities: we have found a global minimum, and are not able to move anymore, or we have found a local minimum, but still, we are not able to move anymore. So, not only we are not able to find new solutions, but also we don't now if our current solution is a global minimum or a local minimum. These are characteristic of simple local search. In Tabu Search we still don't know if our current solution is a global optimal solution, but we can try to escape from local minimum.

Now we need to determine what move is "Tabu": the easiest option is a set of moves that we have done recently. For example, we could have a local minimum with a sequence $s_1 = (4, 2, 1, 3, 6, 5)$ with $T = 22$, and another local minimum with the same T, like $s_2 = (2, 4, 1, 3, 6, 5)$. Note that in this problem, with this instance, the situation of local minimum described never happens; but this can be a pretty common situation with larger instances, because the swap of two jobs before their due date that remains before their due date, gives the same value of the objective function T.

So if we want to avoid it, we need to block some moves. In this example, we are allowed to bounce from s_1 to s_2 with the move $(1, 2)$ but we do not want to go back to s_1. So, when we go from s_1 to s_2 we add the move $(1, 2)$ to our tabu list, and when we are in s_2 we can't do the move $(1, 2)$ (the equivalent of $(2, 1)$) again and go back to s_2, so we will take another sequence, different from s_2.

2.4 Tabu Tenure

In the previous section we have shown that we will block the move that we have just done to avoid cycling; we also said that "the easiest option is a set of moves that we have done recently". In fact, blocking only the last move in general could not be so effective, because one move often is not enough to go out from "valleys" determined by a local minimum. So, we choose our set as the last n moves that we have done. When we make a move, it will be added to our Tabu list, and after n iterations it will be removed. Of course, we have 15 possible moves, so we can't have $n >= 15$.

Another time, this is a simple instance, and we can find the global optimal solution just with a simple greedy algorithm (look at section (3) for more) for every starting solution; so the length of the tabu list is not a problem in this case. In general, we need to pay attention to the length of the list to find a good balance between intensification and diversification. In this case we choose $n = 3$ as tabu tenure.

3 Algorithm implementation

The algorithm has been implemented in `Python` language with the choices made in section (2). Code is available here. Also, a report of the results of the first 10 iterations of the method, obtained from `tabusearch.py` is reported in table (1).

Best solution is found at iteration 3 (and iteration 4 has the same value as well), in yellow.

Is this the global optimal solution? Well, we don't know. As explained in section (2.3), local search methods do not guarantee optimality. Also, we want to answer another question. How much is tabu list important to find a solution in this example?

Answer for the first question is available in `bruteforce.py`. Fortunately, in this case, we can make a simple check: we have only six jobs, and with such a small amount, we can try every possible

Iteration	Sequence	Value	Tabu Moves
0	1, 2, 3, 4, 5, 6	36	empty
1	1, 2, 3, 4, 6, 5	29	(5, 6)
2	1, 2, 4, 3, 6, 5	23	(5, 6), (3, 4)
3	1, 4, 2, 3, 6, 5	19	(5, 6), (3, 4), (2, 3)
4	4, 1, 2, 3, 6, 5	19	(3, 4), (2, 3), (1, 2)
5	4, 1, 2, 6, 3, 5	21	(2, 3), (1, 2), (4, 5)
6	4, 1, 2, 6, 5, 3	23	(1, 2), (4, 5), (6, 5)
7	4, 2, 1, 6, 5, 3	26	(4, 5), (6, 5), (2, 3)
8	2, 4, 1, 6, 5, 3	26	(6, 5), (2, 3), (2, 1)
9	1, 4, 2, 6, 5, 3	23	(2, 3), (2, 1), (1, 3)
10	1, 4, 2, 6, 3, 5	21	(2, 1), (1, 3), (6, 5)

Table 1: Results of the first 10 iterations, iteration 0 is the starting feasible solution

sequence (720 possible sequences) and find which one has the best value among all. The answer is that the global optimal value is 19, and there are two possible sequences with this value, we can see each one of them in table (1).

To answer the second question, in `greedylocalsearch.py` has been implemented a simple local search for each starting sequence possible, that explores the same neighborhood described in (2.2) and stops when no better solution is found in the neighborhood, without any tabu list. The result is that every possible sequence of the 720 available converges to the optimal solution with an average of 3.325 iterations. Of course, Tabu Search is a method that tries to escape local minimum, so in general it provides better solutions than a simple local search. We have obtained this result just because this is an example with a small instance.