Tabu Search

Global Optimization

Medgyes Csaba, Mészáros Bálint

December 18, 2022

Tabu search

- One of the most popular metaheuristics algorithms
- TS was first proposed by Glover in 1986 and was also developed by Hansen.
- TS tries to find the best admissible solution in the neighborhood of the current solution in each iteration, considering recent solutions as 'Tabu' to prevent cycling.
- We are going to demonstrate it with the Single Machine Total Weighted Tardiness Problem (SMTWTP) which is an NP-hard problem

SMTWTP

- Suppose that we have a machine that can handle only one job at a time, and there is an N number of jobs (or tasks) to be processed (without interruption).
- Each job $i \in N$ requires an integer processing time P_i, and has a positive weight W_i indicating the importance of the job and a due date d_i.
- We can indicate the completion time of job i as C_i and the tardiness of the job can be calculated as $T_i = max\{C_i - d_i, 0\}$, so if the job is processed before its due date $C_i \leq d_i$, the will be no tardiness ($T = 0$).
- The objective is to order the N jobs in a way that minimizes the total weighted tardiness of the whole process, i.e.: $\min \sum W_i T_i$.

The Steps of the Tabu Search Algorithm

Step 0:

- The initial step is to create an initial solution so the algorithm can iterate over it and find a better one.
- In most cases, this initial solution is assigned randomly (unless you have a better understanding of the problem).

Step 1:

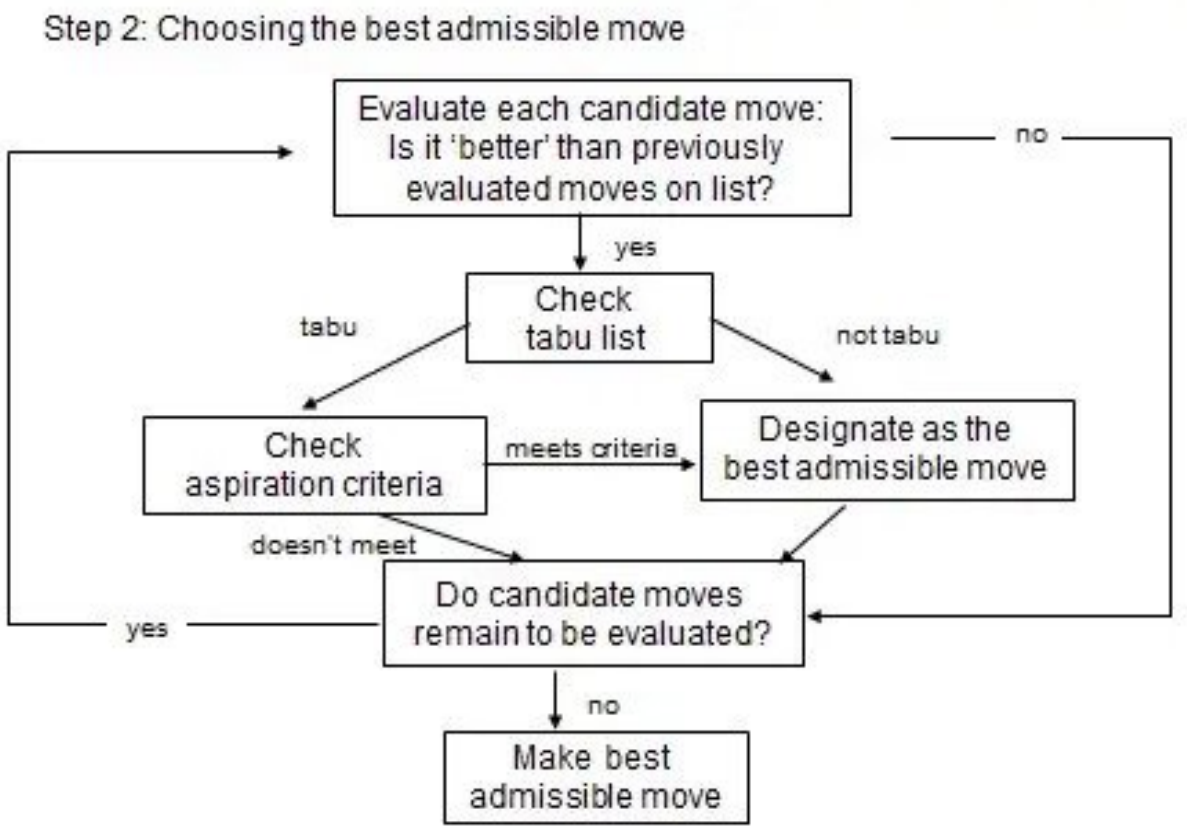
- The next step is to create the list of candidate solutions from the current solution $\mathbb{S}$ (initial solution in Iteration 0), we call these solutions neighbors or the neighborhood of $\mathbb{S}$.
- To find the neighbor solutions from the current solution $\mathbb{S}$, we need to define a neighborhood function (Example on the next slide)

Remark

Example for neighborhood function: Let's assume that the current solution of the problem is $\mathbb{S} = [3, 8, 10, 4, 1, 6, 2, 5, 9, 7]$ and we defined our neighborhood function as a swap move: replace the order of two jobs, then one neighborhood solution would be [8, 3, 10, 4, 1, 6, 2, 5, 9, 7] where jobs 8 and 3 are swapped, another would be [8, 3, 5, 4, 1, 6, 2, 10, 9, 7] swapping 5 and 10.

Step 2:

- From the neighborhood solutions list created in Step 1, we choose the best admissible (Non-tabu or meets aspiration criteria) solution by checking each solution as in the diagram below:



Tabu List

The list TS uses to record the recent solutions and prevents them to reoccur for a specified number of iterations.

- Tabu Tenure: The size of the Tabu list, i.e., for how many iterations a solution component is kept as Tabu. Tabu Tenure has a great impact on the TS performance and we can say that the smaller the tenure, the higher the probability of cycling.
- Tabu Attribute: If we have big Tabu tenure and big problem instances like 10000 number of jobs in our problem, recording tall visited solutions in the Tabu List will be very consuming and expensive. Instead, we store the moves performed rather than the whole solution. Tabu Attributes defines the solution component kept in the Tabu List.

Example for Tabu Tenure:

- Assume that the Tabu Tenure is 2, in Iteration 1 the Tabu List is empty.
- Let's assume that the best admissible solution found in Iteration 1 is [8, 3, 5, 4, 1, 6, 2, 10, 9, 7] then the Tabu List becomes [8, 3, 5, 4, 1, 6, 2, 10, 9, 7].
- In Iteration 2, the best solution is still [8, 3, 5, 4, 1, 6, 2, 10, 9, 7], however it's not admissible because it's Tabu, so we take the next best solution. Let assume it's [8, 3, 5, 4, 1, 6, 2, 10, 7,9]. Tabu list now equals to [8, 3, 5, 4, 1, 6, 2, 10, 9, 7], [8, 3, 5, 4, 1, 6, 2, 10, 7,9].
- In Iteration 3, [8, 3, 5, 4, 1, 6, 2, 10, 9, 7] solution leaves the Tabu List, and the next best admissible solution goes into the list (because the list size is 2).

Aspiration Criteria: If the solution found in the current iteration has a value (objective function value) better than the currently-known best solution's value, but the move (like don't swap job 3 with 1) is Tabu, we can use the Aspiration Criteria to override the Tabu state of this move, thereby including the otherwise-excluded solution in the allowed set.
Step 3:

- Check the defined stopping criteria, this can be the max number of iterations reached or the time of running, if the stopping criteria are not met, go to step 4, if the stopping criteria are met terminate and return the best solution.

Step 4:

- Update Tabu list, Aspiration Criteria, and go to Step 1

Since now we understand the algorithm we should check out an implementation in Python. Thanks for your attention!

https://github.com/csabamedgyes/global_optimization_tabu_search