# PFSP heuristics project

Dorian Dumez

May 8, 2017

## 1 Code

### 1.1 How to use

The whole project can be compiled by making "make" in the code directory. Beside all .o file can be deleted with "make clean" in the same directory. Finally the main can be called on all instances with the script "multiple_run.sh" at the root of this project, results are stoked in the "tmp.txt" file.

The main code call the tabu search and the iterative local search with given parameter on the given instance. The only two common parameter are the instance, specified with "–instance path_to_instance", and the time budget (always the same for the three function), specified with "–tmax time_budget_as_long". Then the output will be de mean value over 5 execution (or 1 for deterministic algorithm) of the measurement. Score, relative deviation and execution time can be measured, this is chose by comment/uncomment preprocessor variable at the beginning of "main.cpp".

The two tabu search have same parameter but they can be different. First, the tabu list length control the time that a move stay tabou, it's controlled with "–tabuListLenghtE integer_value" for the exchange tabu search and "–tabuListLenghtI integer_value" for the insert tabu search. Secondly, the long time memory impact is a floating point value that quantify the penalisation weight of the long time memory, it's controlled with "–longTimeMemoryImpactE double_value" and "–longTimeMemoryImpactI double_value". Finally, the restart threshold is the acceptable relative deviation between the current solution and the best one, if this exceed the current solution become again the best one, this is set with "–restartThresholdE double_value" and "–restartThresholdI double_value".

### 1.2 Initial solution

For all of following algorithm the initial solution is constructed with the rz heuristic. Indeed regarding to experiments we did in the first part a random initialisation hasn't any advantage over the rz heuristic. Beside in [3] they use it too, so i look like a good initialisation.

## 1.3 Tabu search

For my tabu search I choose to use the exchange or the insert neighbourhood. Indeed they were best one in descent algorithm, beside in [4] they use their tabu search with these neighbourhood.

In a tabu search the whole neighbourhood is crossed to search the best solution. But tabu movement, movement which have been done too recently, aren't checked. With the exchange a movement is the exchange of job at the index i and j. With insert it's move the job which was at the place i to the place j. Beside for both of them the movement is symmetric, i and j can be inverse without changing the modification, so the tabu list take care of that. To conclude the basis of our tabu search is to cross neighbourhood by selecting the best non-tabou neighbour (a neighbour which can be reach with a non-tabou movement) until the budget time is used.

Starting from this basic version I add some feature. They are inspired by my knowledge, the article of fred glover [2] and a usage of a tabu search on the same problem in [4]. But in [4] they have slightly different usage of this algorithm because it's only the local of a genetic algorithm.

The first improvement of our tabu search is the well known aspiration criteria. Now we crossed the whole neighbourhood each time, and even test the tabou move. Then if the move give us access to a better solution than the best found so far this movement is keep. I mean that the tabou criteria can be ignored if a better solution than the best one is found. This move is so classic and logic that is not a parameter and it was included is our first iteration of the tabu search.

The second improvement is the long time memory. Indeed, because of the symmetric property of our move, we use only the half of the tabu matrix. I mean when we do a move we stock the next date (number of the iteration) when it will be available, so the tabu list data-structure is a nbJob $\times$ nbJob matrix of long (for the date). So we can use the other half of this matrix to count how many times a move have been done. Then the goal is to diversify the search so we penalised too used movement. I mean instead of just go to the neighbour which have the best value according to the evaluation function we go to the one which have the best evaluation. This evaluation is now $value(solution) + longTimeMemoryImpact * numberOfUsage(move)$, longTimeMemoryImpact is a parameter fixed by the user. Beside we can notice that this add on allow to avoid solution cycle (some can be longer than the tabu list length) by penalised move of these cycle. As a drawback if this factor is too hight their no intensification anymore.

The third improvement is the restart to the best so far. The goal of feature is to do more intensification in promising zone of the search space. Indeed during a tabu search we travel a lot across the solution space, so we might go into non interesting zone. Obviously we doesn't want to spend to much time with bad solution. Then the idea is when we go to a too bad solution we can think we are in a non-promising zone, because we always go the best solution in the neighbourhood (plus the small difference of the long time memory). In these case we want to go back in a promising zone as quickly as possible, so go back to the best solution found so far is a good way. Technically when we go to a solution which have a value superior to $(1 + \text{restartThreshold}) * value(best)$ the current solution become again this best solution. The restart threshold is a parameter given by the user. Contrary to the long time memory when this threshold is too low their no exploration, and even get out a local minima zone may be impossible.

### 1.4 ILS

## 2 Tuning

### 2.1 Tabu search

For the tuning of parameter I start by doing it myself by multiple try. Then I notice that these 3 parameter are strongly linked so they can't be study independently. And visualize at the same time 3 parameter is complicated so I just found a not too bad setting . But most important I feel extreme value of these one, out of them the feature become useless or destabilize to much the algorithm :

- between 6 and 17 for the tabu list length, in [4] they use 7 but with a restricted candidate list
- between 0.001 and 1 for the impact memory
- between 0.05 and 0.4 for the restart threshold

With slightly extend bound I run the irace algorithm [1]. Indeed I extend these bound to allow the algorithm to find exotic setting or to disable an incapacitating feature. It run during 20 hours (4800 run of the tabu search, on 3 core, forth one for the system) on my computer with size 50 instances. I chose to only use small instance because a run already need 45 seconds, $500 * 0.09$ (execution time of VND transpose $\rightarrow$ exchange $\rightarrow$ insert with rz heuristic) and I can't parallelyze more.

So i do the tuning run previously described twice : one time with the exchange neighbourhood and one time with the insert one. Best setting output by irace are compiled in the table 1.

| neighbourhood | tabu tenure | long time memory impact | restart threshold |
|---|---|---|---|
| exchange | 10 | 0.001 | 0.3 |
| insert | 15 | 0.544 | 0.061 |

Table 1: Setting of tabu search

We notice that setting with exchange neighbourhood are quite balance, but a bit centre on intensification. Indeed this combo of tabu tenure and long time memory impact allow some intensification but avoid cycling, and the restart threshold is not that hight so the algorithm will explore a bit but still based on intensification. On the contrary setting output for insert neighbourhood are more exotic. The tabu tenure is more large, so it's allow more intensification and unfortunately cycling. But the long time memory impact is very hight so cycle wont take so much time beside the research will be diversified very quickly. Then to allow some intensification the restart threshold is very low, so the algorithm will come back very often to the best solution, and due to the long time memory impact take an other way.

### 2.2 ILS

## 3 Analysis

For all the test, all algorithm had been run on each instance. For tabu search only one time because both tabu and rz algorithm are deterministic. And because of the probability involve in ILS an average had been done over 5 run.
To cram table I use some abbreviation :

- tabuE for the tabu search with exchange neighbourhood
- tabuI for the tabu search with insert neighbourhood
- ILSE for ILS with a iterative improvement over the exchange neighbourhood
- ILSI for ILS with a iterative improvement over the insert neighbourhood

During the first part we prefer VND with transpose then exchange then insert initialized with rz heuristic. This algorithm need, on average, 0.09 seconds to converge on size 50 instance and 1.05 on bigger one. So all of these algorithm will be test with $0.09 \times 500 = 45$ seconds on small instance and $1.05 \times 500 = 525$ seconds on big one.

### 3.1 Statistical comparison

| size | tabuE | tabuI | ILSE | ILSI |
|---|---|---|---|---|
| 50 | 0.55 | 4.51 | ? | ? |
| 100 | ? | ? | ? | ? |
| all | ? | ? | ? | ? |

Table 2: Relative deviation average in %

| size | tabuE | tabuI | ILSE | ILSI |
|---|---|---|---|---|
| 50 | 0.21 | 3.90 | ? | ? |
| 100 | ? | ? | ? | ? |
| all | ? | ? | ? | ? |

Table 3: Relative deviation standard deviation

### 3.2 Behaviour analysis

## Conclusion

## References

[1] Manuel López-Ibànez & Jérémie Dubois-Lacoste & Leslie Pérez Cáceres & Thomas Stutzle & Mauro Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[2] Fred Glover & Manuel Laguna & Rafael Marti. *Principle of tabu search.* 2007.

[3] Quan-Ke Pan and Rubén Ruiz. Local search methods for the flowshop scheduling problem with flowtime minimization. *European Journal of Operational Research*, 222(1):31 – 43, 2012.

[4] Lin-Yu Tseng and Ya-Tai Lin. A genetic local search algorithm for minimizing total flowtime in the permutation flowshop scheduling problem. *International Journal of Production Economics*, 127(1):121 – 128, 2010.