PFSP heuristics project

Dorian Dumez

May 17, 2017

1 Code

1.1 How to use

The whole project can be compiled by making "make" in the code directory. Beside all
.o files can be deleted with "make clean" in the same directory. Finally "main" (the
executable) 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".

We can say that we have two tabu search, one with the exchange neighbourhood
and one with the insert one, it's choose with "–neighbourTabu neighbour_index" (1 for
exchange and 2 for insert). But these two tabu search use same parameter so the neigh-
bourhood to use is just one of them. First, the tabu list length control the time that
a move stay tabou, it's controlled with "–tabuListLenght integer_value". 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 "–longTimeMemoryImpact double_value".
Finally, the restart threshold is the acceptable relative deviation between the current
solution and the best one (found so far), if this is exceed the current solution become
again the best one, this is set with "–restartThreshold double_value".

Contrary to tabu search ILS have way more parameter. The first one is the same
as tabu : the neighbourhood relation to use. It's choose with "–neighbourILS neigh-
bour_index" (1 for transpose,2 for exchange,3 for insert, 4 and 5 for the two VND). All
local search can be done in first or best improvement, controlled by "–DD boolean"
(0 for first improvement and 1 for deepest descent). The third one is choice of the
move used for perturbations, same as the first parameter it's "–neighbourPerturb neigh-
bour_index" (1 for exchange and 2 for insert). The next two are here to tune the pertur-
bation procedure. perturbFrac is here to chose the fraction of the solution that should
be change, use it with "–perturbFrac double_beetween_0_and_1". perturbRadius allow to

limit the power of a perturbation by bounded it, it's controlled with "–perturbRadius double_beetween_0_and_1" (0 allow only small perturbation and 1 all of them). Finally come the most parametrized choice : the acceptance criterion of new solution. It's control with "–acceptanceCrit index", where :

- 1 is for only improvement

- 2 is for always

- 3 is for metropolis, which need other parameter :

    - T0 the initial temperature, chose with "–T0 double"
    - alpha the cooling factor, chose with "–alpha double"
    - l the length of cooling plateau, chose with "–l long"
    - warmupThreshold the lower bound of the temperature (if it's reach a warmup is used), chose with "–warmupThreshold double"
    - T1 the temperature after a warmup, chose with "–T1 double"

- 4 is for the metropolis type proposed in [5]. It's require a parameter lambda controlled with "–lambda double"

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 [4] 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 [6] they use their tabu search with these neighbourhood.

In a tabu search the whole neighbourhood is crossed to search the best solution. But tabou 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 [3] and a usage of a tabu search on the same problem in [6]. But in [6] 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 × 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 penalise 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 $(1 + restartThreshold) * value(best)$ 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.

After the first tuning session I notice that I can slightly modified the long time memory and the tabu list for the exchange neighbourhood to make it more meaningful. Basically a move was the exchange of two slots, the ith job and the jth one. But if we say that it's the exchange of the job at th ith place and the on at the jth place it's more logic. Because it's mean that the job A is before the job B or the contrary. First we can notice that it doesn't change anything with the tabu list, because they aren't move anyway. But it give a physical meaning to the long time memory : how many time we've tried A before B or B before A.

1.4 ILS

My IlS follow the basic outline of this algorithm 1, it's not wrote but we always keep track of the best solution found.

The local search can be done with all option we seen in the first part of the project. Transpose is available because of his run time, indeed this allow to do a lot of local search,

Algorithm 1 outline ILS

```
1: procedure ILS
2:     s = Construction()
3:     s = LocalSearch(s)
4:     while EndCriterion do
5:         s' = Perturb(s)
6:         s' = LocalSearch(s')
7:         If Acceptable(s') Then s = s'
8:     end while
9: end procedure
```

even if they should have a lower quality. All other are available because don't give that bad quality solution and aren't too slow, so I would like to test it. Beside all of these local search can be done in first or best improvement.

For the perturbation only exchange and insert are available, because transpose doesn't modify that much a solution. A perturbation will be random movement in these neighbourhood. Then a very important choice must be done : how much the solution should be changed. We formalize that with the PerturbRatio, the ratio of job that are changed (in fact a movement change 2 job so it's 2 times this ratio). So if it's 0 we don't perturb, so it's just a descent and the algorithm will be stick in the local minima during all the while loop. As contrary if it's 1 (or more than 0.5) we do a complete restart every time, so there no intensification, beside as we seen in the first part random construction is quite bad. But if we take a balance ratio, for example 5% (so 10% of the job are move), we perturb enough to get out local minima but it's not a restart. Beside the local search become a bit faster, indeed the solution is already quite good so the convergence is faster. With this idea we can add an other parameter : PerturbRadius. The radius of perturbation restrict the perturbation by limited possible move. A perturbation move is allow only if the two random number aren't too spread, in the exchange neighbourhood that mean that the two exchanged job aren't too spread in the schedule and in the insert that the job isn't insert too far away from his current position in the schedule. In practice we take the first random number, $rnd1$, freely in $[\![0, nbJob - 1]\!]$ and the second one only in $[\![rnd1 - nbJob * PerturbRadius - 1, rnd1 + nbJob * PerturbRadius + 1]\!]$. So we have just $2 * nbJob * PerturbRaduis + 2$ choice for the second number, so if $PerturbRadius \leqslant 0.5 - \frac{1}{nbJob}$ it's a restriction of the possible perturbation. The ± 1 is here as a security, in that way, even if PerturbRadius is too low related to the instance size, their always possibles move. Same as PerturbRatio a small PerturbRaduis will encourage intensification with the risk that will stay stuck in a local minima, and a big one will encourage diversification. But we shall see them as a couple because their are strongly linked. Indeed they can be both balance to performed some movement which conserve the general shape of the solution. But they can be more exotic, for example a high PerturbRatio with a small PerturbRadius can work because it will do a lot small modification, so it's still conserve the shape of the solution.

Their also few possibility for the acceptance criterion, I mean the function which decide if the new solution become the current one. I take all possibility described in [4] and I add the metropolis condition (the classical condition of simulated annealing)

even if it's parameterfull. The first one is to keep the solution only if it's improve the current solution, so only if it's the new best. This choice completely kill diversification in favour of intensification, so it shall be counterbalance by perturbation. The second one, always accept, is the contrary. Because it always accept the new solution the perturbation shouldn't change too much the solution to allow the algorithm to do a bit of intensification. The third one is the metropolis condition, this one can accept worst solution based on probability. This probability depend of the current temperature of the algorithm, which is closely related to the execution time. More formally $T = T_0 \times \alpha^{\lfloor date/l \rfloor}$ then the probability of keeping a degrading solution is $e^{\frac{-\delta}{T}}$ where

- δ is the value of the solution minus the value of the current solution

- date is the number of loop that had been performed

- T_0 is the initial temperature, passed as parameter

- l is the length of a cooling plateau, give as parameter

- α is the cooling factor, an other parameter

The value of T_0 should be quite high to allow the algorithm to escape the local minima basin of the first local search (which is quite good according to the first part). But in practice it should be found by experiment because it depend of the magnitude and the variation of the objective function. The value of l should be proportional to the size of the neighbourhood relation because it's related to the time we'll spend to cross it. But it's also should be determined by experiment because it depend of the landscape of the objective space. Finally α determined the time we give to the algorithm to converge. Classically this factor is quite high, over 0.8, to give time to the algorithm and don't force him to converge in a bad local minima basin. But it shouldn't be too high or the algorithm will take a lot of time to converge, so maybe loose many time in bad region. With only that, after a first convergence the algorithm may stay stuck in his local minima, which may not be the global one. So a warm up strategy had been implemented, if the temperature go bellow a threshold we augment it. In practice if the temperature go bellow warmupThreshold we put it at the value T_1. Obviously $T_0 > T_1 >$ warmupThreshold. More precisely the threshold should be quite low to allow the algorithm to do some intensification in promising zone, but it shouldn't be too low to avoid to spend too much time stuck in a local minima and reject all solution. Beside T_1 should be an average between T_0 and warmupThreshold because we neither want to restart the algorithm neither warm up to just move in our local minima basin without escape it. The previous criterion can be very efficient but it require a lot of parameter, all strongly linked, and their are closely linked to the instance type. So I add an other criterion, described in [5], as a compromise. In fact it's the metropolis criterion but with a constant temperature $\lambda \times \frac{\sum_{i=1}^{nbJob} \sum_{j=1}^{nbMac} p_{ij}}{10 \times nbJob \times nbMac}$ where nbJob is the number of job, nbMac the number of machine and λ a parameter. According to [4] λ is a robust parameter but according to [5] it should be quite low. Finally M. Stuetzle advise me to test only some value : 0.3, 0.6, 0.9, 1.2 and 1.5.

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 more 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 [6] 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 [2]. In fact 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. As say previously before I slightly modified the exchange version so I do an other training with irace. But this one was based on the result of the previous version so with an initial solution, smaller bound for parameter but also a smaller budget. 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
exchange2	9	0.26	0.048
insert	15	0.544	0.061

Table 1: Setting of tabu search

We notice that setting with exchange neighbourhood (first iteration) 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. The second iteration with exchange still quite balance but in the other way : the long time memory push to diversification and the restart on intensification. On the contrary setting output for insert neighbourhood are more exotic. Both tabu list and long time memory will really push to diversification (because associated parameter are very high), and the restart will take care of intensification. So it will act a bit like the second version of exchange but in a more exaggerated way.

2.2 ILS

Because of the huge amount of possible setting of ILS I just do one big run of irace. Indeed their is 80 combination of categorical parameter, plus some of them with real one. Beside all of the parameter can't be tested separated and their value compared by statistical test because many of them are strongly linked. The only experiment that could have been done is choosing one parameter (for example the local search) and let irace tune other one to compare them after, but it's too time consuming (maybe around 5×20 hours to have good results), beside with only one long run irace will do the same, and maybe better because it must have more time for the best possibility. So for this big run I give 10560 run (40h on 3 core) as budget. Beside I give to the algorithm quite good solution that I found by hand on few instances. Finally bounds on parameter had been used in the same way as the tabu race. Beside constraint on the value of T_0, T_1 and warmupThreshold had been add, and according to [5] (and M. Stuetzle advice) only $0.3, 0.6, 0.9, 1.2$ and 1.5 are available as value for λ.

After the irace run I obtain the following parameter :

- Local search : VND with transpose then exchange then insert (code 4) in first improvement

- For the perturbation :

    - Insert neighbourhood
    - Fraction of perturbation of 0.068 (so 13.6% are modified)
    - Radius of perturbation of 0.083 (so with the size 50 instances a radius of 4)

- For the acceptance criterion :

    - Metropolis criterion
    - α (the cooling factor) at 0.938
    - l (the plateau size) at 128
    - An initial temperature of 572
    - A warm up threshold of 24 (reach after 6243 iterations)
    - A temperature after warm up of 1012

First of all we notice an unexpected value of the temperature, for an unknown reason constraint on these two hadn't been taken in account by irace (few moment latter I found a mistake in my scenario file, it doesn't take in account the forbidden file), and value it found are very curious. It will start by exploring the neighbourhood of the construct solution, and only after try something else. But finally by other experiment it appear that the algorithm do only around 900 iterations (for the two size instance), so irace disable the warm up by using a too low warm up threshold, so the warm up temperature doesn't matter anymore. Considering all of that this setting are relay based on intensification because we don't perturb that much and then use a powerful local search (so it may fall again in the same local minima) beside the acceptance criterion setting aren't that permissive.

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

- tabuE2 for the tabu search with exchange neighbourhood and modified long time memory

- tabuI for the tabu search with insert neighbourhood

- ILS for iterative local search

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

Table 2 and 4 are very clear, we have a order :

 i - ILS

 ii - tabuE

 iii - tabuE2

 iv - tabuI

In fact quality solution average always follow this order, for all instance size and overall, beside all wilconxon test's p-value are very low (far way smaller than 0.05). Beside the stability of an algorithm can't be a discriminatory criterion because, as we can see in table 3, the standard deviation follow this order too (and for all category). Finally all these algorithm have the same time budget so they are equal relatively to this.

size	tabuE	tabuI	tabuE2	ILS
50	0.55	4.51	1.40	0.30
100	1.01	3.88	2.06	0.50
all	0.77	4.17	1,73	0.40

Table 2: Relative deviation average in %

Results about the tabu search are quite unexpected :

- in [6] they found that the insert neighbourhood was better than the exchange one, and with my experiment the insert one is completely outperformed even if they have the same option and the same irace run

size	tabuE	tabuI	tabuE2	ILS
50	0.21	3.90	0.52	0.15
100	0.31	2.52	0.55	0.26
all	0.35	3.26	0.62	0.23

Table 3: Relative deviation standard deviation

	tabuE	tabuE2	tabuI	ILS
tabuE	NaN	1.67e-11	3.06e-11	4.84e-10
tabuI	1.67e-11	NaN	1.56e-07	1.67e-11
tabuE2	3.06e-11	1.56e-07	NaN	1.85e-11
ILS	4.84e-10	1.67e-11	1.85e-11	NaN

Table 4: p-value of the wilcoxon test

- what a I think as an improvement for the exchange neighbourhood is a drawback. But I tune it with a small budget because I was thinking that is setting must be close to the previous version. But, as we can observe in table 1, is not the case, so it might require a full tuning sequence (a 20h irace run) like other.

On the other side I got good surprise : all of these algorithm scale good. For tabu I feared the algorithm cycling because the tabu tenure became to low compared to the size of the neighbourhood relation, which scale at the square. But seems it doesn't, the long time memory may performed his job properly. Beside an completely illogic fact append with the insert neighbourhood tabu got better ARPD with big instances than with small one. And I feared scaling problem even more for ILS because l (the length of a cooling plateau) is suppose to be proportional to the size of the neighbourhood. Obviously to prevent scaling problem I set the perturbation parameter as ratio but for l it's wasn't possible. Indeed with both size 50 and 100 instances the algorithm perform around 900 local search. So i can't let the algorithm cross the neighbourhood that much time (square of the time for 50 size instance) as it should do, because it won't have the time to, and then doesn't have enough time to converge. Beside this scaling factor is quite obvious for descent in a neighbourhood but it's absolutely not the case for VND algorithm (which had been chose as local search by irace). So I let as constant this factor l because try to scaling it was meaningless and dangerous.

Finally ILS is more robust that I thought. Indeed temperature parameter are really instance linked because it related to the value of solution. In fact the probability to accept a bad solution is $e^{\frac{-\delta}{T}}$ so with a given instance T will be linked to the magnitude of the difference between goo and bad solution. I mean that if we are on a instance with relay low cost and and other one with high one then the relative difference between the initial solution and the best one will the same but in the first case the difference of value might be around 1 000 and for the other one around 100 000. Then it's obvious that the temperature should be different or the behaviour of the algorithm will be extreme for one of them (because of a relay too low or too high temperature). In my case I feared an other scaling problem because value of solution on the big instances are far way bigger than the one of the small instances. But in practice ILS, see table 2 and 3, still good and stable with the size 100 instances.

Even with all these odd results ILS is clearly the best algorithm on a general point of view, but what's relay append in detail ? Beside according to these result I think the strong point of ILS is the number of complex local search performed. I think that because irace choose VDN1, and for a long time it was in best improvement, but completely discard simple descent method really quickly. Beside with a too low temperature (that's append in the case of size 100 instance) their is practically no worsering move so I think the usage of GRASP should be study.

3.2   Behaviour analysis

The previous test was statistical one, which evaluated global quality of algorithm. Now we will study with in detail the behaviour of the two algorithm we prefer in the previous section : tabuE and ILS, others ones was outperformed by far.

Graph 1 and 2 show the qualified run time distribution for some quality solution. This come from execution of the first 5 instances of size 50. For ILS each of them have been done 25 time, and for tabu only one. To see how they behave the maximal execution time have been set to 450 seconds (10×45 seconds) but a line show the time limit we use in other experiment. Both of them are resume in the table 5 that show the average time to reach the same solution quality.
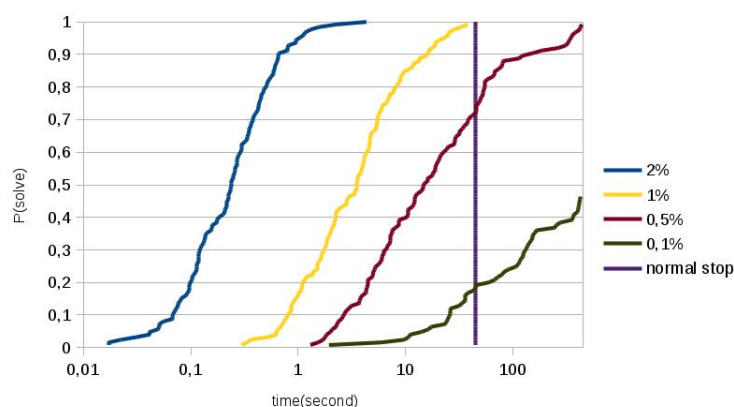

Figure 1: Qualified run time distribution of ILS

I choose a logarithmic scale for the computation time because both algorithm reach, at least, 2% relative deviation too fast to see it.

Because of the fact that a tabu search is deterministic we have too few data to plot an interesting qualified run time distribution graph. So to be able to compared ILS and tabu on this point I make the average table 5.

[1] : average over the first 5 instances when the algorithm reach the require quality solution before 450 seconds (10×45)

[2] : the algorithm never reach a such precision in 450 seconds

[3] : average over the first 5 instances repeated 25 times (with different random seed, time(NULL) had been used) when the algorithm reach the require quality solution before
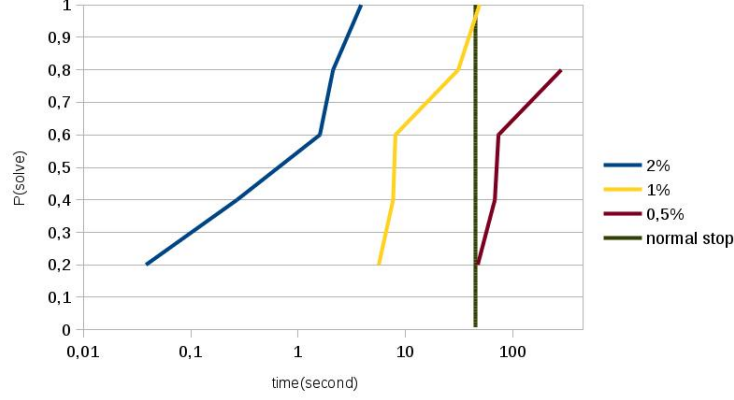
Figure 2: Qualified run time distribution of tabu

	2%	1%	0.5%	0.1%
tabuE	1.57^1	20.32^1	118.45^1	[2]
ILS	0.364^3	5.72^3	52.76^3	136^3

Table 5: Qualified run time distribution average

450 seconds

All these data are based on iteration that reach the given solution quality before 450 seconds. Only one run of ILS doesn't reach 1% and 0.5%, and only 58 over 125 succeed to passed bellow 0.1%. For tabu one over 5 doesn't succeed to reach 0.5% and none of them finish the 0.1%.

Both trace plot show the evolution of the average relative (in %) deviation over the time (in second). They use the execution trace of one run on each instances for both algorithm. The red line is for ILS and the blue one for tabuE. To have a smooth plot a polynomial regression of third degree had been used.

First we'll study the behaviour of ILS, then we'll watch over tabu and finally compared them.

As we can see on graphs 3 and 4 ILS have a very similar behaviour on both instance size. So it's contradicted our hypothesis that ILS would have a exotic (accept only few worstening solution) comportment on big instance due to the non scaling of the temperature. Beside on these graph we see that this algorithm act in 3 phase :

1 - a very aggressive phase : the relative deviation fall quickly. This may be related to the usage of VND : it's allow us to jump from great local minima to other one, so to fall quickly in a great basin of local minima.

2 - after the algorithm wont improve during a long time, and even worsening more and more. This might be the time to escape our local minima basin. But at this time the temperature should start to become lower so the algorithm might just travel to
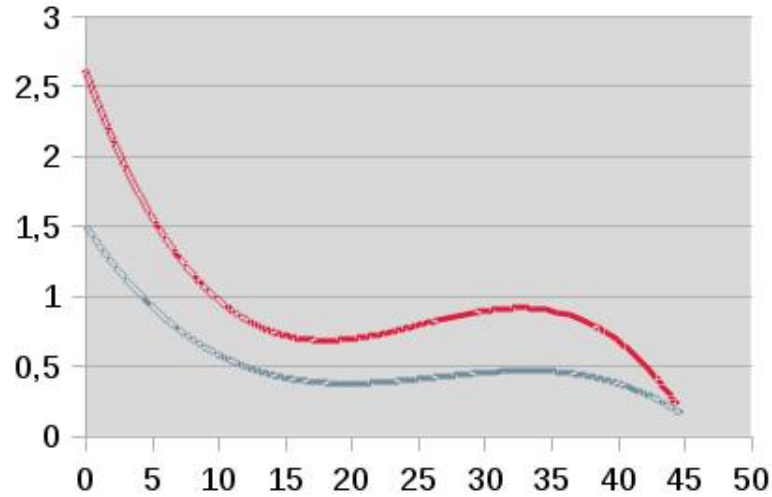
11
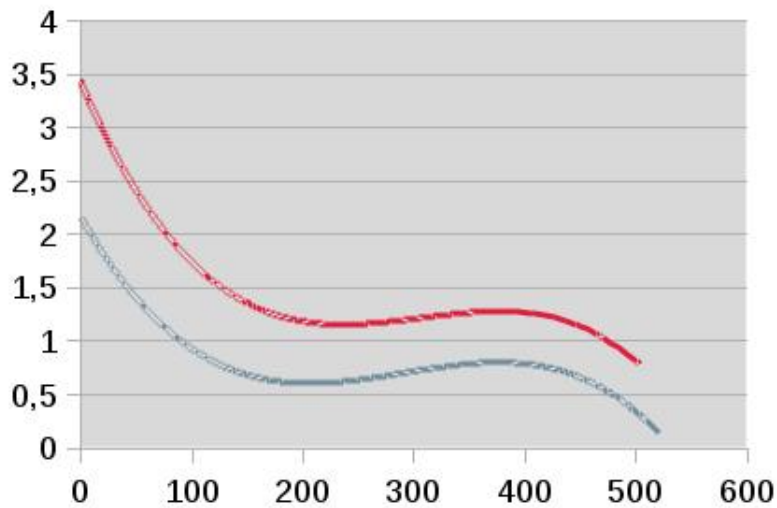
Figure 3: Solution quality trace on size 50 instances

Figure 4: Solution quality trace on size 100 instances

an near local minima basin.

3 - Finally, when the temperature start to become really low the algorithm start his final intensification phase. The result is that it deeply fall into his local minima basin and no more accept bad solution, it just run to the better solution and stop exploration

But when the algorithm stop we can see that it's not stabilized yet. So we can think giving him a bit more time will allow him to reach even better solution by finishing this intensification phase until it get really stuck.

Then we'll use the graph 1, this show the ratio of execution that reach a given quality solution over the time. And we can clearly see that around 20% of run reach the 0.5% relative deviation between 45 and 100 second. So our hypothesis seems true : if we give a bit more time to the algorithm it allow him to reach the minima of his basin so reach better solution quality. And i think this is the minima of his basin because after it's not

improving during a long time. After this stagnation period the number of run that reach 0.1% explode and the one that doesn't reach 0.5% yet do it. Beside I compute that the expected time before the warmup is around 310 seconds, so it's occur with this new wave of improvement. That add argument to the hypothesis according one the algorithm finish is race to best solution until it get stuck. Because this is the goal of the warmup : let the algorithm fall into a very good place, and go again on exploration to be able to use the remaining time. In our case, because irace didn't accede to this part of the run he didn't tune it, but we can observe that the warmup threshold can be a bit higher to avoid this long time with no improvement. For example set it at 190 will set the warmup around 110 second, so just at the beginning of the useless part. With this new parameter the temperature after warm should be tune. But in this case we just have to tune the warmup threshold around this value of 190 and a the warm up temperature, other parameter can stay like this because the first part of the execution seems good. Or we can decide to duplicate them to allow a different behaviour for the second part of the execution. Beside this can be coherent because the second part doesn't have the same role as the first one. During the first one we wanted to find a good minima basin and go deep into it, now we want to escape it and find an other one. But to decide the behaviour of the algorithm we should study the topography of these problem : determined if good minima basin are near or not.

As we can see on graph 3 and 4 tabu have similar behaviour on both instance size, but it doesn't performed that well the last second with big instances. Like ILS we can split the execution in few part :

1 - a very aggressive phase : same as ILS movement of tabu are powerful so it allow it to quickly find good solution. IN fact we can see movement of tabuE as a descent in best improvement in exchange neighbourhood as long as it doesn't fall into local minima (because of the aspiration criteria the tabu list will be ignored). And in the first part we see that this local search was quite good so it explain the first phase

2 - a learning phase : during this phase tabu will explore the space. During this time the long time memory will force the algorithm to exploration, and if it's go bad he go back to our best solution. So same as ILS we travel across our minima basin.

3 - a new local minima basin : tabu will search a new minima basin until it find a good one, enough good to allow intensification and outpass long term memory. And when it's found the aspiration criterion make the job.

Same as ILS, with the 3 and 4 curve, we can think that give a bit more time to tabu will allow him to find better solution. In fact this is true, according to graph 2, give him 100 second will allow tabu to reach 0.5% on many instances. To help him to do that we can think about an adaptable tabu tenure, proposed in [1], to follow the different phase.

The graph 5 allow to compare the two algorithms. Each blue and red point corresponding to one instance, this abscissa to the relative deviation of tabu on it and his ordinate to the relative deviation average (over 5 run) of ILS on the same instance. Small instances are represented by blue point and big one by red points, yellow point are here to visualize the $x = y$ curve.
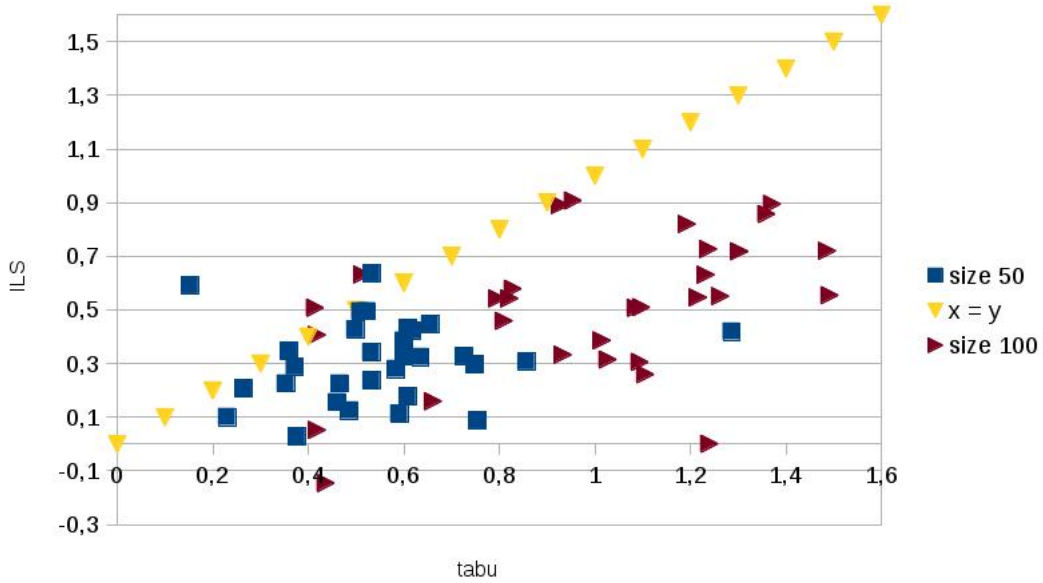
Figure 5: Correlation plot

First we can see on graph 3 and 4 that at any moment ILS have better solution than tabu. And our analysis of graph 1 and 2 told us that ILS have is more promising for longer run : ILS just need some tuning to reach more 0.1% and tabu need a new feature for 0.5%. Beside the global analysis, with table 2, 3 and 5 told us the same. Between these two scale of analysis we can use the correlation plot of the relative deviation after a classical run (45 or 525 second depending of the size) in graph 5. Again we can say that ILS is better because most of the point, for both size, are bellow the $x = y$ curve. So we can't even say that tabu is better than ILS on particular instance because he is clearly better on only one instance, and because of the shape of the point we can say that hard instance for tabu and ILS are roughly the same.

Conclusion

In the first part we prefer VND1 and now our best choice is ILS, using this VND. So for very quick run we have few possibility depending of the time budget. But for longer run ILS is clearly the best even if it need a bit more tuning for extra long run. Finally, regarding to the behaviour of ILS, a GRASP should be try. But some long time memory feature will be needed to avoid the quick stagnation of GRASP after his first very aggressive phase. We can think about path relinking, long time memory bias in the construction and the reactive component.

References

[1] Roberto Battiti and Giampietro Tecchiolli. The continuous reactive tabu search: Blending combinatorial optimization and stochastic search for global optimization. *Annals of Operations Research*, 63(2):151–188, 1996.

[2] 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.

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

[4] 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.

[5] Rubén Ruiz and Thomas Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177:2033–2049, 2006.

[6] 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.