PFSP heuristics project

Dorian Dumez

April 23, 2017

1 Code

1.1 How to use

In the code directory the command "make" (in the code directory) can be use to compile the whole project. Beside "make clean" (in the code directory too) delete all .o files.

After compiling the project experiment can be done on a single instance with "./main path_to_the_instance optimal_value". Then all possibility are use on this instance, for stochastic setting an average is done over 5 runs ("NBEXEC" preprocessor variable at the beginning of the "main.cpp" file). Depending of the settings of the code execution time,value of solutions or proportional difference is output. This can be chose before compiling by the pre-compilator variable, commented or not, at the beginning of the "main.cpp" file. Values for each settings are separated by ":" in the text output.

To have these result on all instances the multiple_run.sh script can be use. This bash script will execute the main on each instance and store all results in the res.txt file.

To compute few statistic R script have been created :

- "compute_average.R" which compute mean value for each algorithm. Values are taken in the file "tmp.txt", so it can be use to compute both average deviation and computation time. Values are output in the "avg.txt" file. We notice that an idicate line can be uncomment if the average is done with relative deviation in $[0; 1]$ and the result is wanted in percentage.
- "compute_standard_deviation.R" which calculate standard deviation, it's have been design in the same way as "compute_average.R" and both take their input in "tmp.txt". Values are output in the "sd.txt" file. The same line can be uncomment to follow the same principle as "compute_average.R".
- "wilcoxon_test.R" compute p-value of the wilconxon pairwise test on each pair of column. Beside it compute the Bonferroni correction on each of them. Output table are stoked into the "wilconxon.txt" file. Input file is "tmp.txt".

1.2 Datastructure of a solution

The solution is represented by a permutation of jobs. It's stored in an static array, the compartment n (we start at the index 0) store the id of the $n + 1$th job to do.

To compute the score we loop on both jobs and machine to compute end date of each jobs on each engine. Then we sum up, with weight, ends date on the last machine. So compute the score of a solution is done in $\Theta(nm)$, where n is the number of jobs and m the number of machine.

Beside, to avoid this too heavy computation time, we design a partial computation function. She use previous computed data to speed up this process. In fact when we compute end date, we store it into the solution's datastructure. And we can notice that end date of a job only depend of the previous one, so if only next one are modified we doesn't need to recompute it, because it doesn't change. In practice this function work like the classical computation of the score but start to compute end date from the index give in parameter, based on previous one. With that the complexity become $O(nm)$.

1.3 Transpose neighbourhood

The transpose neighbourhood is composed of all solution that can be reach by exchange of successive jobs. To scan it we just have to test, for all jobs, if the solution in which the job and the next are switch is better.

In practice we loop on jobs, for each one except the last, we switch him with the next one, then compute the score. If it's better than the actual solution we stop, else we switch them again and go to the next job. If the "dofor" mode is activated we doesn't stop, and just go to the next when we improve. If we are in deepest descent we always undo the switch, and if it's better than the best found so far in this scan we note the current index, after we can do this move in $O(1)$.

To avoid to compute the complete score each time we use the "recomputeWCT" function. But it's work only if jobs before the index parameter haven't been change since the previous computation of the score. Before we notice that we always compute the whole score before the loop to have the score of solution passed in parameter. So we just have to do the loop in the reverse order, start by switch last jobs, to be able to use the partial computation all the time.

1.4 Exchange neighbourhood

The exchange neighbourhood is composed of all solution that can be reach by exchange to job in the queue. To scan it we have, for each job, to test for all other if their switch improve.

In practice we loop on job, and inside we do an other loop to go all over jobs which are after in the queue. Indeed the switch is identical in both way, and we doesn't want to compute two times each neighbour. If it's better than the actual solution we stop, else we switch them again and go to the next job, when all are test we try to move the next job with next ones. If the "dofor" mode is activated we doesn't stop, and just go to the next when we improve. If we are in deepest descent we always undo the switch, and if it's better than the best found so far in this scan we note currents indexes, after we can do this move in $O(1)$.

With the same idea as before we use the "recomputeWCT" function. And to do it we again loops in the reverse order, so we try to exchange jobs with previous ones in the queue. But at the end of all second loop we need to fully compute the score to get the

good end date table. Indeed if we change the last job after working on first one, the end date table might not be correct (in fact it's correct only if the first job has been change and the switch retain).

1.5 Insert neighbourhood

The insert neighbourhood is composed by all solution that can be reach by insert a job between two other. To scan it we need, for all jobs, to try all other places in the queue.

In practice we use two loop, the first one go all over jobs, and the second one move it. Indeed try all possible indexes for a job is equal to switch it with the next one (in a cyclic vision, with a modulo) n times. So for all jobs we do this loop, if an improvement is find we stop all, else if move this job isn't worth we use a loop like the second one, but in reverse order, to bring him back at his original position. In the "dofor" mode we do the first loop for all jobs, and the second one run until an improvement (or the end of possibility). Beside in this mode to cross the neighborhood more efficiently we take care of testing move when his environment had been change. I mean when we insert a job before his current position instead of move the new job at the next index we try the one just after the new position of our job. And when the job is move forward we take care of doesn't skip a job because this part of the schedule is shift. Finally for the deepest descent we do all loop until the end, keep tracks of indexes of the best improvement, but in this case the application take a $O(n)$ to be done.

This time we use "recomputeWCT" slightly differently as previously. Indeed with the cyclic nature a reverse order wouldn't help. But we can still use it thank you to a modulo operation. After his behaviour will be near to this usage in the exchange neighbourhood.

1.6 VND

The implementation of VND is very classic. It have been implement in a generic way, with an array of neighbourhood to be re-used easily. Finally we need to notice that the first neighbourhood is used in descent so it's call in a slightly different way to use the "dofor" improvement if the neighbourhood is set for.

2 Testing

2.1 Test's average of neighbourhood

"tr" point out the transpose neighbourhood, "ex" exchange and "in" for insert. Finally "DD" means we use it int deepest descent. And we need to notice that all these experiment have been done in "dofor" mode.

size	tr	trDD	ex	exDD	in	inDD
50	32.4	31.7	4.0	3.9	5.4	6.5
100	41.2	41.2	4.7	4.7	6.4	9.1
all	36.7	36.4	4.3	4.3	5.9	7.8

Table 1: Relative deviation average in % with random construction

size	tr	trDD	ex	exDD	in	inDD
50	3.7	3.7	2.9	2.7	2.7	2.8
100	4.6	4.6	3.6	3.5	3.4	3.6
all	4.1	4.1	3.2	3.1	3.0	3.2

Table 2: Relative deviation average in % with rz construction

size	tr	trDD	ex	exDD	in	inDD
50	3.79	3.04	0.50	0.65	0.74	1.22
100	3.51	3.47	0.59	0.63	0.80	1.00
all	5.75	5.76	0.64	0.76	0.90	1.70

Table 3: standard deviation of the relative deviation with random construction

size	tr	trDD	ex	exDD	in	inDD
50	0.86	0.86	0.84	0.71	0.61	0.60
100	0.56	0.54	0.62	0.66	0.59	0.68
all	0.86	0.85	0.81	0.78	0.68	0.74

Table 4: standard deviation of the relative deviation with rz construction

size	tr	trDD	ex	exDD	in	inDD
50	0.001	0.006	0.05	0.249	0.12	0.47
100	0.005	0.046	0.46	3.6	1.27	6.13
all	0.003	0.02	0.25	1.9	0.69	3.25

Table 5: execution time average with random construction

size	tr	trDD	ex	exDD	in	inDD
50	0.005	0.005	0.03	0.06	0.04	0.08
100	0.038	0.040	0.36	0.84	0.45	1.21
all	0.02	0.02	0.19	0.44	0.24	0.63

Table 6: execution time average with rz construction

size	tr	trDD	ex	exDD	in	inDD
50	0.0002	0.0007	0.006	0.014	0.011	0.053
100	0.008	0.007	0.043	0.244	0.111	0.486
all	0.002	0.020	0.210	1.702	0.584	2.874

Table 7: execution time standard deviation with random construction

size	tr	trDD	ex	exDD	in	inDD
50	0.0002	0.0003	0.013	0.028	0.012	0.033
100	0.001	0.001	0.121	0.327	0.134	0.482
all	0.016	0.017	0.188	0.454	0.227	0.659

Table 8: execution time standard deviation with rz construction

2.2 Statistical test of neighbourhood

We perform the wilcoxon test on score, in a pairwise way, we will use a significance level of 0.05 for interpretations. These result are compiled in the table 9. Beside we do the same calculation for the execution time, these results are in the table 10. A population isn't comparable to itself so NaN is used in these cases.

"tr", "ex" and "in" are use in the same way as previously for transpose, exchange and insert. "rnd" is used when the start solution isn't construct, and "rz" when the rz heuristic have been used. "DD" is always here when the descent have been done in deepest descent, i.e best improvement. Otherwise the pivoting rule is first improvement.

	rndtr	rndtrDD	rndex	rndexDD	rndin	rndinDD	rztr	rztrDD	rzex	rzexDD	rzin	rzinDD
rndtr	NaN	7.83e-01	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rndtrDD	7.83e-01	NaN	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rndex	2.45e-11	2.45e-11	NaN	9.79e-01	2.58e-11	2.45e-11	4.04e-02	4.04e-02	7.86e-10	7.14e-11	2.45e-11	4.31e-11
rndexDD	2.45e-11	2.45e-11	9.79e-01	NaN	2.45e-11	2.45e-11	1.21e-01	8.87e-02	5.36e-10	8.30e-11	2.58e-11	1.67e-10
rndin	2.45e-11	2.45e-11	2.58e-11	2.45e-11	NaN	2.59e-10	5.55e-11	5.01e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rndinDD	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.59e-10	NaN	2.58e-11	2.58e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rztr	2.45e-11	2.45e-11	4.04e-02	1.21e-01	5.55e-11	2.58e-11	NaN	1.41e-01	2.93e-10	7.76e-11	2.45e-11	2.45e-11
rztrDD	2.45e-11	2.45e-11	4.04e-02	8.87e-02	5.01e-11	2.58e-11	1.41e-01	NaN	3.11e-10	7.76e-11	2.45e-11	2.45e-11
rzex	2.45e-11	2.45e-11	7.86e-10	5.36e-10	2.45e-11	2.45e-11	2.93e-10	3.11e-10	NaN	4.00e-02	3.76e-02	3.83e-01
rzexDD	2.45e-11	2.45e-11	7.14e-11	8.30e-11	2.45e-11	2.45e-11	7.76e-11	7.76e-11	4.00e-02	NaN	2.75e-01	3.28e-01
rzin	2.45e-11	2.45e-11	2.45e-11	2.58e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	3.76e-02	2.75e-01	NaN	3.36e-02
rzinDD	2.45e-11	2.45e-11	4.31e-11	1.67e-10	2.45e-11	2.45e-11	2.45e-11	2.45e-11	3.83e-01	3.28e-01	3.36e-02	NaN

Table 9: wilcoxon pairwise test, solution quality

	rndtr	rndtrDD	rndex	rndexDD	rndin	rndinDD	rztr	rztrDD	rzex	rzexDD	rzin	rzinDD
rndtr	NaN	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rndtrDD	2.45e-11	NaN	2.45e-11	2.45e-11	2.45e-11	2.45e-11	9.04e-09	1.10e-05	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rndex	2.45e-11	2.45e-11	NaN	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.03e-07	3.51e-06	2.65e-02	1.44e-10
rndexDD	2.45e-11	2.45e-11	2.45e-11	NaN	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rndin	2.45e-11	2.45e-11	2.45e-11	2.45e-11	NaN	2.45e-11	2.45e-11	2.45e-11	2.45e-11	5.11e-10	2.45e-11	2.32e-03
rndinDD	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	NaN	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rztr	2.45e-11	9.04e-09	2.45e-11	2.45e-11	2.45e-11	2.45e-11	NaN	7.39e-09	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rztrDD	2.45e-11	1.10e-05	2.45e-11	2.45e-11	2.45e-11	2.45e-11	7.39e-09	NaN	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rzex	2.45e-11	2.45e-11	2.03e-07	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	NaN	3.89e-11	1.22e-03	2.86e-11
rzexDD	2.45e-11	2.45e-11	3.51e-06	2.45e-11	5.11e-10	2.45e-11	2.45e-11	2.45e-11	3.89e-11	NaN	1.79e-07	4.06e-06
rzin	2.45e-11	2.45e-11	2.65e-02	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	1.22e-03	1.79e-07	NaN	7.51e-11
rzinDD	2.45e-11	2.45e-11	1.44e-10	2.45e-11	2.32e-03	2.45e-11	2.45e-11	2.45e-11	2.86e-11	4.06e-06	7.51e-11	NaN

Table 10: wilcoxon pairwise test, execution time

We compute the average of the relative deviation, for all instances and all neighbourhood relation (in deepest descent and first improvement), when we start with a random solution and the rz heuristic. After we compute a pairwise wilcoxon test, compared value are always output by the same algorithm (same neighbourhood, same browsing, different initialisation) on the same instance so when a comparison is done only the initialization differ. With a random start the relative deviation average is 15.9% and with the heuristic it's 3.5%. Finally the p-value of the wilcoxon pairwise test is $3.91e^{-59}$. So we can conclude that an initialisation with the rz heuristic is better as a starting point for a local search.

Then the same study have been done with the pivoting rule. With the first improvement the relative deviation average is 9.59% and with the deepest descent it's 9.85%. Beside the p-value of the wilconxon pairwise test (same idea as before, the only things

that differ between two compared value is the pivoting rule) is 0.000162. Surprisingly the first improvement with a for loop give slightly better result than a deepest descent.

Finally we do this a third time with the neighbourhood relationship. With the transpose relation the relative deviation average (on every pivoting rules and starting point) is 20.73% (we notice the big difference between the two initialisation), with exchange it's 3.76% and with insert it's 5.02%. As before a wilconxon pairwise test have been computed, p-value are in the table 11. So with that we can said that exchange is better than insert which is better than transpose, and the relation is transitive. But these results have bias because transpose and insert neighbourhood suffer much that exchange of the random initialisation. Indeed average only on rz heuristic of transpose is 4.15%, it's 3.20% for exchange and 3.14% for insert. Beside p-value of this wilcoxon test are in the table 12. So with the rz initialisation the insert neighbourhood is the better one.

	tr	ex	in
tr	NaN	4.79e-39	1.83e-40
ex	4.79e-39	NaN	6.02e-20
in.	1.83e-40	6.02e-20	NaN

Table 11: wilconxon p-value of neighbourhood, solution quality

	tr	ex	in
tr	NaN	4.79e-39	1.83e-40
ex	4.79e-39	NaN	6.02e-20
in.	1.83e-40	6.02e-20	NaN

Table 12: wilconxon p-value of neighbourhood, solution quality with srz

With all of that the best setting might be the insert neighbourhood browse in first improvement with a for loop starting by the rz heuristic. According to the table 1, 2 and 9 it's true. Except that we can't said that population of a best improvement in a exchange neighbourhood (initialised by the rz heuristic) is different with a threshold of 0.05.

Now go on the executions time point of view.

The average computation time average overall in first improvement is 0.23 seconds against 1.04 seconds for the deepest descent. Beside the p-value of the wilcoxon test is $6.34e^{-58}$, so as expected first improvement is faster.

The average computation (over all other settings) time with the transpose neighbourhood is 0.018 seconds, 0.699 seconds with exchange and 1.208 with insert. And the p-value of the pairwise wilconxon test, table 13. So, again as expected, transpose is the fastest one, follow by exchange and insert is the slowest one, but still quite fast.

To conclude about the usage of a single neighbourhood for solution quality we can prefer the insert neighbourhood in first improvement or the exchange on in best improvement, both initialize in the rz heuristic (according to table 1, 2 and 9). But if we compare these two on execution time the solution with the insert neighbourhood is faster (according to table 6 and 10). Standard deviation of these two, on two point of view, are

	tr	ex	in
tr	NaN	1.83e-40	1.83e-40
ex	1.83e-40	NaN	4.10e-31
in.	1.83e-40	4.10e-31	NaN

Table 13: wilconxon p-value of neighbourhood, execution time

very low so it's doesn't really matter. And if we primary take care of execution time the transpose neighbourhood in first improvement initialize by random solution is the fastest one (according to table 5, 6 and wilcoxon pairwise test, execution time). But the solution quality is so bad, that I believe that the best choice is to use this solution but with the rz heuristic initialisation. Indeed this is the second fastest setting (so it's quite fast), but the solution quality is far way much better.

2.3 Test's average of VND

VND1 indicate the VND algorithm which use transpose, then exchange and finish with insert. VND2 is for the one which use transpose, then insert, then exchange. DD mean we use neighbourhoods in deepest descent. Same as previously the transpose neighbourhood is use in "dofor" mode.

size	VND1	VND1DD	VND2	VND2DD
50	3.5	3.6	4.2	4.3
100	4.4	4.7	5.4	5.4
all	3.9	4.1	4.8	4.8

Table 14: relative deviation average in % with random construction

size	VND1	VND1DD	VND2	VND2DD
50	2.2	2.3	2.4	2.6
100	3.0	3.0	3.2	3.2
all	2.6	2.7	2.8	2.9

Table 15: relative deviation average in % with rz construction

size	VND1	VND1DD	VND2	VND2DD
50	0.68	0.62	0.52	0.81
100	0.91	0.57	0.76	0.52
all	0.91	0.79	0.89	0.84

Table 16: standard deviation of the relative deviation with random construction

size	VND1	VND1DD	VND2	VND2DD
50	0.61	0.61	0.56	0.52
100	0.58	0.66	0.51	0.59
all	0.70	0.73	0.65	0.62

Table 17: standard deviation of the relative deviation with rz construction

size	VND1	VND1DD	VND2	VND2DD
50	0.28	0.29	0.44	0.44
100	4.38	3.69	7.96	6.34
all	2.30	1.96	4.14	3.34

Table 18: execution time average with random construction

size	VND1	VND1DD	VND2	VND2DD
50	0.09	0.11	0.07	0.10
100	1.05	1.25	0.97	1.34
all	0.56	0.67	0.51	0.71

Table 19: execution time average with rz construction

size	VND1	VND1DD	VND2	VND2DD
50	0.040	0.026	0.048	0.038
100	0.461	0.353	0.923	0.402
all	2.088	1.732	3.849	2.992

Table 20: execution time standard deviation with random construction

size	VND1	VND1DD	VND2	VND2DD
50	0.036	0.043	0.036	0.046
100	0.388	0.425	0.302	0.323
all	0.553	0.649	0.499	0.666

Table 21: execution time standard deviation with rz construction

	rndex	rndexDD	rndin	rndinDD	rzex	rzexDD	rzin	rzinDD
rndVND1	0.09	0.08	0.41	0.79	0.05	-0.12	-0.15	-0.17
rndVND1DD	0.04	0.03	0.35	0.71	0.005	-0.16	-0.19	-0.20
rndVND2	-0.07	-0.08	0.20	0.52	-0.10	-0.25	-0.28	-0.29
rndVND2DD	-0.07	-0.08	0.19	0.51	-0.11	-0.26	-0.29	-0.30
rzVND1	0.51	0.50	0.96	1.48	0.44	0.20	0.14	0.14
rzVND1DD	0.48	0.47	0.93	1.44	0.41	0.17	0.12	0.11
rzVND2	0.42	0.41	0.84	1.32	0.36	0.12	0.08	0.07
rzVND2DD	0.39	0.38	0.80	1.28	0.32	0.10	0.06	0.04

Table 22: relative improvement average of VND to single neighbourhood, in %

2.4 Statistical test of VND

We perform the wilcoxon test on score, in a pairwise way, to the obtained p-value of the null hypothesis (i.e . population are the same). A population isn't comparable to itself

so NaN is used in these cases. First, in the table 23, VND version are compared to each other. Secondly, in the table 24, VND version are compared to the usage of a single neighbourhood. Finally, after the comparison in term of solution quality, we compared VND version regard of the execution time, result are in the table 24, and 26 for the comparison with single neighbourhood usage.

	rndVND1	rndVND1DD	rndVND2	rndVND2DD	rzVND1	rzVND1DD	rzVND2	rzVND2DD
rndVND1	NaN	2.89e-02	4.01e-09	2.09e-08	3.51e-11	1.30e-10	1.07e-10	2.03e-10
rndVND1DD	2.89e-02	NaN	7.86e-08	5.85e-08	2.45e-11	3.01e-11	3.17e-11	4.53e-11
rndVND2	4.01e-09	7.86e-08	NaN	8.47e-01	2.45e-11	2.72e-11	2.58e-11	2.58e-11
rndVND2DD	2.09e-08	5.85e-08	8.47e-01	NaN	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rzVND1	3.51e-11	2.45e-11	2.45e-11	2.45e-11	NaN	2.49e-01	1.83e-02	3.91e-04
rzVND1DD	1.30e-10	3.01e-11	2.72e-11	2.45e-11	2.49e-01	NaN	1.33e-01	1.26e-02
rzVND2	1.07e-10	3.17e-11	2.58e-11	2.45e-11	1.83e-02	1.33e-01	NaN	3.59e-01
rzVND2DD	2.03e-10	4.53e-11	2.58e-11	2.45e-11	3.91e-04	1.26e-02	3.59e-01	NaN

Table 23: wilcoxon pairwise test, solution quality

	rndtr	rndtrDD	rndex	rndexDD	rndin	rndinDD	rztr	rztrDD	rzex	rzexDD	rzin	rzinDD
rndVND1	2.45e-11	2.45e-11	6.37e-04	1.39e-03	2.45e-11	2.45e-11	2.38e-01	2.50e-01	2.52e-06	6.37e-08	9.96e-10	6.93e-08
rndVND1DD	2.45e-11	2.45e-11	7.06e-02	8.32e-02	2.45e-11	2.45e-11	6.16e-01	4.90e-01	3.50e-09	1.75e-10	5.55e-11	7.14e-10
rndVND2	2.45e-11	2.45e-11	3.27e-06	2.72e-06	2.03e-10	2.45e-11	2.26e-06	1.67e-06	9.18e-11	3.51e-11	2.45e-11	3.89e-11
rndVND2DD	2.45e-11	2.45e-11	3.78e-06	2.26e-06	1.15e-09	2.58e-11	2.80e-07	2.03e-07	1.24e-10	2.45e-11	2.45e-11	2.45e-11
rzVND1	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	5.61e-08	7.53e-08	8.31e-06	2.00e-08
rzVND1DD	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	4.17e-07	7.22e-07	1.13e-04	4.34e-07
rzVND2	2.45e-11	2.45e-11	2.45e-11	2.72e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	7.47e-06	1.07e-03	1.28e-03	6.95e-07
rzVND2DD	2.45e-11	2.45e-11	2.58e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	1.49e-04	1.06e-02	1.76e-02	5.59e-05

Table 24: wilcoxon pairwise test, solution quality

	rndVND1	rndVND1DD	rndVND2	rndVND2DD	rzVND1	rzVND1DD	rzVND2	rzVND2DD
rndVND1	NaN	4.70e-06	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rndVND1DD	4.70e-06	NaN	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rndVND2	2.45e-11	2.45e-11	NaN	1.87e-06	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rndVND2DD	2.45e-11	2.45e-11	1.87e-06	NaN	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rzVND1	2.45e-11	2.45e-11	2.45e-11	2.45e-11	NaN	7.80e-03	4.76e-02	3.22e-04
rzVND1DD	2.45e-11	2.45e-11	2.45e-11	2.45e-11	7.80e-03	NaN	8.15e-04	7.03e-01
rzVND2	2.45e-11	2.45e-11	2.45e-11	2.45e-11	4.76e-02	8.15e-04	NaN	6.04e-06
rzVND2DD	2.45e-11	2.45e-11	2.45e-11	2.45e-11	3.22e-04	7.03e-01	6.04e-06	NaN

Table 25: wilcoxon pairwise test, execution time

First speak about the solution quality reach by the two order of neighbourhood used by VND. The average relative deviation overall with VND1 is 3.97% and it's 3.86% with VND2, beside the corrected p-value of the pairwise wilcoxon test is $1.42e^{-18}$. So we can conclude that VND1 perform better quality solution. Next we do the same with the pivoting rule (only on VND), the average of relative deviation of first improvement is 3.57% and in best improvement it's 3.66% with a p-value of 0.025 told us that, again, first improvement is better than a deepest descent. Finally the average of the relative deviation with a random construction is 4.47% against 2.77% with the usage of the rz heuristic with a p-value of $2.64e^{-40}$. So the best setting, in terms of quality solution,

	rndtr	rndtrDD	rndex	rndexDD	rndin	rndinDD	rztr	rztrDD	rzex	rzexDD	rzin	rzinDD
rndVND1	2.45e-11	2.45e-11	2.45e-11	1.84e-10	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rndVND1DD	2.45e-11	2.45e-11	2.45e-11	4.93e-03	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rndVND2	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	8.28e-05	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rndVND2DD	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	9.61e-01	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11	2.45e-11
rzVND1	2.45e-11	2.45e-11	1.59e-10	2.45e-11	1.45e-05	2.45e-11	2.45e-11	2.45e-11	1.07e-10	1.32e-03	1.75e-10	5.84e-01
rzVND1DD	2.45e-11	2.45e-11	3.01e-11	2.45e-11	1.01e-01	2.45e-11	2.45e-11	2.45e-11	2.58e-11	1.79e-07	1.18e-10	5.10e-02
rzVND2	2.45e-11	2.45e-11	9.96e-10	2.45e-11	2.83e-08	2.45e-11	2.45e-11	2.45e-11	5.55e-11	1.44e-01	3.31e-10	3.30e-02
rzVND2DD	2.45e-11	2.45e-11	3.70e-11	2.45e-11	9.01e-01	2.45e-11	2.45e-11	2.45e-11	2.45e-11	6.37e-08	2.58e-11	1.57e-01

Table 26: wilcoxon pairwise test, execution time

seems to be VND1 in first improvement initialized with the rz heuristic. If we look at table 14, 15 and 23 it's seems true.

Now speak about execution time. The average overall of VND1 is 1.37 second and it's 2.17 seconds for VND2, with a wilcoxon test's p-value of $7.91e^{-17}$. Secondly the average with a first improvement is 1.88 seconds and it's 1.67 seconds in best improvement, with a p-value of 0.059 so we can't say that executions time are different between the two pivoting rules. And finally the execution time's average starting from a random solution is 2.93 seconds and only 0.61 second from a construct one, with a p-value of $1.83e^{-40}$. So the fastest setting might be VND1 starting from a constructed solution (but the pivoting rule doesn't matter). According to table 18, 19 and 25 the situation is far way more fuzzy : rzVND1, rzVND1DD and rzVND2 may have the same execution time.

Regarding of these two point of view VND1 in first improvement initialized with the rz heuristic is one of the best setting for VND, so we can say that it's might be the optimal setting (with our possibility) of VND for this problem. But compared to our preferred setting for the usage of a single neighbourhood (rzexDD and rzin) the relative improvement is very low : according to table 22 it's bellow 0.2% (according to wilcoxon test, table 24 results are different but we just go to an average relative deviation of 3% to 2.6) . But a descent in these neighbour alone is faster, 0.44 and 0.24 second against 0.56 second in average (with p-value bellow 0.05, see table 26) . I believe it's related to the usage of a for loop in a descent, with can't be use for exchange and insert in the VND algorithm. So we don't have to invest that much time, but don't get a big improvement of the solution quality.

3 Conclusion

Depending to the time budget we have some possibility. If time is the most important criterion rztr must be chose. And after order by the require time we can use rzin and rzVND1. We must notice that all of them are quite fast and give good solutions. But we doesn't have long and very efficient procedure yet. Finally we need to highlight that of these procedure are very stable (standard deviations are very low) regarding to solution quality or execution time.