ANTHONY DEBRUYN

Heuristic Optimisation

Implementation Exercise 1

Iterative improvement algorithms for the PFSP

Dr. F. Mascia, Dr. T. Stützle INFO-H-413

1 Introduction

The goal of this assignment was to...

2 Implementation

The code was implemented in Java, with an object oriented architecture. A *Main* class receives the arguments and defines the modes to use with the algorithm from these. The *IterativeImprovement* class is the main class of the algorithm. It is constructed in the *Main* class with the modes received earlier.

In this class, there is a loop that goes on until the new solution is the same as the previous one (same pointer). The main class of the algorithm uses 3 other types of class, separated in corresponding packages:

Initial Solution Generators: These classes implement all an interface to behave the same way. They are used to generate a first solution for the algorithm. Each class returns another type of solution.

Neighbourhood Generators: These generate the neighbourhood for each permutation/solution.

Pivoting Managers: These manage the way the neighbourhoods are used, and need an instance of neighbourhood generator. For instance, we have one pivoting manager for the first pivoting mode that returns the first better solution it gets from the generator. The managers return the same solution if no improvement was possible.

There are also a class that represent a permutation/solution, and a class for the problem instances.

2.1 Addons for the exercise 1.2

The second part of the implementation took much less time to end, thanks to the object oriented programming. Only 2 new classes for the 2 new neighbourhood generators were needed. New constants were added in the algorithm main class to enable the use of the new modes. In the main class, new arguments cases were added to take them into account, and a new method was added for the batch process of the instances with the VND approach (same as the other one, but with other modes).

Please consult the javadoc for more information on the internal behaviour.

3 Code Use

To launch the algorithm with a desired mode, launch the jar with the following arguments:

Initial Solution Mode:

- $--$ *random_init* : To generate an initial random solution. See the doc for the info on the corresponding algorithm.
- $--$ *slack_init* : To generate an initial solution with the SLACK heuristic.

Neighbourhood Generation Mode:

- $--$ *transpose* : Generate neighbourhood with a transpose method.
- $--$ *exchange* : Exchange method.
- $--$ *insert* : Insert method.
- $--$ *vnd_tei* : Variable neighbourhood descent, transpose→exchange→insert.
- $--$ *vnd_tie* : Variable neighbourhood descent, transpose→insert→exchange.

Pivoting Mode:

- $--$ *first* : Take the first solution from the neighbourhood with a better cost.

$--best$: Take the best solution from the neighbourhood with a better cost. If there are more than one solutions, take the first met.

You can also choose to launch the algorithm on a single file or on a folder containing several instance files:

$--file\ \{filepath\}$: Launch on a single file. The results are only printed on the console.

$--batch\ \{directorypath\}$: Launch on a folder. The results are written in files, each file name containing the mode of the algorithm. All the algorithm combinations are tested on all the instances. The intermediate results are printed on the console like for the single file.

$--batch\_vnd\ \{directorypath\}$: Same as previous one, but test on all the VND algorithms.

4 Exercise 1.1

4.1 The Results

First of all, all the results are in the files with the different modes as names, like "–random–exchange–best". The first column is the name of the instance, the 2nd is the relative percentage deviation, the third is the time in ms to compute the final solution, the fourth is the cost of this solution, and the fifth is the cost of the best solution known.

We list here the average percentage deviation from the best solutions for each algorithm tested, along with the average computation time:

Algorithm	APD	ACT(ms)
–random–exchange–best	26	1362
–random–exchange–first	10	5744
–random–insert–best	16	2909
–random–insert–first	6	12869
–random–transpose–best	354	28
–random–transpose–first	340	37
–slack–exchange–best	32	1244
–slack–exchange–first	11	3944
–slack–insert–best	18	2523
–slack–insert–first	6	8661
–slack–transpose–best	243	43
–slack–transpose–first	240	47

Figure 1: The APD and ACT for each algorithm.

We see that the transpose mode for the neighbourhood generation is the worst, and by far, but is the less costly in time. The insert mode is the best when it comes to the quality of the solution, but is the worst in time cost. The remaining exchange mode is between the other ones, half less time than the insert mode, and a quality of solution a bit lower than the insert mode, but very far from the quality of the transpose mode.

The reason behind the difference between these 3 modes becomes clear when we consider the size of the respective neighbourhood. If N is the number of jobs and p is the current permutation, we get these sizes:

$$|N_{transpose}(p)| = N - 1$$

$$|N_{exchange}(p)| = \sum_{i=0}^{N-2} N - i - 1 = (N-1)^2 - \frac{(N-1)(N-2)}{2} = \frac{N(N-1)}{2}$$

$$|N_{insert}(p)| = (N-1)^2$$

The size of the insert neighbourhood is the biggest, and thus helps not getting stuck easily in a local maximum. The bigger the size, the smaller the probability for a certain permutation to be a local maximum.

We observe that the SLACK heuristic makes the time cost decrease everywhere, except for the transpose mode. As an attempt to explain this, we could say that the size of the neighbourhood, being a lot smaller than the other mode (linear in the number of jobs), has a much lower impact on the time cost. The real impact comes from the heuristic at the beginning.

The APD is always lower for the "first" mode than the "best" mode. Would this be explained by the fact that more diverse solutions are found with the first mode ? Indeed, we see that the time consumed for the first mode is much superior than the time used for the best mode. And if we look at the number of intermediate solutions, the number is much higher. We could say that with the best mode, we explore more locally than with the first mode, and get stuck quickly in a local maximum.

4.2 Difference between the solutions

We used the Student t-test to determine whether there there is a statistically significant difference between the solutions generated by the different perturbative local search algorithms. The obtained p-values for the Student t-test are shown in figure 2. The algorithms corresponding to the numbers are the ones from figure 1, in the same order (alphabetically sorted).

The Student t-test can be used to test statistically the mean equality null hypothesis. The p-value corresponds to the probability that the null hypothesis is incorrectly rejected. If this value is below the significance level, the null hypothesis is rejected and thus the means are very different. If the value is above the significance level, the hypothesis is incorrectly rejected, thus accepted, and the means are approximately equal. The higher the value, the closer the means.

Here is the R code used to write the p-values to a file. The results matrix was created by taking the average percentage deviation of each algorithm and each instance (matrix of dimension 60x12).

```
a <- read.table("R-avRelPer--random--exchange--best.dat")$V1
...
l <- read.table("R-avRelPer--slack--transpose--first.dat")$V1

results <- c(a,b,c,d,e,f,g,h,i,j,k,l)
results <- array(results, dim=c(60,12))

test = numeric(length = 66)
test2 = numeric(length = 66)

x = 1
for (i in 1:11) {
  for (j in (i+1):12) {
    test[x] <- t.test (results[,i], results[,j], paired=T)$p.value
    test2[x] <- wilcox.test (results[,i], results[,j], paired=T)$p.value
    x = x + 1
  }
}

write(test, file = "p values st test", ncolumns = 1)
write(test2, file = "wilcox p-values", ncolumns = 1)
```

We take a significance level of 0.05 ($\alpha = 0.05$).

	1	2	3	4	5	6	7	8	9	10	11	12
1		2.325677e-10	0.0001764137	1.492921e-10	2.324708e-07	1.397426e-08	0.009320388	1.814296e-10	0.0009812694	1.651045e-12	1.002514e-09	3.956704e-09
2			1.380619e-07	5.676628e-05	1.667657e-07	1.022813e-08	1.119052e-10	0.2236329	1.656816e-12	1.904083e-08	6.981541e-10	2.61601e-09
3				2.616446e-12	2.784541e-07	1.86176e-08	4.364759e-06	6.281086e-05	0.2084699	9.391734e-12	2.116133e-09	7.112663e-09
4					1.560699e-07	1.017382e-08	5.164549e-10	4.937766e-05	6.505518e-17	0.3771135	7.162175e-10	2.629253e-09
5						0.3131815	3.198631e-07	1.624651e-07	2.896557e-07	1.378438e-07	0.0002024202	5.743684e-05
6							1.81788e-08	1.005804e-08	2.054526e-08	8.50651e-09	1.793673e-05	7.765756e-07
7								4.646304e-11	3.193926e-05	1.330835e-11	1.541866e-09	5.902744e-09
8									6.202973e-11	1.221381e-07	6.366285e-10	2.46713e-09
9										8.756858e-19	1.987018e-09	7.336251e-09
10											5.066557e-10	1.945128e-09
11												0.4696791
12												

Figure 2: P-values for each combination of algorithms (Student t-test). Paired test, with each possible pair of algorithms.

	1	2	3	4	5	6	7	8	9	10	11	12
1		1.6479e-11	6.566262e-07	1.650747e-11	1.66871e-11	1.668972e-11	0.001887301	3.558256e-11	3.880961e-05	1.649194e-11	1.66871e-11	1.66688e-11
2			1.401733e-07	5.791151e-06	1.670281e-11	1.670805e-11	1.646866e-11	0.7562894	7.25813e-09	1.681026e-08	1.668972e-11	1.667926e-11
3				1.218116e-10	1.669234e-11	1.670281e-11	2.685764e-09	9.410355e-06	0.07339366	1.376776e-09	1.669757e-11	1.667664e-11
4					1.669757e-11	1.667664e-11	1.648417e-11	4.229889e-07	2.965186e-11	0.861429	1.670281e-11	1.668449e-11
5						0.7742204	1.669757e-11	1.669495e-11	1.670019e-11	1.670543e-11	4.520244e-11	3.454208e-11
6							1.66871e-11	1.669495e-11	1.669495e-11	1.669757e-11	1.66688e-11	2.449249e-11
7								1.642735e-11	5.420053e-09	1.657235e-11	1.670281e-11	1.669757e-11
8									2.515346e-08	3.401544e-08	1.670019e-11	1.670019e-11
9										5.411351e-11	1.66871e-11	1.667403e-11
10											1.670019e-11	1.669495e-11
11												0.0002718787
12												

Figure 3: P-values for each combination of algorithms (Wilcoxon test). Paired test, with each possible pair of algorithms.

The Wilcoxon test does not assume that the population is normally distributed. We see on figure 3 that the p-values give the same idea of the difference between the solutions, except for the combination 11x12. We can see that the hypothesis is rejected in the Wilcoxon test but not in the Student t-test.

We can see on the figures that a lot of values are below the threshold α. We can thus conclude that most of the algorithms generate solutions of significantly different quality. We can check it in figure 1.

5 Exercise 1.2

5.1 The results

Algorithm	APD	ACT(ms)
–random–vnd_tei–first	8	2270
–random–vnd_tie–first	9	4476
–slack–vnd_tei–first	12	1675
–slack–vnd_tie–first	9	3058

Figure 4: The APD and ACT for each algorithm (VND).

We can see here (figure 4) that the means are quite close to each other. The 3rd algorithm is the worst, but not by far. Putting the insert neighbourhood in second place seems to increase the computation time. If we take the same number of missing better neighbour in the insert and exchange neighbourhood (if we look at the entire neighbourhood with no luck x times), then more time is elapsed for the insert mode than the exchange mode (since the neighbourhood size is bigger). So more time is elapsed before entering the third kind of neighbourhood.

5.2 Difference between the solutions

	1	2	3	4
1		0.2743695	0.00536978	0.1269593
2			0.01019433	0.9354282
3				0.08984315
4				

Figure 5: P-values for each combination of algorithms (Student t-test). Paired test, with each possible pair of algorithms.

	1	2	3	4
1		0.06738478	0.001592057	0.006999513
2			0.005924327	0.9682321
3				0.1793561
4				

Figure 6: P-values for each combination of algorithms (Wilcoxon test). Paired test, with each possible pair of algorithms.

If we take a significance level $\alpha = 0.05$, we see that the values are what we expected from looking at figure 4. The figure 5 and 6 are almost similar in terms of conclusions. The only difference is that the 1x4 combination is below the threshold for the Wilcoxon Test, and above for the Student t-test.

We can conclude that the quality of the two first algorithms (the 2 different VND approach with random initial solution) looks quite similar. Same goes for the 2 algorithms with the SLACK initial solution

heuristic (less similar if we look at the Student t-test, more if we look at the Wilcoxon Test).

A difference in quality between the two first solution approach is more present for the TEI order of neighbourhood, than for the TIE where the quality is almost the same.