Heuristic Optimization : Implementation exercise 1

Nikita Marchant (nimarcha@vub.ac.be)

May 17, 2017

1 Implementation

1.1 Usage

This project was implemented in C++ with the latest C++17 additions for a better ease when programming. The compilation uses a`Makefile`. You can compile it by running the command `make` in the working directory. It will produce an executable named `fssp`. When ran with the `-h` option, it will show how to invocate it.

When properly invoked, it will output only one line, containing only one integer : the score of the solution it found.

1.2 Organization

The main class `Instance` is inspired by the `PfspInstance` class given with the assignment. Notable changes are that indexes now begin at 0 to follow the convention of C++ and the move a the parsing into the constructor.

The rest of the code is split into 4 other files :

- `main.cpp` : the file responsible for calling th rest of the code and parsing the command line arguments.
- `initialization.cpp` containing the initialization functions
- `neighborhood.cpp` containing the neighborhood generation functions
- `pivoting.cpp` containing the pivoting functions.

Note: another change from the original code given with the assignment is that the random seed is fixed to be able to reproduce experiments easy. If desired, this behavior can be overridden with a command line parameter.

2 Experiments

All the results shown here were computed and extracted from an IPython notebook given along with the source code.[1]

[1] The notebook is called analysis and can be found both in the .html and .ipynb version

2.1 Exercise 1

At first, we compute the relative deviation from the best known solution :

Initialization	Pivoting	Neighborhood	Relative deviation
random	best-fit	exchange	4.526442
		insert	9.536986
		transpose	37.422758
	first-fit	exchange	1.840076
		insert	6.995218
		transpose	36.153227
srz	best-fit	exchange	3.109973
		insert	3.161487
		transpose	4.144504
	first-fit	exchange	2.988648
		insert	2.935848
		transpose	4.130356

Then the mean computation time:

Initialization	Pivoting	Neighborhood	#Jobs	Mean time
random	best-fit	exchange	50	0.206074
			100	3.410456
		insert	50	0.250388
			100	3.855229
		transpose	50	0.017638
			100	0.080919
	first-fit	exchange	50	0.645275
			100	15.081069
		insert	50	0.830786
			100	19.725573
		transpose	50	0.022015
			100	0.083597
srz	best-fit	exchange	50	0.057918
			100	0.771066
		insert	50	0.057227
			100	0.751073
		transpose	50	0.015344
			100	0.042693
	first-fit	exchange	50	0.053892
			100	1.070865
		insert	50	0.071624
			100	1.092688
		transpose	50	0.016870
			100	0.040687

After those two overviews, we can analyze in depth if there is a statistically significant difference between some methods, both in computation time and score.

For that we use the Student t-test with a significance level of 0.05 or 5%.

When comparing the random initialization versus the simplified RZ heuristic, we see that none of the variants yield a significant improvement in the score over another.

The next step is to compare the pivoting methods : in this case, except when using the SRZ and exchange or transpose, the “first-improvement” yields a significantly better score. When analyzing the computation time, we see another tendency : when using the transpose method, the times are similar but otherwise the best-improvement wins.

The last comparison to do is the neighborhoods: here, the result is more complex. When using the SRZ, using exchange of insert does not increase the score. Otherwise, exchange is always better than the others and insert is superior to transpose. When analyzing the computation time, we see that the transpose method is faster all the time and that exchange is faster than insert except with SRZ where there is no difference.

2.2 Exercise 2

At first, we compute the relative deviation from the best known solution :

Neighborhood	Relative deviation
Transpose exchange insert	2.550408
Transpose insert exchange	2.745331

Then the mean computation time:

Neighborhood	#Jobs	Time
Transpose exchange insert	50	0.073599
	100	1.154808
Transpose insert exchange	50	0.064993
	100	0.728432

We can compute the improvement between a single neighborhood and the VND :
Transpose, exchange, insert vs exchange: 122%
Transpose, exchange, insert vs insert: 221%
Transpose, insert, exchange vs exchange: 113%
Transpose, insert, exchange vs insert: 206%

When comparing the scores of both VND neighborhoods with a student test, we see that transpose, exchange, insert has a better score than the other.

We thus can conclude than using VND with the transpose, exchange, insert is preferable over the other methods.