TSP implementation - How-To

Requirements and Dependencies

- Python 3.5
- Numpy
- matplotlib
- scipy (for the plot generation and statistical analysis scripts)

Usage

Configuration

The EA configuration parameters are defined in JSON files (see `default_args.json` for an example).

Here are the key/value pairs that can be specified in the JSON config file:

- **datafile** - The path to the data file, e.g. `data/TSP_Uruguay_734.txt`. Type: string
- **popsize** - Population size. Type: integer
- **initialize_method** - Use random or k-means clustering for initialisation. Can be set to `random` or `kmeans`. Type: string
- **mp_size** - Mating pool size. Must be a multiple of 2. Type: integer.
- **tournament_size** - Tournament size for tournament parent selection. Type: integer
- **recombination** - The recombination/crossover operator used. Can be set to `best_order` or `cut_crossfill`. Type string
- **mutation** - The mutation operator used. Can be set to `cyclic`, `inversion`, `insertion`, `scramble`, `two_opt`, or `permutation_swap`. Type: string
- **crossover_rate** - Crossover rate, a value between 0.0 and 1.0. Type: float
- **mutation_rate** - Mutation rate, a value between 0.0 and 1.0. Type: float
- **generations** - The number of generations to run the EA for. Type: integer
- **box_cutting_points_n** - The number of crossover points for best order crossover. This parameter **must always be larger than or equal to 5** and **must always be less than the chromosome length minus 1**. Type: integer
- **kca_k** - A value between 0.0 and 1.0, which acts as a proportion of the chromosome length to define the number of clusters for the k-means algorithm. Type: float
- **kca_iterations** - The number of iterations to refine the k-means clusters. Type: integer

Note: Make sure you use double quotes (`"`) when editing the JSON file's key/value pairs. Also, make sure there is no trailing comma at the end of the JSON file. To check whether your config is valid JSON, you may run it through <https://jsonformatter.curiousconcept.com/>, which will report any JSON syntax errors (if present) in your config file.

Note: The **testing/** directory also contains example config files that you can use to run the program.

Running the algorithm

Here are the arguments that the `main.py` script takes:

- **--args-file** or **-a:** Used to specify a custom JSON config file. Default is `default_args.json`. **Example** - `python3 main.py --args-file custom_args.json`

- **--test-runs** or **-t:** Number of times to run the algorithm, usually used in conjunction with the **--export** argument. Default is `1`. **Example** - `python3 main.py --test-runs 10`
- **--export** or **-e:** Export the resulting best fitness values to a CSV file. Usually used in conjunction with the **--test-runs** argument to export the data for multiple runs of the algorithm to a CSV file. This option is how we generated the data for statistical comparisons and visualisations used in our report. **Example** - `python3 main.py --export`
- **--visualize** or **-v:** Toggle the real-time visualisation. **Note** that you cannot use this option when you're using the **--test-runs** option. **Example** - `python3 main.py --visualize`
- **--debug** or **-d:** Print performance and debugging output. **Example** - `python3 main.py --debug`
- **--help** or **-h**: Print the help text. **Example** - `python3 main.py --help`

Here are some examples of a few different ways you can run the script:

The following command uses a custom configuration file, runs the algorithm for 10 runs, and exports the data to CSV files:

```
python3 main.py --args-file custom_args.json --test-runs 10 --export
```

The following command uses a custom configuration file, turns on the real-time visualisation, and also prints the performance and debugging output:

```
python3 main.py --args-file custom_args.json --visualize --debug
```

Data Analysis

We have also provided all the data used to conduct the statistical tests and generate the visualisations that were used in the report. All the relevant files are found under `testing/`:

- **testing/mutation/** contains all the data files, the plots and the script related to the different mutation parameters.
- **testing/recombination/** contains all the data files, the plots and the script related to the different recombination/crossover operators.
- **testing/parameters/** contains all the data files, the plots and the script related to the different tuning parameters for the evolutionary algorithm.
- **testing/final/** contains all the data files, the config files, the plots and the script related to the final performance and fitness measures at the end of our report.

For example, in order to conduct the statistical tests and generate the visualisations for all the mutation parameters:

1. Go to the **testing/mutation/** directory.
2. Run: `python3 analyze.py`

Running the above command should print the results of the statistical tests for the various mutation parameters and should generate image files for the visualisations in the same directory (**testing/mutation/**). You can perform the same steps for the **testing/recombination/**, **testing/final/** and **testing/parameters/** directories too.

Fin □