HERMES

December 10, 2018

1 H.E.R.M.E.S.

2 Heuristic Enabled Rapid Modular Evolutionary Search

2.1 Set-up

2.1.1 Notebook Set-up

```
In [1]: # Setup base directory
        import os
        import os.path
        import sys
        sys.path.append(os.path.join(os.getcwd(), '../..'))

        import json
```

2.1.2 Imports

```
In [2]: from src.EACore.EAVarHelper import EAVarHelper
        from src.EACore.EARunner import EARunner
        from src.Setups.TSP.FileLoader import LoadHelper
        from src.Setups.TSP.EAFactory import EAFactory
        from src.Setups.TSP.PopulationInitialization import PopulationInitializationGenerator
        from src.Setups.TSP.FitnessEvaluator import FitnessHelperGenerator
        from src.Setups.TSP.Inputs.Optimums import get_best_path
```

2.1.3 Configuration

Global Variables

- `maximization`: Where the problem is a maximization or a minimization problem
- `test_all`: Whether to use the methods provided, or to iteratively test every combination of methods
- `methods`: Determines which method will be used for each part of the algorithm
- `use_db`: Whether or not to save results to an SQLite3 database
- `db_name`: Filename for SQLite3 database (if use_db is true)
- `print_stats`: Whether or not to print results to the console as they are produced
- `generation_limit`: The number of generations that the algorithm runs for
- `report_rate`: The number of generations to run between displaying and outputting stats

- `runs`: The number of times to run the algorithm
- `data_set`: The input data to use (0: Sahara, 1: Uruguay, 2: Canada)
- `data_type`: The data structure to use for storing individuals (0: Lists, 1: Numpy Arrays, 2: C Arrays)

Modular Function Definitions The `EACore` is module which contains the class `EARunner`. This is a general shell that is designed to handle many different problems, and is not limited to the Travelling Salesperson Problem. To solve a different problem, you need to provide a new population initialization method, a new fitness evaluation method, and a data set. Other methods are problem agnostic.

The methods used by the algorithm are determined by the `methods` list provided in the configuration file. Here is a table of the functions you can choose from.

Fitness Evaluation

0: Euclidean

Population Initialization

0: Random
1: Cluster
2: Euler

Parent Selection

0: MPS
1: Tournament Selection
2: Random Selection

Recombination

0: Order Crossover
1: PMX Crossover

Mutation

0: Swap
1: Insert
2: Inversion
3: Shift

Survivor Selection

0: Mu + Lambda
1: Replacement

Population Management

0: None
1: Metalleurgic Annealing
2: Entropic Stabilizing
3: Ouroboric Culling
4: Genetic Engineering

```
In [3]: config_json = """{
            "maximization": false,
            "test_all": false,
            "methods": [0, 2, 1, 1, 2, 0, 4],
            "use_db": false,
            "print_stats": true,
            "db_name": "stats.db",
            "generation_limit": 5000,
```

```
        "report_rate": 1000,
        "runs": 1,
        "data_set": 2,
        "data_type": 2
    }"""

    config = json.loads(config_json)
```

2.1.4 Create EA Runner

```
In [4]: factory = EAFactory(config["data_set"], config["maximization"])
        ea = factory.make_ea_runner(config["data_type"], config["methods"])
```

With all imports and object initializations out of the way, we can run the EA with a single function call: `run()`

`run()` takes in a few arguments, the only non-optional one being the `generation_limit`. The other arguments are for hw often a generation summary should be printed, the best fitness found, whether that fitness is the true optimum, and a mute boolean, for if the EA is being multi-threaded (so the outputs don't clash and clutter the terminal).

`run()` returns the best fitnees found, the idividuals with that fitness, the generation it ended on (in case of early convergence), a history of best individuals over the generations, and a tuple with the results of clocking the functions.

```
In [5]: ea.run(config["generation_limit"], print_stats=config["print_stats"], report_rate=conf
        pass
```

```
Population initialization:   Euler              Parent selection:          Tourney
Recombination Method:        PMX Crossover      Mutation Method:           Inversion
Survivor selection:          Mu + Lambda        Management Method:         Engineering
Loaded: TSP_Uruguay_MST.txt
Generation: 1000
  Best fitness: 103275.95994813472
  Avg. fitness: 103275.95994813464
  Copies of Best: 60
Generation: 2000
  Best fitness: 102401.07556411297
  Avg. fitness: 102401.07556411282
  Copies of Best: 60
Generation: 3000
  Best fitness: 102049.60807013858
  Avg. fitness: 102049.6080701386
  Copies of Best: 60
Generation: 4000
  Best fitness: 101304.7742155538
  Avg. fitness: 101304.77421555386
  Copies of Best: 60
Generation: 5000
  Best fitness: 100580.63304406664
  Avg. fitness: 100580.63304406677
```

```
  Copies of Best: 60
Best solution fitness: 100580.63304406664
Number of optimal solutions:  60 / 60
Best solution indexes: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2
Best solution path: array('i', [407, 409, 415, 421, 580, 561, 541, 555, 556, 559, 570, 577, 59
PITime = 2.32,  PSMTime = 1.24,  RMTime = 4.48,  MMTime = 15.84,  SSMTime = 0.13,  PMMTime = 3
Func time: 388.41     Total time: 390.77
--- 390.76635584700125 seconds ---
```

2.1.5 Configure Parameters

Parameters can be changed between runs. Parameters are stored in `ea.vars`. Here I will demonstrate increasing the population threshold (the number of copies of the best individual that may exist before population management begins replacing them), and the mutation rate to increase diversity.

```
In [6]: ea.vars.set_population_threshold_by_percent(0.70)
        ea.vars.mutation_rate = 0.60
        ea.run(config["generation_limit"], print_stats=config["print_stats"], report_rate=conf
        pass
```

```
Population initialization:   Euler                   Parent selection:          Tourney
Recombination Method:        PMX Crossover           Mutation Method:           Inversion
Survivor selection:          Mu + Lambda             Management Method:         Engineering
Generation: 1000
  Best fitness: 101603.29159115907
  Avg. fitness: 101603.29159115907
  Copies of Best: 60
Generation: 2000
  Best fitness: 100374.69963633735
  Avg. fitness: 100374.69963633735
  Copies of Best: 60
Generation: 3000
  Best fitness: 97935.8031847623
  Avg. fitness: 97935.80318476235
  Copies of Best: 60
Generation: 4000
  Best fitness: 97312.12319538763
  Avg. fitness: 97312.1231953875
  Copies of Best: 60
Generation: 5000
  Best fitness: 95885.9852284485
  Avg. fitness: 95885.98522844845
  Copies of Best: 60
Best solution fitness: 95885.9852284485
Number of optimal solutions:  60 / 60
Best solution indexes: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2
```

```
Best solution path: array('i', [457, 561, 541, 555, 556, 513, 488, 491, 487, 502, 510, 485, 50
PITime = 2.13,  PSMTime = 1.31,  RMTime = 6.12,  MMTime = 17.33,  SSMTime = 0.15,  PMMTime = 3
Func time: 331.46     Total time: 333.63
--- 333.62956233099976 seconds ---
```