Evolutionary Computation - lab assignment 10

Szymon Bujowski, 148050, source: https://github.com/bujowskis/put-evolutionary-computation

Problem description

The goal of this assignment is to improve the algorithms that were implemented earlier. Either a modification of previously implemented algorithms or a completely new method should be proposed to improve the average score and the best result obtained in previous experiments (on all instances) at the same time. The new algorithm needs to be described in detail. For example, mechanisms such as:

- generating better starting solutions,
- further improvements of the efficiency of local searches.
- global memory of moves evaluations,
- new mechanisms of candidate moves;
- other operators of neighborhood, perturbation, recombination,
- simultaneous use of different operators,
- use of machine learning mechanisms,
- ...

The proposed method should be precisely described and its algorithm should be provided in pseudocode.

Report – analogous as before

Implemented algorithms and pseudocodes

Notes

- 50% was the required number of nodes in a solution, and "required number of nodes" will be referred to this way in the pseudocode
- Pseudocode is simplified to the best known configuration of Local Search - that is: Steepest LS, Random starting solution, two edges exchange intra moves
- LS with candidate moves was used
- Tried a "long-running" version (run Cycle Stripper and save best result found so far until MSLS running time timeout), however it's not fast enough to do meaningful number of main loop runs

Cycle Stripper

```
1. Generate initial cycle containing all nodes - randomize nodes order
and apply LS on the cycle
2. While number of nodes in cycle is higher than required number of
nodes, at each iteration:
2.1. Calculate change in objective function resulting from removing a
given node within cycle and connecting its neighbors directly
2.1.Ad.1.
  Assume:
    “n1” and “n2” are neighbors of node considered for removal “r”,
    “E(x,y)” is the edge length between nodes “x” and “y”,
    “C(x)” is additional cost of node “x”
  Then the formula for removal change is:
    $E(n1, n2) - [E(n1, r) + E(r, n2) + C(r)]$
  The lower the value, the better
2.2. Choose the node with the lowest removal change. Remove it from
the cycle
2.3. (Additionally) if LS interval is provided and equal to number of
iterations since last LS, apply LS on the cycle
3. (Additionally) if requested, apply LS on the resulting cycle
4. Return the cycle
```

Results

Note: All best solutions were checked with the solution checker

Statistics - comparison to less complex methods

Method	**TSPA** avg (min - max)	**TSPB** avg (min - max)
Random	265351 (236147 - 290929)	213808 (193230 - 236483)
NN at end	85109 (83182 - 89433)	54390 (52319 - 59030)
NN at any	73173 (71179 - 75450)	45870 (44417 - 53438)
Greedy cycle	72590 (71488 - 74350)	51389 (48765 - 57262)
Greedy 2-regret	99830 (85699 - 107981)	64565 (60110 - 70419)
Greedy 2-regret weighted objective	74037 (72590 - 76005)	56039 (52645 - 61863)
Cycle Stripper - no LS	73740 (71105 - 76394)	48381 (45793 - 51649)
Cycle Stripper - LS at end	72571 (70376 - 75279)	46589 (44418 - 49867)
Cycle Stripper - LS every 1 iter, and at end	71904 (69587 - 75833)	46004 (43797 - 49376)
Cycle Stripper - LS every 3 iter, and at end	72051 (70010 - 74871)	46048 (44055 - 48985)
Cycle Stripper - LS every 5 iter, and at end	72082 (70154 - 75120)	45927 (43765 - 48091)
Cycle Stripper - LS every 10 iter, and at end	72113 (70118 - 74718)	46011 (43569 - 49995)

Statistics - comparison to more complex methods

Method	**TSPA** avg (min - max)	**TSPB** avg (min - max)
Local Search (steepest, random, two_edges)	74222 (71725 - 78624)	48662 (46269 - 52049)
Local Search (steepest, greedy, two_edges)	71998 (70110 - 74270)	45171 (44140 - 51146)
Multiple Start Local Search (MSLS)	72324 (71458 - 72825)	46801 (45825 - 47480)
Iterated Local Search (ILS)	69498 (69259 - 70321)	43858 (43568 - 44484)
Large-Scale Neighborhood Search (LSNS) - no LS	69935 (69230 - 71274)	44437 (44984 - 46112)
Large-Scale Neighborhood Search (LSNS) - with LS	69774 (69230 - 70258)	44373 (43550 - 45506)
Evolutionary Algorithm heuristic, no LS	70631 (70082 - 70996)	44453 (43759 - 45383)
Evolutionary Algorithm heuristic, with LS	70169 (69424 - 70688)	44050 (43555 - 44822)
Evolutionary Algorithm random, no LS	73164 (72330 - 74375)	47593 (46332 - 48515)
Evolutionary Algorithm random, with LS	70913 (70041 - 71706)	45098 (44274 - 45669)
Cycle Stripper - no LS	73740 (71105 - 76394)	48381 (45793 - 51649)
Cycle Stripper - LS at end	72571 (70376 - 75279)	46589 (44418 - 49867)
Cycle Stripper - LS every 1 iter, and at end	71904 (69587 - 75833)	46004 (43797 - 49376)
Cycle Stripper - LS every 3 iter, and at end	72051 (70010 - 74871)	46048 (44055 - 48985)
Cycle Stripper - LS every 5 iter, and at end	72082 (70154 - 75120)	45927 (43765 - 48091)
Cycle Stripper - LS every 10 iter, and at end	72113 (70118 - 74718)	46011 (43569 - 49995)

Running times (s) - comparison to less complex methods

Method	**TSPA** avg (min - max)	**TSPB** avg (min - max)
Random	0.00008 (0 - 0.00133)	0.00008 (0 - 0.00150)
NN at end	0.00237 (0.001 - 0.00366)	0.00240 (0.00145 - 0.00376)
NN at any	0.15767 (0.14758 - 0.1954)	0.16658 (0.15078 - 0.53)
Greedy cycle	0.1476 (0.14421 - 0.15667)	0.16 (0.14757 - 0.31)
Greedy 2-regret	0.54 (0.42 - 1.25)	0.44 (0.42 - 0.48)
Greedy 2-regret weighted objective	0.44 (0.43 - 0.48)	0.44 (0.43 - 0.53)
Cycle Stripper - no LS	0.60 (0.49 - 0.99)	0.92 (0.63 - 1.24)
Cycle Stripper - LS at end	0.61 (0.50 - 0.77)	0.62 (0.52 - 0.87)
Cycle Stripper - LS every 1 iter, and at end	2.53 (2.42 - 2.75)	2.53 (2.42 - 2.75)
Cycle Stripper - LS every 3 iter, and at end	1.32 (1.19 - 1.46)	1.32 (1.19 - 1.46)
Cycle Stripper - LS every 5 iter, and at end	1.04 (0.94 - 1.17)	1.04 (0.94 - 1.17)
Cycle Stripper - LS every 10 iter, and at end	0.84 (0.74 - 0.99)	0.84 (0.74 - 0.99)

Running times (s) - comparison to more complex methods

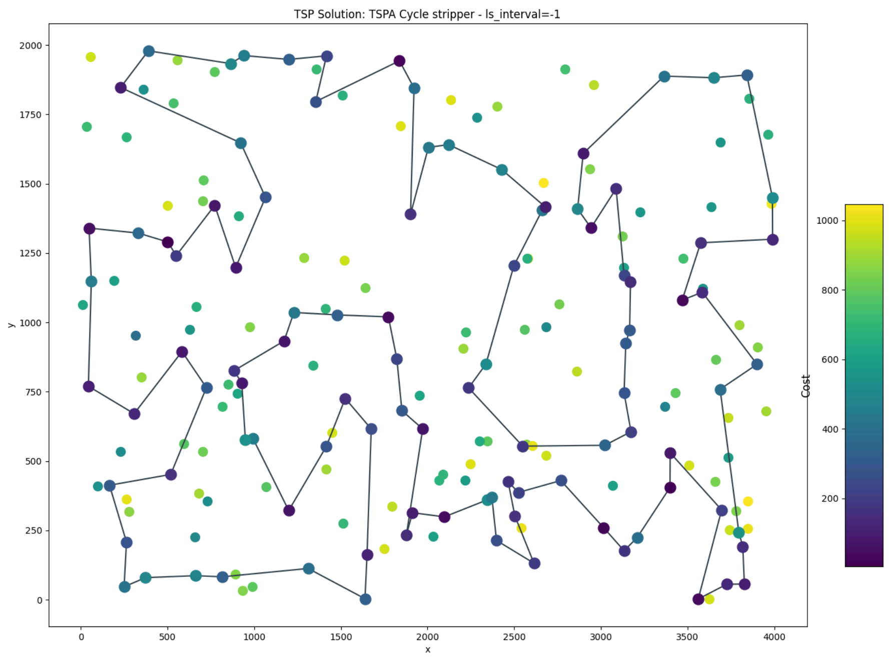
Method	**TSPA** avg (min - max)	**TSPB** avg (min - max)
Local Search (steepest, random, two_edges)	0.22 (0.18 - 0.42)	0.22 (0.17 - 0.32)
Local Search (steepest, greedy, two_edges)	0.39 (0.27 - 0.67)	0.32 (0.26 - 0.59)
Multiple Start Local Search (MSLS)	47.48 (42.77 - 53.01)	45.31 (44.61 - 45.89)
Iterated Local Search (ILS)	47.37 (47.26 - 47.51)	47.39 (47.26 - 47.54)
Large-Scale Neighborhood Search (LSNS) - no LS	47.29 (47.22 - 47.36)	47.28 (47.20 - 47.37)
Large-Scale Neighborhood Search (LSNS) - with LS	47.26 (47.20 - 47.32)	47.39 (47.18 - 47.54)
Evolutionary Algorithm heuristic, no LS	51.53 (51.36 - 51.68)	53.90 (51.47 - 55.08)
Evolutionary Algorithm heuristic, with LS	51.88 (51.25 - 53.65)	53.74 (53.53 - 54.18)
Evolutionary Algorithm random, no LS	50.75 (50.61 - 50.83)	53.92 (53.59 - 54.30)
Evolutionary Algorithm random, with LS	51.95 (50.71 - 54.07)	54.05 (53.71 - 54.63)
Cycle Stripper - no LS	0.60 (0.49 - 0.99)	0.92 (0.63 - 1.24)
Cycle Stripper - LS at end	0.61 (0.50 - 0.77)	0.62 (0.52 - 0.87)
Cycle Stripper - LS every 1 iter, and at end	2.53 (2.42 - 2.75)	2.53 (2.42 - 2.75)
Cycle Stripper - LS every 3 iter, and at end	1.32 (1.19 - 1.46)	1.32 (1.19 - 1.46)
Cycle Stripper - LS every 5 iter, and at end	1.04 (0.94 - 1.17)	1.04 (0.94 - 1.17)
Cycle Stripper - LS every 10 iter, and at end	0.84 (0.74 - 0.99)	0.84 (0.74 - 0.99)

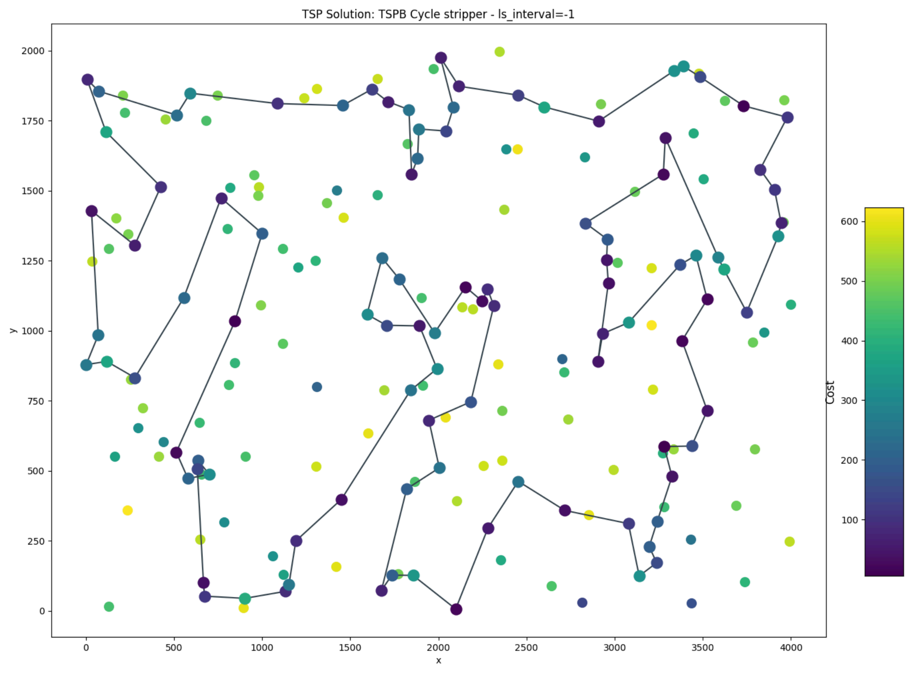
Best solutions

Note: additional cost is depicted using a color scale

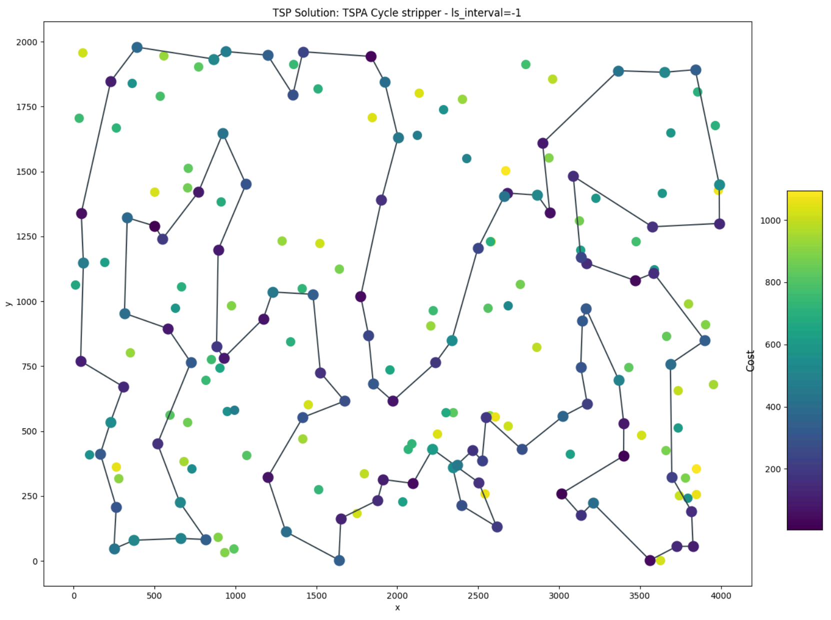
Note: not all solutions are shown here, but the visualizations are available on GitHub

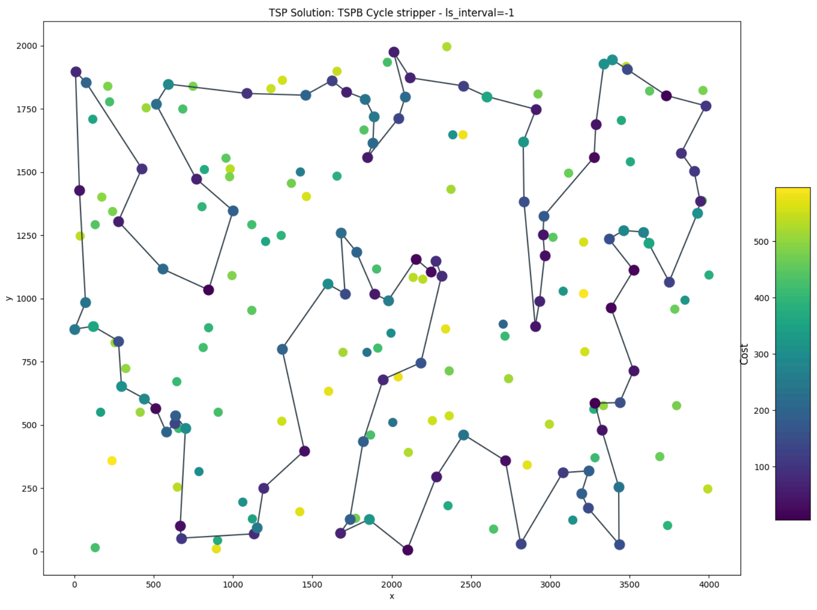
Cycle Stripper - no LS





Cycle Stripper - LS at end





Conclusions

- Initial start from LS on all nodes in the cycle provides global information, which is then exploited by local greedy search during nodes removal
- Cycle Stripper generally seems to perform better on TSPA due to more significant influence of nodes' additional costs
- Less frequent ls interval
    - Technically - when higher, it should be more or less cost at some expense of quality
    - In reality - keeping it relatively small (e.g. every 10 nodes) seems to work just as well as more frequent LS's
    - One needs to strike balance between striving for LS optimality and greedy exploitation
- Based on comparison to less complex methods:
    - Even no LS method outperforms most methods
        - Except greedy cycle (TSPA avg)
        - Except NN at any
    - After applying LS (at least at the end) the method yields better avg and best results than MSLS or EA with no LS (though unfortunate, higher max do happen)
        - Especially considering the method takes fraction of the time
    - Running time (of at least no LS version) is
        - Comparable to that of greedy 2-regret heuristic
        - The same order of magnitude as NN at any, and greedy cycle heuristic
- Based on comparison to more complex methods:
    - Even no LS method outperforms plain LS starting from random solution
        - It performs slightly worse than LS starting from greedy solution, i.e. NN at any followed by LS
    - Due to running time, it's unable to be used as base for "best of all runs" methods such as MSLS
    - However, the method belongs more in the "quick heuristics" class than these more complex ones (its running time is also relatively much quicker)
- Method could be sped up by means of dynamic programming (1 removal results only in change of removal costs of its neighbors) - no LS "in between" could be sped up even further by dynamic programming (1 removal results in only local change in removal costs of removed node's neighbors, the rest stays the same)
- Limitations
    - if one wants solution with all nodes, method turns into plain LS
    - Running time is dependent on that of LS (if it's used)