SOP optimization with simulate annealing

- ## Problem Description:

The Sequential Ordering Problem (SOP) with precedence constraints consists of finding a minimum weight Hamiltonian path on a directed graph with weights on the arcs and on the nodes, subject to precedence constraints among nodes.

- ## Instances Description:

Instances are provided by TSPLIB that it is a library of sample instances for the TSP (and related problems like SOP, ATSP, HCP) from various sources and of various types.

Each instance file consists of two part as **specification part** that contains information about the instance data and **data part**.

- ## Algorithm Description:

The algorithm designed base on the related paper as *(An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem).*

Constructive heuristic used for generating initial solution in the way that from the bingeing each time minimum possible length edge based on precedence condition selected.

For neighboring method to move from current solution to another, **Lexicographic Search** using **forwarding and back warding path-preserving-3-exchange** applied.

The only difference is that in this algorithm lexicographic search doesn't applied on whole search space by iteratively change the parameters "h, i, j", instead random "h" generated and according to that random "i,j" created to do the search.

With the use of loop with size half of dimension forward and with same size loop backward exchanging applied.

It means that in each simulated annealing iteration best solution selected from a list of solutions with size of problem dimension.

- Initial Constructive heuristic
- O(dimension/2) forward searching with random "h, i, j" parameters.
- O(dimension/2) backward searching with random "h, i, j" parameters.
- Selecting the best from search as next solution

- ## Algorithm time complexity:

```
for it in range(int(dimension/2)):
        h = randrange(0, dimension-3)
        i = h + 1
        ...
        for j in range(i, len(solution)):
                for dep in deps[solution[j]]:
                        ...
```

As code shows the forward and backward search consist of 3 loops so the time complexity is $O(n^3)$.

```
def get_neighbor(problem, dependencies, state, cost):
  …
  new_state1 = fpp3exchange(problem, dependencies, state)
  new_state2 = bpp3exchange(problem, dependencies, state)
  …
```

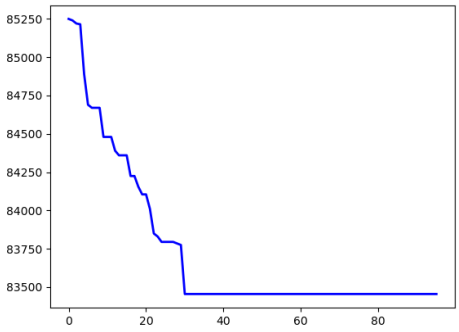
and the neighboring function calling both of them for selecting new solution so the searching algorithm complexity is O(n3).

For updating the temperature 3 methods (**linear** and **logarithmic** and **exponential**) applied to find the best to work with.

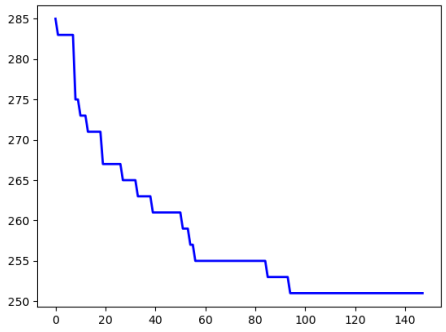
- ## Algorithm Progress:

```
T = 1
ALPHA = 0.8                (for using in temperature updating)
TEMP_MODE = EXP (temperature updating method)
INIT_HEURISTIC = True      (using initial heuristic)
NUM_ITERATIONS = 500
```

- Algorithm progress plot for sample instances:



p43.4.sop



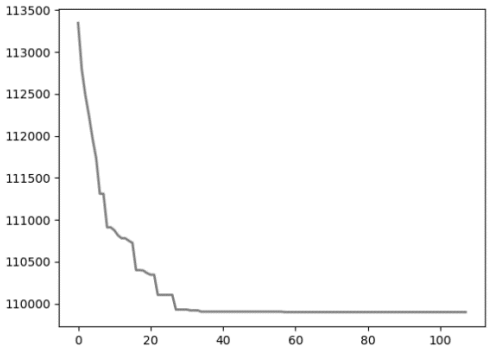
jpeg.4753.54.sop

The whole results (main, max, avg) came in table in excel file named “Results”.

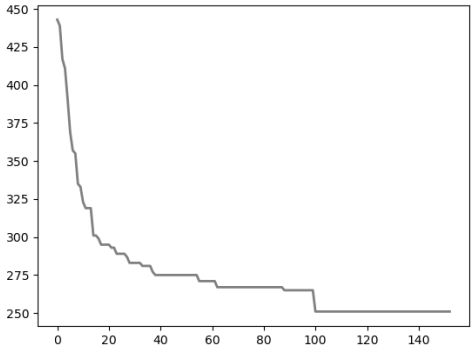
- Initial methods comparison:

T = 1
ALPHA = 0.8
TEMP_MODE = EXP
NUM_ITERATIONS = 500

- Random

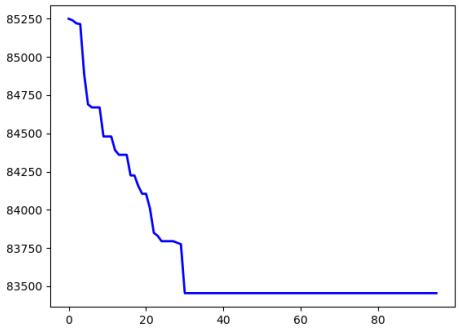


p43.4.sop

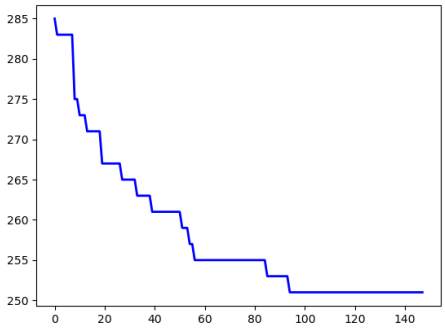


jpeg.4753.54.sop

- Heuristic:



p43.4.sop



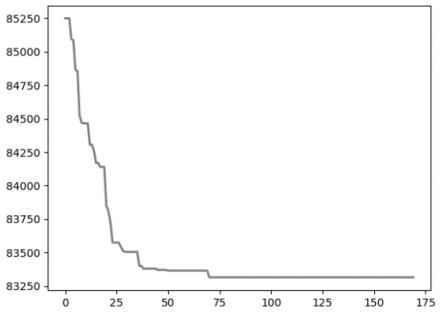
jpeg.4753.54.sop

as result shows with heuristic method algorithm start from much better initial solution and in some cases leads to better final solution.

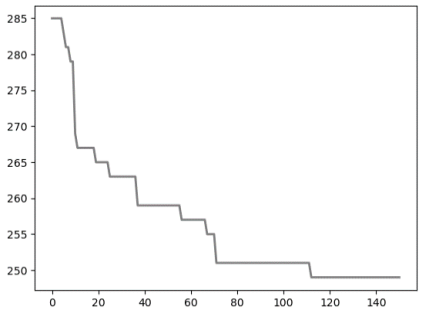
- Temperature update methods comparison:

  T = 1
  ALPHA = 0.9
  INIT_HEURISTIC = True
  NUM_ITERATIONS = 500

  - Linear (ALPHA * T):

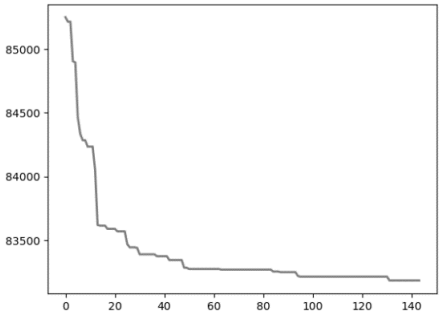


p43.4.sop

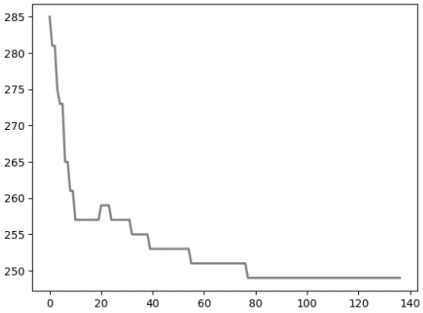


jpeg.4753.54.sop

  - Logarithmic (T0 / math.log(step)):

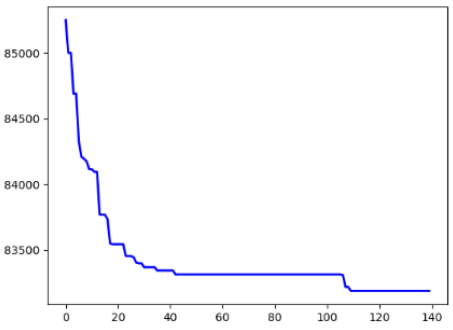


p43.4.sop

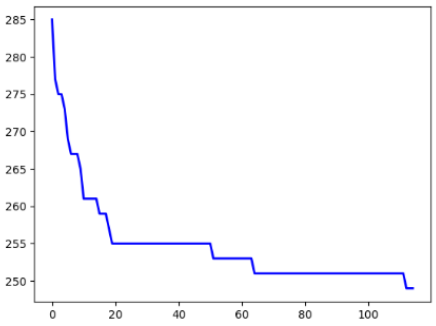


jpeg.4753.54.sop

  - Exponential exp(-ALPHA * step)*T0:



p43.4.sop



jpeg.4753.54.sop

as plots show exponential method perform a little bit better search in compare with other.

- ## Comparison with BKSs:

Instances run with bellow config:

```
T = 1
ALPHA = 0.9
TEMP_MODE = EXP
INIT_HEURISTIC = True
NUM_ITERATIONS = 500
```

Instances with run time under 30 seconds ran 20 times and other with ran 10 times.

On the "E" instances folder, results were near to the BK answers except bellow instance types: kro124p.*, prob.100, prob.7. *, rbg109a.sop.
it seems that from view of this algorithm, these problems were harder than other.

On the "H" instances folder, results were almost similar to the BK answers (with maximum difference equal to 7).

the "M" instances were much more time consuming and the results weren't as good as "H" folder.

The whole results came in excel file named "Results".

- ## Algorithm analysis:

Strength:

this algorithm is much faster that algorithm explained in the related origin paper cause instead of searching whole space with time complexity $O(n^3)$, perform the search just for ***"problem.dimension"*** times.

The results are really close to the paper method in most of the cases.

Weakness:

because of searching the less problem area that related paper method, in some instances it reaches a little bit worst result.

In overall the algorithm is a less time-consuming version of paper method with good acceptable results.