SOP optimization with Simulated annealing, GRASP

- ## 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 and best solution selected among them.
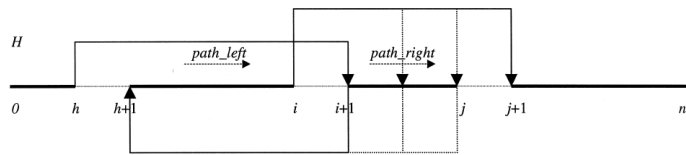

Figure 6. Lexicographic forward path-preserving-3-exchange.
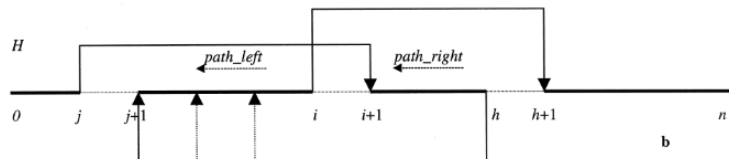

Figure 7. Lexicographic backward path-preserving-3-exchange.

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.
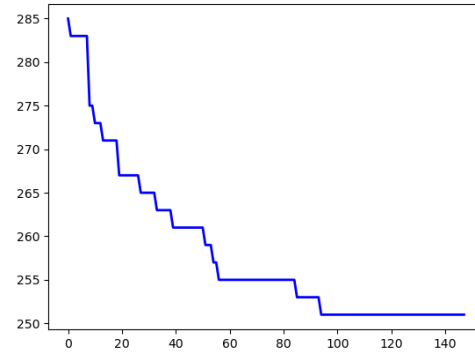
- Simulated annealing algorithm progress:

```
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
```

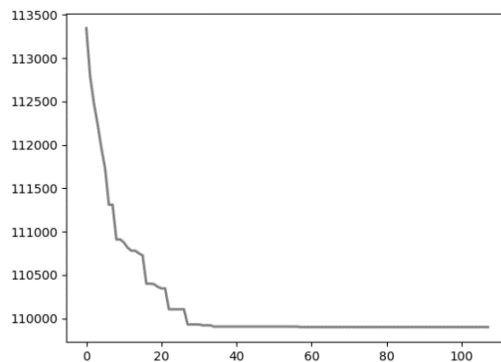- Algorithm progress plot for sample instances:
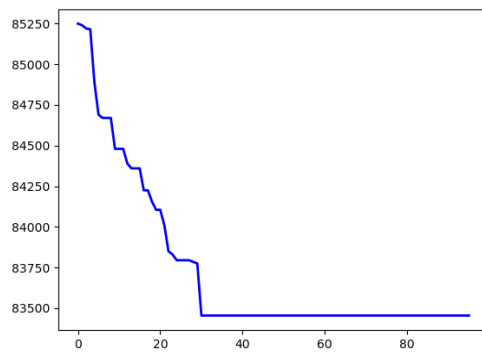
p43.4.sop

jpeg.4753.54.sop

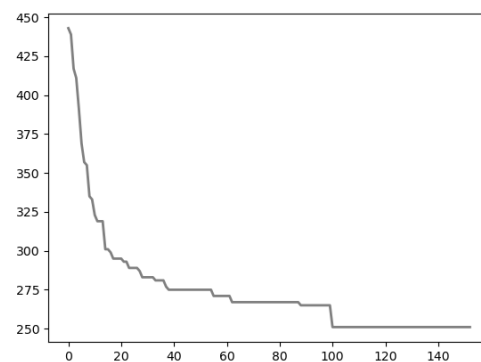The whole results (main, max, avg) came at the end.

- Simulated annealing Initial methods comparison:

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

  - Random

p43.4.sop

jpeg.4753.54.sop

  - Heuristic:
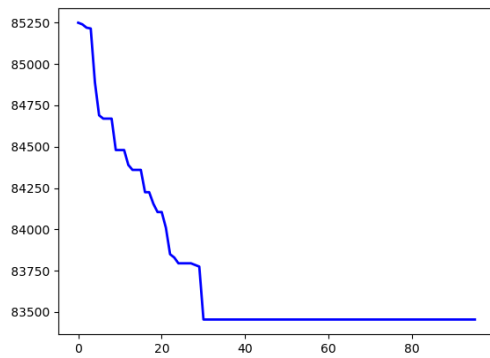
p43.4.sop
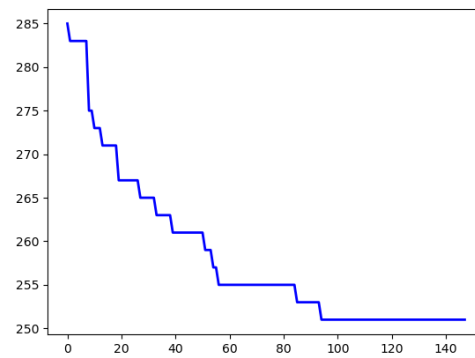
jpeg.4753.54.sop

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

For 10 instances as test heuristic method gave better solution.

- Simulated annealing temperature update methods comparison:

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

- Linear (ALPHA * T):

p43.4.sop
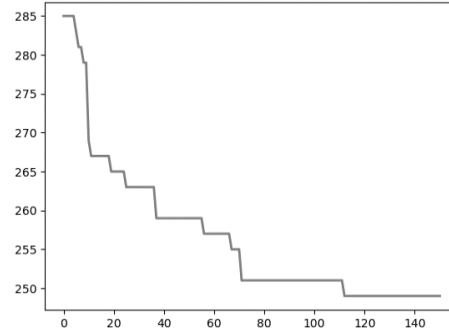
jpeg.4753.54.sop

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

p43.4.sop

jpeg.4753.54.sop

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

p43.4.sop

jpeg.4753.54.sop
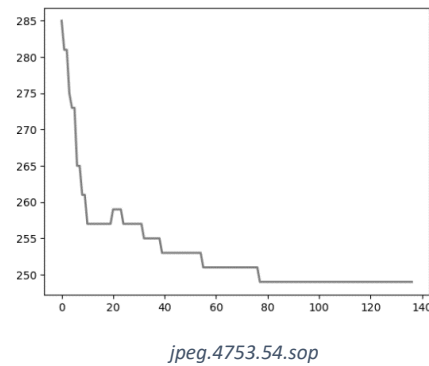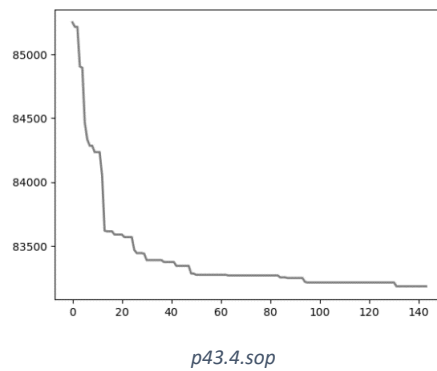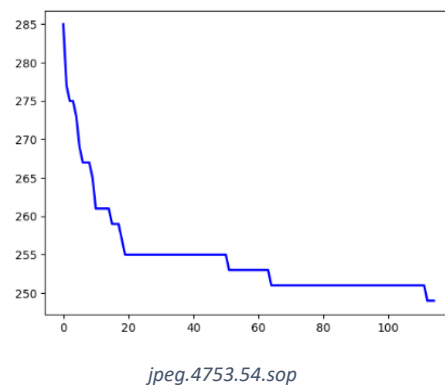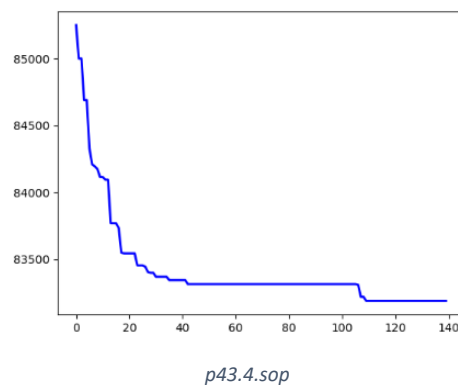
as plots show exponential method perform a little bit better search in compare with other.

- Simulated annealing Comparison with BKSs:

Instances run with bellow config:

```
T = 1
ALPHA = 0.9
TEMP_MODE = EXP
INIT_HEURISTIC = True
NUM_ITERATIONS = 500
```
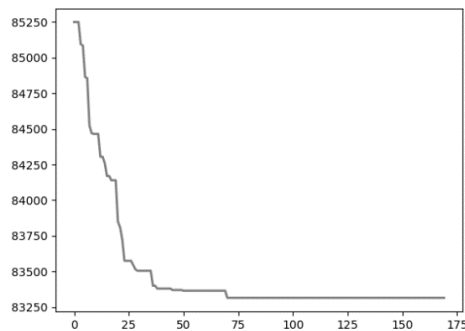
Instances with run time under 30 seconds ran 20 times and other with ran 10 times.

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 (main, max, avg) came at the end.

- Simulated annealing 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.

- Simulated annealing results:

Instance	BKS		BEST	AVG	MAX	min_time	avg_time	max_time	Diff of Best
ft53.2.sop	8026		9473	10312.8	11041	1.1050	1.2013	1.3194	1447
ft70.2.sop	40419		44076	45597.4	46640	2.0604	2.3552	2.7556	3657
kro124p.1.sop	39420		46934	48537.15	49917	7.9588	8.9360	9.8073	7514
kro124p.3.sop	49499		59438	62762.4	66501	3.4378	4.8182	5.6962	9939
p43.1.sop	28140		28290	28598.75	28810	0.8178	0.9116	1.0894	150
p43.4.sop	83005		83140	83270.5	83445	0.3390	0.3805	0.4697	135
prob.100.sop	1123		3158	3870.05	4849	4.0428	4.2812	4.5359	2035
prob.5.sop	243		421	548.15	682	0.7549	0.8216	0.9136	178
prob.7.40.sop	1071		1788	2301.3	2981	0.6532	0.7296	0.8986	717
prob.7.60.sop	912		1952	2545.5	2968	1.4327	1.5482	1.6655	1040
prob.7.70.sop	879		2310	2881.25	3525	1.9058	2.1013	2.2778	1431
rbg050a.sop	400		407	439.4	478	0.8008	0.9093	1.0082	7
rbg050b.sop	397		403	432.75	463	0.8028	0.9041	1.1270	6
rbg050c.sop	467		468	480.9	494	0.7749	0.8580	0.9614	1
rbg105a.sop	1023		1064	1104.55	1143	1.9378	2.1143	2.4140	41
rbg118a.sop	1423		1424	1450.35	1507	1.8151	1.9227	2.1562	1
rbg124a.sop	1361		1366	1397.25	1436	1.8051	1.9065	2.0226	5
rbg126a.sop	1381		1398	1421.6	1481	1.9942	2.2430	2.6624	17
rbg143a.sop	1765		1774	1801.35	1832	2.1233	2.2379	2.4345	9
rbg219a.sop	2544		2578	2605.35	2632	6.6901	7.2614	7.8829	34

rbg247a.sop	3062		3101	3140.35	3187	8.1642	8.7001	9.8681	39
rbg341a.sop	2568		3117	3217.9	3342	26.782	29.577	33.403	549
ry48p.2.sop	16666		18290	20884.7	23105	0.9273	1.0249	1.1865	1624
ry48p.3.sop	19894		22029	23826.5	25251	0.7129	0.8665	1.0328	2135
prob.7.65.sop	915		1649	1930.65	2188	1.6960	2.2015	2.6070	734
rbg109a.sop	198		1046	1081.1	1110	2.0425	2.5092	2.7716	848
rbg117a.sop	1494		1497	1516.55	1548	1.4388	1.6960	1.8951	3
rbg150a.sop	1750		1783	1829.5	1866	3.7965	4.4883	5.2027	33
rbg174a.sop	2033		2059	2114.5	2146	5.1369	6.4195	7.2307	26
rbg190a.sop	2241		2269	2290.0	2311	5.2462	6.6285	7.6026	28
rbg285a.sop	3482		3557	3604.55	3668	14.519	15.807	18.102	75
rbg358a.sop	2545		2884	3001.15	3141	37.397	41.883	47.849	339
gsm.153.124.sop	1109		1110	1121.05	1129	0.6336	0.7397	0.9579	1
gsm.462.77.sop	577		578	581.45	587	0.5404	0.5812	0.6931	1
jpeg.3184.107.sop	791		798	808.0	817	0.8498	0.9889	1.1526	7
jpeg.4753.54.sop	245		247	256.5	269	0.3554	0.4670	0.7355	2
susan.260.158.sop	1016		1022	1035.65	1055	1.7578	2.1034	2.4285	6
typeset.15577.36.sop	155		155	160.65	171	0.2309	0.2598	0.3679	0
typeset.1723.25.sop	64		64	69.85	78	0.1578	0.1951	0.3143	0
typeset.19972.246.sop	2018		2018	2021.6	2034	1.3684	1.4857	1.7511	0
typeset.4724.433.sop	3466		3468	3478.2	3496	6.0954	6.8138	8.1372	2
typeset.16000.68.sop	84		84	85.2	90	0.6667	0.8752	1.1691	0
typeset.10835.26.sop	127		127	130.9	137	0.1950	0.2186	0.2806	0
R.200.100.1.sop	61		340	402.3	453	27.408	28.773	30.956	279
R.200.100.60.sop	71749		72804	74300.15	75808	1.8221	1.9594	2.2568	1055
R.200.1000.30.sop	41196		46190	49303.0	52981	2.2330	2.5983	3.2566	4994
R.200.1000.60.sop	71556		72846	74722.2	76561	1.9925	2.5362	2.8859	1290
R.300.1000.60.sop	109471		110993	112747.95	114203	5.2474	6.6597	9.1146	1522
R.400.1000.15.sop	38963		64354	66147.15	68407	21.304	22.863	25.351	25391
R.500.1000.1.sop	1316		3532	3733.14	3926	631.41	738.70	858.61	2216
R.600.100.60.sop	23293		24300	24479.8	24711	42.649	49.824	66.338	1007
R.600.1000.1.sop	1337		3676	3681.5	3687	1073.2	10774.5	1081.7	2339
R.600.1000.60.sop	214608		224197	226373.6	223894	29.580	33.008	39.903	9589
R.700.1000.15.sop	65678		121526	123669.0	126399	77.331	81.251	92.250	55848
R.700.1000.60.sop	245589		257974	259705.3	261393	80.584	92.586	100.08	12385