<u>Solving a Travelling Salesman Problem using nature inspired search and optimisation algorithms</u>

Note: All code can be run in jupyter notebook
Joe Maliszewski

Write a report to report your results. The report should include

1. Brief introduction of the SA, GA and TS algorithms. You need to justify your design decisions, e.g., encoding scheme for GA, and explain these algorithms by using a flowchart and pseudo-code.
2. Discuss what the parameters are and how you tuned them.
3. You should also list all the average result and standard deviations obtained from the 30 runs of the algorithms
4. Discuss how you compare the results obtained by SA and Tabu search statistically.

Simulated Annealing

Simulated Annealing (SA) is an objective function and can work as an extended version of the "hill-climbing algorithm", in order to find the better optimum solution among a large number of local optima. The hill climbing algorithm is ineffective in these situations as it gets stuck on local minima, where SA can escape these, thus is able to find better solutions.

Annealing is in reference to how metals cool and anneal. The slower a metal cools from a high temperature, the more stable and low energy the structure of ions will be, and therefore it finds a more optimal solution.

The algorithm uses hill climbing, with the additional use of the parameters Temperature, cooling rate, and random probability.

The algorithm works on the premises that there is a probability that a worse solution can be accepted, which allows for local minima to be escaped from.

At a high temperature, the probability of accepting a worse solution is higher, and as it cools the temperature becomes lower, until it is back to normal local optimisation.

“

Algorithm

Step 1. Initialize random solution and high temperature
Step 2: create a random solution by swapping two cities in array
Step 3: calculate route size (smaller the better fitness score)
Step 4: Select or reject depending on P_accept equation
(P_accept = exp((Best_sol -New_sol)/T))
Step 5: update and repeat until T = 0
“

Pseudo code:

“

```
Input: Initial_sol
        Best_sol = Initial_sol

While T > 0:
        New_sol = Create_rand_solution(Best_sol)
                If (New_sol <  Best_sol):
                        Best_sol = rand_sol
                if(New_sol > Best_sol):
                        If P_accept > random_chance
                                Best_sol = New_sol
                        Else:
                                reject
Return Best_sol
```

“

Simulated Annealing:choosing parameters

SA	Temp	T-alpha	iterations	RESULT
1	10	0.003	3000	15263
2	20	0.006	3000	19844
3	50	0.016	3000	16326
4	100	0.0333	3000	16629
5	150	0.05	3000	18488
6	200	0.06	3000	14547
7	250	0.083	3000	18180
8	300	0.1	3000	16381
9	500	0.16	3000	16801
10	1000	0.3	3000	21182
11	1500	0.5	3000	20248
12	40	0.01	4000	18607
13	50	0.01	5000	15021
14	60	0.01	6000	15518
15	70	0.01	7000	16676
16	80	0.01	8000	15897
17	90	0.01	9000	14077
18	100	0.01	10000	14391
19	200	0.01	20000	11551
20	300	0.01	30000	11503
21	400	0.01	30000	15192
22	500	0.01	30000	11639
23	1000	0.01	100000	11726
24	300	0.001	300000	11564
25	310	0.01	3000	23628
26	280	0.01	3000	19043
27	260	0.01	3000	24132
28	200	0.01	3000	20570
29	100	0.01	3000	17522
30	50	0.01	30000	16566

Results and parameters choice

param :	T = 200	T-alpha = 0.06	Itr= 3000

AVERAGE	16716.46667
STDEV	1282.879674

The final parameters chosen above was the best result given in the 3000 iteration restriction category. Parameters were tested using trial and error. It is quite evident from the results that a higher number of iterations however would provide significantly better results. An example of the would be “Trial 20” with 30,000 iterations yielding route distance of `11503`. The final average however for the 3000 iteration is acceptable, considering a substantially better result than hill climbing alone or pure randomness. As one may predict the data shows that a starting high temperature and a slow decrease in temperature over time, proves to yield better results.

Genetic Algorithm (GA)

Genetic Algorithms utilizes the concept of evolution as an objective function. If follows the same rules of survival of the fittest, and is a heuristic approach to solving optimization problems.

Algorithm

1. Create a population. By randomizing the order of genes of the initial solution we create new solutions. A new solution is represented as an individual. Where in the TSP, a city is a gene, and an array of ordered cities is an individual
2. The population is the first generation. And the strongest need to be selected. In order to be selected they must have a fitness score. The fitness score is 1/route_distance of an individual. This can be then sorted into a rank (array) of fitness
3. Now the rank_index is created, selection is now possible. Elitism was implemented which ensures the best individuals in the population automatically get selected. The others under “fitness proportionate selection”, which is representative of a weighted roulette wheel.
4. Once the individuals are selected, they are added to the mating pool.
5. Once added to the mating pool, they can breed. The individuals are paired. When two parents breed, a random section is one parent replaces that same section in the other parent, then fills the rest of the gene with remaining genes.

6. children are then produced.
7. The children are then mutated, where there is a small possibility an individual may have one of their genes swapped within itself.
8. This replaces the old population and the cycle repeats. This repetition is called a generation.

Pseudo code:

```
Population = create_population(init_solution)
For gen in generations:
        Indx = Fitness_ranked(Population)
        Selection_indx = selection (Fitness_ranked)
        Mating_pool = extract_indivuals_from_population(Selection_indx)
        Children = breed_population(mating pool)
        new_generation   = mutate(Children)
Return population
```

Genetic Algorithm tuning parameters

GA	polulation_size	elites	mutation_rate	generations	Results
1	100	20	0.01	10	33228
2	100	20	0.01	20	28279
3	100	20	0.01	50	20387
4	100	20	0.01	100	15408
5	100	20	0.01	200	12989
6	100	20	0.01	300	13561
7	100	20	0.01	400	11997
8	100	20	0.01	700	12557
9	100	20	0.01	1000	11149
10	100	20	0.01	3000	18049
11	50	20	0.01	3000	11393
12	200	20	0.01	3000	10894
13	300	20	0.01	3000	11465
14	400	20	0.01	3000	11683
15	500	20	0.01	3000	11767
16	20	20	0.01	3000	40860
17	200	2	0.01	3000	12815
18	200	8	0.01	3000	13684
19	200	10	0.01	3000	11301
20	200	18	0.01	3000	11612
21	200	30	0.01	3000	11288
22	200	40	0.01	3000	11448
23	200	40	0	3000	11722
24	200	40	0.1	3000	10775
25	200	40	0.02	3000	11403
26	200	40	0.03	3000	10680
27	200	40	0.04	3000	11058
28	200	40	0.05	3000	10704
29	200	40	0.06	3000	11217

30	100	40	0.03	3000	10936

Results and parameters choice

AVERAGE	11034.46667
STDEV	626.7766876

pop = 100	elite = 20	mutation = 0.03	gen = 3000

This algorithm appeared to yield the best results out of all natural search algorithms tested. The number of generations makes a substantial improvement to the results. Furthermore increasing the mutation rate start also presents some significant improvements to overall result (trail 28 amd 30). The number of elites past approximately above 10 does not yield massive improvements, as it sharply hits its ceiling relatively quickly.

Tabu Search

Tabu search makes use of time spans and short term memory to remember recent local optimums to avoid them in relative future. It retains this memory by creating a Tabu list (an array)in that is “taboo”, and to avoid those solutions, therefore rejected previously found local minima. Unlike hill climbing, it can escape local minima to find better solutions with its memory function. The best ever solution found on the cycle however is retained in long term memory.

Algorithm

1. Initial solution
2. Create an array of candidates (different possible solutions). If a candidate is in the TABU list, it is rejected and a newly calculated candidate will take its place. (Tabu list is clear at start)
3. Calculate the fitness of each candidate and give back the best one.
4. The best candidate from the group is added to the tabu list automatically. The Tabu list has a max capacity. If capacity is reached the oldest tabu solution is chucked, and the new one added - one in one out.
5. This best candidate checked against the best current found solution. If better, it replaces the best overall solution.
6. Loops back to create a new array of candidates until the stop condition is met, in this case the number of iterations. The best solution now is the input for the next loop

Pseudo code

```
best _sol = Initial sol

While (stop condition is not met):
    Empty_candidate_list()
    Candidates = create _candiates_list(best sol)
        If candidate in candidates in Tabu list
        reject
        Create_new_candidate
        Add_candidate_to_list(Candidates)
    Winner = get _fittest_canditate(Candidates)
    Add_tabu_list(Winner)
        If Add_tabu_list at limit
            Remove_first_element(tabu_list)
    Is winner better best_sol
        If yes:
            best _sol = winner
        else :
            continue
```

Tabu search choosing parameters:

TS	num_candidates	iterations	TABU_size_limit	Results
1	1000	300	50	18079
2	900	300	50	18154
3	800	300	50	18320
4	700	300	50	17144
5	600	300	50	17498
6	500	300	50	15632
7	400	300	50	15499
8	300	300	50	17068
9	200	300	50	16292
10	100	300	50	15939
11	80	300	50	14545
12	70	300	50	16434
13	50	300	50	15785
14	30	300	50	15745
15	20	300	50	16891
16	10	300	50	17562
17	5	300	50	21461
18	2	300	50	22797
19	80	300	3	16959
20	80	300	10	18035
21	80	300	20	14198
22	80	300	70	16920
23	80	300	100	15876
24	80	300	200	16020
25	80	300	500	15700
26	80	300	1000	18430
27	80	100	20	17750
28	80	500	20	16190

29	80	1000	20	14267
30	80	20000	20	17031

TABU_SIZE_LIM = 500	NUM_CAND = 80	Iterations = 3000

AVERAGE	16668.43333
STDEV	1497.267066

From the results, tabu search was the least successful algorithm. Changing the set parameters did not make a substantial difference to the overall result. An idea may be to develop the acceptance function, allowing for the tabu list to have exceptions in some cases. The parameters chosen were more or less random, due to little correlation. Despite these results, although not fantastic, do show the capacity to produce a substantially reduced route. It therefore suggests it is functionally sound, however development of exception cases could be an area to explore.