AI Search

cmkv68 - Durham University

Utilities for our TSP Methods

In experimentation, both methods are subject to these techniques.

Supreme Tour Caching: In the event that a good tour been discovered, the tour corresponding to its Tour file is saved locally as a cache in a `.json` file. This supreme tour can be loaded again as a reference file for new results to be compared to. This allows us to keep a good tour on hand, ensuring that we get the best result out of any run.

Mutation Techniques

Neighbouring Mutation - A neighbouring mutation is produced by finding a random position `N` within the path and switching its position with the city `N+1`. For example, if city `2` was chosen in the tour `1,2,3,4,5,6,7,8`, then the mutation would return `1,3,2,4,5,6,7,8`. If the city chosen is the last city in the list, then it would be replaced with the first city in the list.

Random Mutation - Random mutation is produced by finding two random positions in the path and switching their positions with each other. For example, in the tour `1,2,3,4,5,6,7,8`, a Random Number Generator is used to get the cities `2`, and `6`. These two cities are swapped leading to the tour path `1,6,3,4,5,2,7,8`.

Partial Shuffle Mutation [1] - Integers are swapped randomly between two randomly generated positions. All integers between the first and the second offset are swapped randomly.

The mutation methods were tested against the tours, `AISearchfile042`, `AISearchfile175` and `AISearchfile535`. The test will use the same path switching method, and the results would be the average of 5 Simulated Annealing runs, to reduce the significance that entropy applies to the mutation methods.

Overall, the Random Mutation swap with a 1x multiplier performed the best, followed by the Random 2x. The neighbouring 1x performed last. This led to using the Random 1x swap being used in both throughout annealing and genetic algorithms. The case for more mutations detrimentally affecting the performance of the mutation may be related to the fact that the tour would deviate further from its tour direction.

TourfileA - Simulated Annealing

The initial implementation consisted of setting alpha `a = 0.9` and temperature `T = 2000` variables. This was followed by generating an initial state (our path) randomly. A while loop is then used where a possible successor state is generated using a 1x mutation. The tour lengths of the states are compared, with the initial state being replaced with the successor if

its tour length is better. Otherwise, it will switch dependent on our probabilistic approach. The probability `a` is deduced as so:

```
- if initial/2 > possible_sucessor
    a = 0.5
- if initial/3 > possible_sucessor
    a = 0.3
- else:
    a = 0
```

`T` is then multiplied by `a`. This loop continues until `T = 0`, or 100,000 steps are performed.

Initial	Test	12	17	21	26	42	48	58	175	180	535
Annealing	28	68	2085	4255	1945	2356	35469	89216	45198	736040	148321

Experimentation

The initial path generation remained random. This was kept to reduce possibilities of leading towards a local maximum. I decided to keep the 1x Random **generation of Neighbouring Path (Mutation)**, as my results have shown that it has performed the best.

Cooling Method

My initial method revolved around the formula `T = a * T`, where Alpha `a < 1`, a method described by Kirkpatrick et. al [3]. I experimented with different `a` values with limited improvements being visible in the results. The logarithmic formula `T = 1/log(step)` was later experimented with, which has shown to improve performance when used in conjunction with the `e^(lo - l1 / T)` Path Switching formula (where `lo` is the neighbouring tour length, `l1` is the current tour length).

Probabilistic Path Switching

Probability allows the states to escape some local minima, an issue with the hill climbing algorithm. I have experimented with the three methods below as my probability function, where `lo` is the neighbouring tour length, `l1` is the current tour length, and `T` is the temperature.

```
1. Initial (Probabilistic approach used during initial run.)
2. e^(-lo / T)
3. e^(lo - l1 / T)
```

These formulas all grow proportionally to the number of steps incremented already. They grow in an exponential fashion to simulate the temperature of a object cooling. The cooler the object, the less likely we will take the neighbouring tour. With experimentation, I have chosen to go forward with 3. `e^(lo - l1 / T)` as my experiments have shown greater improvements amongst the other two formulae.

TourfileB - Genetic Algorithm

My initial implementation worked by initiating a population size of 10,000, with 100,000 generations. Each member of the population is a tour, where each tour is randomly generated

for the first generation. Cycling through all the generations, two random parents are chosen to crossover using the split method, where half of one parent's tour is merged with half of the other parents' tour to create an offspring. Duplicates and missing cities in the offspring are added randomly. Mutation occurs depending on probability, where the child tour would have a randomly chosen city's position N would be switched with its $N+1^{th}$ position. The offspring is added to the population for the next generation. Off-springs will continue to be created until the population size is met. The best offspring of the generation is kept as the supreme. After all generations were lapped, the supreme from the last generation is used for the tour to be outputted as a file.

The initial implementation produced the following results.

Initial	Test	12	17	21	26	42	48	58	175	180	535
Genetic	28	72	2092	4243	1857	2289	35412	91341	44998	745530	151313

Experimentation

I have chosen to keep the population and generation size unchanged, in order to check improvements between the initial implementation and the adjusted. The Mutation method has been chosen to be random 1x, as it performed the best in my testing.

Generation of Population

Several methods have been tested, such as mutating the cached supreme tour, and greedy. I found issues with the Supreme tour where the success of the algorithm is dependent on the RNG (Random Number Generator) taking the tour out of some possible local minima.

I started with loading cached data from the results, but this faced issues as the results would tend towards a local maximum. Random generation was used again until I used a best-first greedy algorithm to generate the children. This improved results for most of the tours, but it lacked entropy to escape issues related to escaping more local maximums. Introducing mutation to the tours after the greedy tours were produced assisted in escaping the local maximum.

Choice of Parents (Pre-Crossover)

I have attempted two methods of choice for choosing Parents, including my initial random choice and choosing the Supreme (where the best tours will cross over with each other; when a parent has been used to cross over then they cannot be used again). I have found that the random parents have performed better.

Crossover Methods

Several crossover methods have been tried, but the most successful methods I found were the *Split Method* and *Partially Mapped Crossover* (Goldberg & Lingle 1985) [2]. I performed tests between the two and found that the split method has shown to be the most effective approach.

Results

Annealing had a 99.9% mutation rate. Genetic used an 80% mutation rate. This is for both initial and modified instances.

Initial	Test	12	17	21	26	42	48	58	175	180	535
Annealing (i)	28	68	2085	4255	1945	2356	35469	89216	45198	736040	148321
Genetic (i)	28	72	2092	4243	1857	2289	35412	91341	44998	745530	151313
Annealing (m)	28	56	1521	2972	1536	1343	16945	34237	26399	4310	70565
Genetic (m)	28	56	1444	2676	1568	1545	18588	38899	22737	189123	57260

I have found that the nearest neighbour approach has dramatically affected the genetics performance in the 180 Cities tour file. This may be related to how the tour was set up, where a basic greedy best first approach might not be appropriate in this case.

Tours	Test	12	17	21	26	42	48	58	175	180	535
Annealing	0	17.65	27.05	30.15	21.03	43	52.23	61.62	41.59	99.41	52.42
Genetic	0	22.22	30.98	36.93	15.56	32.5	47.51	57.41	49.47	74.63	62.16

The above table shows the reductions in tour lengths as a percentage between the modified and the original tours produced by their respective methods. Other than the test method (which has already reached the optimum), every tour file has experienced improvements.

Tour file	Test	12	17	21	26	42	48	58	175	180	535
Shortest Tour	28	56	1444	2676	1536	1343	16945	34237	22737	4310	57260
Best Method	A	A	B	B	A	A	A	A	B	A	B

The above table shows the shortest tour produced by the modified methods, and the method used to produce the tour. Method A is the Simulated Annealing, and B being the genetic algorithm.

References

[1]. Syberfeldt, A., Gustavsson, P., Svantesson, J., Almgren, T. (2014) "A Case Study of Evolutionary Simulation Based Optimization in Aircraft Engine Manufacturing" In: Industrial Simulation Conference, Skövde, June 11-13, 2014

[2]. Goldberg, D.E., and R. Lingle. "Alleles, Loci, and the Traveling Salesman Problem." Proceedings of the First International Conference on Genetic Algorithms and Their Application, edited by Grefenstette J., Lawrence Erlbaum Associates, Hillsdale, NJ, 1985, pp. 154-159.

[3]. S. Kirkpatrick; C. D. Gelatt; M. P. Vecchi (1983). "Optimization by Simulated Annealing", Science, New Series, Vol. 220, No. 4598., pp. 671-680.