Name: Michal Pluta User-name: vsdc48

Algorithm A: Genetic Algorithm (GA)

Algorithm B: Particle Swarm Optimisation (PS)

Description of enhancement of Algorithm A:

The biggest improvement to my implementation was the use of diversity mechanisms, especially mutation.

I implemented 6 different mutation operators including RSM (Reverse Sequence), PSM (Partial Shuffle), THROAS, THRORS, random swap, and full shuffle which helped maintain genetic diversity within the population. Inspired by (Otman, 2012), my algorithm randomly chooses the mutation to perform, with each mutation operator being weighted according to how good I found it to be at promoting genetic diversity. Additionally, the weight of the full-shuffle mutation scales exponentially with the number of iterations since the last incumbent solution. It follows the curve: $min\left(8, \left\lfloor 1.02^{\left(\frac{\text{iterations since incumbent solution}}{10}\right)} \right\rfloor\right)$, which increases gradually from 1 to 8 throughout roughly 1000 iterations. This allows the algorithm to vary how harshly it promotes diversity. I experimented with a few cross-over operators, in particular OX, and SCX, however, I found the OX operator to be the most successful. I tried to vary the crossover operator during runtime but found that SCX, while quicker at converging, led to sub-optimal tours, which is likely due to its close resemblance to the nearest-neighbour algorithm.

I switched from roulette selection to tournament selection which improved performance and led to higher quality tours. I implemented the 2-opt local-optimisation algorithm and chose to treat it as a mutation. It randomly occurs and can be thought of as two parent individuals randomly spawning a genius child individual, which helps the algorithm converge.

I found that running 2-opt fully resulted in lower genetic diversity, and instead, only run 10 passes of 2-opt per call.

To improve convergence, I implemented random extinctions which occur after a threshold of iterations, and purge between 20-40% of the least-fit individuals in the population. Additionally, as per (Eremeev, 2021), I added random classical restarts, which trigger after a threshold iteration, and if the current iteration number > 2 × iteration of the incumbent solution.

As with PSO, I tried to initialise the individuals in my population with tours formed using nearest neighbour instead of fully random, however, I found this to hurt the genetic diversity of the population. My implementation outperformed the basic version by roughly a 5-10% reduction on city sets smaller than 48, and between 25-97% on city sets larger than 48.

Description of enhancement of Algorithm B:

The first modification I chose to make was implementing a different velocity function to remove the $O(n^2)$ time complexity of bubble-sort. I did this by implementing a velocity function with non-consecutive swaps (permutation), which drastically improved the performance of the algorithm. I experimented with different topologies of PSO and found that the star topology (global best) (Shami, 2022) was the most efficient and consistent at finding good solutions.

Upon testing different parameter configurations, I found that by bounding the epsilon proximity parameter between 0.8 and 1, the algorithm was more consistent at finding good solutions. By adding an annealing factor to the inertia (theta) (Shami, 2022), and scaling it exponentially from 1 down to 0, particles were less likely to leave a local minima's area, resulting in better tours. I also implemented velocity normalisation which triggers if $|\mathrm{velocity}| > 2.5 \times \mathrm{num\_cities}$. This means the complexity/length of each velocity can be contained, avoiding velocity explosion (Shami, 2022), and improving performance.

While all these improvements helped with performance, the quality of the tours was lacking. I decided to try using nearest-neighbour for initialising particle tours, which was achieved by randomly generating a partial tour (of length 2-3), and then completing it using nearest-neighbour. This led to significant tour quality improvements and by integrating the 2-opt local optimisation algorithm (Croes, 1958) (Uddin et al.), my PSO algorithm saw near-optimal performance.

One drawback to this was that particles often got stuck in local optima and failed to escape, which was likely caused by the choice of topology. I decided to get a bit creative and implemented dazing (this might already exist, but I couldn't find any relevant literature). To put simply, the idea behind dazing is to avoid particles tending towards the same global optima. In practice, if a particle hasn't improved upon its personal best in a certain number of iterations, it has a small chance of getting dazed where its position is randomly reset, velocity is set to 0, and it is blinded (social velocity = 0) for a duration of 10 iterations. This adds some diversity to the algorithm and avoids premature convergence.

Similarly, extinction events were implemented to escape local optima which resulted in higher-quality tours.

Overall, my PSO algorithm outperformed the basic implementation by a consistent reduction of 40-70% and in some cases up to 99.7%. (Each basic algorithm was run 5 times (60s each) on each city set; the median time of each run was compared to the enhanced algorithm's best overall solution)

DESCRIPTION OF ALGORITHM ONLY IF THE ALGORITHM IS NOT COVERED IN LECTURES

Description of *non-standard* Algorithm A:

Description of *non-standard* Algorithm B: