SOS EX1

Josef Rabmer Leonhard Ender

1 Introduction

The optimization methods we chose for exercise 1 are Genetic Algorithms (GA) and Ant Colony Optimization (ACO). We used these algorithms to find solutions for the Job-shop Scheduling Problem, the Quadratic Assignment Problem and the Constrained Vehicle Routing Problem.

2 JSP

The Job-shop Scheduling Problem is the problem of scheduling a given number of jobs for execution on a given number of machines. There are different variants of it; in the one we chose, a job consists of a number of steps (or parts) that need to be executed in a specific order and each step can only be executed on exactly one of the machines and takes a specific amount of time. Machines can only execute on job at a time. The goal of the optimization is to minimize total execution time.

An example for an instance of this version of the JSP would be:

- Three machines m_1, m_2 and m_3
- Two jobs j_1 and j_2, with the following steps (in this order):
  - j_1: 10 min on m_2, 5 min on m_1, 15 min on m_3
  - j_2: 25 min on m_2, 13 min on m_3, 5 min on m_2

For the performance evaluation, we used four different instances:

	ft06	la01	la29	la40
Jobs	6	10	20	15
Steps/Machines	6	5	10	15

Table 1: Instances of the JSP used for performance evaluation

2.1 Formulation

2.1.1 GA

Genetic Algorithms are inspired by the principles of natural evolution and genetic processes. In GAs, a population of candidate solutions evolves through selection, crossover, and mutation, gradually improving over generations.

For JSSP, a solution, or "chromosome," typically encodes a sequence of operations to be performed. One common representation uses operation-based encoding, where the sequence of job IDs specifies the order in which operations are scheduled. For example, a chromosome [1, 2, 1, 3, 2, 3] might represent the order of operations for three jobs.

Apart from finding the right encoding, all genetic algorithms need to define a fitness function and the other genetic operators. The fitness function in a GA evaluates the quality of each solution. In our scenario this is minimizing the total make-span of all of the jobs. One can do this by just finding the start time of the first job and end time of the last job.

The algorithm begins by initializing a population of randomly generated chromosomes, which ensures diversity in the solution space. Through selection mechanisms better-performing solutions are chosen to produce offspring. Crossover, a key GA operation, combines portions of two parent chromosomes to create new solutions. For the application to the JSSP problem, we went with a specialized crossover method Order Crossover (OX). This ensures that offspring remain valid scheduling sequences. Finally we use mutation to randomly swap two jobs in order to introduce genetic variety.

2.1.2 ACO

Since the Ant System is designed to work on graphs, the JSP problems needs to be represented as a graph (in this case, a directed weighted graph). The approach we used is outlined by Dorigo et. al. in [DMC96], where they use it as an argument for the robustness of the Ant System approach.

The basic idea is to represent each step of every job as a node, e.g. step 5 of job 7 would be node N_{57}. Every node has an arc (i.e. a directed edge) to the next step in the job (e.g. N_{57} has an arc to N_{67}, which in turn has an arc to N_{77}) as well as an undirected edge to every step that belongs to a different job (e.g. N_{57} has an undirected edge to every step of all jobs except job 7). In addition, there is a start node N_0, which has an arc to the first step of every job is is only used as a starting point for the ants.

An ant constructs a possible solution to the problem by starting at N_0 and traversing the graph until every node has been visited. It keeps a list of the next step for every job, from which it selects the next node to go to, as well as a list of nodes that have already been visited (i.e. steps that already have been completed). Once the ant has traversed the entire graph, the order in which it visited the nodes is a solution to the JSP.

The decision on which node to visit next is made based on the weight of the graph edges, which is initially equal for every edge[1] and is refined every iteration by adding

[1]Note that Dorigo et. al. propose initializing the weights by using a fast greedy heuristic for the JSP. This could improve the performance of the ACO (especially if it is run with few iterations), but we left it out in our solution for simplicity.

pheromones based on how many ants took the edge and how good their solution was.

2.2 Performance

2.2.1 GA

We evaluated the GA implementation with four differnts test cases. As one can see from the first test set, initial good solutions were found but no significant further improvements could be made after that

Iterations	ft06	la01	la29	la40
50	57	666	1567	1276
100	55	666	1392	1276
1000	55	666	1392	1276

Table 2: Results (makespan time) for JSP using GA.

2.2.2 ACO

Performance of the ACO was evaluated using four problem instances of different sizes, each with different numbers of iterations of the ACO.

Iterations	ft06	la01	la29	la40
10	44	666	1188	1179
50	43	666	1178	1134
100	43	666	1146	1123

Table 3: Results (makespan time) for JSP using ACO with 20 ants. The ACO parameters used were $\alpha = 1, \beta = 1, \rho = 0.91$.

3 QAP

The Quadratic Assignment problem deals with the following real-life issue: There are a set of n facilities and a set of n locations. For each pair of locations, a distance and a weight or flow is specified (e.g., the amount of supplies transported between the two facilities). The problem is to assign all facilities to different locations with the goal of minimizing the sum of the distances weighted by the corresponding flows. An example application would be to position components on a motherboard.

3.1 Formulation

3.1.1 GA

The QAP lends itself quite well to the application of Genetic Algorithms. A solution is a permutation of the facilities to the locations. This permutation can be represented as a chromosome in GAs. For example, a permutation of facilities $[f_1, f_2, ..., f_n]$ represents assigning the facility f_1 to location 1, f_2 to location 2, and so on. In order to solve the problem, we then have to minimize the total cost given by:

$$\sum_{a,b \in P} w_{a,b} d_{f(a),f(b)}$$

Where w is the weight function, d a distance function and f an assigment of facilities to locations.

Apart from the fitness function, it is also important to specify the other genetic operators.

- Selection: Roulette Wheel Selection
- Crossover: Two point crossover
- Mutation: Swap mutation

Important here is the use the swap mutation because we need to ensure that after every mutation of crossover the new solution is still valid and there are no duplicates. Swap mutation swaps two random values and therefore there are no issues here. For two point crossover it is important to watch for possible violations and then fix these correspondingly.

3.1.2 ACO

We solved the Quadratic Assignment Problem with the Ant System by utilizing an approach laid out by Dorigo et. al. in [DMC96] (the same paper used for the JSP) and, in more detail, by Colorni and Maniezzo in [MC99]:

An ant creates an assignment by going through all locations and successively picking facilities to place there. Similar to how an ant would pick the next town to go to in the TSP, the choice which location to pick is based on an initial heuristic (see [DMC96, p. 38] for the exact heuristic used) and on the pheromones that were left there by ants previously.

After each iteration, the assignments created by the ants are evaluated and pheromones are deposited accordingly.

3.2 Performance

3.2.1 GA

Genetic algorithms are know to reach good solutions very quickly and this is confirmed here by the results. After only 10 generations it usually reaches a decent solution. Depending on what metric is used to generate the initial population this can improve significantly. There can be some improvements seen with a higher number of generations. At generation 400, the results do get slightly better but there are no big differences anymore. This points to the fact that the solutions still in the running likely approach a local extrema.

	nug12	nug20	nug30
Optimal	578	2570	6124
Iterations			
10	648	3030	7062
400	630	2965	6802
1000	613	2932	6589

Table 4: Results for QAP using GA with 1000 generations

3.2.2 ACO

Notably, the Ant System did not perform very well for this problem, even though the approach is mostly the same as in [MC99], which evaluates the algorithm on some of the same instances and achieves drastically better results (often even finding the optimum). Although it is not entirely clear how many iteration were used[2], the fact that our version does so much worse (and does not seem to improve when increasing the number of iterations even beyond 100) implies that either there is a fault in the implementation, the parameters were not dialed in correctly (even though we did try a multitude of variations), or the differences between the approaches has a greater influence then anticipated. These difference mainly are a more sophisticated heuristic for the initial weighting and the use of a local search after an ant has found a solution (see [MC99, p. 772]). It is notable that Maniezzo and Colorni found in their experiments that reducing the influence of the heuristics lead to stagnation around suboptimal solutions [MC99, p. 773]

	nug12	nug20	nug30
Optimal	578	2570	6124
Iterations			
10	670	3038	7562
50	654	3080	7504
100	648	2964	7452

Table 5: Results for QAP using ACO with 20 ants. The ACO parameters used were $\alpha = 0.5, \rho = 0.9$.

4 CVRP

The Vehicle Routing Problem (in the version we used) is the problem of finding an optimal path for a vehicle to deliver goods from a central depot to a number of customers, with the restriction that each customer needs a certain amount of goods an the vehicle only has limited carrying capacity. This means that it might have to return to the depot (potentially more than once) to refill. It is possible to optimize for different cost factors, we chose to optimize for total path length. This makes the VRP a generalization of

[2]The paper only specifies the execution time as 10 minutes. Since it was published in 1999, this is not comparable to execution times on a modern computer.

the Traveling Salesman Problem (TSP), as a vehicle with infinite capacity would simply choose the shortest round tour.

4.1 Formulation

4.1.1 GA

For solving the Vehicle Routing Problem, one can follow a similar approach to that of the TSP. The major difference is the addition of the vehicle capacity and the starting depot. A natural candidate for the encoding of the solutions is therefore a list of taken routes. A route is a list of nodes and also always starts at the depot node and ends at it.

We can then take a similar approach to TSP and minimize the distance while also making sure the capacity and route validity constraints are not broken. The fitness function is therefore very similar to TSP as well as the selection or mutation operators. One does have to ensure that after crossover, all resulting solutions are still valid.

4.1.2 ACO

Since the Vehicle Routing Problem is a generalization of the TSP, it can be solved by an Ant System in a very similar way: Ants start out at the depot and consecutively visit every node, but unlike int the TSP, they can only keep going until their capacity runs out, at which point they have to return to the depot and continue from there. This approach is presented in [BM04].

Note that this approach only works for this version of VRP. If we for example chose to optimize for time (given a certain number of vehicles) instead of total distance, a different approach would be needed.

4.2 Performance

4.2.1 GA

The following measurements are from using GAs for solving the Vehicle routing problem.

Iterations	A-n54-k07	A-n80-k10
100	3273	2927
400	2823	2623
1000	2692	2343

Table 6: Results (total path distance) for VRP using GAs.

4.2.2 ACO

The Ant System performs reasonably well on the VRP, and increasing the number of iterations results in significant improvement. There are some notable flaws in the resulting paths though: Because the ants (simulating a vehicle) keep going until their storage is depleted, and nodes which exceed the demand are removed as potential next stops, they can end up in a situation where only one node has a small enough demand to be eligible, and they will then deliver to that node even if it is very far away, instead of returning to the depot early to refill.

Iterations	A-n54-k07	A-n80-k10
10	1477	2647
50	1224	1998
100	1100	1938

Table 7: Results (total path distance) for VRP using ACO with 20 ants. The ACO parameters used were $\alpha = 1, \beta = 1, \rho = 0.95$.

References

[BM04] John E. Bell and Patrick R. McMullen. "Ant colony optimization techniques for the vehicle routing problem". In: *Advanced Engineering Informatics* 18.1 (2004), pp. 41–48. ISSN: 1474-0346. DOI: `https://doi.org/10.1016/j.aei.2004.07.001`. URL: `https://www.sciencedirect.com/science/article/pii/S1474034604000060`.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. "Ant system: optimization by a colony of cooperating agents". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 26.1 (1996), pp. 29–41. DOI: `10.1109/3477.484436`.

[MC99] V. Maniezzo and A. Colorni. "The ant system applied to the quadratic assignment problem". In: *IEEE Transactions on Knowledge and Data Engineering* 11.5 (1999), pp. 769–778. DOI: `10.1109/69.806935`.