COS 314
Artificial Intelligence
Assignment Two

26 April 2025

Dewald Colesky u23536030

Table of contents:

1. Questions
   1.1. Pheromone Representation and Update Mechanisms
   1.2. Ant movement rules and route construction strategies.
   1.3. Constraint Handling Techniques
   1.4. Parameters and their tuning strategies
2. Experimental setup
3. Results
4. Analysis
5. Appendix
   5.1. Readme
   5.2. References

Pheromone Representation and Update Mechanisms

In this ACO design for TOP:

- **Pheromone trails** are represented as a matrix (i,j) where each element indicates the likelihood of traveling between nodes

- **Pheromone Update**: Pheromones evaporate or have the best pheromones deposited with (Q * current edge pheromone) / Calculated score

```
    coord edgePos(alledges[ec].first);
    for(auto & eu : alledges[ec].second)
    {
    if(areCoordsEqual(eu.second.first,added))
    {
        eu.first = (params.Q*(eu.first)/p.second);
        continue;
    }
    eu.first *= (1-params.evaporationRate);
```

```
//now compute scores
for(const auto & i : scores)
{
    std::pair<coord, double> Pij;
    Pij.first = i.first.first;
    Pij.second = pheromoneCalc(i.second, i.first.second, scores);
    calculatedScore.push_back(Pij);
    Logger::info("Raw Pheromone scores: " + Pij.first.to_string() + "\t" + std::to_string(Pij.second), update);
}
```

    - Only the current best solution contribute to pheromone increases, encouraging exploration of good routes.

Ant Movement Rules and Route Construction Strategies

Ants construct solutions using a probabilistic rule:

- At each step, an ant chooses the next node based on a combination of:
    - **Pheromone strength** at (i,j)
    - **Heuristic information** score-to-distance ratio
    - **Random reset:** Around 1% of the time a node is reset and another option is chosen from the calculated options.

- The probability of moving from node i to node j is given by:

$$p(i,j) = \frac{[\tau(i,j)]^{\alpha}[\eta(i,j)]^{\beta}}{\sum_{k \in \text{allowed}}[\tau(i,k)]^{\alpha}[\eta(i,k)]^{\beta}}$$

```
double ProblemInstance::pheromoneCalc(double Tij, double Nij,std::vector<std::pair<std::pair<coord,double>, double>> scores)
{
    double top = std::pow(Nij, params.pheromoneImportance) * std::pow(Tij, params.heuristicImportance);
    double bottom = 0.0;
    for(auto & i : scores)
    {
      //  Logger::info("i.first.second: " + std::to_string(i.first.second), "wfge.txt");
        if(std::isnan(i.first.second))
        {
            i.first.second = i.second;
        }
       // Logger::info("i.second: " + std::to_string(i.second), "wfge.txt");
        bottom += (std::pow(i.first.second, params.heuristicImportance) * std::pow(i.second, params.pheromoneImportance));
    }

    return top / bottom;
}
```

- where:
    - α controls the influence of pheromones,
    - β controls the influence of heuristic information.

Ants continue to add nodes to their route until iteration or tmax constraints prevent further expansion.

Constraint Handling Techniques

To ensure feasible solutions:

- **tmax** is strictly enforced:
    - An ant cannot select a node if visiting it would cause the route's total travel to exceed tmax.
- **Vehicle Capacity Constraints:**
    - Each ant constructs multiple tours, splitting the nodes across available vehicles without exceeding individual constraints.
    - Visited nodes may not be seen by any other vehicles
- **Node selected:**
    - Only viable nodes are considered
    - Each unvisited node is checked in an iteration
    - Each viable node has the probabilistic function calculated
    - A node is chosen randomly based on the probabilistic scores

Parameters and Tuning Strategies

- **Parameter Tuning:** alpha, beta, rho, Q were tuned to gain overall best parameter setup (results in spreadsheet).

```
16  void parameterTuning(std::vector<std::string> f)
17  {
18  std::vector<double> alphaValues = {0.1, 0.2, 0.5, 1.0};
19  std::vector<double> betaValues = {0.1, 0.3, 0.6, 1.0};
20  std::vector<double> evaporationRates = {0.3, 0.5, 0.7};
21  std::vector<int> qValues = {1, 2, 5};
22
23  for(auto &filename : f)
24  {
25      std::vector<TuneResult> tuningResults;
26    //  for(int k =0; k < 10;k++)
27      {
28          for (double alpha : alphaValues) {
29              for (double beta : betaValues) {
30                  for (double evap : evaporationRates) {
31                      for (int q : qValues) {
```

- **Pheromone Importance (α)**: Controls the influence of pheromone trails; set at 0.525
- **Heuristic Importance (β)**: Controls the influence of heuristic information; set at 0.775
- **Pheromone Evaporation Rate (ρ)**: 0.5.
- **Pheromone deposit Factor (Q)**: 1.5.
- **Initial Pheromone Level**: Uniform across all edges at 0.1.
- **Tuning Strategies:**
    - Parameters were tuned through brute force testing.
    - All problem instances were solved with various settings to determine robust values.
    - After initial experiments, slight adjustments were made based on convergence behavior and solution quality.
    - Parameters to test with were somewhat random to test extremes and average values

Experimental Setup

Problem Definition

The problem being solved is a version of the Team Orienteering Problem (TOP), but instead of trying to travel the shortest distance, the goal is to get the highest score. Every time a node is visited, it adds to the total cost (or score), and the aim is to find a route that collects the most points while visiting each node at most once and returning to the start. The purpose of the evaluation is to see how well Ant Colony Optimization (ACO) performs on different problem setups by looking at how good the solutions are, how quickly they improve, and how long they take to run.

Algorithm Used

Ant Colony Optimization (ACO)

- **Parameter Tuning:** alpha, beta, rho, Q were tuned to gain overall best parameter setup.

```
16  void parameterTuning(std::vector<std::string> f)
17  {
18  std::vector<double> alphaValues = {0.1, 0.2, 0.5, 1.0};
19  std::vector<double> betaValues = {0.1, 0.3, 0.6, 1.0};
20  std::vector<double> evaporationRates = {0.3, 0.5, 0.7};
21  std::vector<int> qValues = {1, 2, 5};
22
23  for(auto &filename : f)
24  {
25      std::vector<TuneResult> tuningResults;
26    //  for(int k =0; k < 10;k++)
27      {
28          for (double alpha : alphaValues) {
29              for (double beta : betaValues) {
30                  for (double evap : evaporationRates) {
31                      for (int q : qValues) {
```

- **Pheromone Importance (α)**: Controls the influence of pheromone trails; set at 0.525
- **Heuristic Importance (β)**: Controls the influence of heuristic information; set at 0.775
- **Pheromone Evaporation Rate (ρ)**: 0.5.
- **Pheromone deposit Factor (Q)**: 1.5.
- **Initial Pheromone Level**: Uniform across all edges at 0.1.
-
- **Max Iterations**: 10 * m * n.
- **Stopping Criteria**: Max iterations without improvement: 4 * m * n.

- **Randomness**:The random seed used in ACO runs can either be user-defined or generated automatically.If not specified by the user, the program combines multiple sources of entropy to produce a robust random seed:

```cpp
unsigned int generateSeed() {
    std::random_device rd; // non-deterministic random number generator
    auto timeSeed = std::chrono::high_resolution_clock::now().time_since_epoch().count();
    auto pidSeed = static_cast<unsigned int>(getpid());
    auto rdSeed = static_cast<unsigned int>(rd());

    // Combine using XOR and bit shifting to improve mixing
    unsigned int seed = static_cast<unsigned int>(timeSeed) ^ (rdSeed << 1) ^ (pidSeed << 2);
    return seed;
}
```

Experimental Environment

Hardware

- **Processor**: AMD Ryzen 5 7600 × 12 cores
- **Memory**: 16 GB DDR5 RAM
- **System**: Micro Star International Co., Ltd MS7E28
- **Operating System**: Ubuntu 24.04.1 LTS

Software

- **Language**: C++17
- **IDE**: Visual Studio Code

Benchmark and Test Set

The following data sets will be tested:

- Problem instances involving 33, 102 nodes with different amounts of vehicles and tmax.
- Each instance will be solved multiple times with 10 runs per data set.
- **Cost and distance is assumed to be related**

Evaluation Metrics

- **Time Taken**: Total time required by ACO to find a solution.
- **Iterations Completed**: Number of iterations performed before convergence or stopping.
- **Solution Quality**: The final total cost achieved by the best vehicle (higher is better).

Experimental Procedure

1. **Input Data**: Read the data representing nodes, vehicles, distances, and costs into the program.
2. **User Input**:Run the program (With fixed or random seed)
3. **Algorithm Execution**:
    - For each problem instance, run ACO for the defined number of runs.
    - Each run will execute for a maximum of (10 * m * n) iterations.
4. **Data Collection**:
    - Save the following data into a file called Utils/logs/results.txt:
        - Seed used for each run.
        - All run solutions and costs.
        - The file and problem instance tested.
        - The best run for each file and algorithm (Ordered results).
        - Time taken to complete.
        - Iterations used.
        - Total cost
        - Total distance
        - Total runtime

Comparison

The performance of ACO will be compared based on:

- **Cost**: Which had the highest cost.
- **Total Cost**: The sum of costs of each instance.
- **Runtime**: How many iterations/time (directly corresponding) are needed to reach the best solutions or stopping criteria.

This analysis will highlight trade-offs between computational efficiency and solution quality in maximizing the cost objective.

Results

ResultsCOS314A2

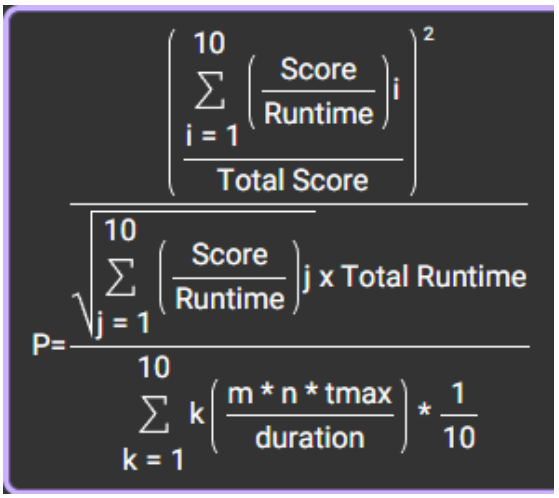
*To see route taken please open the spreadsheet linked

FileNum	File	Seed	Cost	Duration	Runtime	Iterations
1	p3.2.a.txt	3588961391	10	6.50	-0.025	267
Vehicles	2	3045640776	20	6.79	-0.024	266
Tmax	7.5	805353773	20	5.08	-0.023	267
Nodes	33	3713294087	20	6.76	-0.024	267
		3058826263	30	5.15	-0.027	266
		2279845222	60	7.40	-0.023	268
		3010371004	60	6.76	-0.023	267
		1328493137	60	5.08	-0.023	267
		397781147	60	6.76	-0.023	267
		2912821468	60	6.04	-0.023	267
		Total	400	62.33	-0.235	2669
FileNum	File	Seed	Cost	Duration	Runtime	Iterations
2	p3.2.c.txt	2100401784	80	11.08	-0.024	268
Vehicles	2	455442220	0	0.00	-0.023	267
Tmax	12.5	2994187335	70	12.22	-0.024	269
Nodes	33	3856232744	20	11.19	-0.023	267
		1101000685	40	12.40	-0.023	267
		419947862	30	11.28	-0.023	267
		3822526212	40	8.95	-0.023	266
		2793196359	20	11.40	-0.023	266
		1139819698	70	10.44	-0.023	268
		2361228586	80	11.08	-0.023	268
		Total	450	100.05	-0.231	2673
FileNum	File	Seed	Cost	Duration	Runtime	Iterations
3	p3.3.a.txt	3132753787	50	4.49	-0.050	398
Vehicles	3	3561267176	0	0.00	-0.050	398
Tmax	5	84640900	10	3.94	-0.050	398
Nodes	33	1001664736	50	4.49	-0.050	398
		904009226	50	4.49	-0.050	398
		3115884780	50	4.49	-0.050	398
		1276495525	50	4.49	-0.050	398
		2588657403	50	4.49	-0.050	398
		2944692779	50	4.49	-0.050	398
		497035052	50	4.49	-0.050	398

		Total	410	39.90	-0.502	3980
FileNum	File	Seed	Cost	Duration	Runtime	Iterations
4	p3.3.c.txt	2484330328	30	7.40	-0.051	400
Vehicles	3	3843576227	20	8.00	-0.050	398
Tmax	8.3	47257914	50	4.49	-0.050	398
Nodes	33	1829461440	20	8.00	-0.051	399
		906390500	50	4.49	-0.051	398
		2833211966	60	8.23	-0.050	399
		283137333	60	8.23	-0.050	399
		2444135262	60	6.76	-0.051	399
		3642199181	50	4.49	-0.051	398
		2479173747	50	4.49	-0.051	398
		Total	450	64.59	-0.506	3986
FileNum	File	Seed	Cost	Duration	Runtime	Iterations
5	p3.4.s.txt	3678257880	70	24.00	-0.093	531
Vehicles	4	1904109619	30	20.29	-0.092	532
Tmax	26.2	2921974081	50	25.77	-0.091	532
Nodes	33	3667155210	80	23.51	-0.092	532
		3303655187	70	23.32	-0.094	531
		76827555	50	23.50	-0.091	531
		1403401706	70	19.74	-0.094	531
		684747192	50	25.55	-0.092	531
		4161432001	80	22.78	-0.094	531
		1120071022	20	17.86	-0.091	531
		Total	570	226.33	-0.924	5313
FileNum	File	Seed	Cost	Duration	Runtime	Iterations
6	p3.4.t.txt	4029404983	80	24.70	-0.094	533
Vehicles	4	3816074035	80	25.99	-0.093	532
Tmax	27.5	3832680150	70	24.28	-0.092	532
Nodes	33	1560431674	60	26.31	-0.093	534
		2588379533	60	27.44	-0.092	531
		1063754580	50	23.50	-0.091	531
		818462540	70	27.26	-0.093	531
		1823977530	0	0.00	-0.092	533
		3832078516	120	26.24	-0.093	533
		177992253	80	24.39	-0.091	531
		Total	670	230.12	-0.924	5321
FileNum	File	Seed	Cost	Duration	Runtime	Iterations
7	p7.2.a.txt	1896668571	14	8.94	-0.063	818

Vehicles	2	3162175026	14	8.94	-0.063	818
Tmax	10	1669503223	0	0.00	-0.070	818
Nodes	102	1005671417	14	8.94	-0.063	818
		931772212	16	10.00	-0.063	818
		1405850219	14	8.94	-0.071	818
		976639651	16	10.00	-0.063	818
		3836801261	14	8.94	-0.063	818
		2207474079	14	8.94	-0.063	818
		3963966316	16	10.00	-0.063	818
		Total	132	83.67	-0.643	8180
FileNum	File	Seed	Cost	Duration	Runtime	Iterations
8	p7.2.m.txt	3384299186	40	122.66	-0.078	820
Vehicles	2	922621227	58	121.95	-0.078	821
Tmax	130	2135121195	35	123.72	-0.080	820
Nodes	102	3828514461	66	128.24	-0.077	820
		1129902229	68	129.30	-0.077	821
		3550564541	41	129.33	-0.078	820
		729300326	65	128.44	-0.080	822
		3168708657	0	0.00	-0.077	819
		4003079016	49	107.64	-0.078	820
		2155155557	64	129.50	-0.078	821
		Total	486	1120.79	-0.781	8204
FileNum	File	Seed	Cost	Duration	Runtime	Iterations
9	p7.3.h.txt	2035357603	27	48.43	-0.164	1227
Vehicles	3	2496560176	31	49.64	-0.164	1227
Tmax	53.3	606886726	29	52.83	-0.161	1227
Nodes	102	1975106483	52	52.40	-0.162	1227
		4245866971	0	0.00	-0.161	1226
		1339490695	21	50.99	-0.161	1226
		210730651	31	49.24	-0.171	1227
		2400379730	28	35.61	-0.166	1226
		4128479130	16	50.99	-0.164	1226
		2484002049	39	47.37	-0.169	1227
		Total	274	437.50	-1.642	12266
FileNum	File	Seed	Cost	Duration	Runtime	Iterations
10	p7.4.g.txt	1061735816	32	27.68	-0.277	1635
Vehicles	4	985734202	0	0.00	-0.286	1635
Tmax	35	1777622355	27	33.41	-0.286	1635
Nodes	102	3073591284	37	29.34	-0.296	1635

Critical Analysis of Results



Where:

- I,j,k are indexing the 10 runs
- Score/Runtime measures the efficiency
- Total Score is the sum of scores per run
- Total Runtim sums runtime per instance
- m,n,tmax are parameters related to the problem size:
    - m = number of vehicles
    - n = number of nodes (cities)
    - tmax = maximum allowed time per vehicle
- Duration is the total algorithm run time.

Intent:

- Measure relative efficiency between runs
- Rewards efficiency
- Heavy runs get penalized
- Normalize problem solutions (Larger problems will be slower)
- Normalize solutions and compare them to see which had the relative best cost solution (Solutions that have lower costs should be compared relatively)

Handling Missing Solutions:

In some runs, the algorithm failed to produce a valid solution. This can occur when the search process does not successfully construct a complete solution within the allowed parameters (such as tmax or iteration limits).

Possible reasons include:

- **Insufficient exploration:** A low tmax may limit the algorithm's ability to explore enough solution space.
- **Premature convergence:** The algorithm may converge too early on suboptimal or incomplete paths.
- **Random factors:** Due to the probabilistic nature of the algorithm and random seeding, some runs may inherently fail to find a valid solution.

These cases were noted separately and excluded from final performance calculations to maintain consistency across all evaluated instances.

Files	Performance
p3.2.a	8.40
p3.2.c	7.98
p3.3.a	8.65
p3.3.c	8.44
p3.4.s	6.63
p3.4.t	6.33
p7.2.a	26.97
p7.2.m	12.13
p7.3.h	17.50
p7.4.g	18.88
p7.4.q	17.20

Since the formula emphasizes score efficiency (score per second) and normalizes results based on instance size, larger problems naturally achieve higher performance scores.

The instance p7.2.a stands out with the highest performance, primarily due to a lower tmax, which resulted in significantly better outcomes on average. A general trend can be observed where lower tmax values correlate with improved performance, a pattern that is also noticeable across the 33-node problems.

The instances p3.2, p3.3, and p3.4 exhibit similar performance levels, while p7.2, p7.3, and p7.4 form another group with comparable performance.

Because runtime is factored into the performance calculation, the scaling behavior favors larger instances — the normalized performance of the 102-node problems is approximately three times better than that of the 33-node problems.

ConIcusion:

Lower tmax lead to a reduced runtime (with similar iterations) to an identical instance with different tmax. The lower tmax means the solutions converges quicker and results in a higher performance score.

ReadMe

https://github.com/amJohnnyma/COS314

References

- Blum, C. (2005). *Ant Colony Optimization: Introduction and Recent Trends*. *Physics of Life Reviews, 2*(4), 353–373.
- Colorni, A., Dorigo, M., & Maniezzo, V. (1991). *Distributed Optimization by Ant Colonies*. Proceedings of the European Conference on Artificial Life.
- Dorigo, M., & Gambardella, L. M. (1997). *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem*. *IEEE Transactions on Evolutionary Computation, 1*(1), 53–66.
- Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization*. MIT Press.
- Gendreau, M., Laporte, G., & Potvin, J. Y. (2002). *Metaheuristics for the Vehicle Routing Problem*. In *The Vehicle Routing Problem* (pp. 129–154). SIAM.
- Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.
- *Ant Colony Optimization for Vehicle Routing Problems*. Retrieved from https://drops.dagstuhl.de/storage/01oasics/oasics-vol014-atmos2010/OASIcs.ATMOS.2010.142/OASIcs.ATMOS.2010.142.pdf (Accessed April 13, 2025).
- *Vehicle Routing Problem*. Retrieved from https://en.wikipedia.org/wiki/Vehicle_routing_problem (Accessed April 13, 2025).
- *Heuristic Search Methods* (Open Textbooks). Retrieved from https://www.opentextbooks.org.hk/ditatopic/27149 (Accessed April 14, 2025).
- *WolframAlpha* (Used for formula verification). Retrieved from https://www.wolframalpha.com/ (Accessed April 26, 2025).
- Lecture notes