SF2568 Parallel Computations for Large-Scale Problems - Project

Parallel Ant Colony Optimization for the Travelling Salesman Problem

Nolwenn Deschand and Jonathan Haag

March 30, 2023

1 Problem description

The travelling salesman problem (TSP) is a famous problem where the goal is, given a list of cities and the distances between each pair of cities, to find the shortest possible route that visits each city exactly once and returns to the origin city. It is an NP-hard problem, and the exact solution is computionally expensive when the number of cities increases, with a complexity of $O(n!)$. It quickly becomes impossible to solve in a reasonable time and alternative ways to construct approximate solutions are necessary.

One approach to find a good, but not necessarily optimal solution to this problem is ant colony optimization (ACO). This method is inspired by the behavior of biological ants that deposit pheromones on the ground, for example, when searching for food. The traces then attract other members of the colony to take the same route and leave pheromones themselves. As pheromones evaporate over time, shorter paths will have a higher concentration as they are being used more frequently and consequently emerge as being favorable. The idea of ACO applied to the TSP is relatively straight forward: We send out a number of ants, each constructing a tour, that is to visit all cities and return to the origin. In each step the city to visit next is chosen in a probabilistic way depending on the distance between the current city and the cities that have not been visited yet as well as the corresponding pheromone trails. If a path between two cities has been chosen and is part of a good tour (measured by the total length of the tour), the pheromones get increased accordingly, i.e., the best paths will be associated to stronger pheromone trails which influence the next round of ants.

The exploration part of the ants can be parallelized, leading to major speed ups of the computations.

2 Description of our algorithm

2.1 General setup

The general sequential algorithm is given by the following pseudocode [1]:

Algorithm 1 The Ant Colony Optimization Metaheuristic

```
Set parameters, initialize pheromone trails
while termination condition not met do
    ConstructAntSolutions
    UpdatePheromones
end while
```

There are several versions of ACO algorithms, the most populars being Ant System (AS), MAX-MIN Ant System (MMAS) and Ant Colony System (ACS). We chose to implement the MMAS as it has proven to deliver good results while being relatively simple conceptually. Its characterizing elements, especially compared to the original AS, are that only the best ant updates the pheromone trails and that the value of the pheromone is bounded [1]. The ACS extends the ideas of MMAS by introducing a local pheromone update, which can help to diversify the explored paths. However, these local updates lead require increased communication which is not optimal regarding our goal of parallelizing the implementation.

We implemented the following formulas according to the description in [1]:

In the MAX-MIN version of the ACO, the pheromone τ_{ij}, associated with the edge joining cities i and j, is updated as follows:

$$\tau_{ij} \leftarrow [(1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}^{best}]_{\tau_{min}}^{\tau_{max}} \tag{1}$$

where ρ is the evaporation rate, and τ_{max} and τ_{min} are the upper and lower bounds imposed on the pheromone, respectively. The operator $[x]_a^b$ is defined as

$$[x]_a^b = \begin{cases} a & \text{if } x > a, \\ b & \text{if } x < b, \\ x & \text{otherwise,} \end{cases} \tag{2}$$

and $\Delta\tau_{ij}^{best}$ is

$$\Delta\tau_{ij}^{best} = \begin{cases} \frac{1}{L_{best}} & \text{if (i,j) belongs to the best tour,} \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Here, the best tour can refer to either the best global tour or the iteration best or a hybrid approach.

When an ant k is in the city i, the probability of visiting the city j is given by

$$p_{ij}^{k} = \begin{cases} \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{c_{il}\in N(s^p)}\tau_{il}^{\alpha}\eta_{il}^{\beta}} & \text{if } c_{ij} \in N(s^p), \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where $N(s^p)$ is the set of feasibles components, the edges (i,l) where l is a city not visited yet by the ant k.

The parameters α and β (pheromone_power and distance_power in our code) control the relative importance of the pheromone versus the heuristic information η_{ij}, which is given by:

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{5}$$

with d_{ij} the distance between the cities i and j, in our implementation the euclidean distance.

Notice that in the original publication [3], the authors propose a variety of additional measures to enhance the algorithm's ability to find optimal solutions some of which are significantly more difficult to implement and go beyond the scope of this project. Section 4 gives a more detailed overview of our implementation and how it differs from the full-featured version.

2.2 Parallelization strategy

Several parallelization strategies are possible for this problem. We chose to implement the coarse-grained master slave model. In this approach, a master process manages the global information (pheromone matrix, best solution found, etc.) and also controls a group of slave processes that perform subordinated tasks, related to the ACO search space exploration. Each process controls a group of ants, that simultaneously explore the graph and construct tours. After each iteration, the slaves send their best local solution to the master that updates the global solution as well as the pheromone matrix and broadcasts the latter to each slave for the next iteration. Master-slave parallel ACO implementations have been quite popular in the research community, mainly due to the fact that this model is conceptually simple and easy to implement [2], which makes it a good candidate for us.

Adding the communication steps to Algorithm 1, the parallel algorithm reads:

Algorithm 2 The Ant Colony Optimization Metaheuristic, Parallel version

```
Set parameters, initialize pheromone trails
while termination condition not met do
    ConstructLocalAntSolutions
    GatherLocalSolutions
    UpdateGlobalSolution
    UpdatePheromones
    BroadcastPheromones
end while
```

Notice that the computation of the pheromone matrix update and the probability to visit a city is the same for the sequential and parallel algorithm. As noted before implementation details for the individual steps are discussed in Section 4.

3 Theoretical performance estimation

We decided to evaluate the theoretical performance of our algorithms excluding the reading of the input file, the initialization of the parameters and and the writing of results into a file. The reason for that is that, as the initialization happens only once the associated effort is negligible compared to the computation time, especially increasing number of cities. Further, file reading and writing is the most un-optimized part of our code, and while the former is also only performed once, the latter might be obsolete depending on the context (maybe only the final solution is of interest and thus the intermediate results are not needed).

Please also note that the proposed model is rather conservative as we mostly use a worst-case approximation of the runtime. This specifically concerns certain parts of the code are executed rarely, for example, updating the best global solution (often in the beginning but less frequently as we get closer to convergence). However, in our model we simply consider these parts to be executed at all times.

3.1 Sequential algorithm

We start by calculating the execution time for the best serial algorithm, which can be summarized as

$$T_s^* = n_{iter}(n_{ants} \cdot t_{constructTour} + t_{updatePheromones}), \tag{6}$$

with

$$\begin{aligned} t_{constructTour} &= 3 \cdot n_{cities}^2 \cdot t_\alpha + n_{cities} \cdot t_\alpha \\ t_{updatePheromones} &= 2 \cdot n_{cities} \cdot t_\alpha + n_{cities}^2 \cdot t_\alpha \end{aligned} \tag{7}$$

where n_{iter} denotes the number of iterations, n_{ants} the number of ants, n_{cities} the number of cities, $t_{constructTour}$ the time to construct a tour for one ant, $t_{updatePheromones}$ the time to update the pheromone matrix, and t_α the time for an arithmetic operation.

Overall this leaves us with

$$\begin{aligned} T_s^* &= n_{iter} \cdot t_\alpha (n_{cities}^2 (3 \cdot n_{ants} + 1) + n_{cities} \cdot (n_{ants} + 2)) \\ &= O(n_{iter} \cdot n_{cities}^2 \cdot n_{ants} \cdot t_\alpha). \end{aligned} \tag{8}$$

As for the parallel algorithm, the running time for the best serial algorithm mostly depends on three parameters: the number of cities, the number of iterations and the number of ants. Increasing the number of ants or the number of cities will increase the computation time of an iteration.

3.2 Parallel algorithm

3.2.1 Computation time

Moving on to the parallel version, we begin with the computation time

$$T_{comp} = n_{iter} \cdot (L \cdot t_{tour} + t_{keepBestPath} + t_{updatePheromones}) \tag{9}$$

with L the number of ants per process $\in \{ \lfloor \frac{n_{ants}}{P} \rfloor, \lfloor \frac{n_{ants}}{P} \rfloor + 1 \}$, and $t_{constructTour}$ as well as $t_{updatePheromones}$ as in equation 7. Further, it contains the time for the comparison of the results from the different processes

$$t_{keepBestPath} = P \cdot t_\alpha. \tag{10}$$

Therefore, we obtain

$$\begin{aligned} T_{comp} &= n_{iter} \cdot t_\alpha [n_{cities}^2 (3 \cdot L + 1) + n_{cities} (L + 2) + P] \\ &= O(n_{iter} \cdot n_{cities}^2 \cdot L \cdot t_\alpha). \end{aligned} \tag{11}$$

3.2.2 Communication time

For the communication time, we get

$$T_{comm} = n_{iter} (t_{gatherPathsLength} + t_{getBestPath} + t_{broadcastPheromones}) \tag{12}$$

with

$$\begin{aligned} t_{gatherPathsLength} &= log(P)(t_{startup} + 1 \cdot t_{data}), \\ t_{getBestPath} &= log(P)(t_{startup} + 1 \cdot t_{data}) + t_{startup} + n_{cities} \cdot t_{data}, \\ t_{broadcastPheromones} &= n_{citites} \cdot log(P)(t_{startup} + n_{cities} \cdot t_{data}), \end{aligned} \tag{13}$$

where we use the startup time $t_{startup}$ and the time to send an individual data package t_{data}.

Therefore, we have

$$\begin{aligned} T_{comm} = n_{iter}[&2 \cdot log(P)(t_{startup} + 1 \cdot t_{data}) \\ &+ (n_{citites} \cdot log(P) + 1)(t_{startup} + n_{cities} \cdot t_{data})]. \end{aligned} \tag{14}$$

3.2.3 Parallel time

Overall this yields

$$\begin{aligned} T_P &= T_{comp} + T_{comm} \\ &= n_{iter} \cdot [t_\alpha(n^2_{cities}(3 \cdot L + 1) + n_{cities}(L + 2) + P) \\ &\qquad + 2 \cdot log(P)(t_{startup} + 1 \cdot t_{data}) \\ &\qquad + (n_{citites} \cdot log(P) + 1) \cdot (t_{startup} + n_{cities} \cdot t_{data})] \\ &= O(n_{iter}(n^2_{cities} \cdot L \cdot t_\alpha + n_{cities} \cdot log(P)(t_{startup} + n_{cities} \cdot t_{data}))). \end{aligned} \tag{15}$$

Again, the running time for the parallel algorithm mostly depends on the number of cities, the number of iterations and the number of ants. Notice that the communication time is rather small compared to the computation time and that the latter is only marginally increased compared to sequential algorithm.

3.3 Speedup

We can calculate the speedup with the usual approach and simplify directly to

$$\begin{aligned} S_P &= \frac{T^*_s}{T_P} \\ &= O\left(\frac{n^2_{cities} \cdot n_{ants} \cdot t_\alpha}{n^2_{cities} \cdot L \cdot t_\alpha + n_{cities} \cdot log(P)(t_{startup} + n_{cities} \cdot t_{data})}\right) \\ &= O\left(\frac{n_{ants} \cdot t_\alpha}{\frac{n_{ants}}{P} \cdot t_\alpha + log(P)(\frac{t_{startup}}{n_{cities}} + t_{data})}\right). \end{aligned} \tag{16}$$

As one would expect using more processes on a smaller problem (in terms of the number of cities) leads to a decrease in speedup, while for larger problems the startup cost are less impactful (in the limit for $n_{cities} \to \infty : S_P = O\left(\frac{n_{ants} \cdot P}{n_{ants} + log(P) \cdot P}\right)$ as $t_\alpha \approx t_{data}$).

4 Implementation details

4.1 Code structure, MPI, and timing

To make our code more readable, we moved the main functions that we use in the sequential and parallel versions of our ACO into a new file called aco_functions.c. This file includes the functions for picking the next city, update the pheromone matrix, let the ants do a tour, read the input parameters and write the results in a file.

In terms of using MPI for the parallel version, we use **MPI_Gather** to collect all the local solutions in an array on the process with rank 0 over which we can then loop to find the rank of the process with the iteration-best solution. Only if that solution is a new global best, it is broadcast to the other processes using **MPI_Bcast**. We use a barrier to ensure that all processes wait until the master process has completed the comparison and spread the word, and only then send the found path from the corresponding process to the master (if necessary at all). Finally, the master updates the pheromone matrix and broadcasts it to all processes, again using **MPI_Bcast**.

When timing our code, we removed all the parts to write the results into a file, to keep only the computation and communication time. The initialization of the parameters is included in the timing, but is very small compared to the core of the algorithm.

4.2 Problem instances and parameters

The data that we use comes from the Traveling Salesman Problem Library (TSPLIB) [4], a database from Uni Heidelberg with problem instances for the TSP that also contains optimal solutions to compare our results with. We chose to work on 3 instances of different sizes: berlin52, pcb442 and pr2392. Table 1 shows the characteristics of the 3 instances.

Problem	**Number of cities**	**Optimal path length**
berlin52	52	7542
pcb442	442	50778
pr2392	2392	378032

Table 1: The three considered TSP instances with the number of cities and known optimal solution.

Parameter tuning is really important in order to get good results. Bad parameters can result in very different path lengths, sometimes far from the optimal due to too little exploration of the search space. Table 2 shows the used parameters for the experiments, following the recommendations from [3] for the most

part and tweaking where necessary.

Problem	**# iterations**	**# ants**	ρ	α	β	τ_{min}	τ_{max}
berlin52	2000	100	0.3	1	10	$\frac{1}{(0.3*8000)}$	$\frac{\tau_{max}}{(2*52)}$
pcb442	1000	150	0.3	1	10	$\frac{1}{(0.3*60000)}$	$\frac{\tau_{max}}{(2*422)}$
pr2392	1500	256	0.3	1	10	$\frac{1}{(0.3*450000)}$	$\frac{\tau_{max}}{(2*2392)}$

Table 2: Used parameters for the numerical experiments on Dardel.

In all cases, the pheromone trails are initialized with $\tau_{init} = \tau_{max}$ as suggested by authors of the original MMAS paper. In contrast to them, we chose relatively large colony sizes (they typically use $n_{ants} = 25$) since testing $P > n_{ants}$ makes no sense given our parallelization approach. Also, we used a fixed number of iterations instead of a termination condition based on the last update of the global solution as an indicator for convergence (basically saying that if the global solution has not been updated for a number of iterations, the algorithm has converged). Another important distinction is that they extend the basic MMAS by using a 3-opt local search that modifies up to three connections of a found solution to improve it. Since, as they note, this requires significantly more implementation effort and applies to both the sequential as well as the parallel version, i.e., does not affect the parallelization, we decided to omit this strategy despite the improvements it promises. Further, they use a candidate list of length 20 to pick the next city instead of all unvisited cities and apply a hybrid update rule in which either the global and iteration best solution is used for the pheromone update based on a heuristic. It should be noted that while our algorithm does not perform as well as their full-fletched version, the obtained results as shown in section 5 match their findings before adding some of these refinements quite well.

5 Results of a typical problem run

Table 3 shows the best solutions found with our implementation compared to the optimal solutions for the three considered instances.

Problem	**Optimal path length**	**Best found path length**	**Relative error**
berlin52	7542	7542	0.0%
pcb442	50778	52042	2.5%
pr2392	378032	414083	9.5%

Table 3: Best found paths and corresponding relative error for the three problems.

Generally, our results are close to the optimal solutions, where for the small problem instance with only 52 cities the algorithm is able to find the best solution, while for growing problem size it deviates more and more. As stated

before this aligns with the findings of the MMAS authors prior to introducing the 3-opt local search. As a result of the increased running time, which will be discussed in the next section, even the basic parameter tuning becomes much harder for the bigger problems.

To go more into the details, let us look at the results for the small problem, berlin52. Figure 1 shows the found and optimal path, respectively. Notice that they start at the same origin (around (600,600)) and complete the same tour but in different directions. This obviously does not affect the solution, but is interesting the observe.

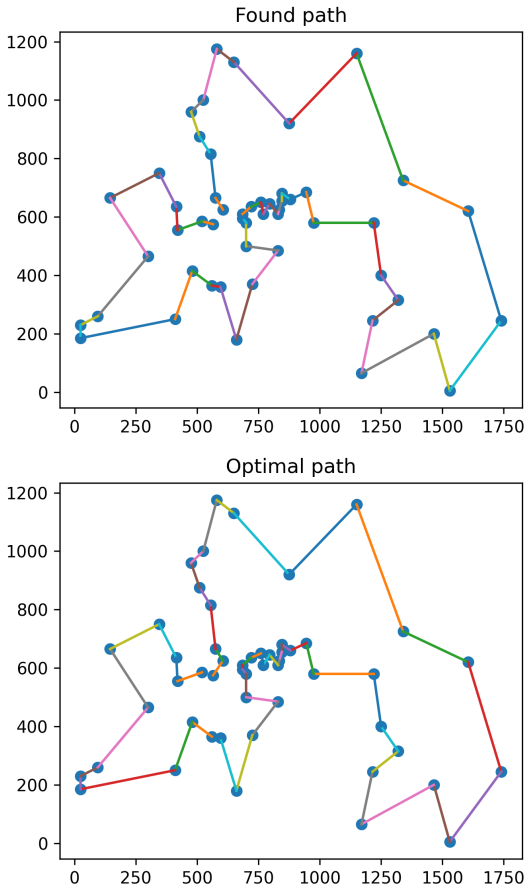

Figure 1: Found (top) and optimal (bottom) path for the TSP berlin52. The different colored connections are used to highlight the different direction of the found paths.

Another interesting thing to look at is the length of the best path over the iterations, i.e., convergence as shown in Figure 2. At the beginning, better paths are often found so that the length of the current best path decreases quickly. This occurs more and more rarely later on, and the last small improvement that we get is around the 400th iteration after which the optimal solution is found. The red line is the length of the optimal path for the problem berlin52.

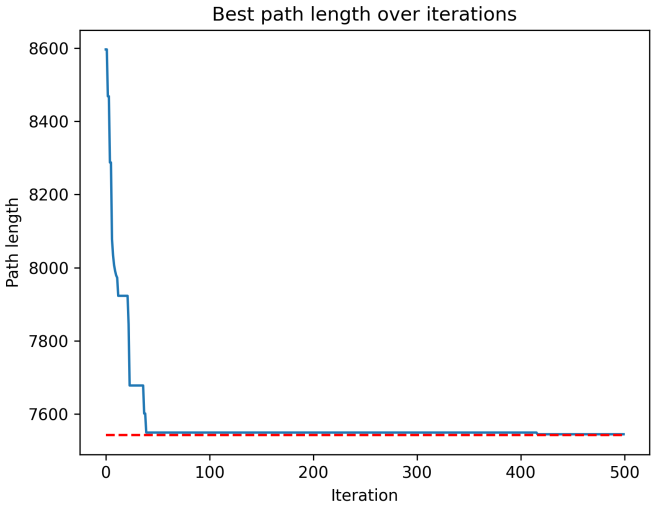

Figure 2: Convergence of our algorithm (blue) to the best solution (red) over the course of the first 500 iterations for berlin52.

Finally, we can visualize the findings after each iteration by constructing a fully connected graph with the cities as vertices and edge weights according to the pheromone trails. Figure 3 does exactly that for a few selected iterations and additionally shows the current global best path at that point of the optimization procedure. We can nicely see the uniform initialization $\tau_{init} = \tau_{max}$ before any tour is constructed and then continuously evaporating trails. Notice that, as indicated by the convergence plot, we are obtain a relatively good solution after only a few dozen iterations after which only small tweaks happen.

6 Experimental speedup investigation

We tested the algorithm for the three problem instances on Dardel using various number of processes P and one and four nodes ($N = \{1, 4\}$) to compare the raw runtimes and speedups among not only the different sized problems but also our theoretical approximation.

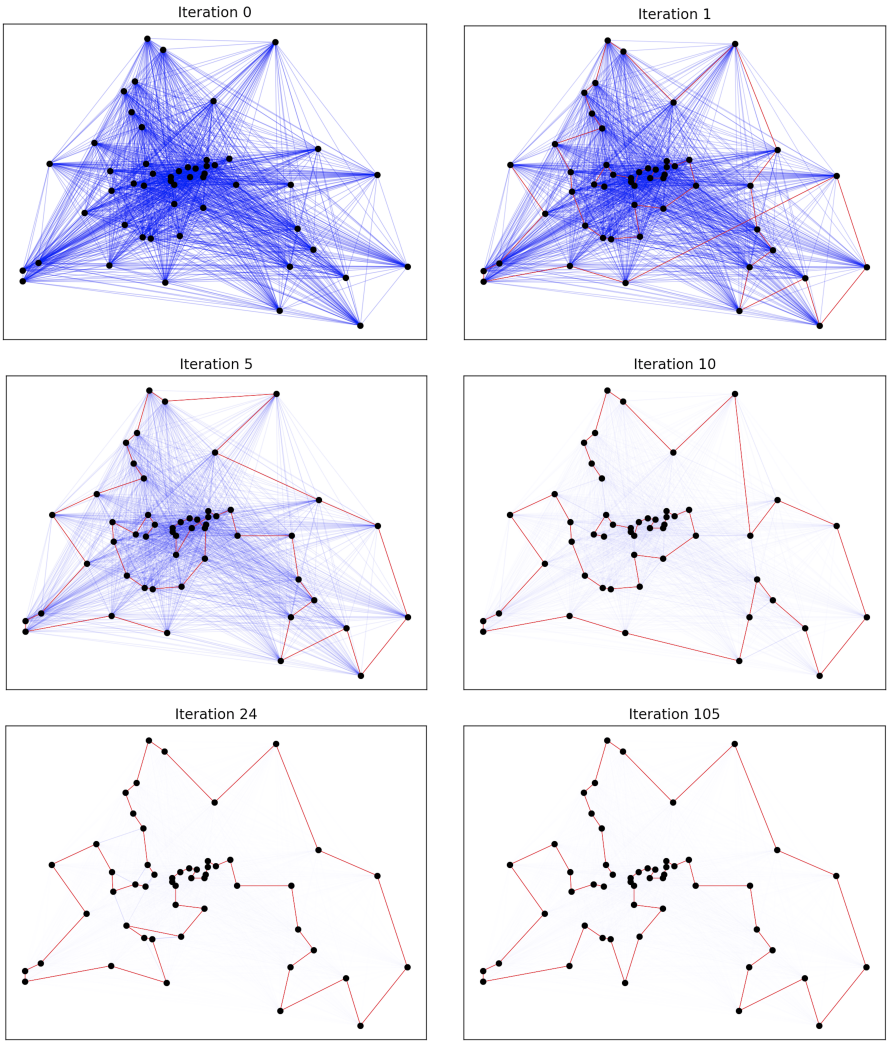

Figure 3: Intermediate best paths and pheromone trails for berlin52.

6.1 Runtimes

First, let us look at the obtained runtimes for berlin52 as summarized in Table 4 and plotted in Figure 4. We can observe a nice improvement of the runtime for increasing P as well as consistently higher runtimes when using four nodes due to the increased communication time. Lastly, the simplified theoretical estimation seems to match those findings quite well too.

	$P=1$		$P=2$	$P=4$	$P=8$	$P=16$	$P=32$
$N=1$	10.47	10.49	5.31	2.71	1.47 1	0.88	0.59
$N=4$	-		-	3.18	1.99	1.34	1.11

Table 4: Runtimes for berlin52 in seconds. For $N = P = 1$ we state the results for the sequential and the parallel version of the algorithm.

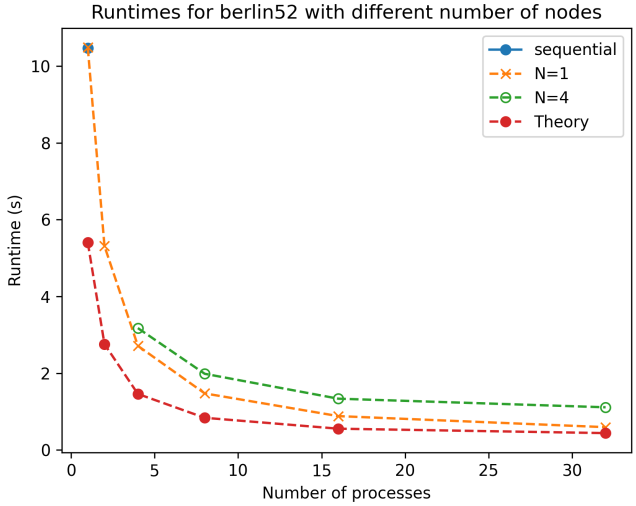

Figure 4: Obtained raw runtimes for berlin52 for different number of processes and nodes. Notice the comparison with the runtime of the sequential algorithm for $P = 1$ and the theoretical estimate $O(n^2_{cities}(\frac{n_{ants}}{P} + log(P)))$ which can be derived from Equation 15.

We can consider the corresponding results for the other two problems, starting with the medium-sized pcb442, as in Table 5 and Figure 5. Here, we observe a similar trend as before although our theoretical model now seems to overestimate the runtimes for increasing P.

	$P=1$		$P=2$	$P=4$	$P=8$	$P=16$	$P=32$	$P=64$	$P=128$
$N=1$	554.6	559.5	282.4	143.3	72.6 1	39.6	21.3	14.3	12.0
$N=4$	-		-	144.6	74.2	41.0	22.5	16.1	12.6

Table 5: Runtimes for pcb442 in seconds. For $N = P = 1$ we state the results for the sequential and the parallel version of the algorithm.

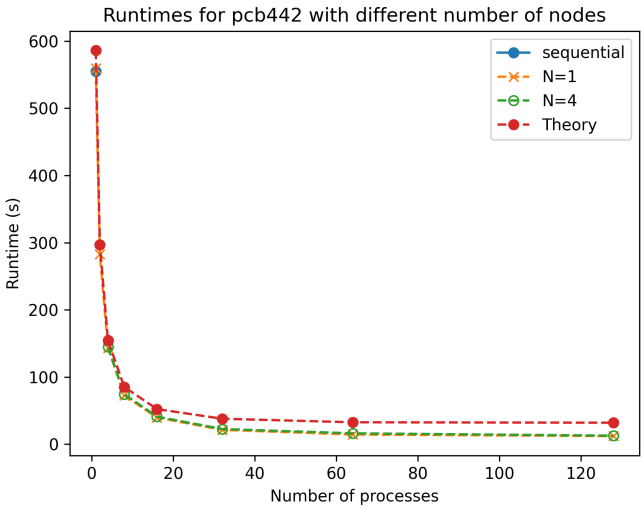

Figure 5: Obtained raw runtimes for pcb442 for different number of processes and nodes. Notice the comparison with the runtime of the sequential algorithm for $P = 1$ and the theoretical estimate $O(n^2_{cities}(\frac{n_{ants}}{P} + log(P)))$ which can be derived from Equation 15.

Lastly, we can look at the results for the very large problem pr2392 as shown in Table 6 and Figure 6. Here, we only test for large P and do not compare to the sequential version as the computational effort would be too large. Also, we use a slightly modified theoretical approximation where we add a factor of ten to part that estimates the computation time as the worst-case approach significantly overstates the influence of the communication. This way we can also counteract the simplifications introduced during the modeling of the computation time where we used a order-of estimate instead of an exact model for the number of required computation steps.

	$P = 32$	$P = 64$	$P = 128$	$P = 256$
$N = 1$	1422.5	753.6	467.3	-
$N = 4$	1447.9	768.0	442.4	275.7

Table 6: Runtimes for pr2392 in seconds.

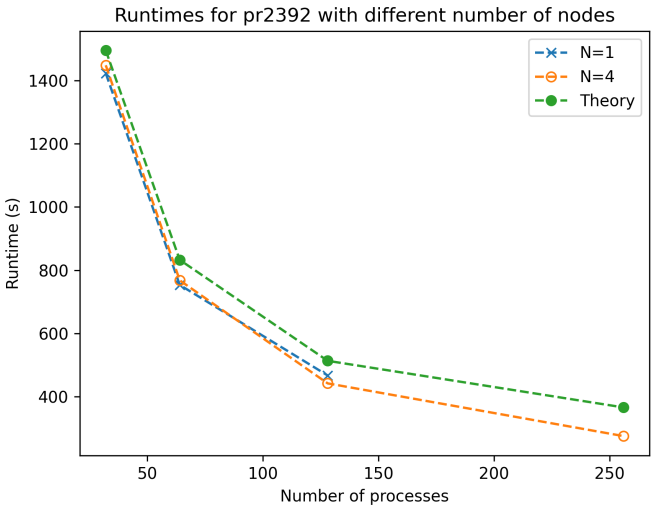

Figure 6: Obtained raw runtimes for pr2392 for different number of processes and nodes. Notice the comparison with the theoretical estimate $O(n_{cities}^2(10 \cdot \frac{n_{ants}}{P} + log(P)))$ which can be derived from Equation 15.

6.2 Speedup

To get a better feeling for the performance of our algorithm, the used parallelization strategy, and the derived theoretical model, we can compute the speedups for the two smaller problems with the usual approach $S_P = \frac{T_s^*}{T_P}$. For berlin52, we can see that the experimental findings match our considerations in Section 3 quite well as the theoretically derived curve lies in between the results on one and four nodes, see Figure 7. This indicates that our model works reasonably well for small problems, slightly underestimating the parallel performance on one node and overestimating it for $N = 4$.

As depicted in Figure 8, the experiments for the second problem yield better speedups than expected with the theoretical model regardless of N. A few reasons can explain these results: First of all, our model is simplified in several ways introducing errors along the way. As said before, it is also rather conservative (taking the worst case that does not occur all the time) implying that our actual runtime should often be lower, especially on bigger problems. This checks out with our findings. Furthermore, notice that without any communication time, the speedup should be around P. It seems that the used model overestimates communication relative to computation time, which ultimately leads to increased errors as the number of processes increases.

Generally though, the results for both cases indicate that the parallel algorithm is significantly faster than its sequential counterpart, which confirms our choice of the used approach. The efficiency $\frac{S_P}{P}$ was above 80% for the most part when

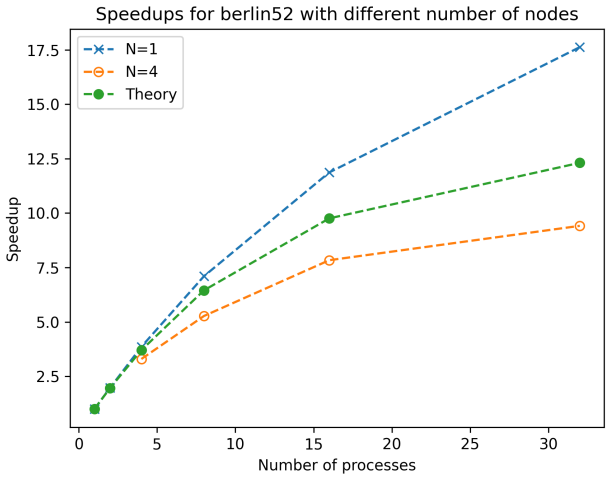

Figure 7: Obtained speedups for berlin52 for different number of processes and nodes. Notice the comparison with the theoretical estimate $O(\frac{n_{ants} \cdot P}{n_{ants}+P \cdot log(P)})$ which can be derived from Equation 16.

$N = 1$ before a decline sets in as we increase P (this happens earlier for the smaller problem). For $N = 4$, the efficiency is generally lower but still not bad, for the most unfavorable setup $N = 4, P = 128$ it turns out to be around 34%.

Finally, note that for the biggest problem no speedup can be provided as no baseline for $N = P = 1$ is available.

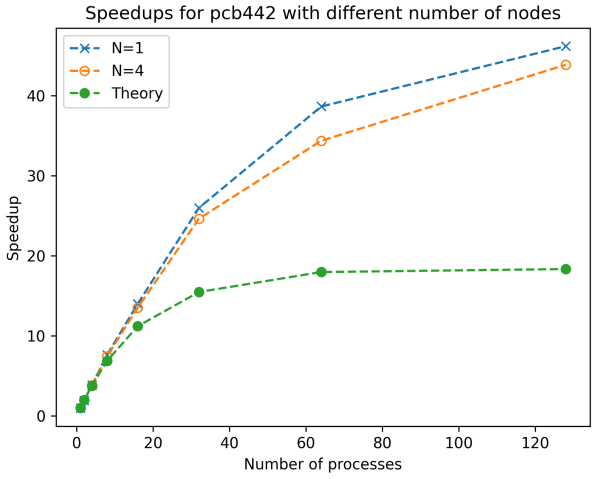

Figure 8: Obtained speedups for pcb442 for different number of processes and nodes. Notice the comparison with the theoretical estimate $O(\frac{n_{ants} \cdot P}{n_{ants}+P \cdot log(P)})$ which can be derived from Equation 16.

7 Conclusions

In this project, we considered a version of ant colony optimization to approximately solve the traveling salesman problem. For this, we implemented the well-established MAX-MIN ant system and derived a parallelization strategy based on a master-slave approach. To test this, we conducted multiple numerical experiments on Dardel for problem instances of varying size and compared the obtained solutions with the known optima. Further, we compared the obtained runtimes and associated speedups with our theoretical considerations regarding the performance of the parallel algorithm. This showed that while our estimate was reasonably accurate for smaller problems, it generally overstated communication cost especially when using large P. The parallel algorithm itself showed great speedups in the tested scenarios.

The parallelization approach was chosen due to its simplicity and promising potential in terms of performance which, to a large extent, has proven to be true. Nonetheless, different strategies exist and one might be interested in comparing their scalability. In that context, it could also be interesting to refine the used theoretical model even though this would suggest a more complicated derivation process. On the other hand, the obtained results indicated that the current estimate only requires a few modifications to deliver good approximations of the runtime and speedup.

Last but not least, it should again be noted that the base version of the algorithm used here has some distinct limitations when it comes to findings good solutions for large TSP instances. Therefore, extending both, the sequential as well as the parallel version, to find more robust solutions would be good next step to extend this project.

References

[1] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. "Ant colony optimization". In: *IEEE Computational Intelligence Magazine* 1.4 (2006), pp. 28–39. DOI: 10.1109/MCI.2006.329691.

[2] Martin Pedemonte, Sergio Nesmachnow, and Hector Cancela. "A survey on parallel ant colony optimization". In: *Applied Soft Computing* 11.8 (May 2011), 5181â5197. DOI: 10.1016/j.asoc.2011.05.042.

[3] Thomas Stützle and Holger H Hoos. "MAX–MIN ant system". In: *Future generation computer systems* 16.8 (2000), pp. 889–914.

[4] *TSPLIB*. University Heidelberg. URL: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.