Analyzing the Open Shop Scheduling Problem Using the Ant Colony Optimization Algorithm

Field of Computer Science
Saeed Sabzeh

Supervisor:
Dr. Jalil Rashidi-Nia

September 2024

Introduction

The Open Shop Scheduling Problem (OSSP) is a well-known and important topic in the field of engineering due to its wide industrial applications. OSSP, as an NP-complete problem, has its own challenges and encompasses a broader solution space compared to other scheduling problems, such as Job Shop Scheduling and Flow Shop Scheduling.

Metaheuristic algorithms, such as Ant Colony Optimization (ACO), have garnered significant attention due to their high efficiency in solving combinatorial optimization problems [9]. In this project, we have explored and solved the OSSP using the Ant Colony Optimization algorithm (ACO). Inspired by the social behavior of ants in nature [3], the ACO algorithm is regarded as a powerful tool for solving scheduling problems due to its high capability in searching and finding optimal solutions for complex problems.

Given the complexity and the broad search space of the OSSP, applying the ACO algorithm can significantly improve the quality and speed of finding optimal solutions. In this report, we will introduce the ACO methodology, explain its implementation for solving the OSSP, and analyze the obtained results.

Open Shop Scheduling Problem

The Open Shop Scheduling Problem (OSSP) is a combinatorial optimization problem where a set of jobs $(J=\{J_1,J_2,\ldots,J_n\})$ must be processed by a set of machines $(M=\{M_1,M_2,\ldots,M_m\})$ [8]. Each job J_i consists of m operations O_{ij}, where each operation requires processing on machine M_j.

Problem Definition:

- **Inputs:**
  - A set of n jobs: $J=\{J_1,J_2,\ldots,J_n\}$
  - A set of m machines: $M=\{M_1,M_2,\ldots,M_m\}$
  - P_{ij} : The processing time of operation O_{ij}, where job J_i is processed on machine M_j.
- **Constraints:**
  - Each machine Mj can process only one operation at any given time.
  - Each job Ji can execute only one operation at a time.
  - The order of executing operations Oij on machines is arbitrary and does not need to follow a specific sequence.
- **Objective:**
  Minimizing the makespan (C_{max}): The primary objective in most versions of this problem is to minimize the total completion time, defined as:

- $C_{max} = max_{i,j}\{C_{i,j}\}$ where C_{ij} represents the completion time of operation O_{ij}.
- The Open Shop Scheduling Problem (OSSP) is more complex than other scheduling problems, such as Flow Shop or Job Shop, due to its flexibility in the sequence of operations. Computationally, OSSP is recognized as an NP-hard problem. For this reason, solution methods for this problem often involve heuristic, metaheuristic, and approximate algorithms.

	Machine 0	Machine 1	Machine 2	Machine 3	Machine 4
Job 0	80	3	65	98	9
Job 1	79	69	51	51	45
Job 2	65	37	75	53	91
Job 3	39	95	58	49	76
Job 4	54	29	27	68	93

Table 1: An OSSP consisting of 5 machines and 5 jobs [8]. Each number represents the time required to complete the corresponding job on the given machine.

Ant Colony Optimization Algorithm

Ant Colony Optimization (ACO) is a metaheuristic algorithm inspired by the natural behavior of ants in finding the shortest path between their nest and food sources. This algorithm is used to solve combinatorial optimization problems, such as the Traveling Salesman Problem (TSP), scheduling, and routing problems [3][5].

How ACO Works:

1. **Inspired by Ant Behavior:**
    - In nature, ants move randomly while searching for food. Once they find food, they deposit a chemical substance called "pheromone" on their way back to the nest.
    - Other ants are more likely to follow paths with a higher amount of pheromone compared to other paths [3].
    - Over time, shorter and more efficient paths accumulate more pheromone, while less efficient paths are gradually abandoned.
2. **Simulation in the Algorithm:**
    - In ACO, several "artificial ants" are created, which move through the problem space. Each ant makes a move at each step and explores a path.
    - The path selection is based on probabilities influenced by the amount of pheromone on the paths and other problem-specific parameters.

   - After completing a path, each ant deposits pheromone on its traversed path proportional to the quality of the path (e.g., shorter distance or lower cost).
   - Through repetition, optimal paths are identified, and ants gradually prefer these paths.

3. **Pheromone Updates:**
   - Pheromones gradually evaporate to prevent premature convergence to suboptimal local solutions.
   - Pheromone levels are updated over time based on the quality of the paths.

Implementing ACO for Open Shop Scheduling

To use the Ant Colony Optimization algorithm for solving the Open Shop Scheduling Problem (OSSP), the problem must be modeled as a mathematical graph. In this model:

- Each operation O_{ij}, where i represents the job and j represents the machine, is treated as a node v_{ij} in a fully connected directed graph $G=(V,E)$ [4][6].

In this graph:

- $V=\{v_{ij}\}$: The set of nodes, representing the operations.
- $E=\{(v_{ij},v_{kl})\}$: The set of directed edges, representing the precedence between operations.

The objective is to find a path in this graph that satisfies all timing and precedence constraints while minimizing the makespan (C_{max}), the total completion time of all jobs.

Each ant k in the ant system traverses a path π_k in the graph, representing a possible sequence of operations. The probability of ant k selecting the edge (v_{ij},v_{kl}) is based on:

- The pheromone level $\tau_{ij,kl}$ on the edge.
- A heuristic function $\eta_{ij,kl}=1/t_{kl}$, defined as the inverse of the processing time t_{kl} for

$$p_{ij}^{k}(t)=\frac{\left[\tau_{ij}(t)\right]^{\alpha}\cdot\left[\eta_{ij}\right]^{\beta}}{\sum_{l\in J_i^k}\left[\tau_{il}(t)\right]^{\alpha}\cdot\left[\eta_{il}\right]^{\beta}},$$

  operation O_{kl}.

The probability is influenced by two parameters, α and β, which control the impact of pheromone levels and heuristic information, respectively.

Pheromone Update Formula:

After all ants complete their paths, the pheromone levels on the edges are updated as follows [3]:

$$\tau_{ij}(t) \leftarrow (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t) \, ,$$

$$\text{where } \Delta\tau_{ij}(t) = \textstyle\sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t),$$

where:

- ρ: The pheromone evaporation rate $(0 \le \rho \le 1)$.
- $\Delta\tau_{ij}^{k}$: The amount of pheromone deposited by ant k on the edge V_{ij}, defined as:

$$\Delta\tau_{ij,kl}^{k} = \begin{cases} \frac{Q}{C_k} & if\ ant\ k\ traversed\ the\ edge\ (v_{ij}, v_{kl}) \\ 0 & otherwise \end{cases}$$

Here, C_k is the total completion time for path π_k traversed by ant k, and Q is a constant [4].

Pseudocode for the ACO Algorithm:

Algorithm 1: Ant Colony Optimization (ACO)

1. **Initialize Pheromones**
   - For each pair of operations (i,j): $\tau_{ij}(0)=\tau_0$
2. **Main Loop** (for each generation):
   - **For each ant**:
     - Initialize the ant's solution as an empty list.
     - Set the **current operation** to the starting operation.
     - **While remaining operations exist**:
       1. Calculate the probabilities for each remaining operation.
       2. Select the next operation based on the computed probabilities.

            3. Add the selected operation to the ant's solution.
            4. Update the current operation to the selected operation.
        - **Evaluate the solution**:
            - Calculate the makespan for the ant's solution.
            - Store the makespan in the solution costs array for the ants.
    - **Update the Best Solution**:
        - Identify the ant with the minimum makespan.
        - If this solution is better than the overall best solution:
            - Update the best solution and its cost.
    - **Update Pheromones**:
        - Evaporate pheromones on all edges.
        - For each pair of operations (i,j) in the best solution:
            - Increase pheromone τ_{ij} as a reward.

3. **Local Search Optimization**:
    - Perform local search on the paths.
    - If the optimized solution is better than the overall best solution:
        - Update the best solution and its cost.
4. **Output the Best Solution and Its Cost**:
    - Return the best solution and the associated cost.

Algorithm 2: Create Solution (create_solution)

1. Each ant starts with an empty path π.
2. While the length of the path ($|\pi|$) is less than the total number of jobs n:
    - Select an operation o_j not yet added to π.
    - Add o_j to π.
3. Add the cost of the new operation to the total cost of π.
4. Return π.

Algorithm 3: Calculate Makespan (calculate_makespan)

1. Initialize scheduling lists for each machine as empty.
2. For each operation in the solution:
   - Identify the job and machine of the current operation.
   - Retrieve the execution time of the operation.
   - Set the operation as not executed.
   - While the operation is not executed:
     - For each other machine:
       - Check if the current job is being executed on another machine.
       - If the job is being executed:
         - Calculate the current machine's time and the job's end time on the other machine.
         - If scheduling conditions are suitable for this machine:
           - Set the operation as executed.
     - If the operation is executed:
       - Update the schedule for the current machine and mark the operation as complete.
     - Otherwise, make the machine wait for the operation.
3. Return the maximum length of all scheduling lists as the makespan.

Algorithm 4: Select Next Operation (select_next_operation)

- Input:
  - current_operation_index: Index of the current operation.
  - remaining_operations: List of unvisited operations.

1. Compute probabilities for selecting the next operation using the probability mechanism.
2. Select an operation from remaining_operations based on the computed probabilities.
3. Return the selected operation.

Algorithm 5: Calculate Operation Probabilities (calculate_operation_probabilities)

1. Input:
   - current_operation_index: Index of the current operation.
   - remaining_operations: List of unvisited operations.
2. Compute the number of remaining operations.
3. For each operation in remaining_operations:
   - Retrieve the pheromone value τ from the pheromone matrix.
   - Calculate η:

$$\eta = \begin{cases} 1 & \text{if it is the last operation} \\ \dfrac{1}{t_{currnt}} & \text{otherwise} \end{cases}$$

   - Compute the probability of selecting the operation using τ and η.
4. Normalize the probabilities so that their sum equals 1.

$$p_{ij}^{k}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[\eta_{ij}\right]^{\beta}}{\sum_{l \in J_i^k} \left[\tau_{il}(t)\right]^{\alpha} \cdot \left[\eta_{il}\right]^{\beta}},$$

5. Return the normalized probabilities.

Algorithm 6: Reward Pheromone Values (rewards(s, k))

- For all pairs of operations o_i and o_j in path s:
  - Increase pheromone τ_{ij} by K/C_s .

Algorithm 7: Update Pheromone (update_pheromone)

1. Evaporate pheromone values in the pheromone matrix based on **ρ**.

$$\text{where } \Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t), \,t) \,,$$

2. Reward the best current solution:
    - Increase pheromone values using the reward function for best_solution and its associated reward.
3. Reward the overall best solution:
    - Increase pheromone values using the reward function for ant_solutions[i_{best}] and its associated reward.

Key Parameters for Optimizing the Performance of the ACO Algorithm

To optimize the performance of the Ant Colony Optimization (ACO) algorithm, setting the correct parameters is essential. Below are explanations of the key parameters that significantly affect the algorithm's performance:

- **Alpha (α):**
  This parameter determines the importance of pheromone (τ) in the ants' path selection. The higher the value of alpha, the more influence pheromone has on the ants' decision-making process.
- **Beta (β):**
  This parameter indicates the impact of the heuristic function (θ) in path selection. A higher beta value signifies more emphasis on the heuristic information in determining the optimal path.

Typically, these two parameters are set to equal values, usually 1. This setting allows the ants to be influenced equally by both pheromone and exploration factors. Generally, this helps the ants make balanced decisions and prevents over-reliance on one factor.

- **Current_Best_Solution_Reward:**
  This parameter defines the reward given to the best solution found in the current iteration (Iteration_Best). Rewarding these solutions encourages exploration and the discovery of new solutions in the search space.
- **Overall_Best_Solution_Reward:**
  This parameter defines the reward given to the best overall solution found so far (Best_So_Far). Rewarding this solution reinforces exploitation and utilizes previously successful solutions.

Common values for these parameters, such as 2/3 for Overall_Best_Solution_Reward and 1/3 for Current_Best_Solution_Reward, help the algorithm maintain a balance between exploration and exploitation. This balance ensures the algorithm effectively explores the search space while benefiting from previous experiences.

- **Rho (ρ):**
  The ρ parameter determines the evaporation rate of pheromones on previously traveled paths. This parameter significantly influences the convergence speed of the algorithm. If the value of ρ is very small, pheromones evaporate slowly, which may cause poor convergence, especially if the number of generations is insufficient. On the other hand, if ρ is too large, pheromones evaporate quickly, leading to fast convergence to local solutions and potentially missing better solutions. Choosing an appropriate value for ρ requires balancing the algorithm's convergence without disregarding good solutions.
- **Number_of_Ants (Number of Ants):**
  The number of ants in each generation affects the exploration in each iteration. Increasing the number of ants means more efforts per generation and consequently a higher likelihood of finding better solutions. Typically, the number of ants is set between 20 to 50.
- **Number_of_Generations (Number of Generations):**
  This parameter defines how many generations the algorithm will run. Increasing the number of generations gives the algorithm more opportunities to improve solutions but also requires more computation time. For larger problems, more ants and generations typically help improve solution quality, but this also increases computational time. For smaller problems, fewer ants and generations may suffice, and increasing these parameters excessively may not further enhance solution quality.

Optimizing Solutions Using Iterated Local Search (ILS)

To improve the scheduling results obtained from the Ant Colony Optimization (ACO) algorithm, the Iterated Local Search (ILS) method is employed. This method helps generate better optimization solutions using a local search algorithm.

- **Local Search Algorithm:**
  Local search is a heuristic technique used for solving discrete optimization problems. The method begins with an initial solution and explores its neighborhood to improve the solution. The process works as follows:

  1. **Initialization:** The algorithm starts with an initial solution and evaluates its makespan (completion time).
  2. **Neighborhood Exploration:** For each pair of operations, their positions are swapped to generate new solutions.
  3. **Evaluation and Improvement:** The new solution's makespan is evaluated. If the new makespan is better, the solution is updated, and the search starts again.
  4. **Termination:** This process continues until no further improvement can be found.

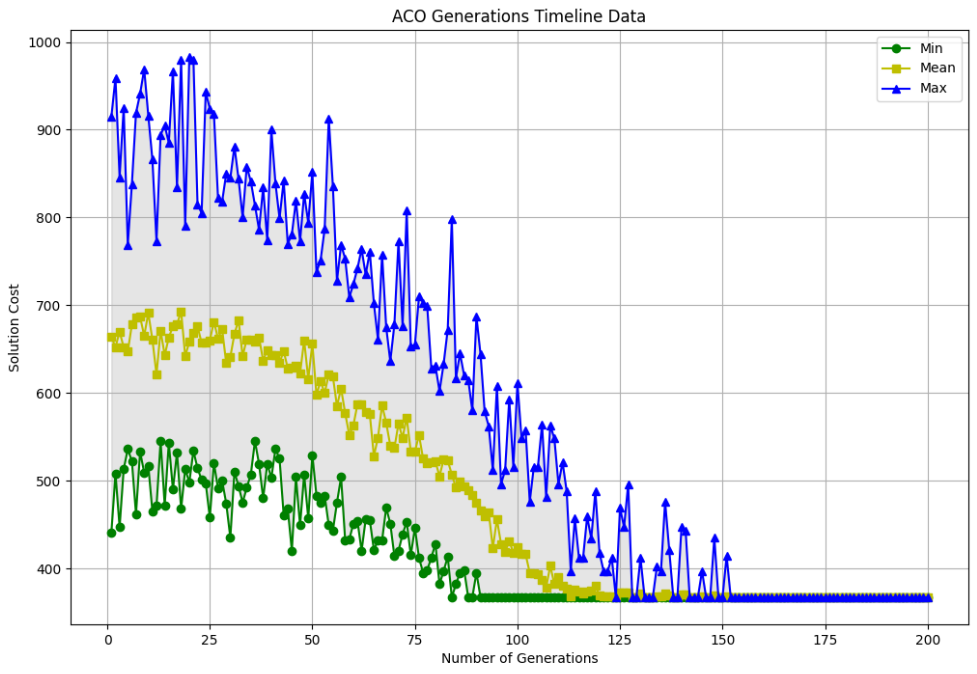
- **Iterated Local Search Algorithm:**
  Iterated Local Search builds on local search by using multiple iterations and perturbations to further improve results. The steps are as follows:
  1. **Initialization:** The algorithm starts with an initial solution and evaluates its makespan.
  2. **Perturbation:** A perturbation function creates a new solution by performing several random swaps of operations.
  3. **Local Search Application:** The perturbed solution undergoes local search to be further improved.
  4. **Comparison and Update:** If the improved solution has a better makespan, it is accepted as the new best solution. Otherwise, the number of unsuccessful attempts is increased, and the algorithm continues with new perturbations.
  5. **Termination:** After a specified number of unsuccessful attempts without improvement, the algorithm terminates.

Reviewing the Progress of the ACO Algorithm in Solving Two OSSP Problem Instances

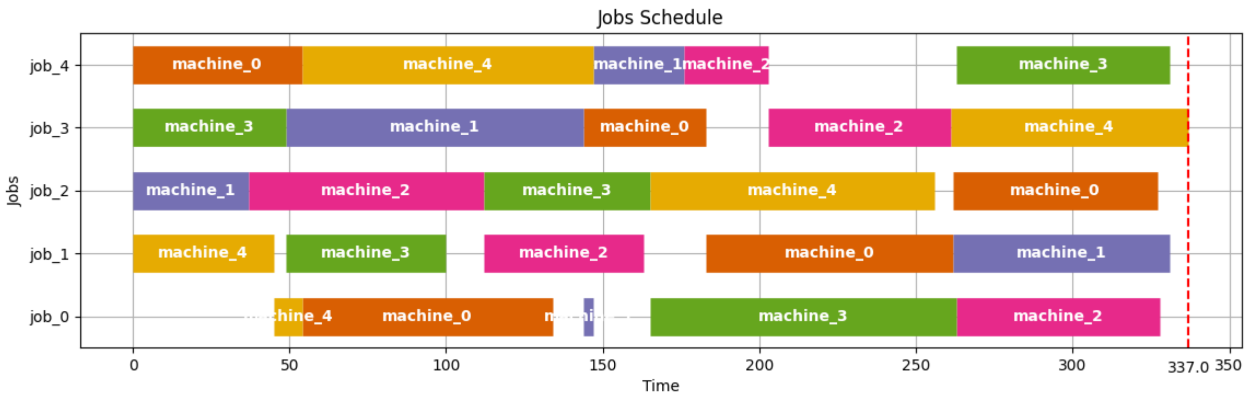
The parameter values used in the execution of this algorithm are as follows:

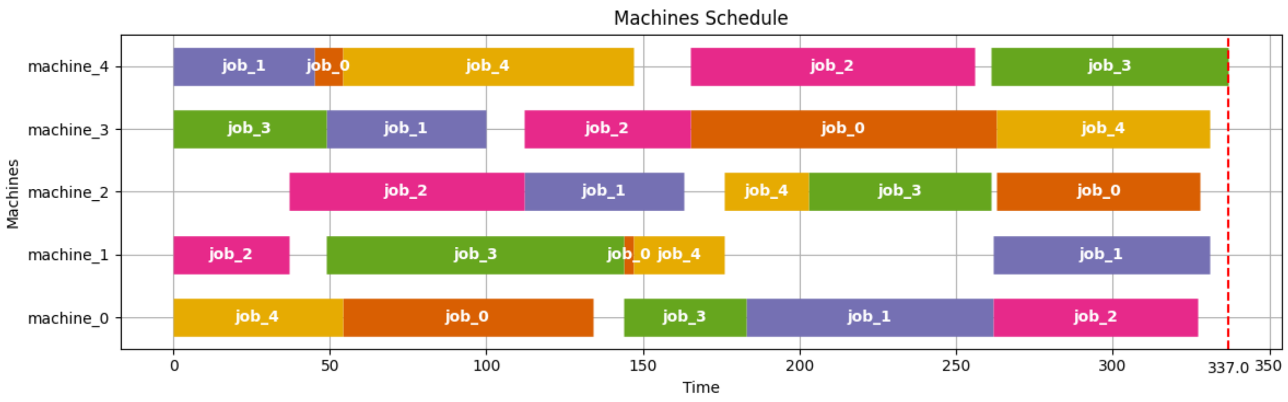
- Alpha (α): 1
- Beta (β): 1
- Rho (ρ): 0.1
- Initial Pheromone (τ): 1
- Overall Best Solution Reward: 2 / 3
- Current Best Solution Reward: 1 / 3
- Number of Ants: 25
- Number of Generations: 200

Example 1: Solving OSSP Problem in Table 1
The following chart illustrates the progress of the ACO algorithm in the OSSP problem presented in Table 1 and listed under "5x5 -3" in Table 3.



The chart below shows the obtained solution and the order of task operations executed on the machines.



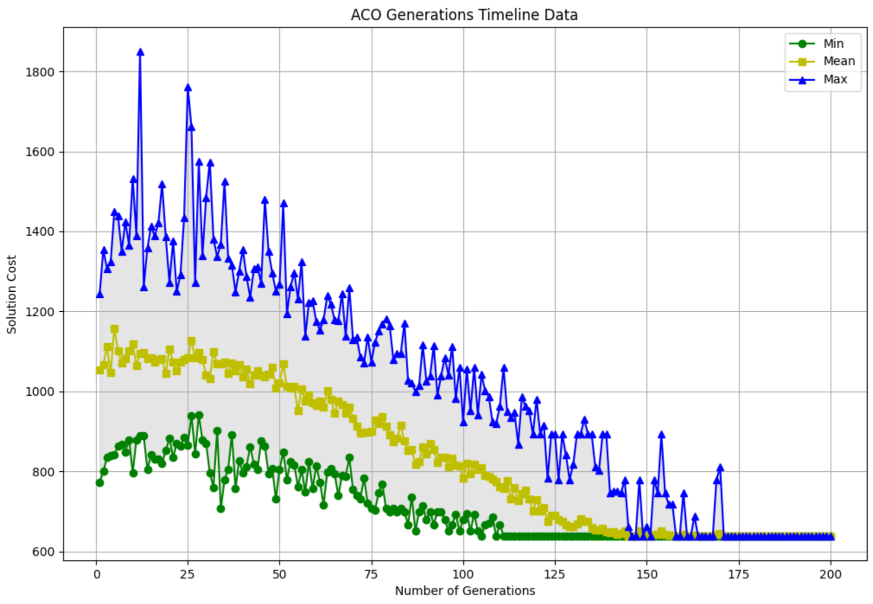


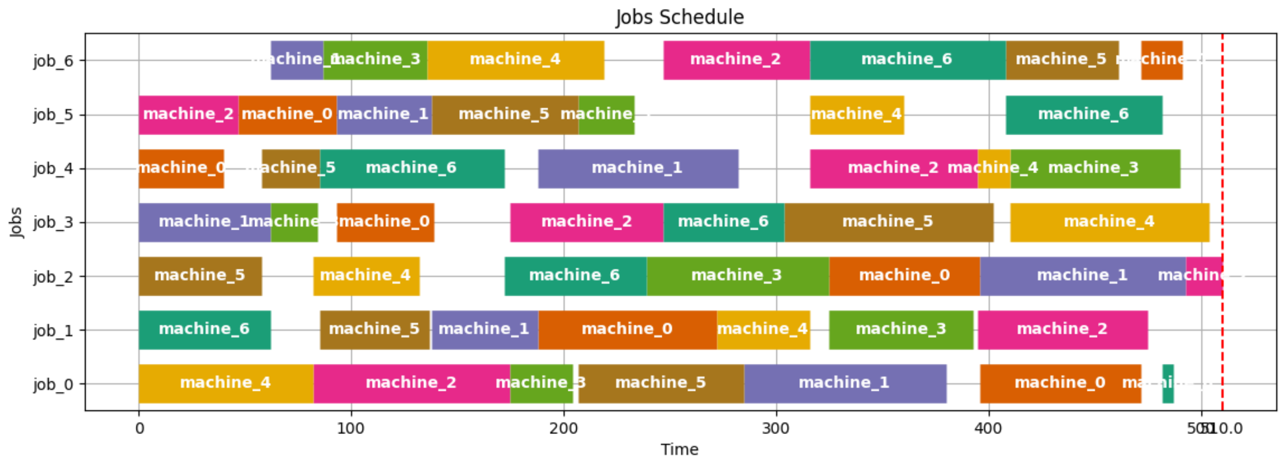
Time required to solve the problem:

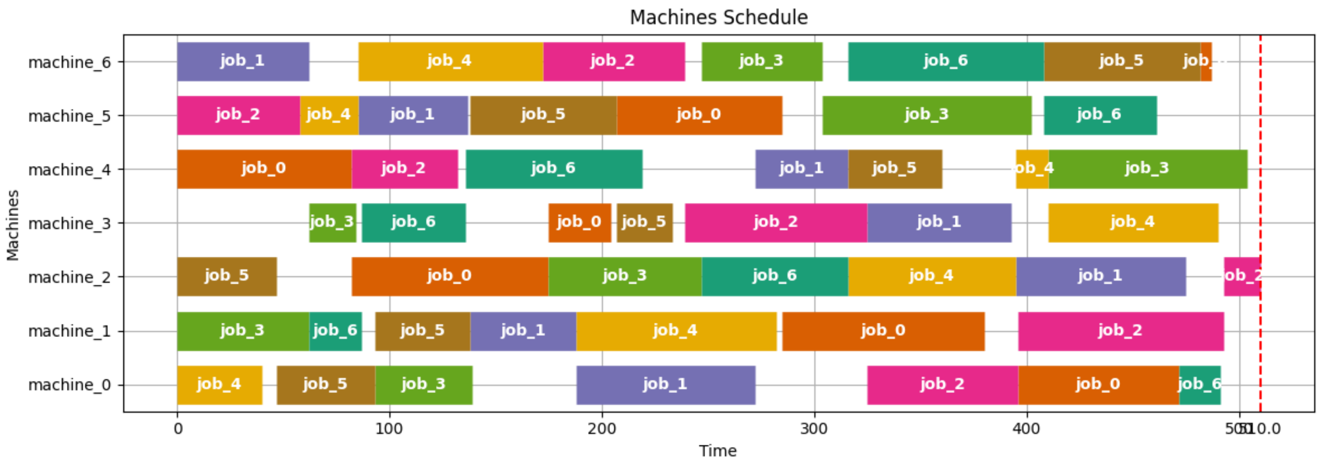
- Time required to find the solution using ACO: 25.66 seconds
- Time required to improve the solution using local search: 26.79 seconds

Example 2: An OSSP problem with 7 machines and 7 tasks.

SS







Results from Experimental Tests

In this section, the results of various Open Shop Scheduling (OSSP) tests with different dimensions are presented, as published in E. Taillard's research titled "Benchmarks for Basic Scheduling Problems" [8]. The information regarding these tests is available through a provided link.

The tables below show the results obtained from running the Ant Colony Optimization (ACO) algorithm, which was previously described. These tables include the best solutions presented in Taillard's work as well as the best results obtained by combining ACO with Local Search, along with the time required to run the complete algorithm.

The parameter values used in the experiment are as follows:

- Alpha (α): 1
- Beta (β): 1
- Rho (ρ): 0.1
- Initial Pheromone (τ): 1
- Overall Best Solution Reward: 2 / 3
- Current Best Solution Reward: 1 / 3
- Number of Ants: 25
- Number of Generations: 200

Table of Results for 4 Jobs and 4 Machines
The following table shows the results from experiments conducted for the 4-job and 4-machine problem:

example	Upper Bound	*ACO*	Final (*+ Local Search*)
4 x 4 – 1	*193*	*193*	*193*
4 x 4 – 2	*236*	*252*	*241*
4 x 4 – 3	*271*	*278*	*271*
4 x 4 – 4	*250*	*263*	*253*
4 x 4 – 5	*295*	*305*	*295*
4 x 4 – 6	*189*	*194*	*193*
4 x 4 – 7	*201*	*209*	*203*
4 x 4 – 8	*217*	*227*	*220*
4 x 4 – 9	*261*	*268*	*267*
4 x 4 – 10	*217*	*221*	*221*

Table of Results for 5 Jobs and 5 Machines

The following table shows the results obtained from experiments conducted for the 5-job and 5-machine problem:

example	Upper Bound	ACO	Final (+ Local Search)
5 x 5 – 1	*300*	*322*	*300*
5 x 5 – 2	*262*	*283*	*266*
5 x 5 – 3	*328*	*367*	*337*
5 x 5 – 4	*310*	*374*	*332*
5 x 5 – 5	*329*	*400*	*348*

Table of Results for 7 Jobs and 7 Machines

The following table shows the results obtained from experiments conducted for the 7-job and 7-machine problem:

example	Upper Bound	ACO	Final (+ Local Search)
7 x 7 - 1	*438*	*567*	*477*
7 x 7 – 2	*449*	*523*	*505*
7 x 7 – 3	*479*	*638*	*510*
7 x 7 – 4	*467*	*565*	*489*

References

[1] Arnaout, J. P., Musa, R., & Rabadi, G. (2010). "Ant colony optimization algorithm to parallel machine scheduling problem with setups." *International Journal of Production Economics*, 123(2), 175-182.

[2] Blum, C. (2005). "Ant colony optimization: Introduction and recent trends." *Physics of Life Reviews*, 2(4), 353-373.

[3] Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press.

[4] Dorigo, M., & Gambardella, L. M. (1997). "Ant colonies for the traveling salesman problem." *Biosystems*, 43(2), 73-81.

[5] Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization*. MIT Press.

[6] Dorigo, M., Birattari, M., & Stützle, T. (2006). "Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique." *IEEE Computational Intelligence Magazine*, 1(4), 28-39.

[7] Hoos, H. H., & Stützle, T. (2004). *Stochastic Local Search: Foundations & Applications*. Elsevier.

[8] Taillard, É. D. (1993). "Benchmarks for basic scheduling problems." *European Journal of Operational Research*, 64(2), 278-285.

[9] Talbi, E. G. (2009). *Metaheuristics: From Design to Implementation*. John Wiley & Sons.