Implementation of Ant Colony Optimization for Task Scheduling in Heterogeneous Multiprocessing Environment

Nhan Tran

December 2025

1 Problem Definition

In multiprocessing environments, different processes, such as GPU, TPU, CPU, have different computation times for different tasks. The goal is to find a solution to schedule a set of tasks that depend on each other so that the total computation time of a set of tasks is as short as possible. Finding an optimal solution is an NP-hard problem. A solution, proposed by [1], is an algorithm based on ant colony optimization. This algorithm tries to find a solution to schedule processes in a respectable amount of time by simulating characteristics of ant colonies through pheromones as reward. The goals of my implementation are to visualize the algorithm and to have some input and output that can be written and read by an external program looking to schedule a set of tasks with a set of processes.

2 Conceptual Design

The design is simple. An input of tasks, processes computation time, dependency edges, and metaheuristics is sent to the algorithm, and an output of tasks and assigned processes is produced by the algorithm.

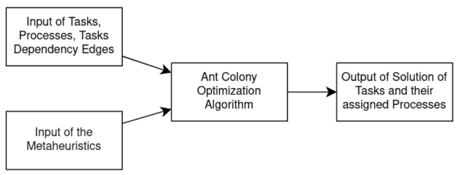
Input of Tasks, Processes, Tasks Dependency Edges
Input of the Metaheuristics
Ant Colony Optimization Algorithm
Output of Solution of Tasks and their assigned Processes

Figure 1: The Design

The algorithm first build a tasks graph out of the tasks and their dependencies. Importantly, all of the tasks have to have an edge connected to another task, and there are no cycles. In other words, the graph generated is a directed acyclic graph. Then, the core algorithm repeatedly build ant solutions, applying local optimizations, and update pheromone trails until a max iteration is reached.

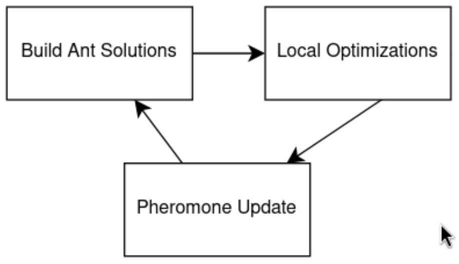

Figure 2: Algorithm Loop

A ready task is a task that has all its parent tasks assigned a process. To build a solution, an ant pick a ready task and a process to assign that task. It will do this repeatedly until all tasks are assigned. After the first ant build a solution, the next ant build a solution. This process will happen until all available ants have a solution. A solution for an ant is a sequence of tasks and processes. Then, a certain number of the best ant solutions are picked, and pheromone updates are performed on the best ant solutions.

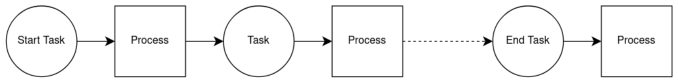

Figure 3: An ant solution

The pheromones are stored in 2 pheromones matrices. The first pheromone matrix, task pheromone matrix, represent the pheromones of all possible edges of between process and task that can be in an ant solution. The second pheromone matrix, process pheromone matrix, represent the pheromones of all possible edges of between task and process that can be in an ant solution.

In the first iteration, the ants pick ready tasks and processes randomly. In subsequent iterations, the ants pick ready tasks and processes based on `STaskRule` and `SProcRule`. If the task is the start task, then there is no need to compute the heuristics since there is only one task to picked.

`STaskRule` is the probability mass distribution function based on the task heuristic and the task pheromone. For all next ready tasks, the heuristic to pick a ready task is computed with the following formula:

$$\text{Task Heuristic(Task)} = \frac{\text{Task Pheromone(Process, Task)}}{\text{Longest Path from Task to End}}$$

where the process is the previous process picked by the ant, and the longest path from task to end is computed recursively. And the probability mass distribution for the task is computed with the following formula:

$$\text{P(Task)} = \frac{\text{Task Heuristic(Task)}}{\text{Sum of all Ready Task Heuristics}}$$

`SProcRule` is the probability mass distribution function based on the process heuristic and the process pheromone. For all processes, the heuristic to pick a process is computed with the following formula:

$$\text{Process Heuristic(Task)} = \frac{\text{Process Pheromone(Task, Process)}}{\text{Average Computation Cost(Task)} \times \text{EST(Task)}}$$

where the task is the task picked by the ant, and EST is the estimate start time of the task. And the probability mass distribution for the task is computed with the following formula:

$$\text{P(Task)} = \frac{\text{Process Heuristic(Task)}}{\text{Sum of all Process Heuristics}}$$

The pheromone updates are performed on the best solutions with the following equation:

$$\text{New Pheromone} = (1 - \text{Decay}) \times \text{Old Pheromone} + \Delta$$

where Decay is a value from 0 to 1, and Δ is computed with:

$$\Delta = 1/\text{Solution Completed Time}$$

3 Implementation Description

The program is written in C. The user interface is developed using Raylib [2], a C graphics library. The ant colony algorithm is stored in `ACORNK.c`. The input mapping to interact with the algorithm and the data is stored in `EntityInput.c`. The main process loop is stored in `Main.c`. The data structure used to stored the pheromone matrices is an array with the length of the number of processes times the number of tasks. The task contains a position coordinate for mapping to the user interface, an ID, the number of child tasks, and the number of parent tasks. The edge contain from task ID, to task ID, and the communication cost. The process contain an array of all the compute costs for each task. The metaheuristics is stored globally and contain the decay parameter, the number of ants, the number of best ants, the max iteration.

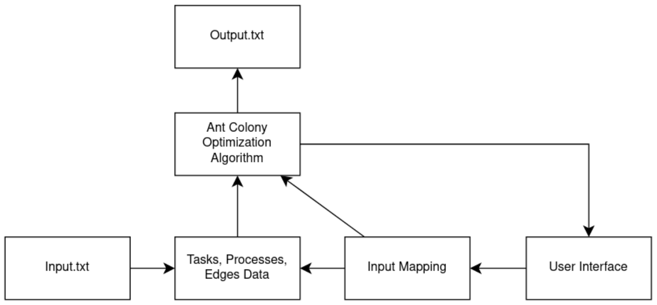
Output.txt
Ant Colony Optimization Algorithm
Input.txt
Tasks, Processes, Edges Data
Input Mapping
User Interface

Figure 4: Implementation Design

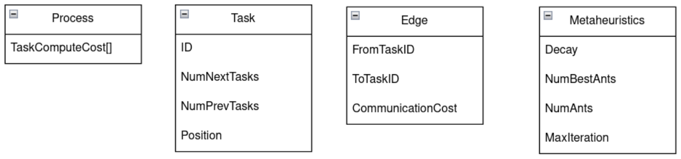

Figure 5: Data

4 User Guide

The user must install Raylib [2], a C graphics library for the user interface. Then, the user can download the project folder and create a build folder. Then, inside the build folder, the user can run the command: `cmake ..` and then `make`, which builds `ACOTaskScheduler` program. The user can run the program by calling `./ACOTaskScheduler`. The user can set the tasks, processes, and edges within `config/input.txt`. The user can also set the metaheuristics within `config/metaheuristics.txt`. The user can add new process by pressing the key `P`. The user can add new task by left clicking on an empty space. The user can add a new edge between 2 tasks by right clicking a task and another task. The user can set the computation cost of a task by a process by left click on a task node, left click on the computation cost text, and entering in numbers. The user can set the edge communication cost by left click the cost text on the edge and enter in numbers. The user can save the new graph by pressing the key `S`. The user can run the algorithm by pressing space key. An output is produced within `config/output.txt`.

5 Self Evaluation

Most of the design goals are achieved. Tasks, processes, and dependencies can be set in the config/graph.txt file. Calculation of a task schedule assignment is completed. A user interface is present to show the user the running algorithm. There is an output file. However, no API is provided for an external process or program to call the algorithm directly.

References

[1] Jeffrey Elcock and Nekiesha Edward. An efficient aco-based algorithm for task scheduling in heterogeneous multiprocessing environments. *Array*, 2023.

[2] Raylib. Raylib, 2025. Accessed: 10 Dec 2025. URL: `https://www.raylib.com/`.