Task Group Allocation for Solving Multi-Load Agent Pickup and Delivery problem

Anonymous Author(s)
Submission Id: «EasyChair submission id»

ABSTRACT

This document outlines the formatting instructions for submissions to AAMAS-2025. You can use its source file as a template when writing your own paper. It is based on the file 'sample-sigconf.tex' distributed with the ACM article template for LaTeX.

CCS CONCEPTS

• **Computing methodologies → Multi-agent planning**.

KEYWORDS

Multi-load agent, Pickup and delivery problem, Task allocation

ACM Reference Format:
Anonymous Author(s). 2025. Task Group Allocation for Solving Multi-Load Agent Pickup and Delivery problem. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 5 pages.

1 INTRODUCTION

The multi-load agent pickup and delivery problem (MLAPD) is a variant of the pickup and delivery problem which is a well-known combinatorial optimization problem in the field of logistics and transportation. In the MLAPD problem, there are multiple agents, each of which can carry multiple tasks at the same time. The goal is to allocate tasks to agents to optimize the completion time of all tasks. The MLAPD problem is a NP-hard problem, and it is difficult to solve it optimally in polynomial time [1].

2 RELATED WORK

3 PROBLEM DEFINITION

In this section, we introduce the multi-load agent pickup and delivery problem, which we address in this paper. Before this, we first give the definitions of map, task, agent, comparative metrics and MLAPD problem.

Definition 3.1 (Map). A map $G = (V, E)$ is the , whose vertices V correspond to locations and edges E represent edges between adjoining locations. The shortest distance of the path between two locations v_i and v_j is denoted as $dis(v_i, v_j)$.

Definition 3.2 (Task). A task $\tau \in \Gamma$ is a tuple $< v_\tau^s, v_\tau^g, t_\tau^r, t_\tau^d, t_\tau^c >$, where v_τ^s and v_τ^g are the pickup location and delivery location, respectively. t_τ^r, t_τ^d and t_τ^c are the release time, deadline and completed time, respectively.

Definition 3.3 (Agent). An agent $a \in A$ is represented as a tuple $< v_a, c_a, \Gamma_a, S_{\Gamma_a} >$, where v_a is location, c_a represents the maximum capacity, Γ_a is the set of tasks, and S_{Γ_a} is the schedule of Γ_a. The $S_{\Gamma_a} =< v_1, v_2, ..., v_n >$, where v_i is a pickup or delivery location of task $\tau \in \Gamma_a$, and the cost of schedule S_{Γ_a} is denoted as $Cost(S_{\Gamma_a}) = \sum_{i=1}^{n-1} dis(v_i, v_{i+1})$.

A task $\tau \in \Gamma$ is released at t_τ^r. When system assigns τ to an agent a, a needs to reach v_τ^s before t_τ^d, and then deliver τ to v_τ^g while recording the completed time t_τ^c. If τ could not be completed before t_τ^d, then the task is considered as failed.

Definition 3.4 (Comparative metrics). In this paper, we demonstrate algorithmic superiority through three comparative metrics, service time (ST), makespan (MS) and completion ratio (CR). $ST = \sum_{\tau_i \in \Gamma} (t_{\tau_i}^c - t_{\tau_i}^r)$, is defined as the sum of differences between the completed times and the release times of all tasks. $MS = max\{t_\tau^c\}$ $(\forall \tau \in \Gamma)$ is time when last task has completed. $CR = \sum_{\tau_i \in \Gamma} X_{\tau_i}/|\Gamma|$, where X_{τ_i} is a decision variable, is completion ratio of Γ. If task τ_i is completed, then X_{τ_i} is 1; otherwise, it is 0.

Definition 3.5 (MLAPD). The multi-load agent pickup and delivery problem (MLAPD) is a tuple $P =< G, A, \Gamma >$, defined by a map G, a set of agents A, and a set of tasks Γ. Moreover, P amounts to finding the allocation of tasks to agents, which optimize three comparative indicators, (ST, MS, CR). Time is discretized into timesteps, and agents can only move to adjacent locations or wait at its current location in one timestep. Howerver, a conflict occurs when two agents are at the same location or pass through the same edge at the same time. The time required to avoid conflicts needs to be considered, since it will affect the comparative metrics. Therefore, we assue that agents can take θ time to avoid a conflict.

4 SOLUTION

In this study, Task-Group Allocation Algorithm (TGA) is developed to solve MLAPD problem efficiently. The TGA method first analyzes the likelihood of carpooling between each task, and then divide the set of tasks into groups via K-Capacity Hierarchical Clustering Algorithm (KCHC). Then, we allocate task groups to agents.

4.1 Carpooling Chance

The biggest difference between multi-load agent and single-load agent is that multi-load agent can load multiple items at the same time. This means that multi-load agent performing a task affects the completion time of the "co-passenger" tasks which are also executed to the same agent at same time.

Example 4.1 (Co-passenger tasks). As shown as in Figure 1, there is an agent, a, and three tasks, τ_1, τ_2 and τ_3. The agent a is located at pickup location of τ_1. The blue lines are the best schedule for

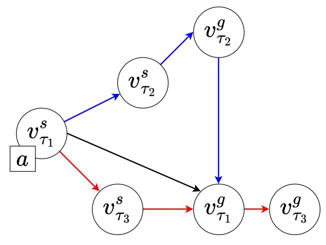


Figure 1: The impact of co-passenger tasks

agent a to complete τ_1 and τ_2. The red lines are the best schedule for agent a to complete τ_1 and τ_3. When we allocate τ_1 and τ_2 to the agent a, the agent a needs to take the blue paths to complete τ_1 and τ_2 by greedy strategy, and it will not impact the completion time of τ_2 but lengthen the completed time of τ_1. Howerver, if we allocate τ_1 and τ_3 to the agent a, a needs to take the red paths to complete τ_1 and τ_3. While it also lengthens the completed time of τ_1, it is obvious that τ_3 has less of an effect on τ_1 as compared to τ_2. Therefore, we need to consider the impact of "co-passenger" tasks when allocating tasks to agents.

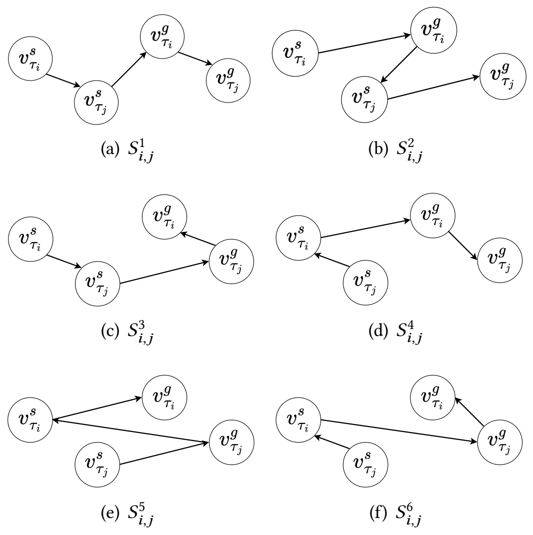


Figure 2: All possible schedules of two tasks

We first analyze the likelihood of carpooling between two tasks. As shown in Figure 2, there are 6 schedules to the completion of τ_i and τ_j, ignoring the location of agent. The schedules are $S^1_{i,j}$, $S^2_{i,j}$, $S^3_{i,j}$, $S^4_{i,j}$, $S^5_{i,j}$ and $S^6_{i,j}$. Therefore, we define the carpooling chance between two tasks τ_i and τ_j as follows.

Definition 4.2 (Carpooling chance). The carpooling chance between two tasks τ_i and τ_j is defined as

$$\delta_{\tau_i,\tau_j} = min\{Cost(S^q_{i,j})\} - dis(v^g_{\tau_i}, v^s_{\tau_i}) - dis(v^g_{\tau_j}, v^s_{\tau_j}) \tag{1}$$

where $q \in [1, 6] \land q \in \mathbb{N}_+$. $\delta_{i,j}$ is difference between the best schedule of two tasks and the sum of the distances between the pickup and delivery locations of the two tasks. The small $\delta_{i,j}$ indicates that τ_i and τ_j being executed by the same agent has less impact on each other's completed time, which can efficiently optimize ST.

Howerver, in this paper, we focus on multi-load agents, which can carry multiple tasks at the same time, not just two agents. Therefore, we need to consider the carpooling chance between multiple tasks. In the beginning, we encounter an issue that we cannot find the best schedule to complete multiple tasks, which is a NP-hard problem. Therefore, we firtly find a novel schedule to complete there tasks. Then we evaluate cost of the schedule and proof it. Finally, we quantify the carpooling chance between multiple tasks.

As shown in Figure 3(a), there is a set of task Γ_K, which contains K tasks, τ_1, τ_2, ..., τ_K. The schedule $\dot{S_{\Gamma_K}}$ of K tasks is shown in Figure 3(b). This schedule starts at the pickup location of a task in Γ_K, then goes to the pickup location of other tasks in Γ_K, and finally to the delivery location of all tasks in Γ_K. More significantly, in $\dot{S_{\Gamma_K}}$, the path between two locations about two tasks in Γ_K is is part of paths of the best schedules of two tasks. Therefore, Theorem 4.3 is proposed to evaluate the cost of the schedule of K tasks, and prove it.

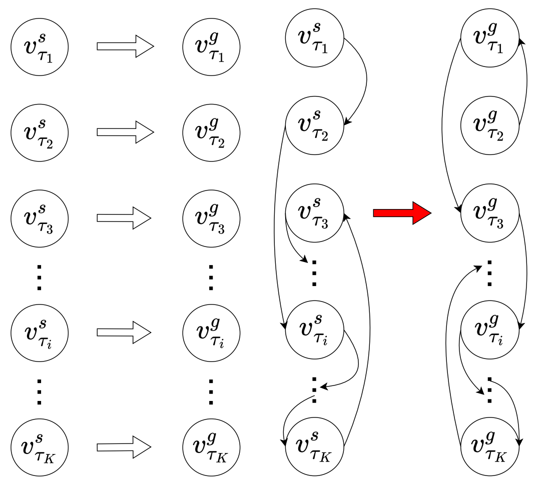
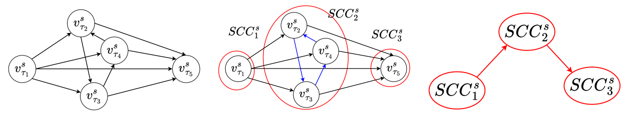


Figure 3: Schedule of K Tasks

Theorem 4.3. *Given a set of tasks Γ_K, there exists a path shown in Figure 3(b) such that the following bound holds.*

$$Cost(\dot{S_{\Gamma_K}}) \leq 2 * \sum_{\tau_i \in \Gamma_K} dis(v^s_{\tau_i}, v^g_{\tau_i}) + 2 * (K-1)\delta \tag{2}$$

where δ is the carpooling chance between two tasks.



(a) Tournament (b) SCC (c) topologize

Figure 4: The tournament and SCC

Proof. Firstly, we need to prove that there is a path to complete Γ_K. As shown in Figure 3(b), the schedule first connects the pickup locations of all tasks in Γ_K, then connects the delivery locations of all tasks in Γ_K. Focusing on the pickup locations set of Γ_K and connecting the pickup locations, we can get a tournament [4], which is a directed graph with exactly one edge between each two vertices, in one of the two possible directions. As shown in Figure 4(a), there are 5 pickup locations, and a path between each two locations. Accroding to the definition of tournament, there is a Hamiltonian path in the tournament [4], that means there exists a path to connect all pickup locations of Γ_K. Similarly, there is a path to connect all delivery locations of Γ_K. Then, we need to find a path to connect the pickup locations and delivery locations. Therefore, we introduce the Strongly Connected Component (SCC) [3], which is a subgraph that is strongly connected. As shown in Figure 4(b), $v^s_{\tau_2}$, $v^s_{\tau_3}$ and $v^s_{\tau_4}$ combine to form a SCC where there is a path between each two locations. When we shrink the SCC to a single vertex and focus on the last SCC of the pickup locations and first SCC of the delivery locations, we can find two locations, $v^s_{\tau_i}$ and $v^g_{\tau_j}$, in two SCCs respectively. Therefore, there is a path between $v^s_{\tau_i}$ and $v^g_{\tau_j}$ to connect the pickup locations and delivery locations.

Secondly, we need to evaluate the cost of the path. In the beginning, we evaluate the cost of the path between the pickup locations. $dis(v^s_{\tau_j}, v^s_{\tau_i})$ is the distance between the pickup locations of τ_i and τ_j, and all possible schedules of two tasks are shown in Figure 2. If the best schedule of two tasks is $S^1_{i,j}$, we can get Equation 3.

$$\begin{aligned} Cost(S^1_{i,j}) &= dis(v^s_{\tau_j}, v^s_{\tau_i}) + dis(v^g_{\tau_i}, v^s_{\tau_j}) + dis(v^g_{\tau_j}, v^g_{\tau_i}) \\ &\geq dis(v^s_{\tau_j}, v^s_{\tau_i}) + dis(v^g_{\tau_j}, v^s_{\tau_j}) \quad (3) \\ dis(v^g_{\tau_i}, v^s_{\tau_i}) + \delta_{\tau_i,\tau_j} &\geq dis(v^s_{\tau_j}, v^s_{\tau_i}) \quad (4) \end{aligned}$$

Then, based on this Definition 4.2, we can get Equation 4. Finally, we check all the possible schedules of two tasks in Figure 2, and we ensure the Equation 4 holds. Similarly, we can evaluate the cost of the path between the delivery locations by the same method and get Equation 5.

$$dis(v^g_{\tau_j}, v^s_{\tau_j}) + \delta_{i,j} \geq dis(v^g_{\tau_j}, v^g_{\tau_i}) \quad (5)$$

Therefore, we can evaluate the cost of the path between the pickup locations and delivery locations. The cost of paths in pickup locations less than or equal to $\sum_{\tau_i \in \Gamma_K \setminus \tau_i} dis(v^s_{\tau_i}, v^g_{\tau_i}) + (K-1)\delta$, which τ_i is the last pickup location in $\dot{S_{\Gamma_K}}$. The cost of paths in delivery locations less than or equal to $\sum_{\tau_i \in \Gamma_K \setminus \tau_j} dis(v^s_{\tau_i}, v^g_{\tau_i}) + (K-1)\delta$, which τ_j is the first delivery location in $\dot{S_{\Gamma_K}}$. The cost of the path $dis(v^g_{\tau_j}, v^s_{\tau_i})$, which connects the pickup locations and delivery locations in $\dot{S_{\Gamma_K}}$, is less than or equal to $dis(v^g_{\tau_j}, v^s_{\tau_j}) + dis(v^g_{\tau_i}, v^s_{\tau_i})$. Therefore, the cost of the schedule is less than or equal to $2 * \sum_{\tau_i \in \Gamma_K} dis(v^s_{\tau_i}, v^g_{\tau_i}) + 2*(K-1)\delta$.

□

Therefore, we can quantify the carpooling chance between multiple tasks as follows, and introduce the Get Carpooling Chance Algorithm (GCC). Accroding to the Theorem 4.3, to be more efficient, we fisrt use Tarjan algorithm [2] to get the SCCs of the set of tasks, and topologize it to get chain of SCC like Figure 4(c). Then, we calculate the carpooling chance in each SCC by Equation 6, and between two SCCs by Equation 7. Finally, we get the carpooling chance between two task groups by Algorithm 1.

$$\Delta_{SCC} = \frac{2 * \sum_{\tau_i \in SCC} \sum_{\tau_j \in SCC} \delta_{\tau_i,\tau_j}}{|SCC|} \quad (6)$$

$$\Delta_{SCC_i, SCC_{i+1}} = \frac{\sum_{\tau_i \in SCC_i} \sum_{\tau_j \in SCC_{i+1}} \delta_{\tau_i,\tau_j}}{|SCC^i| * |SCC^{i+1}|} \quad (7)$$

Algorithm 1 Get Carpooling Chance Algorithm

```
Input: Set of tasks $\Gamma_K$
Output: Carpooling chance $\Delta_{\Gamma_K}$
1: Initialize $\Delta_{\Gamma_K} = 0$
2: Call Tarjan algorithm to get $SCC^s$ and $SCC^g$ of $\Gamma_K$
3: for $SCC^s_i \in SCC^s$ do
4:     $\Delta_{SCC^s_i} = \frac{2 * \sum_{\tau_i \in SCC^s_i} \sum_{\tau_j \in SCC^s_i} \delta_{\tau_i,\tau_j}}{|SCC^s_i|}$
5:     if $SCC^s_i$ is not last SCC then
6:         $\Delta_{SCC^s_i, SCC^s_{i+1}} = \frac{\sum_{\tau_i \in SCC^s_i} \sum_{\tau_j \in SCC^s_{i+1}} \delta_{\tau_i,\tau_j}}{|SCC^s_i| * |SCC^s_{i+1}|}$
7:     end if
8:     $\Delta_{\Gamma_K} += \Delta_{SCC} + \Delta_{SCC^s_i, SCC^s_{i+1}}$
9: end for
10: Do the same for $SCC^g$
11: return $\Delta_{\Gamma_K}$
```

Algorithm Detail: The pseduo-code of Get Carpooling Chance Algorithm is shown in Algorithm 1, where the input is a set of tasks Γ_K containing K tasks, and the output is the carpooling chance Δ_{Γ_K}. First, we initialize Δ_{Γ_K} to 0. Then, we call Tarjan algorithm [2] to get the SCCs of Γ_K (lines 1-2). For each SCC, we calculate the carpooling chance in the SCC and between two SCCs (lines 3-10). Finally, we return the carpooling chance Δ_{Γ_K} (line 11).

4.2 K-Capacity Hierarchical Clustering

The K-Capacity Hierarchical Clustering Algorithm (KCHC) is proposed to divide the set of tasks into groups. The KCHC algorithm is shown in Algorithm 2.

Algorithm Detail: The pseudo-code of K-Capacity Hierarchical Clustering Algorithm is shown in Algorithm 2, where the input is a set of tasks Γ and the max capacity of agents K, and the optput is a set of task groups Π. First, we initialize Π to an empty set (line 1). Then, we treat each task as a task group and add it to Π (lines 2-3). For each pair of task groups, we calculate the carpooling chance between them by Equation 1 (lines 4-6). Next, we iteratively merge the task groups with the minimum carpooling chance until the set of task groups does not change (lines 7-16). Finally, we return the set of task groups Π (line 17).

Algorithm 2 K-Capacity Hierarchical Clustering Algorithm

```
Input: Set of tasks $\Gamma$, the max capacity of agents $K$
Output: Set of task groups $\Pi$
 1: Initialize $\Pi = \emptyset$
 2: for $\tau_i \in \Gamma$ do
 3:     Treat $\tau_i$ as a task group $\Gamma_i$, and add $\Gamma_i$ into $\Pi$
 4:     for $\Gamma_j \in \Pi$ and $\tau_j \in \Gamma_j$ do
 5:         $\Delta_{\Gamma_i,\Gamma_j} = \delta_{\tau_i,\tau_j}$
 6:     end for
 7: end for
 8: while $\Pi$ has changed do
 9:     Select minimum carpooling chance $\Delta_{\Gamma_p,\Gamma_q}$ from $\Pi$
10:     Merge $\Gamma_p$ and $\Gamma_q$ into $\Gamma_{p+q}$
11:     Remove $\Gamma_p$ and $\Gamma_q$ from $\Pi$ and add $\Gamma_{p+q}$ into $\Pi$
12:     for $\Gamma_j \in \Pi$ do
13:         if $|\Gamma_j| + |\Gamma_{p+q}| \le K$ then
14:             $\Gamma = \Gamma_{p+q} \cup \Gamma_j$
15:             GetCarpoolingChance($\Gamma$)
16:         end if
17:     end for
18: end while
19: return $\Pi$
```

Analysis: When Γ contains N tasks, the time complexity of Algorithm 2 is $O(N^2)$. $O(N^2)$ is the time complexity of calculating the carpooling chance between two tasks (lines 2-7).

4.3 Task Group Allocation Algorithm

Before we introduce the Task Group Allocation Algorithm (TGA), we first define the detour cost of the schedule of agent a to complete Γ. The detour cost is defined as the difference between the cost of the schedule and the sum of the distances between the pickup and delivery locations of the tasks in Γ.

$$DCost(S_{\Gamma_a}) = Cost(S_{\Gamma_a}) - \sum_{\tau_i \in \Gamma_a} dis(v^s_{\tau_i}, v^g_{\tau_i}) \quad (8)$$

where S_{Γ_a} is the schedule of agent a to complete Γ. Therefore, we introduce the Get Schedule of Task Group Algorithm (GSTG) to the schedule of task group by Equation 8.

Algorithm 3 Get Schedule of Task Group

```
Input: Set of tasks $\Gamma$, Agent $a$
Output: Cost of schedule $Cost(S_\Gamma)$
 1: Initialize $Cost(S_\Gamma) = 0$
 2: while $\Gamma$ is not empty do
 3:     for $\tau_i \in \Gamma$, adjacent locations $v_i$ and $v_j$ in $S_{\Gamma_a}$ do
 4:         Insert $v^s_{\tau_i}$, $v^g_{\tau_i}$ between $v_i$ and $v_j$, respectively
 5:         Calculate the detour cost $DCost(S_{\Gamma_a})$.
 6:     end for
 7:     Select task $\tau$ with minimum detour cost
 8:     Remove $\tau$ from $\Gamma$
 9: end while
```

Algorithm Detail: The pseduo-code of Get Schedule of Task Group Algorithm is shown in Algorithm 3, where the input is a set of tasks Γ and an agent a, and the output is the cost of the schedule $Cost(S_\Gamma)$. First, we initialize $Cost(S_\Gamma)$ to 0. Then, we iteratively insert the pickup and delivery locations of tasks in Γ between adjacent locations in S_{Γ_a} (lines 3-6). Next, we select the task with the minimum detour cost and remove it from Γ (lines 7-9). Finally, we return the cost of the schedule $Cost(S_\Gamma)$.

Then, we allocate task groups to agents. However, we cannot indirectly allocate task groups to agents, since we do not consider the location of agents, while dividing tasks into groups. It maybe lead to the situation that agents schedule task groups inappropriately.

Example 4.4. As shown in Figure 5, there is an agent, a, and a task group Γ, which contains three tasks, τ_1, τ_2 and τ_3. The black lines are the best schedule S_Γ of a to complete Γ. Obviously, in black lines, a requires a detour to pickup location of τ_1, which is irrational schedule.

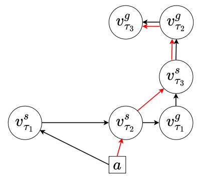


Figure 5: The schedule of agent a to complete Γ

Therefore, to consider the location of agents, we split the task group Γ into multiple subgroups,

$$\Pi_\Gamma = \{\forall \Gamma_i | \Gamma_i \subset \Gamma\} \quad (9)$$

where Π_Γ is the set of subgroups of Γ.

Based on there, we propose the Task Group Allocation Algorithm (TGA) to allocate task groups to agents.

Algorithm 4 Task Group Allocation Algorithm

```
Input: Set of tasks $\Gamma$, set of agents $A$
Output: Set of schedule $\mathcal{S}_S$ of all agents
 1: $K$ = the max capacity of agents
 2: $\Pi$ = K-Capacity Hierarchical Clustering Algorithm($\Gamma$, $K$)
 3: for $\Gamma \in \Pi$ do
 4:     Get $\Pi_\Gamma$ by Equation 9
 5:     $\tilde{\Pi}$ extends $\Pi_\Gamma$
 6: end for
 7: for $a_i \in A$, $\Gamma_j \in \tilde{\Pi}$ do
 8:     Get detour cost $DCost(S_{\Gamma_{a_i}})$ by GSTG Algorithm
 9: end for
10: while $\tilde{\Pi}$ is not empty do
11:     Select agent $a$ and task group $\Gamma$ with minimum detour cost
12:     Remove $\Gamma$ and $\forall \Gamma' | \Gamma' \cap \Gamma \neq \phi$
13: end while
```

Algorithm Detail: The pseduo-code of Task Group Allocation Algorithm is shown in Algorithm 4, where the input is a set of tasks Γ and a set of agents A, and the output is a set of schedule $\mathcal{S}_S$ of all agents. First, we get the max capacity of agents K, and call KCHC Algorithm to find a set of task groups Π (lines 1-2).

Analysis:

REFERENCES

[1] Xiaoshan Bai, Andres Fielbaum, Maximilian Kronmüller, Luzia Knoedler, and Javier Alonso-Mora. 2022. Group-based distributed auction algorithms for multi-robot task assignment. *IEEE Transactions on Automation Science and Engineering* 20, 2 (2022), 1292–1303.

[2] Robert Tarjan. 1972. Depth-first search and linear graph algorithms. *SIAM journal on computing* 1, 2 (1972), 146–160.

[3] Wikipedia contributors. 2024. Strongly connected component — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Strongly_connected_component&oldid=1237069718. [Online; accessed 16-September-2024].

[4] Wikipedia contributors. 2024. Tournament (graph theory) — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Tournament_(graph_theory)&oldid=1234378036. [Online; accessed 15-September-2024].