Self-Driving Vehicle Data Scheduling in Edge-Clouds

Brandon Burtchell
Department of Computer Science
Texas State University
San Marcos, TX
bab334@txstate.edu

Michael Finch
Department of Computer Science
Gonzaga University
Spokane, WA
mfinch@zagmail.gonzaga.edu

Xiao Chen
Department of Computer Science
Texas State University
San Marcos, TX
xc10@txstate.edu

***Abstract*—In an edge cloud environment, data processing in self-driving vehicles goes through local processing, communication, and remote processing. How to schedule these data for timely processing is critical to guaranteeing the safety of self-driving vehicles. This scheduling problem is related to the flow shop scheduling problem, which is NP-complete. In this paper, different from those in the literature that focus on minimizing makespan, our objective is to develop algorithms that produce schedules to minimize the average waiting time AWT and by taking the priorities of the data types into account since vehicle data are time-sensitive and different data types have different emergency levels. In regard to this objective, we propose two heuristic algorithms: the Priority and AWT-based NEH (PAN) algorithm, and the Priority-based Aalla's (PAA) algorithm. Simulation results show that our proposed algorithms outperform the preexisting ones and while PAN is a better algorithm when considering our metrics, PAA is more efficient while still producing similarly viable results.**

***Index Terms*—average waiting time, cloud, makespan, mobile edge computing, priority, self-driving vehicle**

I. Introduction

Self-driving vehicles have attracted a lot of attention from companies and research organizations in recent years and will reshape the transportation in the future [2]. Self-driving vehicles combine a variety of sensors to perceive their surroundings, such as radar, lidar, sonar, GPS, odometry and inertial measurement units [9, 14]. The data collected by these sensors need to be processed in a timely manner to identify appropriate navigation paths, as well as obstacles and relevant signage [9]. Meanwhile, with the development of cloud computing, more and more mobile applications offload computation-intensive jobs to remote cloud data centers [8]. Although such operations could substantially enhance the capability of mobile devices, a long communication delay is inevitable. To mitigate this problem, some data are processed locally on the edge resources to be closer to the user. Thus, the paradigm that combines the resources at the edge and the cloud, called *edge-clouds*, also known as edge computing, has become more and more popular [11].

In an edge computing environment, each data type i collected by sensors in a self-driving vehicle goes through three stages:

- local processing in the vehicle with time l_i,
- transmission to cloud (communication) with time c_i,
- and remote cloud processing with time r_i.

Different data types have different priorities: some are more urgent than others. Thus, we factor in the priority of each data type. We assume that a higher number represents a higher priority. A sample of data is shown in Figure 1.

data type i	l_i	c_i	r_i	p_i
1	4	3	5	5
2	2	4	3	9
3	7	5	4	2
4	6	1	2	6

Fig. 1. A data sample.

With these data, in this paper, we investigate data scheduling algorithms for self-driving vehicles in an edge computing environment. Data types in a self-driving vehicle are time-sensitive. To improve safety and security of self-driving vehicles, we explore schedules to minimize the average waiting time (AWT) of n data types using their three-stage processing times and priorities.

Our problem resembles the flow shop scheduling problem [1] where all jobs pass the same sequence of machines. 'Machines' are the 'stages' here. However, most papers in the literature [3, 4, 5, 6, 10, 12, 13] aim to minimize makespan (defined as the total amount of time for a schedule to finish), rather than AWT of the schedules. Besides, they do not consider the priorities of the data types. Therefore, it is necessary for us to study the problem. After looking into a series of three-stage scheduling algorithms, we come up with two algorithms that best suit our goal. The first algorithm is called the *Priority and AWT-based NEH* (PAN) algorithm, which is enlightened by the NEH algorithm in [12]. And the second one is called *Priority-based Aalla's* (PAA) algorithm inspired by the Aalla's algorithm in [3]. Simulation results show that they perform the best by our metric comparing with existing algorithms. Algorithm PAN produces better schedules than PAA but PAA is more efficient when the number of data types is large.

The key differences of our work from others are as follows:

- Developing scheduling algorithms with the objective of minimizing AWT instead of makespan

- Adding data type priority in scheduling
- Conducting simulations to evaluate the performance of the proposed algorithms

The rest of the paper is organized as follows: Section II briefly summarizes the related work. Section III defines the problem. Section IV presents our solutions. Section V describes the simulations we have conducted, and the conclusion is in Section VI.

II. Related Work

The scheduling problem we study here is related to the flow shop scheduling problem [1]. In a flow shop scheduling problem, there are m machines that should process n jobs. All jobs have the same processing order through the machines. The order of the jobs on each machine can be different.

Most previous research on the flow shop problem is generally concerned with minimizing makespan. When $m = 2$ machines, the problem can be solved optimally in $O(nlogn)$ time by Johnson's algorithm [10]. If there are $m = 3$ machines or more, then the problem is NP-complete [7]. Extended Johnson's algorithm [10] can find optimal solutions with three machines when certain conditions are met. The algorithm creates two partial schedules H and L based on whether $l_i \leq r_i$, then sorts the elements in H increasingly and L decreasingly before concatenating them together.

Johnson's algorithm would go on to serve as the basis for many future heuristic algorithms. For instance, the CDS algorithm [4] compares the above Johnson's schedule against its own similarly obtained schedule, created by only sorting according to l_i and r_i. CDS chooses the best schedule according to a minimized makespan. Similar to the CDS algorithm, Algorithm 2 [5] first extends Johnson's algorithm, then shuffles the schedule according to critical data types in order to check if a schedule with smaller makespan can be obtained. Another algorithm named Palmer's Heuristic [13] weights the processing times of l_i and r_i with a multiplier, then sorts the entire list of data types by their total weighted processing time. This is an early utilization of a "slope" that occurs between stages of a single data type.

Compared to the above algorithms, the Nawaz, Enscore, and Ham (NEH) algorithm [12] appears to be the best polynomial heuristics in practice, but at a higher complexity [15]. Finally, a more recently developed algorithm called Aalla's algorithm [3] offers small makespans by calculating a slope factor that minimizes idle times between data types. By utilizing a greedy approach, the algorithm appends the next best data type to the current schedule in each round.

Different from these work, our goal is to find a schedule to minimize AWT for data types with different priorities.

III. Problem Definition

A self-driving vehicle installs many sensors. These sensors measure different parameters of the vehicle. The data collected by the sensors are processed in three stages sequentially: local processing in the vehicle, sending to cloud (communication), and cloud (remote) processing. Suppose there are n data types. Each data type has a vector of four parameters: l_i, c_i, r_i, and p_i. The first three parameters represent the processing times in three stages, respectively, and the last one is the priority of the data type.

The waiting time wt_i of a data type i is the time that it is not processed in any of the three stages after its generation. It is easier to describe it through its *turn around time*. The turnaround time tr_i of a data type i is the period from the time the data type was generated to the time the final remote stage of the data type is complete. Then the waiting time wt_i of a data type i is:

$$wt_i = tr_i - (l_i + c_i + r_i) \tag{1}$$

Then the average waiting time AWT of all n data types is:

$$AWT = \frac{1}{n}\sum_{i=1}^{n} wt_i \tag{2}$$

Since self-driving vehicles receive large amount of data from various sensors and these data are time-sensitive, our primary goal is to develop algorithms to schedule n data types to

$$\text{minimize } \{AWT\} \tag{3}$$

Our defined problem is related to the flow shop problem [1], which is NP-complete when $n \geq 3$. It becomes more difficult when priority is added. It is a combinational search problem with $n!$ possible sequences. If one could enumerate all $n!$ sequences, the sequences with minimum average waiting time could be identified, but this procedure is quite expensive and impractical for large n.

IV. Our Solutions

In this section, we propose two algorithms namely PAN and PAA to generate schedules to minimize AWT.

A. Priority and AWT-based NEH (PAN) Algorithm

Our first solution Priority and AWT-based NEH (PAN) algorithm is inspired by the Nawaz, Enscore, and Ham (NEH) algorithm [12]. The NEH algorithm aims to minimize makespan and does not consider priorities of the scheduled elements. We extend it to address our goal. The detailed PAN algorithm is presented in Figure 2.

In the PAN algorithm, we first do the preprocessing of each data type by dividing its processing times in the three stages by its priority. In this way, a higher priority data type will have smaller processing times so that it can be scheduled earlier. We then initialize the unscheduled list U by including all the data types. Next, we sort the data types in U in non-increasing order by the total processing time T_i ($T_i = l_i + c_i + r_i$) of each data type. Then we consider the two possible permutations (τ_A, τ_B) of the first two data types in U, choosing the permutation that has the minimal AWT and assigning it to the partial schedule π. For the remaining unscheduled data types, we consider every possible position a data type i can be inserted in the current partial schedule π. Each possible insertion schedule is stored in τ_i, then the partial schedule π is updated by the

PAN: Priority and AWT-based NEH Algorithm

```
1: **Inputs:** a set of data types with their processing times in three stages and their priorities.
2: **Output:** a schedule $\pi$ that minimizes $AWT$.
3: For each data type, divide $l_i, c_i$, and $r_i$ by $p_i$;
4: $U = \{1, \ldots, n\}$; /* initialize the unscheduled list $U$ by including all the data types */
5: for each data type $i$ do
6:     calculate $T_i = l_i + c_i + r_i$; /* total processing time */
7: end for
8: sort $U$ in non-increasing order based on $T_i$;
9: /* consider the schedules with the first two data types */
10: $\tau_A = \{U_1, U_2\}$;
11: $\tau_B = \{U_2, U_1\}$;
12: $\pi$ = $\tau_A$ or $\tau_B$ with minimal $AWT$;
13: for $i = 3$ to $n$ do
14:     /* create possible schedules and choose the best */
15:     $\tau_i =$ insert $U_i$ at every possible position in $\pi$;
16:     $\pi = \tau_i$ with minimal $AWT$;
17: end for
18: return $\pi$
```

Fig. 2. The PAN algorithm

τ_i with the minimal AWT. This process is repeated until all data types have been scheduled.

If we apply the PAN algorithm to the example in Figure 1, it returns the schedule $\pi = \{2, 4, 1, 3\}$, which is visualized by the Gantt chart in Figure 3. This schedule has an $AWT = 5.5$.

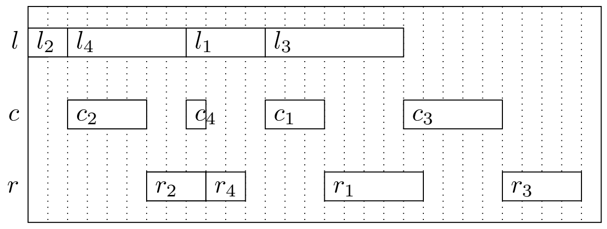

Fig. 3. The Gantt chart of the schedule produced by the PAN algorithm

B. Priority-based Aalla's (PAA) Algorithm

Our second solution Priority-based Aalla's (PAA) algorithm is built on the idea of the Aalla's algorithm [3]. The Aalla's algorithm minimizes the schedule makespan by greedily reducing the idle time between two adjacent items. Therefore we think it will be helpful to reduce the average waiting time in our problem.

The details of our proposed algorithm PAA are described in Figure 4. First, like PAN, we do the pre-processing of the data by dividing the processing times of a data type by its priority. Next, for each data type i (here renamed as b to facilitate description), we calculate a slope factor W_b, defined by:

$$W_b = c_b + 2(r_b) \tag{4}$$

In this slope factor, we give more weight to the processing times in the later stages. W_b is static once calculated for each data type. Next, we initialize counter N to the total number of data types n, set the current schedule π to $\emptyset$, and put all the data types to the unscheduled list U. We initialize the first data type a to zero.

Then in the main loop, as long as not all the data types are scheduled, we iterate the following steps. First, for each data type b in the unscheduled set U, we calculate the total weighted idle time $I_{a,b}$ between data type a, the previously scheduled job (or 0, if a is the first date type) and b, the current job being considered for appending. The equation for $I_{a,b}$ is defined by:

$$I_{a,b} = 2(i_{a,b,c}) + i_{a,b,r}, \tag{5}$$

where $i_{a,b,s}$ is the idle time between two data types in one stage ($s = c$ or r). In defining the total weighted idle time, we give a higher penalty to the idle times in earlier stages as any idle times on these stages tend to delay processing on later stages.

Next, to decide the best candidate to append to a partial schedule, we define a ratio $P_{a,b}$:

$$P_{a,b} = I_{a,b}/W_b \tag{6}$$

Among the unscheduled data types in U, we choose the best data type j which has the minimal Pa, b. In case there is a tie, we pick j that has the maximal W_b. We then append data type j to the current schedule π, remove it from U, set a to j in preparation for the next round, and decrement counter N by one. This process is repeated until N becomes 1, or all data types have been scheduled.

If we apply the PAA algorithm to the table in Figure 1, it returns the schedule $\pi = \{2, 1, 4, 3\}$, visualized by the Gantt chart in Figure 5. This schedule has an $AWT = 5.25$.

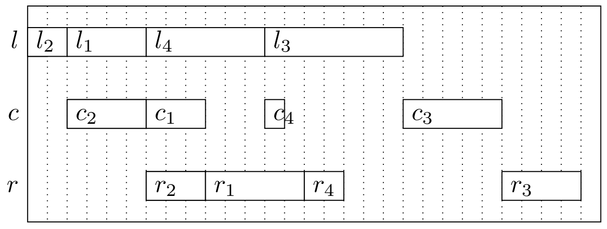

Fig. 5. A Gantt chart of the schedule produced by the PAA algorithm

V. Simulations

In this section, we evaluate the performance of our proposed algorithms by comparing them with existing algorithms using a simulator written in Python.

PAA: Priority-based Aalla's Algorithm

```
1: **Inputs:** a set of data types with their processing times in three stages and their priorities.
2: **Output:** a schedule $\pi$ that minimizes $AWT$.
3: For each data type $b$, divide $l_b, c_b$, and $r_b$ by $p_b$
4: **for** each data type $b$ **do**
5:     calculate $W_b$;
6: **end for**
7: $N = n$, where $n$ is the number of data types
8: $\pi = \emptyset$;
9: $U = \{1, \ldots, n\}$;
10: $a = 0$;
11: **while** $N > 1$ **do**
12:     **for** $b$ in $U$ **do**
13:         calculate $I_{a,b}$;
14:         $P_{a,b} = I_{a,b}/W_b$;
15:     **end for**
16:     /* choose best $j$, append to $\pi$, remove from $U$ */
17:     $j = b$ with min$\{P_{a,b}\}$, and max$\{W_b\}$ to break ties;
18:     $\pi = \pi \cup \{j\}$;
19:     $U = U \backslash \{j\}$;
20:     $a = j$;
21:     $N = N - 1$;
22: **end while**
23: **return** $\pi$
```

Fig. 4. The PAA algorithm

A. Algorithms Compared

We compared PAA and PAN to the following existing scheduling algorithms, each modified to consider priority of the data types for fair comparison.

- Extended Johnson's algorithm [10]
- Algorithm 2 [5]
- Campbell, Dudek, and Smith (CDS) algorithm [4]
- Nawaz, Enscore and Ham (NEH) algorithm [12]

In addition to these algorithms, we also compared our algorithms to the brute-force algorithm, which finds the schedule with the minimum AWT.

B. Metrics

For our primary metric, we utilized AWT in Equation (2). Furthermore, in order to measure the advantage of considering priority p_i, we define another metric, *satisfaction* S_i, defined by the following equation to evaluate how satisfied each data type is.

$$S_i = \begin{cases} 1 - \frac{x-j}{n+1-j}, & \text{if } x \geq j, \\ 1, & \text{otherwise} \end{cases} \tag{7}$$

S_i is a ratio that represents how close the order number x of a scheduled data type i in the generated schedule is to its priority rank j, where j represents the position of i in a schedule sorted by priority only. Then the average satisfaction AS of all data types in a schedule is defined by:

$$AS = \frac{1}{n}\sum_{i=1}^{n} S_i \tag{8}$$

For AS, a higher value means a better result.

To summarize the effectiveness of every algorithm in terms of AS and AWT, we define a score metric Ω to combine them. Metric Ω is a ratio that represents how well both metrics are met, with a higher value representing a better score.

$$\Omega = \hat{AS}/\hat{AWT} \tag{9}$$

$\hat{AS}$ and $\hat{AWT}$ are the normalized values of AS and AWT, respectively. The normalization allows us to put metrics on different scales together.

C. Experiment Settings

In the simulations, we feed all the algorithms with the same data, which contain the processing times of the data types in the three stages and their priorities. The processing times are randomly generated in the range of $[1, 99]$ and priorities are randomly selected in the range of $[1, 10]$.

To test our algorithms with different numbers of n, we tried $n = 6$, $n = 10$, $n = 20$, $n = 50$, and $n = 100$. Each value of n had 100 runs. In each round, once an algorithm generated a schedule, we calculated its AWT, AS, and Ω. And after these 100 rounds were finished, we computed the average metrics for each algorithm across the rounds.

D. Results of Comparing All Algorithms

From the simulations, we found that both PAN and PAA had higher Ω values than the rest of the algorithms for all values of n, with PAN scoring higher than PAA. Figure 6 displays the Ω values when $n = 6$.

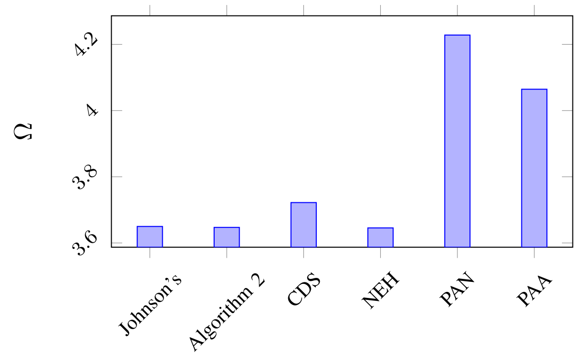

Fig. 6. Ω scores of algorithms, $n = 6$

These high Ω values show that PAN and PAA do a better job of both minimizing AWT and maximizing AS. Given that PAN has the highest Ω, we conclude that PAN is the best suited algorithm for this research problem.

To confirm our claim, we will compare both PAN and PAA more closely against the brute-force algorithm that generates the minimal AWT next.

E. PAA and PAN vs Brute-Force

We compared PAN and PAA with the brute-force algorithm in terms of AWT and AS. The brute-force algorithm finds the optimal schedule from all permutations of n data types.

Obviously, the brute-force algorithm produced schedules with the smallest AWT overall. However, our proposed PAN and PAA did not appear to be drastically far off from the ideal result. Evaluating PAN and PAA, we found that they had very similar AWT, with PAA having a slightly lower AWT. Figure 7 shows the comparison of the three algorithms in terms of AWT when $n = 6$.

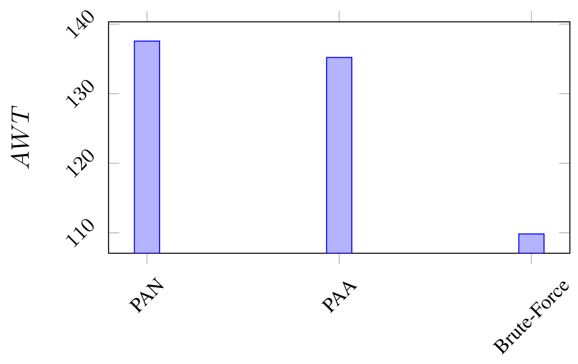

Fig. 7. AWT of proposed algorithms and brute-force search, $n = 6$

Next, we analyzed the AS of the three algorithms. From these results, we found that PAN produced schedules with a significantly higher AS than PAA. Figure 8 shows the comparison when $n = 6$. We conclude that this difference in satisfaction accounts for the PAN having a higher overall score. Compared to brute-force, which did not consider priority at all, both PAN and PAA had higher AS making them more effective algorithms for our application.

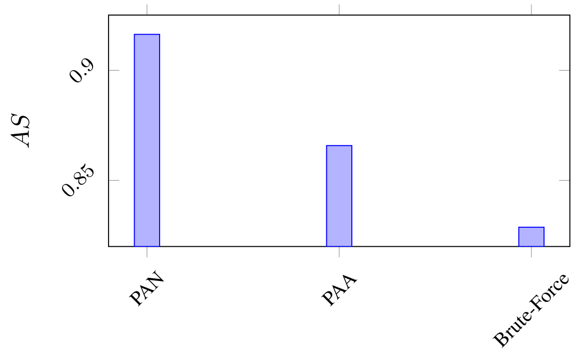

Fig. 8. AS of PAN, PAA, and brute-force algorithms, $n = 6$

Despite the high satisfaction values of PAN and PAA, brute-force algorithm's optimal AWT results in a higher Ω than PAN and PAA (Figure 9). Nonetheless, the scores of PAN and PAA are not significantly lower than that of brute-force.

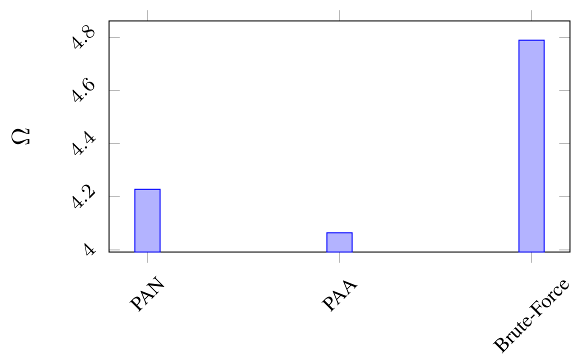

Fig. 9. Ω scores of proposed algorithms and brute-force, $n = 6$

F. Efficiency of the Algorithms

To further compare our proposed algorithms, we looked into the run times of PAN, PAA, and the brute-force. To measure these times, we added timers into our simulation to track how long it took each to produce a schedule. Once we had calculated the run times for each round's schedules, we calculated the average run time for each algorithm across all rounds.

First, comparing our proposed algorithms to brute-force, we see that both PAA and PAN are considerably more efficient (Figure 10). From this result, we conclude that while brute-force produces minimal AWT, PAN and PAA still produce quality results while also being much more efficient. For this reason, we argue that PAA and PAN are more effective than the brute-force algorithm.

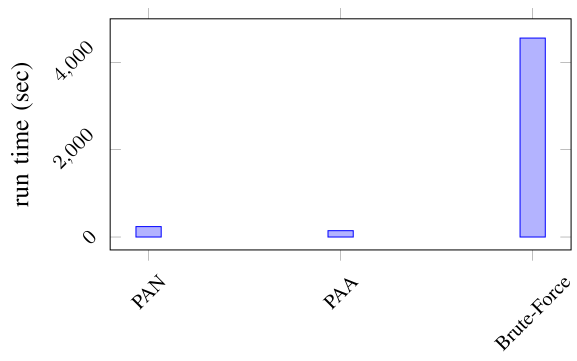

Fig. 10. Average run time of proposed algorithms and brute-force, $n = 6$

Looking at our two proposed algorithms, it is clear that the average run time for PAN is greater than that of PAA (Figure 11). Knowing that PAA produces only one final schedule whereas PAN produces many more partial schedules, we can understand how PAA is a more efficient algorithm than PAN.

Furthermore, we analyzed the average run time across several different values of n data types to see how the run times would vary with larger data tables. From this experiment, we found that the larger the value of n and the more data types to

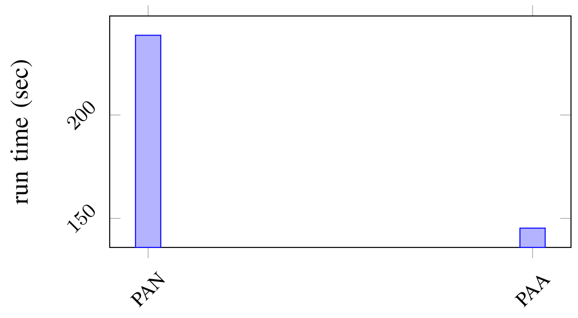

Fig. 11. Average run time of proposed algorithms, n=6

process, the greater the difference in run time between PAN and PAA (Figure 12).

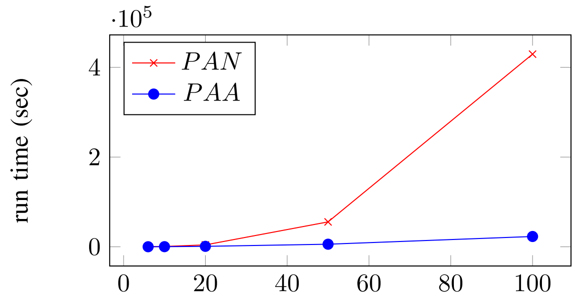

Fig. 12. Average run time of proposed algorithms across different values of n

From this insight, we conclude that while PAN can produce better schedules in terms of AWT and AS combined, PAA is much more efficient when there are large quantities of data types to be scheduled.

VI. Conclusion

In this project, we have explored a series of priority-based three-stage scheduling algorithms for scheduling time-sensitive data from self-driving vehicles in an edge cloud environment. In order to ensure that data are processed in a timely manner, our objective has been to develop algorithms to reduce the average waiting time by considering data type priorities. To meet this objective, we have proposed two heuristic algorithms PAN and PAA. Simulation results have shown that our proposed algorithms outperform the existing ones and while PAN is a better algorithm when considering our metrics, PAA is more efficient while still producing similarly viable results. Therefore, PAA is better suited when the data size is large.

In the future, we will explore more efficient and effective three-stage scheduling algorithms. Additionally, we hope to look into implementing and testing the algorithms we have proposed in real self-driving vehicle data systems.

Acknowledgment

This material is based upon work supported by the National Science Foundation under REU grant #1757893.

References

[1] Flow-shop scheduling. https://en.wikipedia.org/wiki/Flow-shop_scheduling.

[2] Self-driving car. https://en.wikipedia.org/wiki/Self-driving_car.

[3] Rajasekhar Aalla. A heuristic algorithm for flowshop sequencing problems. Master's thesis, Texas Tech University, Lubbock, Texas, 1992.

[4] Herbert G. Campbell, Richard A. Dudek, and Milton L. Smith. A heuristic algorithm for the n job, m machine sequencing problem. *Management Science*, 16(10):B630–B637, 1970.

[5] Bo Chen, Celia A. Glass, Chris N. Potts, and Vitaly A. Strusevich. A new heuristic for three-machine flow shop scheduling. *Operations Research*, 44(6):891–898, 1996.

[6] Y. Duan and J. Wu. Joint Optimization of DNN Partition and Scheduling for Mobile Cloud Computing. In *International Conference on Parallel Processing (ICPP 2021)*, 2021.

[7] M. R. Garey, D. S. Johnson, and Ravi Sethi. The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[8] Z. Han, H. Tan, G. Chen, R. Wang, Y. Chen, and F. C. M. Lau. Dynamic virtual machine management via approximate markov decision process. In *Proc. of INFOCOM*, 2016.

[9] J. Hu and et al. Cooperative control of heterogeneous connected vehicle platoons: An adaptive leader-following approach. *IEEE Robotics and Automation Letters*, 5(2):977–984, 2020.

[10] Selmer Martin Johnson. *Optimal Two- and Three-Stage Production Schedules with Setup Time Included.* RAND Corporation, Santa Monica, CA, 1953.

[11] Y. Li and S. Wang. An energy-aware edge server placement algorithm in mobile edge computing. In *Proc. of IEEE International Conference on Edge Computing (EDGE)*, pages 66–73, 2018.

[12] Muhammad Nawaz, E Emory Enscore, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

[13] D. S. Palmer. Sequencing jobs through a multi-stage process in the minimum total time—a quick method of obtaining a near optimum. *Journal of the Operational Research Society*, 16(1):101–107, 1965.

[14] A. Taeihagh and H. Lim. Governing autonomous vehicles: emerging responses for safety, liability, privacy, cybersecurity, and industry risks. *Transport Reviews*, 39(1):103–128, 2019.

[15] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74, 1990.