EXPLORING SCHEDULING POLICIES FOR UAV SWARMS

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Clara Friederike Steinhoff

August 2021

© 2021 Clara Friederike Steinhoff

ABSTRACT

UAV swarms have gained popularity over the past years, with increased accessibility contributing to new research areas and applications. The increased complexity of handling of UAVs swarms and assigning multi-phase applications results in the need for efficient task assignment for UAV swarms. The project extends an existing UAV swarm simulator to enable comparison of scheduling algorithms and techniques. The simulator is easily augmentable and provides a simple-to-use interface for the specification of complex applications. The evaluation of five scheduling algorithms shows that existing solvers for global problem optimization outperform naive scheduling approaches in terms of performance and assignment quality. Adaptive task batching and the utilization of cloud resources can lead to better task assignments with lower overhead and reduce the computational burden on the swarm.

BIOGRAPHICAL SKETCH

Clara was born on the 20th of June 1995, on a sunny day in Berlin, Germany. A mischievous child, Clara spent her early years living in Africa with her parents, a baby brother, and two dogs. Back in Berlin for her school years, Clara spent countless years eating ice cream and reading books past her bedtime with a torch under her blanket. Upon finally graduating Schiller high school, Berlin in 2013 Clara - undecided as she was at the time - spent a voluntary year abroad in the Philippines. Returning to Germany, Clara decided to pursue University studies, initially enrolling for physics, before dabbling in media technology, and finally falling in love with computer science. It was in these years, that Clara would meet her wonderful boyfriend Florian whom she would eventually follow to the US. After finishing her Bachelors thesis on the subject of vectorization of the VVC decoder in ARM Neon, and relishing in the news of being accepted to Cornell's MS program, Clara embarked on a new chapter. At Cornell, Clara worked as a TA for the first time under the amazing Anne Bracy, learnt (and is still learning!) to play tennis during beautiful Ithaca summers, and conducted her masters thesis on the topic of UAV swarm scheduling policies alongside her advisor Christina Delimitrou. After two years of growing at the challenge of her studies and making the best out of a pandemic, Clara is excited to start the next step in her journey by joining the cloud computing company Exotanium.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Christina Delimitrou. Her guidance and direction throughout the project have been invaluable, and I greatly appreciate the new perspectives she provided. I would also like to thank Stephen Wicker for his constructive feedback on coursework and thesis, and just generally being an affable minor advisor.

I want to thank Florian Suri-Payer, for the unconditional encouragement and support, and being more confident in my abilities than even myself. And, finally, I want to acknowledge my friends and family, for bearing with me and having a sympathetic ear whenever I faced struggles.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Unmanned aerial vehicles (UAVs), and more specifically the use of UAV swarms to execute tasks, continue to be a well investigated topic as new technologies and trends emerge. UAV swarms can be employed for a wide range of tasks and may be utilized in remote and hazardous environments to prevent human endangerment. Typical tasks involve generating data through sensing and subsequently processing that data. Example applications are smart farming [1], surveillance and inspection [2], disaster sensing [3] and mobile crowd sourcing [4]. While most of these tasks could also be executed by a single UAV, there are advantages of swarm based systems. The capability of a single UAV is limited due to computational and power constraints. Multiple UAVs are able to overcome those limitations by collaborating, leading to higher coverage and faster execution of a tasks and missions. Additionally a swarm provides redundancy through numbers, increasing scalability and survivability in case one ore multiple UAVs should fail [5]. Another method of overcoming the constraints of UAVs is the offloading of computationally expensive tasks to an edge server or the cloud.

One of the core issues associated with UAV swarms is the coordination of such a system. Smaller groups of UAVs can be operated by humans controllers, but this becomes a more difficult feat for bigger swarms. Automated work assignment becomes crucial in order to maintain efficient task distribution and execution for larger swarms and workloads. In general a UAV swarm can be viewed as a distributed processing system with additional constraints [6], since the tasks are bound to a physical location, and power and sensor limitations

need to be taken into account. Some of the approaches of scheduling workloads under those constraints will be addressed in the next section. A UAV swarm can be comprised of multiple UAV types with different processing and sensing capabilities. This poses an additional challenge since under- and over-utilization of resources may occur [7].

Simulation plays an important role in determining the quality of a UAV swarm system. It allows for the exploration of different scenarios without the overhead and cost of setting up and using an actual UAV swarm. Algorithms and applications can be debugged and verified before deploying them to a physical system, preventing potentially dangerous situations [8]. However, a simulation may not be able to completely and accurately replicate a physical setup due to its complexity. Most simulations focus on the aspects that are relevant to the system and choose abstract some features of the real world. Despite these drawbacks simulations are useful to examine promising ideas before applying them to a physical system.

This project focuses on the implementation of a simulation tool which allows for the evaluation of algorithms for real-time task assignment in UAV swarms. It takes into consideration heterogeneous UAV swarms and workloads with the option to offload computation to the cloud. The simulator was first implemented for class project [9] and has since been extended.

CHAPTER 2

BACKGROUND

2.1 Scheduling

Scheduling algorithms are used to determine when and how tasks or jobs are executed by the UAV swarm (and potentially the edge and cloud servers). There are several objectives that an algorithm can try to optimize, such as the minimization of power consumption [10] [4], the minimization of execution latency [7], avoiding UAV failures [11], maximizing the reward associated with a task [12] [13] [14], or a trade off between the named objectives. In addition to task assignment, schedulers may also be involved with path planning and routing of the UAVs [15] [1], as well as planning when to charge the UAVs to prevent complete depletion of the swarm [14].

2.1.1 Problem Formulations

The scheduling problem can be interpreted in a variety of ways, depending on the assumptions that are made. Some systems only consider offline scheduling, meaning that the entire workload is known beforehand. The problem in this scenario can be viewed as the vehicle routing problem [16], where the fleet of vehicles is the swarm of UAVs. Another paper models it as a variant of the Knapsack problem, where the UAVs represent the boxes and the tasks the items to fit within the boxes [12].

Real-time task scheduling, where tasks may arrive at any time, can be de-

scribed as the assignment problem, where an assignment between a set of UAVs and tasks needs to be found. Similarly, the problem can also be seen as an instance of the stable matching problem [4]. In Wei et al.'s framework each UAV maintains task queues [17]. They describe the problem of inserting a new task into a UAV's task queue as the traveling salesman problem. The UAV is a salesman and the locations to visit are given by task locations, including the ones in the queue previously and the new one. This approach assumes that the UAV has been assigned the task by a controller, if that is not the case the problem of inserting a task into any UAVs schedule can simply be represented by a multi traveling salesman problem [18].

2.1.2 Algorithms

A variety of scheduling algorithms have been explored for the UAV swarm task scheduling problem. The proposed algorithms may also address the routing problem in addition to task assignment.

Linear Programs and Solvers

A lot of the aforementioned work includes a form of integer or linear problem optimization representation of the scheduling problem. As such it comes to no surprise that ready to use optimization software is used to solve the problem [19] [12] [20]. A popular choice is the CPLEX solver by Intel which implements the simplex algorithm. While the solver will find an optimal solution, it is not necessarily computationally efficient. As a result this approach is not necessarily suited for real-time scheduling in which the solver would have to be called

for every incoming task [20]. The complexity issue can be addressed through heuristic algorithms and accepting potentially sub-optimal solutions. Some of these heuristic algorithms are covered in the next sections.

Nature Inspired Algorithms

Numerous heuristic optimization algorithms are inspired by principles of nature, examples being genetic algorithms, particle swarm optimization, and ant colony algorithms. These types of algorithms can find solutions at a computationally lower cost, but require tuning of different parameters.

Genetic Algorithms (GAs) are used for route planning [4] [18] and to solve the task assignment problem [21]. A genetic algorithm simulates the evolution of a random population of solutions into a good population of solutions. Each solution is associated with a measure of goodness; the fitness. The evolutionary process usually involves the combination of fit solutions into a new solutions until a certain termination point is reached. In the case of route planning [4] a solution is given by the ordering of way points on the route. The fitness of a solution is given by the reciprocal of the total sensing cost. The bundling of tasks that should be executed by one UAV can be achieved through a clustering algorithm which groups tasks based on their location [18]. In task assignment [21] a solution represents a feasible assignment between UAVs and tasks. A solution is considered good if the expected cost associated with it is low.

Particle Swarm Optimization (PSO) is used to solve offline task assignment for UAVs. In a PSO the solutions are particles within the search space. Each particle has a velocity associated with it and constantly adjusts its position. Dif-

ferent formulations of the problem, such as vehicle routing problem with time windows [16] and a variant of the parallel machine scheduling problem [15], have been explored with PSOs. Optimization objectives are the minimization of total makespan [16] and the minimization of total cost of the execution [15]. Khosiawan et al. compare results of their PSO implementation to a GA implementation, and demonstrate that the PSO is simpler while also being more effective [15].

Ant colony algorithms such as Ant Colony Optimization (ACO) [22] [23] and Dynamic Ant Colony Labor Dision (DACLD) [24] have been successfully deployed for UAV swarm coordination and cooperation. Ant colony algorithms model the behaviour of agents in the form of ants, where behaviour within a search space is influenced by pheromone trails left by other ants. Rosalie et al. combine an ACO implementation to maximize coverage with chaos dynamics to generate unpredictable flight patterns in a military context [22]. The UAVs leave repelling pheromones along their flight path to avoid coverage of the same area within a short period of time. Wu et al. use the principle of ACLD to assign tasks to UAVs in a distributed manner [24]. Each task is associated with a pheromone, and if a certain threshold is reached an ant, representing a UAV, will follow the stimulus and execute the task. Hu et al. use a clustering algorithm to group tasks, then assign the task clusters to UAV teams with the Hungarian algorithm, and finally use ACO to assign tasks within a team [23]. They conclude that this hierarchical approach can greatly reduce the complexity of the task assignment problem.

Game Theory

In game theory agents make strategic decisions based on incentives and rewards. Utilizing elements of game theory has gained popularity in work covering UAV swarm scheduling and coordination.

Zhang et al consider the assignment as a non-cooperate dynamic game with incomplete information [10]. The UAVs bid on arriving tasks based on the expected payoff, which is determined by execution cost and reward. A similar auction based approach is found in other works [25] [13] [1]. Generally the first phase is a distributed bidding phase, in which each UAV calculates the reward for each task and adds selected ones to its bundle. A consensus phase follows to resolve any possible conflicts that might have occurred during the first phase.

Mukherjee et al. use a Nash bargaining game to offload data to other UAVs in a swarm of heterogeneous UAVs in order to prevent under-utilization of resources [7]. The bargaining game is between the probability of the node that generated the data processing it as well, and offloading the processing to another UAV in the swarm. The goal of the game is to minimize processing lag and offloading delays.

A Nash bargaining game is also applied in the work of Motlagh et al. in order to find a trade-off between two scheduling objectives; power consumption and delay [26]. The implementation of the trade-off based scheduler achieves fair results in terms of power consumption and delay. However, this comes at the cost of a drastically increased runtime of the algorithm for larger problems.

2.2 Computation Offloading

Computation offloading describes the offloading of computationally intensive tasks from a UAV to another, more capable server. The resources of UAVs are limited, so computation is expensive in terms of power consumption, but also latency. Delegating computation preserves the UAVs resources for other tasks that specifically require the UAV's capabilities, such as taking photos or sensing. It can also lower the execution time needed for the computation. The offloading process is associated with a cost; the transfer of data to the server which will process it. It requires some sort of network connection and will incur transmission latencies. The transmission process can also consume the UAVs power and lead to bandwidth saturation [3]. Alternatively the data could be stored on the UAVs and be processed later, this is only applicable for systems that do not require real-time processing of data.

Mukherjee et al. [7] propose the idea of computational offloading within a heterogeneous swarm. The swarm is comprised of different UAVs, most are fitted with scalar sensors, while one is fitted with multimedia sensors. The amount of data generated by the latter is substantially higher. In order to effectively utilize the available resources the data is distributed among the swarm members for processing. Song et al. [19] enable splitting of a job into subtasks when a UAV runs out of power, so the job can be handed off to another UAV. This allows the swarm to tackle long-running jobs that might be infeasible to execute otherwise.

Luo et al.'s [3] framework utilizes cloud resources to offload data to cloud servers for real-time processing. Video data is pre-processed by the UAV that

captured it, and the results are used by a scheduler to determine whether to send the video data to the cloud for more complex analysis. The pre-processing of data can avoid unnecessary data transfers and latencies associated with it. Pre-processing of data on UAVs is also considered in the work of Liu et al. [27] [28]. In addition to cloud resources, edge servers are also taken advantage of as an intermediary between UAV and cloud. This allows for a trade off between latency and compute power. The edge server can provide low latency and energy efficient resources, while more demanding computations can be sent to the cloud at a higher migration cost. Systems following the UAV-edge-cloud approach can benefit from lower cloud resource costs if data is (pre-)processed on the edge instead [29].

CHAPTER 3

SYSTEM DESIGN

The proposed simulation framework is comprised of multiple components. The UAV swarm consists of a set of potentially heterogeneous UAVs and executes a workload of tasks or jobs. A job groups multiple tasks into a single unit and allows for specification of dependencies between tasks. The workload and arrival rate of tasks is not previously known, tasks are scheduled in real-time as they arrive. Task assignment is handled by a controller which runs in a centralized fashion on a base station on the ground. A decentralized controller running on each UAV would be feasible as well, but is not considered in this project. The controller is driven by a specified scheduling algorithm. Cloud resources for computational offloading are available as well.

3.1 Interface

The system can be easily configured or augmented for simulations. The configuration is handled through a set of files; one file is used to set the parameters for the simulation, and additional files are used to define jobs. A job configuration is a task graph in JSON format for easy modification and portability. This allows a user to specify large simulations and jobs with multiple phases on an abstract level instead of focusing on the low level details. Simulations can be run to evaluate behaviour of UAV swarms under certain conditions before transferring the simulation setup to a real life scenario. Compatibility plays an important role to enable easy transfer of specifications between simulation and a physical system [8].

3.2 Scheduling Algorithms

The system is designed for real time scheduling. The scheduling problem under these circumstances can be considered an instance of the assignment problem. Whenever a task arrives or a UAV finishes its assigned task, the objective is to find a good match between available UAVs and pending tasks. A scheduling algorithm can optimize a match in multiple ways. The objective considered in this project is the minimization of power consumption for energy efficiency. Since the system is considered real time, the runtime of a scheduling algorithm also has to be taken into account. Long execution times could lead to further delays and cause issues in a physical system.

All scheduling algorithms need to take into account the constraints that are set by the UAVs and tasks, depending on their type. In order to execute a task a UAV needs to have the resources to execute the task. These resources are defined in terms of equipment as well as power. A UAV needs to have the sensors required to execute the task, e.g. taking a photo requires a camera mounted to the UAV. Additionally, the UAV should have enough battery capacity left to travel to the task location, execute the task and return to the base station in case it needs to be recharged. Power consumption associated with a task includes travel, task execution and, if needed, data transfer. A job specification may pose additional constraints since task dependencies need to be considered. A task can only be executed, if its preceding tasks have been completed.

It is assumed that the controller in charge of scheduling has a global view of the system. General information about different task types, such as average execution times and sensor requirements, is known. Over the course of the simu-

lation the location newly arriving tasks becomes available as well. A centralized controller will have information about all of the UAVs in the swarm, while a decentralized controller would only hold information about the specific UAV that it is running on. The information associated with a UAV includes its location, its velocity, its battery status and the processing power of its CPU. The centralized controller obtains the information through the exchange of messages whenever the status of a UAV changes.

3.2.1 FIFO

The FIFO policy is included in the system as a naive baseline to compare the performance of other, more sophisticated, scheduling policies against. A queue, sorted by arrival time, is utilized to keep track of pending tasks. To assign a task to a UAV the first item of the queue will be dequeued and assigned to the first available UAV that is able to process it.

3.2.2 Local Optimization

A greedy algorithm is used to find a solution with low power consumption through local optimization. In each step of the assignment process an available UAV is assigned the task with the lowest expected power cost for that UAV. This approach may not find the globally optimal solution, but can be a simple approximation of that solution.

3.2.3 Global Optimization

In contrast to local optimization, global optimization is more complex and can achieve better, if not optimal, results.

Given a set of available UAVs $U = \{u_1, u_2, ..., u_m\}$, a set of pending tasks $T = \{t_1, t_2, ..., t_n\}$ and a cost matrix C, where the entry $C(i, j)$ describes the cost associated with UAV u_i executing the task t_j, the global optimization problem can be expressed as follows:

Minimize

$$\sum_{i \in U} \sum_{j \in T} C(i, j) * x(i, j)$$

Constraint to

$If\ |U| < |T|$:

$$\sum_{j \in T} x(i, j) = 1, \forall i \in U$$

$$\sum_{i \in U} x(i, j) \leq 1, \forall j \in T$$

$Else$:

$$\sum_{j \in T} x(i, j) \leq 1, \forall i \in U$$

$$\sum_{i \in U} x(i, j) = 1, \forall j \in T$$

Since the assignment problem may be rectangular the constraints are split into two cases. The first case describes the problem when more tasks need to be scheduled than UAVs are available, the second all other scenarios. The entries of the cost matrix are calculated according to the scheduling objective, in this project the cost is the power consumption of the UAV executing the task. If a

UAV cannot execute the task, e.g. it does not have enough remaining battery of the sensors required by the task, the cost is set to infinity.

The solution for the global optimization problem can be found with a variety of read-to-use solvers and (meta)-heuristic algorithms.

Hungarian Algorithm and Minimum Cost Flow Algorithm

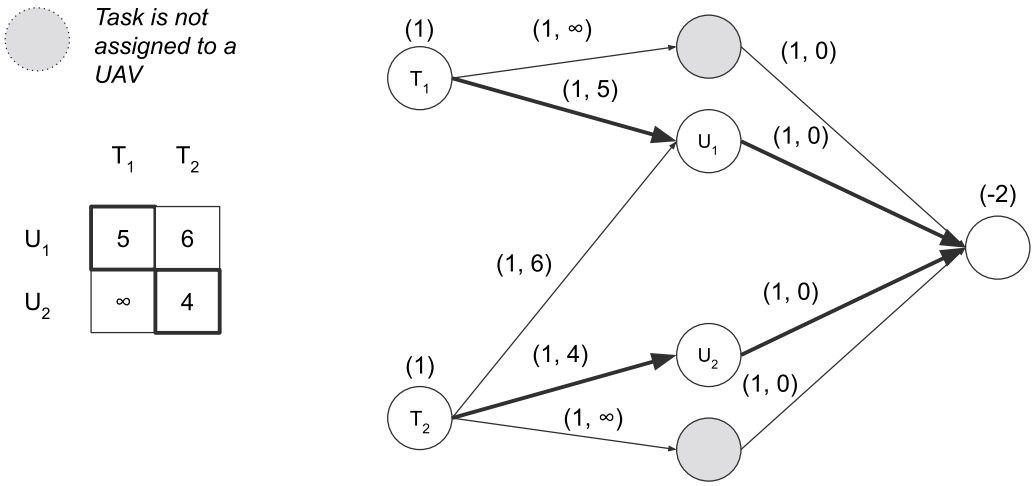


Figure 3.1: Cost matrix and graph representation. Tasks are supply nodes, connected to the nodes of UAVs capable of executing the task. The edges are labeled with (capacity, cost). The grey nodes represent not scheduling a task, penalized with an infinite cost. Flow is directed towards a demand node with a demand equal to the number of tasks.

The cost matrix mentioned in the problem formulation can also be used as input for the Hungarian Algorithm. An efficient variant of this, the Jonker-Volgenant algorithm, uses a shortest-augmenting path approach based on graph theory [30]. A Scipy implementation of the Jonker-Volgenant algorithm was used in this project. After execution of the algorithm the obtained assignments are ver-

ified, if the cost of an assignment is infinity, it is not valid. Only valid assignments are passed on to the UAVs.

A minimum cost flow graph representation (see figure 3.1) of the problem can be used as the input for solving algorithms as well. The tasks are represented as supply nodes and the UAVs as intermediary nodes to the demand node. Each task node has edges to the nodes of the UAVs that are able to execute it. The capacity of all edges is set to one, the cost of the edge is the cost of the task execution. The cost of not scheduling a task is considered by adding an additional node per task that represents leaving the task unassigned. The cost of not scheduling a task can be modeled as infinity or as the power cost incurred by a UAV idling. A Google OR-Tools solver, which utilises a cost-scaling push-relabel algorithm, is used to solve the minimum cost flow problem in this project. From here on the algorithm solving the Minimum Cost Flow Problem will be referred to as MCFP.

Conversion from one representation of the problem into another is simple. Both solvers are able to find optimal solutions. However, the network flow problem solver is able to solve any network flow problem, while the Jonker-Volgenant algorithm is optimized for the 2D assignment problem specifically.

Ant Colony Optimization

Ant Colony Optimization imitates the behaviour of ants finding paths when searching for food. After discovering a food source ants lay down pheromone trails as a guide for other ants.

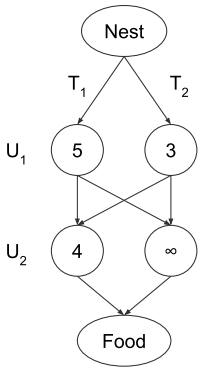


Figure 3.2: ACO search graph based on the cost matrix.

Over time the pheromone will evaporate and lose intensity. Longer paths will require more time to traverse, providing more time for pheromones to evaporate. The problem can be encoded as a graph which the ants traverse to get from their nest to a food source (see figure 3.2). The graph is an alternative representation of the cost matrix (see figure 3.1). In addition to the cost matrix a pheromone matrix is maintained to keep track of the pheromone intensity for each node. The algorithm simulates a group of ants traversing the graph for multiple iterations. Each ant traversal leads to local pheromone updates to the visited nodes. The shortest path for the ants within the iteration is kept track off. After one iteration is completed a global pheromone update takes place to update the pheromones of the shortest found path. The ACO implementation and tuning of parameters in this project is based on work of Piao et. al [31]. In theory ACO is able to find good or even near-optimal solutions while following simple concepts. In practice the quality of solutions depends on the implementation and tuning of the algorithm for the specific problem it is applied to.

3.3 Computation Offloading

The system allows for computational offloading to ease the burden on a single UAV. Computationally intensive tasks may be offloaded to other UAVs in the

swarm, as well as the cloud. Offloading of work within the swarm is implicitly possible through the definition of jobs. As an example, instead of considering taking a photo and running object detection on it as a single task, it can be split into two tasks as part of a job. The two tasks belonging to the job are then linked through a dependency. This is advantageous in a heterogeneous UAV swarm where one type of UAV generates a large amount of data which incurs a high processing time and another generates a comparatively small amount of data. The splitting of tasks allows for the distribution of computational load across the entire swarm.

Cloud resources are a powerful alternative to computation within the UAV swarm. Offloading computation to the cloud can drastically reduce the processing time of data, although it is slightly penalized by network delay caused by data transfer. Cloud offloading can be used in addition to any of the proposed scheduling algorithms. If cloud offloading is enabled, a task that can be run on the cloud will be processed by the cloud instead of being assigned to a UAV.

CHAPTER 4

SYSTEM IMPLEMENTATION

The simulator is implemented in Python 3 following an object oriented programming paradigm. The simulator, compute resources, jobs, tasks and controllers are represented by a class each. This allows for easy configuration and extension of the simulator. The project code can be found on GitHub: `https://github.coecis.cornell.edu/cfs232/droneswarm`.

4.1 Simulation

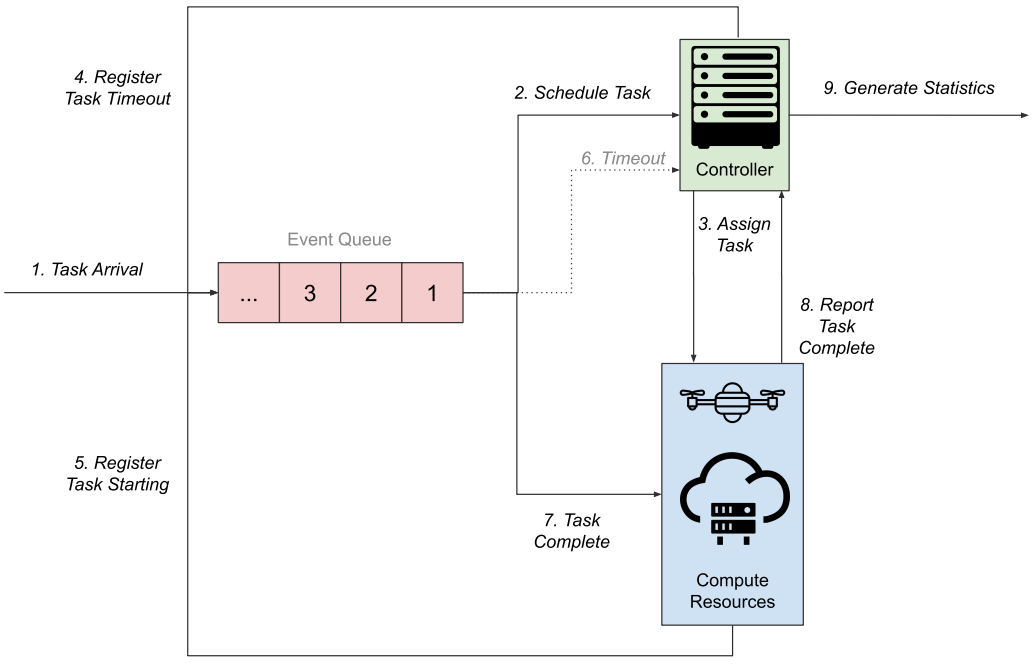


Figure 4.1: Simulator framework.

The simulation is driven by an event queue. The elements of the queue are essentially triggers for function calls that handle certain simulation events. Triggers are added to the queue for arriving tasks or jobs, UAVs finishing tasks,

and to monitor the UAVs battery status. The event queue is sorted in ascending order by timestamps that correspond to the time that the event should occur. The simulation loop pulls events from the queue, updates the simulation to the timestamp of the event and calls the specified function with the provided arguments. The task assignment within the framework is visualized in figure 4.1.

4.2 System Setup

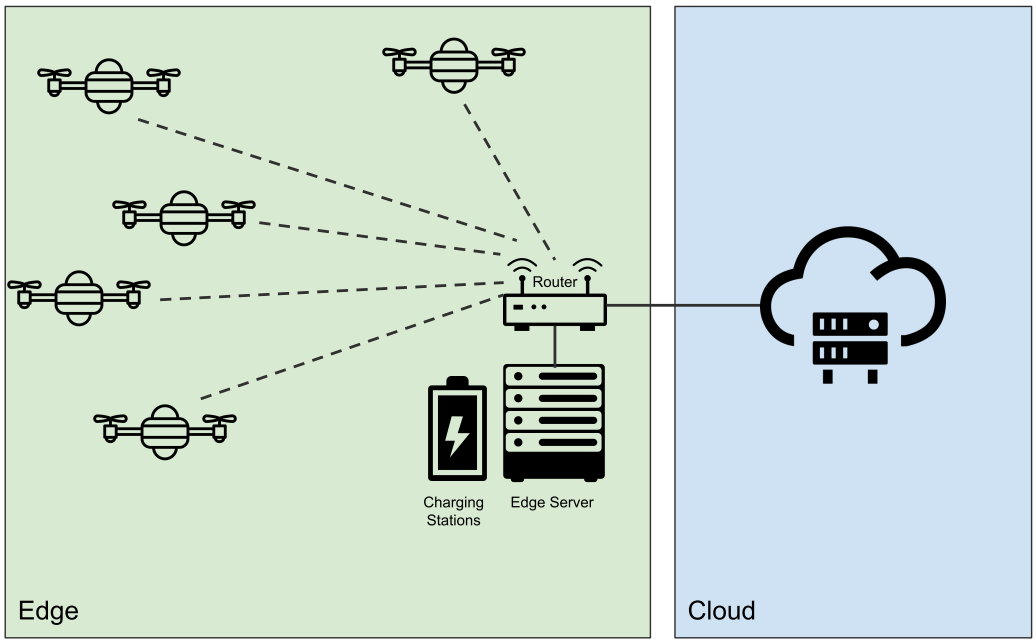


Figure 4.2: System setup.

4.2.1 Environment

The simulation is set in a in a grid of size 10mx10m. For purposes of simplification is assumed that there are no obstacles within the grid, obstacle avoidance is not considered. The base station is located at the center of the grid and is the

is the starting point for the UAVs in the simulation. It consists of an edge server, a router, and charging stations for the UAVs. Charging spots are assumed to always be available. The bandwidth of the router can be specified, but is assumed to be symmetrical for both up- and downloads. Network delays are considered to a certain extent, but are modeled in a simplified manner. The calculation of the network delay takes into account the amount of data that is transferred, the bandwidth and potential network congestion due to high load.

4.2.2 UAVs

The UAV swarm can consist of different types of multi-rotor UAVs, these can be specified in a configuration file. A type of UAV is associated with a number of parameters, such as a velocity, and the sensors it is equipped with. The velocity is assumed to be constant throughout the simulation. The power consumption and CPU capability is parameterized as well, since these are important constraints that need to be taken into consideration for scheduling.

A UAV consumes power throughout the simulation by traveling, executing tasks, transferring data and idling. Each UAV type has a certain battery capacity which defines the maximum charge a battery can hold. The battery will deplete at a specified rate, and the UAV will return to the base station to recharge once a battery load of under 15% is reached. At this point the UAV is considered unfit the execute any further tasks. Ideally no UAV should deplete before it can return to the base, so the threshold is chosen to ensure that a UAV is able to reach the base station with the remaining battery.

The latency for a task is scaled by a CPU scaling factor to approximate the

execution times of a task on different CPUs. This factor is defined relative to the execution time the task would have on the cloud. The hardware of UAVs is generally far less powerful than the cloud resources available.

4.2.3 Cloud Resources

The cloud provides additional compute resources that can be taken advantage of to reduce the computational burden on the UAVs. A task can only be assigned to be processed on the cloud if it is explicitly executable by the cloud, these are tasks that do not involve sensing. The cloud is abstracted to a single resource that is infinite and always ready. Scheduling a task to run on the cloud will incur a higher network latency. The latency used in the simulator is based on experiments uploading data to the Microsoft Azure cloud with the Azure Speed Test Tool [32].

4.2.4 Workload

The workload for the simulations can consist of multiple task and job types. Execution latencies for tasks are based on experiment latencies of actual applications.

Tasks

A task is a unit of work that cannot be subdivided into smaller subtasks. A task in the simulation will arrive according to a specified task arrival distribution.

The task duration is calculated based on a defined average latency and standard deviation from the latency according to a Gaussian distribution. A task deadline is set upon arrival, if a task is not completed by the deadline it is considered a failed task. The deadlines are soft deadlines, and tasks are still executed if they do not meet their deadline.

The location of the task within the grid is randomly generated. Non-sensing tasks do not have a designated location since they can be executed at any location, including the cloud. The sensors required for sensing tasks, e.g. a camera to take a photo, are specified in the task description. The amount of data generated by a task type can be specified as well. A task may also be part of a job, as described in the next section.

Jobs

A job consists of multiple tasks that may or may not depend on each other. Task dependencies can be defined through a JSON representation of a task graph. The tasks within a job do not have their own arrival time, deadline, or priority. The arrival time and priority is inherited from the job, so all tasks of a job arrive at the same time and have the same priority. The job deadline only applies to the job as a whole, so a task within a job cannot fail by itself, only the entire job. Scheduling a job is broken down into scheduling all of its tasks, and a job is considered complete as soon as all tasks belonging to the job have been completed. Data flow between tasks is taken into account based on the specified task dependencies.

4.2.5 Controller

The controller class is the unit driving the task assignment process. Each controller schedules task batches according to the scheduling policy associated with it. The batching process can be configured to be based on a batch interval, the batch size, or an adaptive batch size. Batching allows for better optimization with the more advanced scheduling algorithms.

The controller maintains information about the current status of the system. UAVs are kept track off based on their status, UAVs can be either available, busy, charging, or depleted. Similarly tasks are into pending, in-flight, completed, timed out, and failed tasks. The controller sets a timeout for each task once it is assigned to a UAV to keep track of straggling tasks. If a task is not completed before the set timeout, the task will be considered timed out.

The scheduling algorithms discussed in this project optimize based on power consumption. The estimated power cost of UAV i executing task j is calculated by the controller as follows, where c represents a cost, l a latency, s a scaling factor: $c_{total}(i,j) = c_{wakeup}(i) + c_{travel}(i,j) + c_{task}(i,j)$ where $c_{travel}(i,j) = dist_{xy}(i,j) * s_{xy}(i) + dist_{z}(i,j) * s_{z}(i)$, and $c_{task}(i,j) = (l_{execution}(j) * (s_{compute}(i) + s_{idle}(i))) * s_{battery}(i)$, and $l_{execution} = l_{task}(j) * s_{slowdown}(i) + l_{network}(i,j)$.

For a controller's cost estimate average latencies and approximations are used, these may differ from actual latencies. Wakeup cost is only non-zero if the UAV is in standby.

CHAPTER 5

EVALUATION

5.1 Methodology

5.1.1 Applications

The applications used in the experiments are either a single task or consist of a job with multiple phases. Jobs that are split into multiple phases of simple tasks follow the concept of modular design. Smaller application modules have gained popularity with the rise of serverless functions and microservices, since they are faster to develop and deploy.

The single tasks in this project are taking a photo and sensing the temperature at a certain location. A job with two phases involves data generation, such as the aforementioned tasks, and processing of the data. Processing tasks of photos take the form of image classification, facial recognition and image segmentation. A three-phase job related to computer vision consists of taking the photo, pre-processing it by running basic classification and following up with a more in depth recognition.

5.1.2 Parameters

The parameters defining task latencies and UAV features are based on empirical values. The UAV parameters are modeled after trial runs with Parrot AR 2.0 drones by the group that originally implemented the simulator. Their report

mentions that further validation of certain power scaling parameters might be needed [9], unfortunately access to the UAVs was limited due to Covid-19. The task latencies are based on measurements obtained by running implementations of the applications.

5.1.3 Task Arrival

The rate at which tasks arrive in real time to be processed can impact the behaviour of different scheduling algorithms and techniques. To observe potential differences experiments can be run with a selection of five arrival distributions. The task arrival interval is set to 5s and the frequency of arrivals for the interval is set to 100 tasks for most distributions. A bursty task arrival entails all tasks for an arrival interval arriving at the beginning of the interval, creating a burst of pending tasks. Arrivals according to the exponential and Poisson distributions are sampled within a time frame based on the interval and frequency for each task. To model rare events that do not saturate the swarm a lower task frequency per interval, namely 10, is selected. Rare event arrivals are sampled from the Poisson distribution as well. Finally, a mixed distribution combines bursty and Poisson rare task arrivals at a 1:1 ratio. Experiments, unless specified otherwise, use a mixed task distribution.

5.1.4 Generating Results

The experiments are run on an Intel® Core™ i7-10510U CPU @ 1.80GHz × 8 (4 physical cores). Experiments are run multiple times and results are averaged

to ensure a fair comparison with regard to the inherent variance of events in the simulation. The exception to this rule are large scale experiments. They are only run once since the simulation duration drastically increases and they are less prone to a large degree of divergence. The experiments are run for a swarm size of 100 UAVs unless specified otherwise.

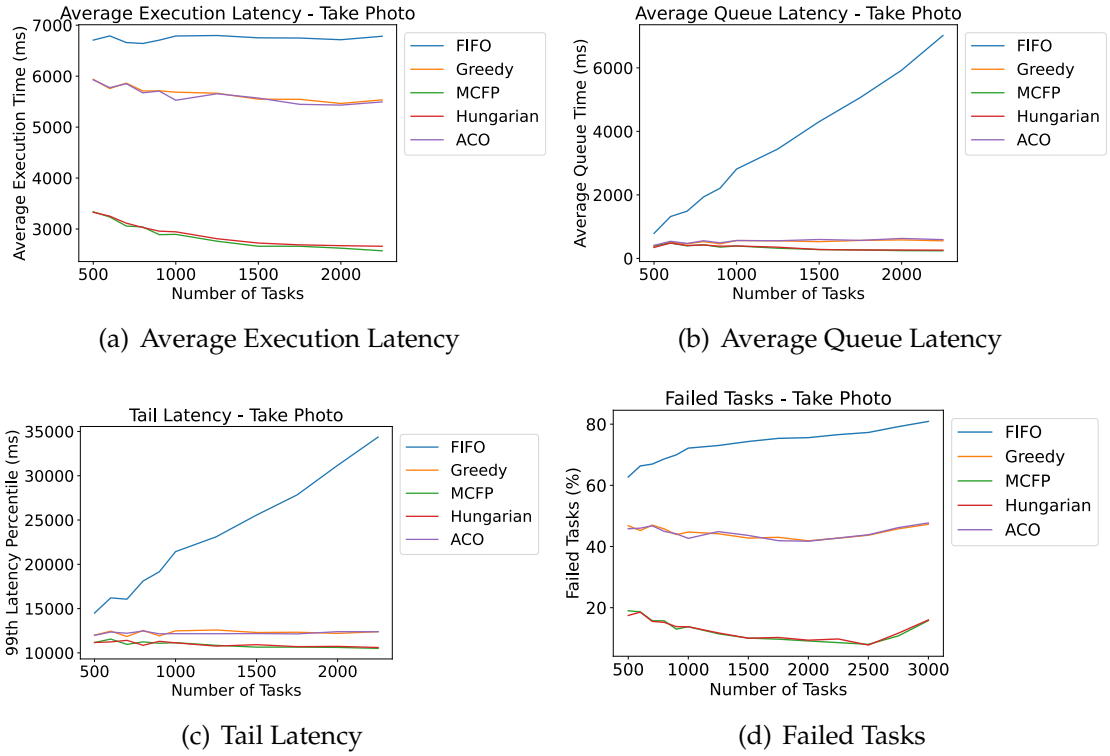
5.2 Results and Discussion

5.2.1 Scheduling Algorithm Comparison

The scheduling algorithms are compared for different swarm setups and workloads. All scheduling algorithms, except FIFO, optimize for power consumption. Metrics considered are average and tail latencies, the percentage of failed tasks and the average time it took the scheduler to find an assignment for a task (average scheduling time).

Homogeneous UAV Swarms

Comparison of the scheduling algorithms shows that the read-to-use solvers used for the Hungarian and MCF problem are able to achieve the most promising results (see figure 5.1). As anticipated FIFO performs the worst; FIFO is not an optimizing policy, but a simple approach with little overhead. The greedy algorithm and ACO, local optimization and solution approximation, lie between the aforementioned algorithms in terms of performance. The greedy algorithm does not find globally optimal task assignments due its local optimization ap-



(a) Average Execution Latency

(b) Average Queue Latency

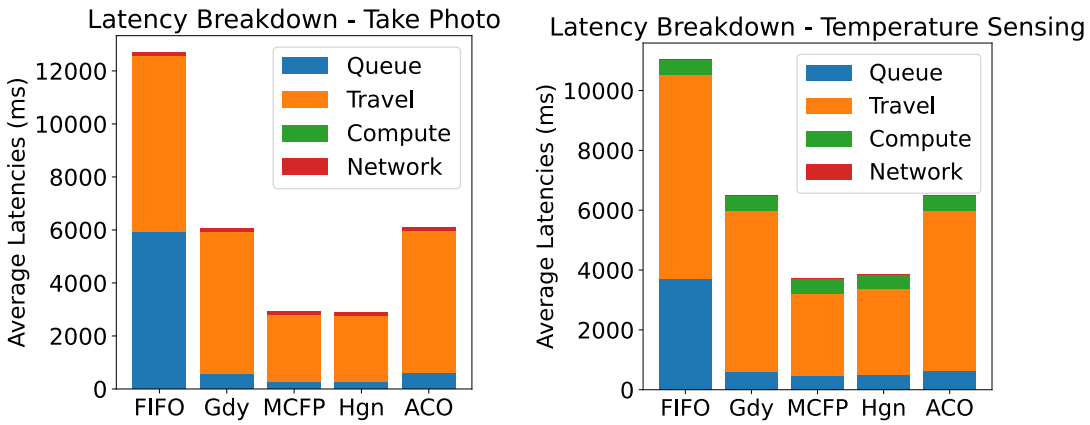
(c) Tail Latency

(d) Failed Tasks

Figure 5.1: Comparison of scheduling algorithms for the single-phase task of taking a photo.

proach. The ACO has a similar performance to the greedy algorithm. This is somewhat expected since it is a meta-heuristic, but the quality of assignments (or lack thereof) could also be a result of misconfigured parameters in the implementation.

The decrease in average execution time for the optimizing algorithms can be explained by their optimizing nature. A larger workload will lead to more tasks in the task queue, meaning that more tasks are available for the assignment optimization. Since the simulation grid is limited in size, the probability of a task being closer to a UAV is higher with a higher number of pending tasks. As a result the travel latency can be reduced and the average latency and failed tasks converge to a minimum. Figure 5.1 (d) shows that this is in fact merely a



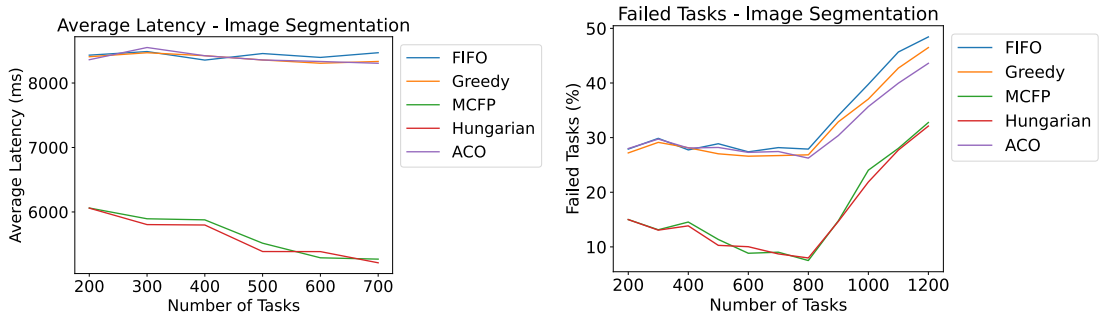
(a) Latency Breakdown Taking Photos (2000 Tasks)

(b) Latency Breakdown Temperature Sensing (1100 Tasks)

Figure 5.2: Latency breakdowns of the total latency from task arrival to task completion.

dip, as failed tasks increase again under a high workload.

Figure 5.2 shows the latency breakdown of two task types across the scheduling algorithms. Plot (a) shows the latencies for 2000 photos taken, and (b) for 1100 temperature sensing tasks. The workload was chosen to be moderately intense to reflect average execution times when UAVs have not yet depleted and are kept busy. The photo-taking task has a short compute time and an increased network latency compared to the temperature sensing task. Taking a photo does not require a lot of time, but generates a larger amount of data. The latency that can be influenced by a scheduling algorithm directly is the execution time. The execution time is composed of the travel and execution time. The algorithm may be able to reduce the computation latency if the swarm is heterogeneous and computation latency varies per UAV type. Travel latency can be optimized for tasks that need to be executed at a certain location, e.g. sensing tasks. FIFO, although keeping queue time low in theory, has the highest queue and travel latency. The FIFO policy results in higher execution latencies, so



(a) Average Latency

(b) Failed Tasks

Figure 5.3: Comparison of scheduling algorithms for the multi-phase job of image segmentation.

UAVs are occupied with a task for a longer time, leading to higher queue times for the remaining tasks. In contrast, MCMF and the Hungarian algorithm are able to reduce execution time which in turn reduces the queue time. ACO and local optimization are a sort of middle ground between the other approaches, achieving some degree of optimization.

The performance of the algorithms for multi-phase jobs might be slightly unexpected when compared to the performance for a single-phase task. This is caused by a different metric generation for jobs. A job is considered in-flight when its first task has been assignment, and is considered complete once its last task has been completed. None of the schedulers take the belonging of a task to a job into consideration. This may lead to a scenario where the first task of the job has been completed, but the remaining tasks of the job are still pending, since the assignment of other tasks is more efficient in terms of power consumption. A multi-phase job often consists of data generation and processing, where the data processing might be more energy consuming than its generation. Under such conditions starting a job would be cheaper than finishing it. Straggling tasks can lead to a long job execution time, although efficiency is maintained on

a task scheduling level. This is where FIFO has an advantage. Although FIFO might not optimize the assignments in terms of power consumption, it ensures that jobs are executed in order. This is reflected in improved performance in terms of job latencies (see figure 5.3). However, for a higher workload the optimizing algorithms still outperform FIFO in terms of failed tasks.

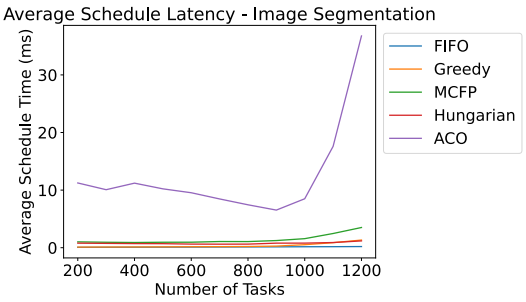
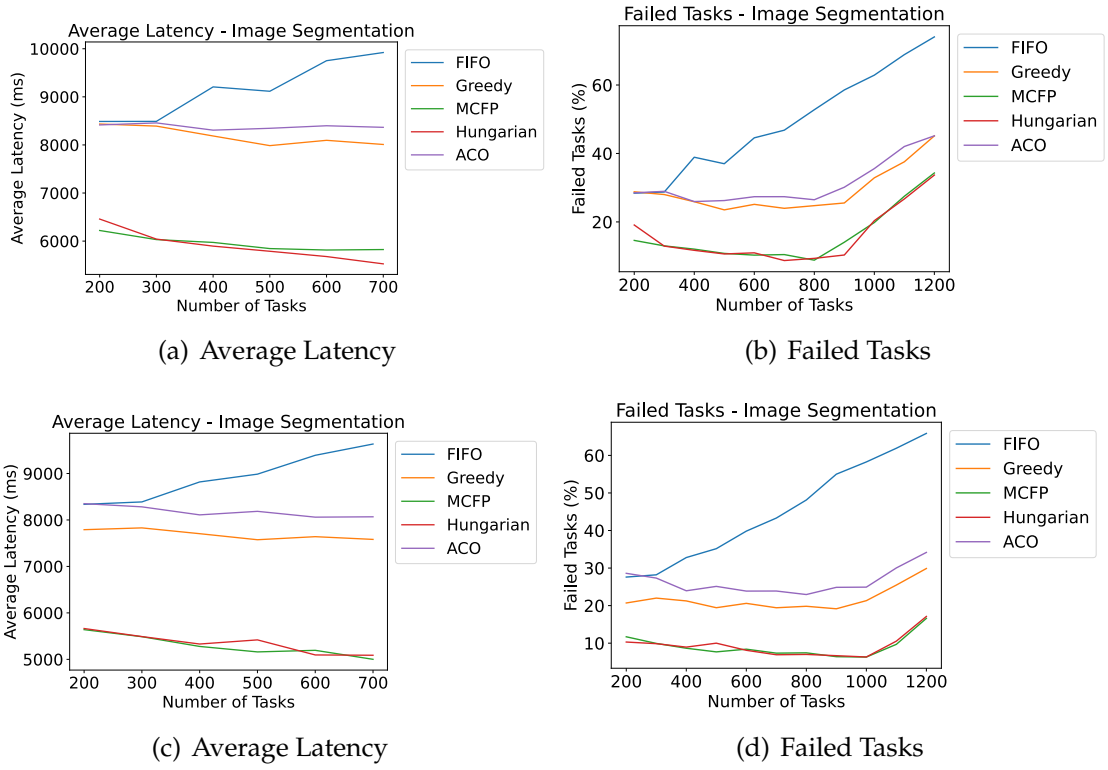


Figure 5.4: Scheduling algorithm latencies (average latency of the algorithm to assign a task to a UAV)

The algorithms do not only differ in performance, but also in their runtime. Figure 5.4 shows that all algorithms are fairly fast, except for ACO. FIFO and the greedy algorithm are fast due to their simplicity. The solvers for the MCFP and Hungarian problem are highly optimized implementations, which makes them very efficient in terms of latency. The ACO implementation likely suffers from a sub-optimal implementation, making it significantly slower than all other scheduling algorithms. This latency could be improved by either tuning or parallelizing the algorithm, or using available Python implementations.

Heterogeneous UAV Swarms

A heterogeneous UAV swarm is composed of multiple types of UAVs. UAVs may differ in their sensing and processing capabilities, as well as their power and velocity parameters. The previous section showed that multi-phase job execution time might suffer from straggling tasks when the data processing task



(a) Average Latency (b) Failed Tasks

(c) Average Latency (d) Failed Tasks

Figure 5.5: Heterogeneous UAV swarms, (a) and (b) show results for a swarm where 50% of UAVs are equipped with cameras. (c) and (d) show results for the same setup, but additionally the UAVs without cameras have more powerful CPUs.

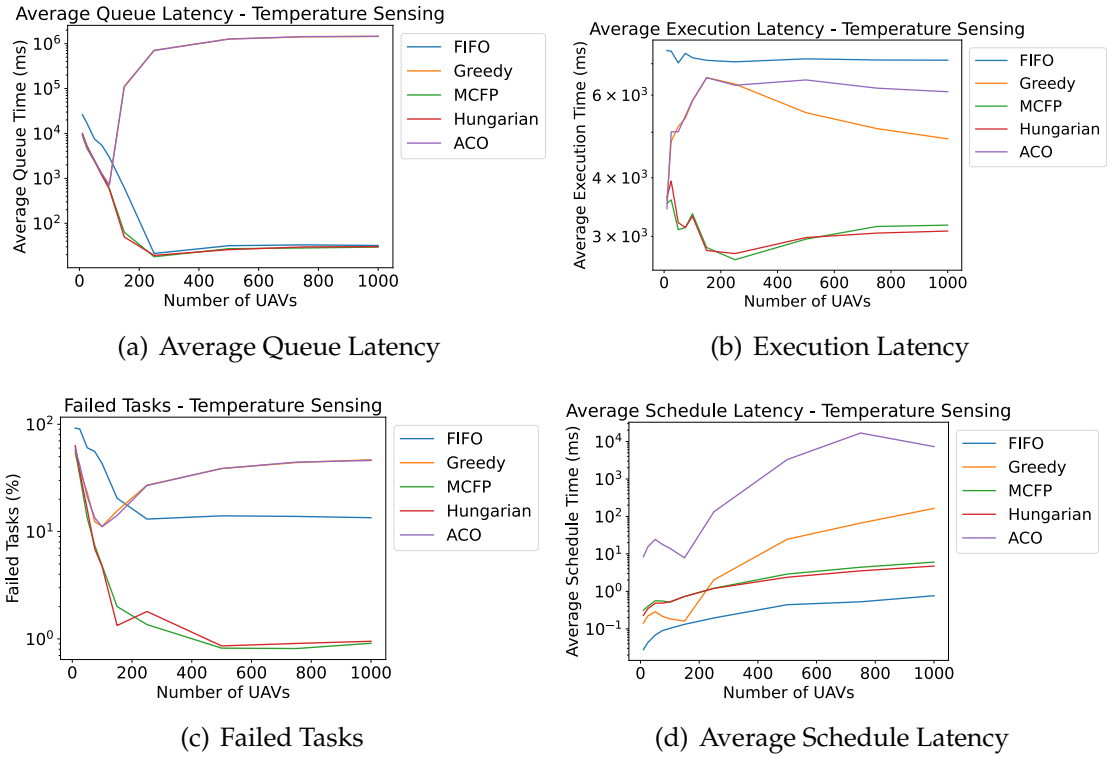
is more expensive in terms of energy consumption. Related work mentions that heterogeneous swarms with dedicated processing UAVs can help with efficient task execution [7]. Figure 5.5 shows experiment results for heterogeneous UAV swarms. The job workload is the same as in the previous section. The swarms are comprised of two types of UAVs with a 50/50 split. In plots (a) and (b) the UAVs are essentially the same, the only difference being that one set of UAVs is fitted with cameras while the other one has no sensing capabilities. The setup in plots (c) and (d) is similar, additionally the UAVs without camera are equipped with more powerful CPUs. The scheduling algorithms, except FIFO, are able to maintain efficient task assignment and achieve similar performance to a homo-

geneous UAV swarm. Plots (a) and (b) demonstrate that optimizing algorithms are able to achieve good performance for multi-stage jobs and swarms with less sensory equipment. Plots (c) and (d) show that modifying a share of the UAV swarm to have higher computational capabilities can improve overall performance in terms of latency and failed tasks. This is most apparent for the MCFP and Hungarian algorithms. This principle is demonstrated with CPU capabilities, but other modifications (e.g. longer lasting batteries or faster UAVs) could also aid UAV swarm performance.

Large UAV Swarms

As smaller UAVs become more affordable, the utilization of large UAV swarms becomes more feasible as well. Efficient task scheduling is crucial as swarm size increases. To evaluate the performance of scheduling algorithms for larger UAV swarms, experiments were run with 10-1000 UAVs, with a workload of ten temperature sensing tasks per UAV.

The results for the larger swarms resemble the previous experiment results to a certain extent (see figure 5.6). At a certain swarm size the number of failed tasks remains almost constant for FIFO and the optimal solvers, while it increases for the other algorithms. The greedy algorithm and ACO algorithm suffer from long queue times, which dominates the total latency of the tasks, leading to a higher number of failures. FIFO is a policy that minimizes queue time, which benefits the overall latency. The optimal solvers are able find assignments with a short execution time, so more tasks can be executed in a certain time frame. These aspects contribute to better results as the overall problem size increases. Plot (d) shows that the differences in algorithm latencies become



(a) Average Queue Latency

(b) Execution Latency

(c) Failed Tasks

(d) Average Schedule Latency

Figure 5.6: Results for large UAV swarms with 10-1000 UAVs under a workload of 10 tasks per UAV.

more pronounced for larger swarm sizes. The greedy algorithm benefits from its simplicity for smaller experiments, but is inefficient as the problem size increases. The ACO algorithm especially suffers from long run-times, which is why the scale of the larger experiments was limited. This could likely be ameliorated by further tuning of the implementation. The high schedule latency is an additional factor that worsens the queue time, since tasks are considered queuing until they are assigned to a UAV.

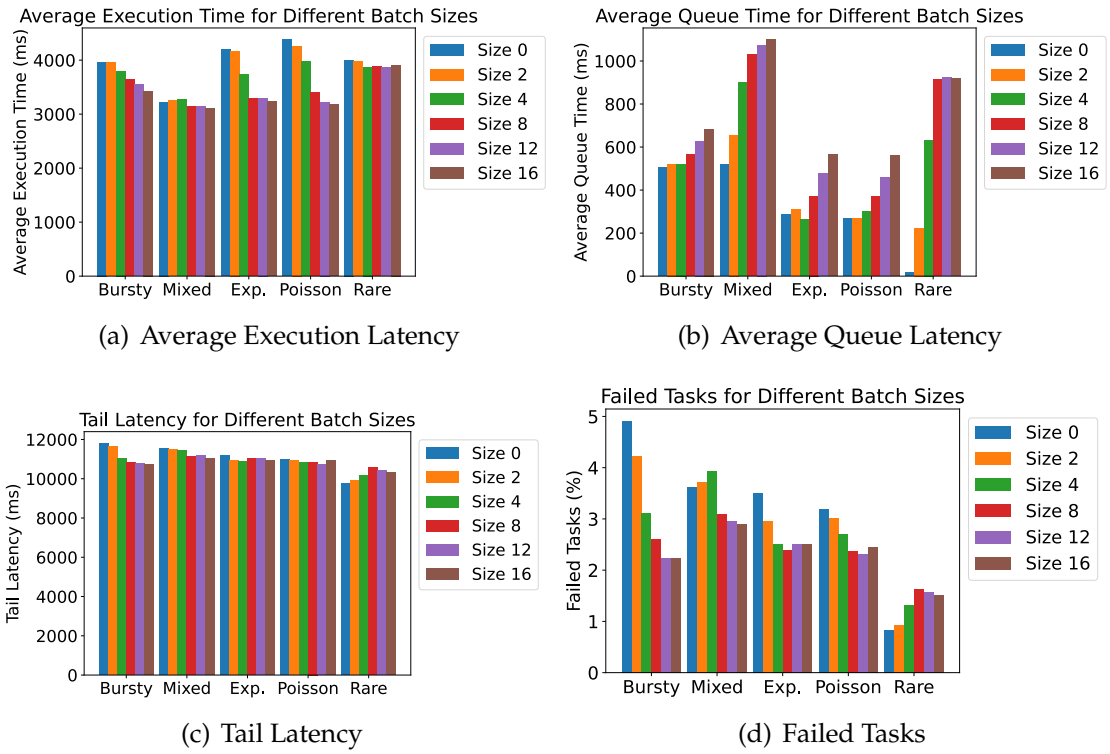
5.2.2 Task Batching

Task batching refers to the processing of multiple tasks at once. Here the processing refers to the task assignment process of the controller. Task batches might be scheduled after a certain period of time, after a certain batch size has been reached or a combination of both. These batching techniques are referred to as interval batching and (fixed) size batching hereinafter.

In the system tasks are maintained in a queue until they are scheduled to a UAV. In contrast to a regular batching scenario, tasks are not necessarily removed from the queue after a batch is processed. Tasks are only removed if an assignment to UAV could be found. The number of available UAVs might not correspond to the pending tasks, and available UAVs might not meet the tasks sensor and power requirements.

Batch size has an impact on optimizing scheduling algorithms. The following experiments demonstrate these effects for the MCFP solver, although they are, to some degree, applicable to any optimizing algorithm. The figures show results for a swarm of 100 UAVs executing 1000 temperature sensing tasks. The interval for interval batching is set to 100ms. The results obtained for multi-phase jobs do not show these effects as clearly since the metrics for jobs are generated differently, as discussed previously. The belonging of tasks to jobs is not necessarily taken into account when scheduling, so the effects of batching are not visible as clearly for jobs. The effects of batching do still hold true for the tasks within a job.

A larger batch size has a positive impact on average execution latency, tail latency and failed tasks, since a larger task pool enables more efficient assign-



(a) Average Execution Latency

(b) Average Queue Latency

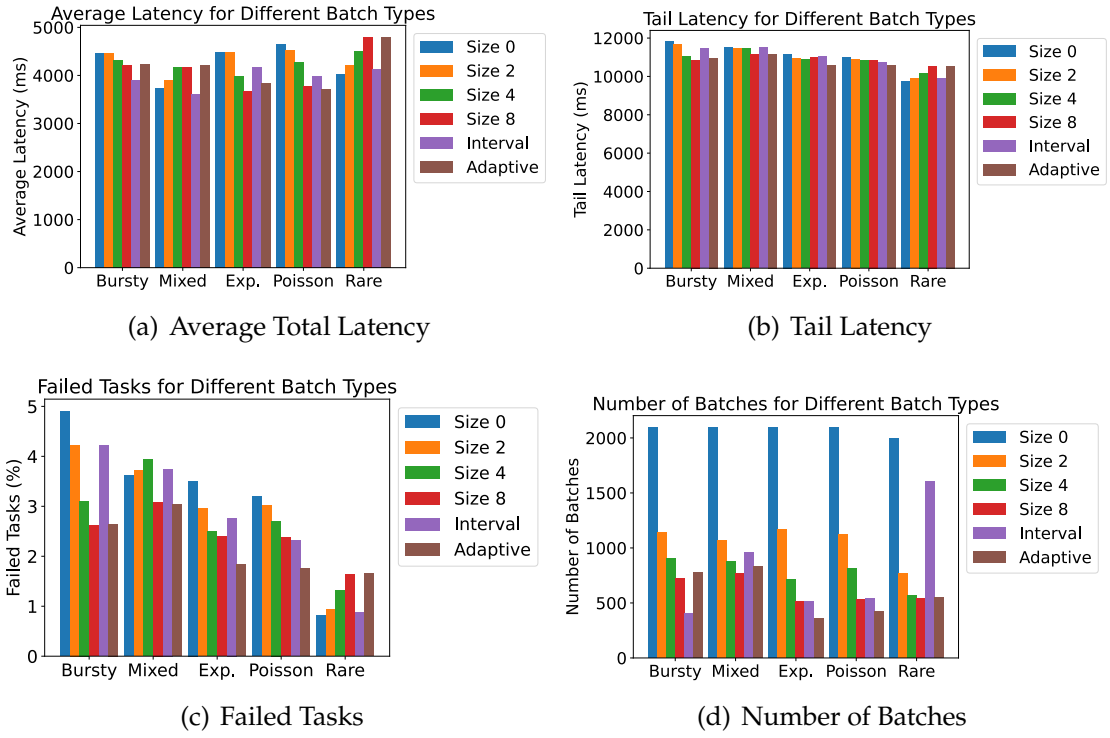
(c) Tail Latency

(d) Failed Tasks

Figure 5.7: Impact of batch size on average latencies and failed tasks for the execution of 1000 temperature sensing tasks.

ments (see figure 5.7). This statement only holds true to a certain extent, as rare events may suffer from a higher total tail latency due to the increased queue time. Queue time increases for larger batches across all distributions, but it is most extreme for rare events. Batches will take longer to fill up with rare events, causing a drastic increase in queue time. To ensure that batches are scheduled within a reasonable wait time, a batch interval is specified in addition to the batch size. If a batch was not filled and processed within the interval, a timeout will trigger the batch to be scheduled regardless of size.

The ideal batch size varies according to the arrival rates of tasks and current load of the system. Adaptive batching can be used to change the batch size throughout the execution to find a trade off between the number of batches



(a) Average Total Latency

(b) Tail Latency

(c) Failed Tasks

(d) Number of Batches

Figure 5.8: Comparison of batching techniques for 1000 temperature sensing tasks; adaptive batching, interval batching and fixed size batching.

scheduled and resulting task latencies and failures. The adaptive batching policy uses the queue time as a guide for in- or decreasing the batch size. If the queue time of recently scheduled tasks is above a certain threshold, the batch size is decreased. It is assumed that task queue time is high due to a decrease in task arrivals, and batches requiring a longer time to be filled. If the queue time is below the threshold, the batch size is increased to take advantage of the benefits of larger batches. The threshold is set to 2s, leading to a slight increase in average queue time, and thus higher average total latency for some distributions (see figure 5.8). The tail latency and failed tasks are improved with adaptive batching, with the exception of rare events. Batching can also drastically decrease the number of task batches that are processed during execution (see figure 5.8). Compared with interval batching, adaptive batching finds a good

trade off between batch size and scheduling performance. This is true for all but bursty task arrivals. A bursty distribution benefits most from interval based batching, since all tasks of an arrival interval can be scheduled in one interval batch of 100ms. Adaptive batching is reactive, meaning that batch size will only gradually increase to match the bursty arrivals.

While a larger batch size will reduce the numbers of task batches scheduled during execution, it can also increase the time needed by the optimizing scheduling algorithm to process a batch (see figure 5.9). The increase depends on the algorithm used, but for efficient implementations it will be in the range of a few milliseconds.

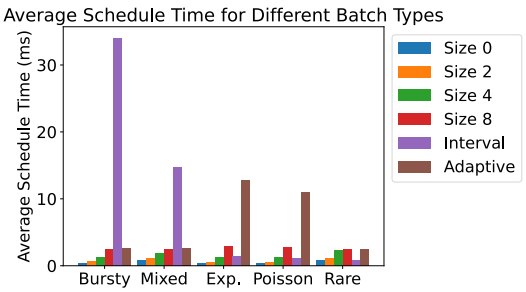


Figure 5.9: Influence of batch policy on scheduling algorithm latencies.

It should be noted that the adaptive batching based on queue time only remains efficient as long as the queue time is an accurate representation of task arrival and processing. Once UAVs start to run out of battery, tasks are executed at a significantly slower rate than before, leading to increased queue times. Decreasing the batch size will no longer help performance since the queue time no longer reflects arrival rate, but the decrease of processing units instead. Switching from adaptive batching to regular interval batching, once UAVs need to recharge, could be an option.

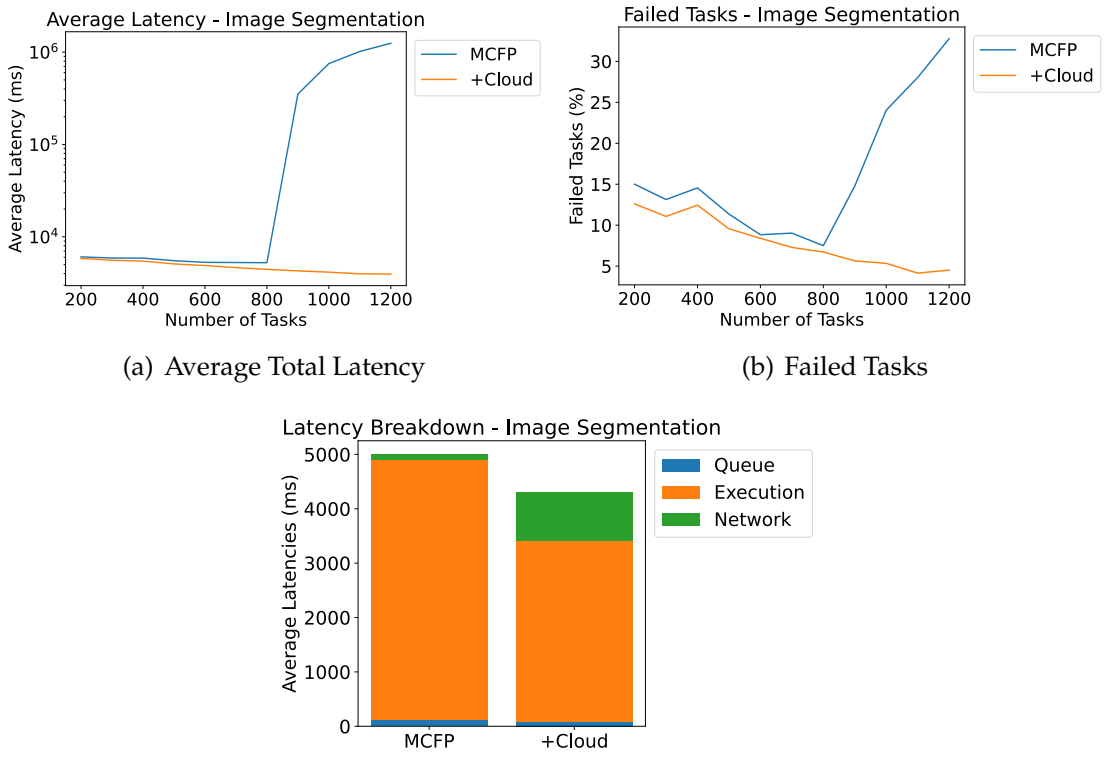
5.2.3 Computation Offloading

In a previous section it was shown that offloading tasks to dedicated compute UAVs can be beneficial for the execution of multi-phase jobs. A common approach to computation offloading is the utilization of cloud resources for computationally intensive tasks. The cloud provides considerably more powerful compute resources than regular UAVs. Transferring sensor data to the cloud for processing can speed up computation while preserving power resources of the UAV swarm. However, the data transfer will incur a higher network delay. Computation offloading may not be ideal for all applications; processing tasks with a short duration could be penalized by the higher network latency. The experiments are run with the two-phase job image segmentation. Image segmentation is computationally expensive, more so than image recognition. This makes the job an ideal candidate for cloud offloading.

Figure 5.10 shows that the utilization of cloud resourced can be beneficial to the swarms performance, especially under a larger workload. Heavy computation will lead to faster depletion of the swarm, resulting in higher execution latencies and failed tasks. Cloud offloading also results in a different latency allocation. Execution latency is reduced while the network delay increases.

5.3 Limitations

Since the focus of the simulator is the evaluation of scheduling algorithms, other aspects of the execution are not addressed or are kept abstract. One example of this is the network setup; it is only an approximation of latencies and does



(a) Average Total Latency

(b) Failed Tasks

(c) Latency Breakdown (800 Tasks)

Figure 5.10: Cloud offloading in addition to UAV task execution, tasks are offloaded to the cloud if they do not require sensing by the UAVs.

not take into account potential interruptions or packet loss. At this point in time the simulator does not take obstacle avoidance, potential collisions, and other UAV flight related issues into consideration as well. Obstacle avoidance could potentially be modeled by considering it a task of a job that needs to be completed before the execution of all other tasks can commence.

The power related configuration parameters are based on estimations by the group that initially implemented the simulator. The estimates are derived from experiments with an actual UAV swarm setup. However, their report mentions that some parameters might require further configuration and verification [9]. Once access to a UAV swarm is possible the parameters should be reconfigured.

Additionally it would be interesting to obtain data for UAV types other than the Parrot AR 2.0 drone.

CHAPTER 6

CONCLUSION

This project concerns the augmentation of a UAV swarm simulator for the evaluation of real-time scheduling policies. The simulator provides a simple interface to configure simulations and specify multi-phase jobs. The system also allows for easy implementation of new scheduling algorithms. The algorithms evaluated in this project are FIFO, a greedy local optimization algorithm, and three global optimization algorithms. To solve the global optimization problem two solvers, namely MCFP and Hungarian, and a meta heuristic, ACO, are employed.

Experiments with different applications and swarm configurations showed that ready-to-use solvers are able to find the best solutions with a low execution time. The greedy algorithm, despite being quite simple, is able to achieve decent performance compared to a basic FIFO policy. The ACO, while in theory a good approach, is not without issues and needs further tuning of parameters and implementation. It does not outperform the greedy algorithm and has a high execution time.

Task batching and the utilization of cloud resources are additional policies that can assist UAV swarm task assignment. An adaptive batching policy based on queue time is introduced and is able to maintain, and in some cases improve, task assignments for optimizing schedulers. The number of batches needing to be processed can be drastically reduced while only minimally increasing the processing overhead for a batch. Cloud resources can be utilized for computationally intensive tasks and lighten the load on the UAV swarm. Future work could investigate a smart cloud scheduling policy that takes into account the

trade off between increased network delay and sped up execution. Additionally the edge server could be used for lighter processing tasks or pre-processing of data before sending it to the cloud.

BIBLIOGRAPHY

[1] Jie Hu and J. Yang. Application of distributed auction to multi-uav task assignment in agriculture. *International Journal Of Precision Agricultural Aviation*, 1(1), 2018.

[2] Guang Zhou, Jinwei Yuan, I-Ling Yen, and Farokh Bastani. Robust real-time uav based power line detection and tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 744–748, 2016.

[3] Chunbo Luo, James Nightingale, Ekhorutomwen Asemota, and Christos Grecos. A uav-cloud system for disaster sensing applications. In *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*, pages 1–5, 2015.

[4] Zhenyu Zhou, Junhao Feng, Bo Gu, Bo Ai, Shahid Mumtaz, Jonathan Rodriguez, and Mohsen Guizani. When mobile crowd sensing meets uav: Energy-efficient task assignment and route planning. *IEEE Transactions on Communications*, 66(11):5526–5538, 2018.

[5] İlker Bekmezci, Ozgur Koray Sahingoz, and Şamil Temel. Flying ad-hoc networks (fanets): A survey. *Ad Hoc Networks*, 11(3):1254–1270, 2013.

[6] Grzegorz Chmaj and Henry Selvaraj. Distributed processing applications for uav/drones: A survey. In Henry Selvaraj, Dawid Zydek, and Grzegorz Chmaj, editors, *Progress in Systems Engineering*, pages 449–454, Cham, 2015. Springer International Publishing.

[7] Anandarup Mukherjee, Sudip Misra, Anumandala Sukrutha, and Narendra Singh Raghuwanshi. Distributed aerial processing for iot-based edge uav swarms in smart farming. *Computer Networks*, 167:107038, 2020.

[8] Fabio D'Urso, Corrado Santoro, and Federico Fausto Santoro. An integrated framework for the realistic simulation of multi-uav applications. *Computers and Electrical Engineering*, 74:196–209, 2019.

[9] Drew Zagieboylo, Mark Zhao, Joo Yeon Chae, and Francois Mertil. Drone swarm simulator, 2018.

[10] Juan Zhang, Yulei Wu, Geyong Min, Fei Hao, and Laizhong Cui. Balancing energy consumption and reputation gain of uav scheduling in edge computing. *IEEE Transactions on Cognitive Communications and Networking*, 6(4):1204–1217, 2020.

[11] M. De Benedetti, F. D'Urso, G. Fortino, F. Messina, G. Pappalardo, and C. Santoro. A fault-tolerant self-organizing flocking approach for uav aerial survey. *Journal of Network and Computer Applications*, 96:14–30, 2017.

[12] M. Alighanbari and J.P. How. Decentralized task assignment for unmanned aerial vehicles. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 5668–5673, 2005.

[13] Luca Bertuccelli, Han-Lim Choi, Peter Cho, and Jonathan How. Real-time multi-uav task assignment in dynamic and uncertain environments. In *AIAA Guidance, Navigation, and Control Conference*, pages 10–13, Chicago, Illinois, 09 2009. American Institute of Aeronautics and Astronautics.

[14] Yoav Gottlieb and Tal Shima. Uavs task and motion planning in the presence of obstacles and prioritized targets. *Sensors (Basel, Switzerland)*, 15(11):29734–29764, Nov 2015.

[15] Yohanes Khosiawan, Youngsoo Park, Ilkyeong Moon, Janardhanan Mukund Nilakantan, and Izabela Nielsen. Task scheduling system for uav operations in indoor environment. *Neural Computing and Applications*, 31(9):5431–5459, Sep 2019.

[16] Xiaowei Jiang, Qiang Zhou, and Ying Ye. Method of task assignment for uav based on particle swarm optimization in logistics. In *Proceedings of the 2017 International Conference on Intelligent Systems, Metaheuristics and Swarm Intelligence*, ISMSI '17, page 113–117, New York, NY, USA, 2017. Association for Computing Machinery.

[17] Yi Wei, M. Brian Blake, and Gregory R. Madey. An operation-time simulation framework for uav swarm configuration and mission planning. *Procedia Computer Science*, 18:1949–1958, 2013. 2013 International Conference on Computational Science.

[18] Zhangjie Fu, Yuanhang Mao, Daojing He, Jingnan Yu, and Guowu Xie. Secure multi-uav collaborative task allocation. *IEEE Access*, 7:35579–35587, 2019.

[19] Byung Duk Song, Jonghoe Kim, Jeongwoon Kim, Hyorin Park, James R. Morrison, and David Hyunchul Shim. Persistent uav service: An improved scheduling formulation and prototypes of system components. In *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 915–925, 2013.

[20] Hakim Ghazzai, Hamid Menouar, Abdullah Kadri, and Yehia Massoud. Future uav-based its: A comprehensive scheduling framework. *IEEE Access*, PP:1–1, 06 2019.

[21] Zhenyue Jia, Jianqiao Yu, Xiaolin Ai, Xuan Xu, and Di Yang. Cooperative multiple task assignment problem with stochastic velocities and time windows for heterogeneous unmanned aerial vehicles using a genetic algorithm. *Aerospace Science and Technology*, 76:112–125, 2018.

[22] Martin Rosalie, Grégoire Danoy, Serge Chaumette, and Pascal Bouvry. Chaos-enhanced mobility models for multilevel swarms of uavs. *Swarm and Evolutionary Computation*, 41:36–48, 2018.

[23] Xiaoxuan Hu, Huawei Ma, Qingsong Ye, and He Luo. Hierarchical method of task assignment for multiple cooperating uav teams. *Journal of Systems Engineering and Electronics*, 26(5):1000–1009, 2015.

[24] Husheng Wu, Hao Li, Renbin Xiao, and Jie Liu. Modeling and simulation of dynamic ant colony's labor division for task allocation of uav swarm. *Physica A: Statistical Mechanics and its Applications*, 491:127–141, 2018.

[25] Xiaowei Fu, Peng Feng, and Xiaoguang Gao. Swarm uavs task and resource dynamic assignment algorithm based on task sequence mechanism. *IEEE Access*, 7:41090–41100, 2019.

[26] Naser Hossein Motlagh, Miloud Bagaa, and Tarik Taleb. Energy and delay aware task assignment mechanism for uav-based iot platform. *IEEE Internet of Things Journal*, 6(4):6523–6536, 2019.

[27] Baichuan Liu, Huawei Huang, Song Guo, Wuhui Chen, and Zibin Zheng. Joint computation offloading and routing optimization for uav-edge-cloud computing environments. In *2018 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 1745–1752, 2018.

[28] Baichuan Liu, Weikun Zhang, Wuhui Chen, Huawei Huang, and Song Guo. Online computation offloading and traffic routing for uav swarms in edge-cloud computing. *IEEE Transactions on Vehicular Technology*, 69(8):8777–8791, 2020.

[29] Wuhui Chen, Baichuan Liu, Huawei Huang, Song Guo, and Zibin Zheng. When uav swarm meets edge-cloud computing: The qos perspective. *IEEE Network*, 33(2):36–43, 2019.

[30] David F. Crouse. On implementing 2d rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679–1696, 2016.

[31] Chunhui Piao, Xufang Han, and Yalan Wu. Improved ant colony algorithm for solving assignment problem. In *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, volume 15, pages V15–476–V15–480, 2010.

[32] Blair Chen. Azure speed test. `https://www.azurespeed.com/`. Online, Accessed: 2021-05-28.