A Survey of Scheduling Algorithms for the Edge

Tim van Kemenade
Vrije Universiteit Amsterdam
t.a.w.van.kemenade@student.vu.nl

Matthijs Jansen
Vrije Universiteit Amsterdam
m.s.jansen@vu.nl

Alexandru Iosup
Vrije Universiteit Amsterdam
a.iosup@vu.nl

Abstract

The increase in connected IoT devices and the ever-increasing demand for real-time processing, like Industrial IoT (IIoT), and vehicular edge, have increased the need for computing resources closer to the devices in the form of the edge. Positioned as a layer between the cloud and endpoints, the edge aims to bring computational, storage, and network resources closer to user devices, thereby reducing latency and doing this in an energy efficient way. Distributing these resources is done via a scheduler to efficiently allocate resources to tasks depending on the set out requirements. This systematic literature survey will focus on scheduling in the edge. The importance of the edge becomes more visible with mobile devices in densely packed and offloading on IoT devices that impose restrictions such as computing, mobility, delay, and power constraints. Through a systematic literature survey, we present a comprehensive taxonomy of the design space, dissecting existing paradigms, algorithmic approaches, offloading types, and optimization goals.

1 Introduction

The cloud provides a large number of resources and high bandwidth to execute large data-intensive tasks [25] but the emergence of IoT devices [71] and mobile devices like smartphones and cars [57] that require real-time processing brought forth a new concept, the edge. The edge is a new layer with medium resources and bandwidth, but physically closer to the endpoints. The endpoints are even more resource-constrained and have the previously mentioned real-time requirements. This edge, while previously considered a data consumer, is increasingly becoming a data producer as well [71]. The concept of the edge has been introduced as a middle layer between the endpoints (user devices) and the cloud [35]. This edge should move compute resources closer to the endpoints to be in closer physical proximity to reduce network hops and end up reducing end-to-end latency [68]. These 3 layers, the cloud, edge, and endpoints, comprise the computing continuum [35, 36] as seen in Figure 1. This computing continuum has been explored in several scenarios, such scenarios include Multi-access Edge Computing (MEC) [77] for mobile devices switching between different edge servers due to a changing location, and offloading on IoT devices, which focuses on offloading to nodes with higher computing capabilities [3].

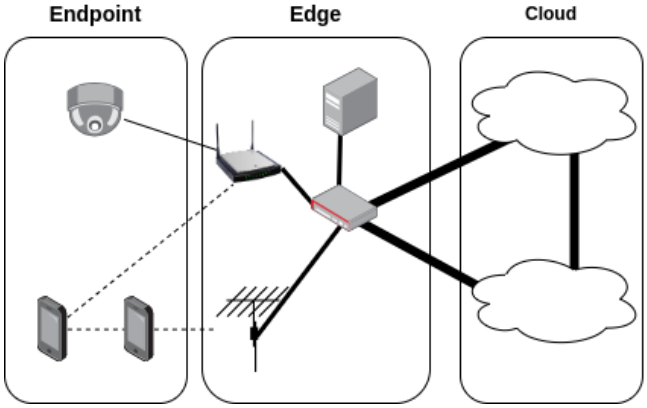

Figure 1: An overview of the compute continuum, showing that endpoints are owned by users, and the edge and cloud are owned by service providers inspired by [35]

Managing the edge and associated resources is a difficult task and part of any resource management system. The edge scheduler is central to this and is responsible for placing tasks on the edge, conforming to the different requirements imposed by the tasks. The IoT devices highlight some problems any edge scheduler needs to solve: the mobility, computational resources, and power restrictions of the endpoint devices. The mobility of devices is especially important in dense edge networks where several edge nodes have an overlapping reach and the moving endpoints can fall outside of the range of the edge node while their tasks are being processed. Power restrictions on battery/solar-powered devices also impose further restrictions which can occur for sensor networks [10]. And the high bandwidth real-time applications are becom-

ing increasingly prominent. Optimizing for these constraints could result in reduced latency, improved scalability, more efficient resource utilization, and better pricing schemes.

Many schedulers have been proposed using methods such as reinforcement learning, particle swarm optimizations, and heuristic solutions. Previous surveys have mapped the different scheduler classes into a taxonomy, focusing on heuristic and metaheuristic schedulers [8], stochastic Markov-based schedulers [70], listing papers on approaches to different components of a larger resource management layer [22,26], listing the algorithmic approach of reviewed papers [30], optimization strategies [67]. These differences are highlighted in Section 2. However, the aforementioned taxonomies are talking about a subset of scheduler classes. They also don't show the influence of certain infrastructure design decisions on the design of the algorithm. *So what scheduling classes do exist? And what infrastructure are they designed for?*

Edge schedulers are designed with goals like delay, power usage, and cost minimization [8]. These goals can be quantified so that proper comparisons can be made. Different schedulers can then be discussed in terms of their capabilities, some of which the edge schedulers excel more at than others. The differences in capabilities highlight certain trade-offs. One often talked about trade-off is the execution delay versus the power usage [55, 57].

This paper contains the following 4 contributions:

1. We design the process of conducting an in-depth systematic literature review in Section 3. This is done by giving an overview of the steps and explaining the steps in more detail.

2. We present a taxonomy of the components to build an edge scheduler in Section 4. This taxonomy is broken down into other taxonomies, focusing on existing system models in Section 5, algorithmic approaches in Section 6, offloading types in Section 7, and optimization goals in Section 8.

3. We perform an in-depth analysis of scheduling classes, like heuristic or machine learning-based approaches, in Section 6. We will target all found approaches in the accepted papers, grouping them by major scheduling classes and for the metaheuristics also depending on the type of metaheuristic to keep an overview of the different approaches.

4. We group papers according to their delay, energy, cost, and QoS optimization goals in Section 8.

2 Related work

Current surveys differ from focusing on the algorithms driving edge scheduling to the infrastructure and platform specifics involved in edge scheduling. The scheduler is a crucial part of the complete resource management which for the edge attempts to achieve optimal computational offloading in terms of execution delay or energy efficiency (or both). These surveys discuss various parts of edge computing which have overlap. The contributions of these surveys usually fall within 2 groups. They either present a taxonomy or a classification of the domain space. However while these overviews (taxonomies and classifications) can be useful on their own, they are often hard to relate to each other. One often cited survey on classifications is Mach et al. [55], which classifies applications according to how much of the application can be offloaded to the edge, knowledge of the amount of data to be processed, and the dependencies between offloadable parts. However, the paper comparison doesn't classify offloadability and instead focuses on energy efficiency and execution delay. Another survey [57] groups the task model, the offload approach, together with papers focusing on the aspects of energy efficiency and execution delay. This trade-off between the execution delay and energy efficiency is often mentioned in surveys with an emphasis on minimizing measured execution delay and power usage [55], highlighting individual papers focusing on one or both as minimization goals [65], or putting emphasis on measurability and benefits [70]. Most taxonomies contain some categorization of algorithms. The categorization of algorithms differs wildly as some focus on the algorithm strategies [8, 30, 32, 44], while others focus on algorithms that are different parts of resource management in the edge [22, 26]. Such taxonomies are often incomplete and typically focus on some classes of schedulers. One taxonomy on resource management focuses on different aspects of resource management like application placement, resource scheduling, offloading, and more [22] which only partially aligns with [26], although measurability was not mentioned in [22]. Ghobaei's [22] taxonomy is extensive but doesn't cover what decisions benefit which algorithms. These 4 groups (taxonomies, classifications, trade-offs, and algorithms) are shown in Table 1 highlighting additional surveys that focus on a subsection of the literature while it is also interesting how the surveys relate to each other. The topics as shown in Table 1 are based on what we found when reviewing the survey. Topics like migration were excluded as we focused on algorithms that rarely discuss migrations.

3 Research Methodology

Research on scheduling in the edge can be found in a wide range of literature with multiple different keywords being used for the same concept. It is important to first define how we find, select, and analyze the literature in Section 3.1. We then discuss the steps, elaborate on how this paper interprets the steps, and on further restrictions imposed on the method.

Survey	Paradigm					Optimization					Offloading				Algorithm					
	MEC	Fog	Cloudlet	Cloud	IoT	Delay	Response time	Energy	Cost	Warm-up	Single	Multiple	Full	Partial	Heuristic	Meta-heuristic	Hyper heuristic	Hybrid	ML	Integer
This survey	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Mach et al. [55]	✓	✓				✓		✓		✓	✓		✓	✓						
Shakarami et al. [70]	✓	✓				✓	✓	✓	✓		✓		✓	✓	✓	✓				
Ghobaei-Arani et al. [22]		✓				✓		✓	✓	✓	✓	✓			✓	✓		✓		
Mao et al. [57]	✓	✓				✓	✓	✓	✓		✓	✓	✓	✓						
Ren et al. [65]	✓	✓	✓			✓		✓			✓		✓	✓						
Jamil et al. [32]		✓			✓	✓	✓	✓	✓						✓	✓	✓	✓	✓	
Belmahdi et al. [8]		✓			✓		✓	✓	✓						✓	✓		✓		
Kumar et al. [44]				✓		✓	✓	✓	✓						✓	✓		✓		
Islam et al. [30]		✓				✓	✓	✓	✓											
Hong et al. [26]		✓									✓	✓	✓	✓	✓					

Table 1: Comparison between this survey and existing surveys

3.1 Selected method

We compare 3 literature aggregation methods for a systematic review that were found using Google Scholar using the query "systematic literature review methodology". We extracted the latest revision of all methods with more than 1000 citations which resulted in 4 methods, [43, 62, 83, 84].

The paper on the snowballing method [83] is excluded from the comparison as it focuses on the selection of literature based on an initial set. It can be used as an extension of the available search methods, but it is far from a complete method of conducting a systematic literature study as it doesn't elaborate on the process of finding proper seed papers.

All the methods use an iterative style. Kitchenham [43] iterates back to the planning stage to revise the search depending on quality criteria that are set at a later stage. Okoli [62] defines the iteration at a different stage as it focuses on developing the protocol, training the reviewers in an iterative manner, and synthesizing data in an iterative manner to ensure completeness. Xiao [84] performs an iterative process to define the problem definition, this iterative process is based on ensuring the quality of work ensuring sufficient work is reviewed and the workload is feasible.

Kitchenham [43] and Xiao [84] use 3 stages, planning, conducting, and reporting. Okoli [62] introduces 4 stages, planning, selecting, extraction, and execution. Selecting and extraction are both part of the conducting stage while execution involves both the last step synthesizing of the conducting stage and the reporting stage itself.

This survey will use the method from Xiao [84]. It was chosen for its flexibility in the search criteria. Xiao [84] adjusts the search criteria based on quality and feasibility rather than just on quality criteria as is done by Kitchenham [43].

	Planning	
1	Formulate the problem	S1 & 2
2	Develop and validate the review protocol	S3.1
	Conducting	
3	Search the literature	S3.2
4	Search for inclusion	S3.3
5	Assess quality	S3.4
6	Extract data	S3.5
7	Analyze and synthesize data	S3.5
	Reporting	
8	Report findings	S4-S10

Table 2: Overview of where the steps in the process of a systematic literature survey of Xiao [84] are executed

Okoli [62] wasn't chosen as the approach is more linear, not revising the selection based on results later on in the quality assessment. Table 2 shows where every step is executed.

3.2 Search the literature

Xiao [84] mentions 3 channels for literature search: electronic databases, backward searching, and forward searching. We will focus on electronic databases as the queries already provide hard search criteria that exclude additional papers from being selected. The selected electronic database to execute the query on is dblp [1]. Semantic Scholar [42] is used for further filtering and processing of the papers found using dblp, but isn't included in the exploration of papers as its search relevance algorithm isn't exact matching like dblp does (dblp uses prefix matching). The queries can be defined over the contents of the title. The queries will only consider English publications. An initial query will be performed, later subqueries will

be a subset of this initial query. The queries are executed on the public dblp search API for publication queries [1], which uses no API key.

The survey is structured as follows. An initial title search is done to include all papers that are potentially related to this survey. This initial search leaves us with the base query in Listing 1. Later sections will focus on a subset of the base query which could be more restrictions on the title. The base query in Listing 1 is based on prefix-matching and searches for one of 'sched', 'resource manag', or 'offload' in the title to restrict the search to scheduler-related papers and one of 'MEC', 'fog, 'edge', or 'cloudlet' in the title to find papers within the compute continuum.

There are 795 papers that were retrieved using the initial query in Listing 1 using dblp. The dblp query was executed on 2023-08-20 and the results can be reproduced using the dblp dataset 2023-08-01 [2]. We are left with 101 papers after all the subqueries have been executed, the algo query in Listing 2. This is reduced to 63 papers that were accepted. Papers were rejected for several reasons as seen in Table 3. The rejection reasons are discussed below.

- No access: The paper isn't open access and institutional access from VU Amsterdam or University of Amsterdam doesn't work.
- Failed inclusion: The paper didn't provide sufficient information in the abstract and introduction to meet the criteria laid out in Section 3.3.
- Failed quality assessment: The full paper didn't meet the criteria as laid out in Section 3.4.
- Unfinished: The paper was marked as unfinished by the publisher, which means that the paper is either still being revised or a case for the paper was published.
- Survey: The paper is a survey. Surveys will be discussed in the related work in Section 2 and won't be included for analysis and synthesis.
- Thesis: Theses aren't as they haven't been accepted for publishment.
- Retracted: The paper was retracted by the publisher. The reason for the retraction won't be discussed in this review.

```
SELECT ALL papers FROM dblp
CASE INSENSITIVE MATCH
(sched|resource manag|offload) in title
AND (MEC|fog|edge|cloudlet) in title
```

Listing 1: Initial dblp query

Reason	No access	Failed inclusion	Insufficient quality	Unfinished	Survey	Thesis	Retracted
Algo Listing 2	11	16	1	1	1	1	1

Table 3: Overview of the amount of papers rejected by reason for all subqueries

```
SELECT ALL papers FROM initial query
CASE INSENSITIVE MATCH
(algo) in title
```

Listing 2: Algorithm subquery

3.3 Search for inclusion

Next, the search for inclusion will review the abstract and introduction of all the selected papers. This is done over the chosen subset in Section 3.2. A paper will be rejected if the main findings aren't in scope after reading the abstract and introduction. Whether the findings are within the scope of the query will be defined below with the subqueries goals in mind.

- Algo Listing 2: It is important that it is immediately clear what type of algorithm is proposed in the paper. It is also handy if an analysis is performed against well-established algorithms for the task at hand which should also be clear from reading the abstract and introduction. The problem statement and setup should be mentioned in a generalized form and can later be expanded upon in the paper. So the requirements for the abstract and introduction are as follows:
  - Mentioned the used algorithm for scheduling.
  - Mentioned the edge model and assumptions made about the model.

3.4 Assess quality

The last step in selecting papers is assessing the quality of the papers that have made it this far. This analysis is performed over all the text, excluding appendices. The quality assessment will be based on the contributions, and the statements made. This step targets potential ambiguity that becomes clear later in the paper and ensures that the focus stays on the edge. It excludes papers that happen to not discuss the edge or algorithms upon further inspection. The criteria were based on the Algo Listing 2 as this query focuses specifically on algorithms for schedulers in the edge.

3.5 Analysis

All selected papers will be analyzed by their statements, tables, graphs, and figures. Findings will be synthesized into this systematic literature review. The analysis is elaborately broad and discusses a wide range of different edge models, used algorithms, offloading types, and optimization goals.

The analysis is synthesized into a single taxonomy in Section4. This is split into several subdomains. What is considered the edge model and its constituents is discussed in Section 5. Then different algorithms used for scheduling tasks from endpoint to edge to cloud is analyzed in 6. The algorithm analysis was done for different paradigms, with different definitions of offloading and goals for optimization in mind. The type of offloading will be discussed first in Section 7, grouping the analyzed offloading models under of the 4 categories. The analysis of different edge models, used algorithms, and definition of offloading allows us to further analyze the optimization goals under which the schedulers were designed, these optimization goals are analyzed in Section 8.

The analysis and synthesis of the papers will be closed off with the conclusion in Section 10. The conclusion will focus on lessons learned and opportunities for edge scheduling in the compute continuum.

4 Building a scheduler for the edge

Building a scheduler for the edge is made out of a few components. The scheduler involves a System model as seen in Section 5, the algorithm as seen in Section 6 is tailored to requirements imposed by the problem model, and the problem model can be split into how tasks are offloaded (or placed in the edge) as seen in Section 7 and what the optimization goal is as seen in Section 8. The system model aims to discuss the layout of the edge, the heterogeneity of nodes, and how communication lanes are set up. The problem model then discusses what type of users the system has, the makeup of tasks they send in, additional requirements on these tasks, and what the scheduler tries to achieve. This problem is solved with the algorithm tailored to the problem which is discussed before the problem model to get a better understanding of the scope of edge schedulers, but in papers is usually done after the system model and problem model or together with the problem model. The analysis results in a taxonomy for edge scheduling in the compute continuum as seen in Figure 2. This taxonomy is scoped around the primary components of defining an edge scheduler and is broken down into smaller taxonomies that display more in-depth what the section encompasses.

5 System models

As seen in Figure 1 which was inspired by [35], we divide the compute continuum into 3 parts. The cloud, the edge, and the endpoints. The cloud won't be discussed, although cloudlets have partial overlap with the cloud as they are smaller clouds in the edge layer. The edge can mean a lot of things, depending on the setup. Endpoints are the users of the edge and are made out of a wide range of devices with different requirements.

The cloud is connected to endpoints via the edge layer to which endpoints connect. It is seen as one singular cloud and provides high resources and high bandwidth, but has a higher latency penalty. Some papers utilize the cloud to offload computationally intensive tasks [40, 48, 58], but this is often not discussed in detail.

The edge is often named the fog or Multi-access Edge Computing (MEC, previously named Mobile Edge Computing). MEC is used for densely packed networks where the users are mobile and the coverage area of the edge overlaps as seen in Figure 5. Connection to the edge often occurs via base stations, which are the wireless towers endpoints connect to. These base stations then either have a local edge server [12, 48] or connect to slightly more distant edge servers [94]. The edge can often be referred to using 3 scenarios as seen in Figure 4.:

1. Edge node directly connected to the cloud [12, 18, 40, 48, 58, 94].

2. Edge cluster that can potentially be connected to the cloud [4, 16, 29, 33, 37, 45–47, 56, 66, 69, 73, 74, 76, 87, 88, 90, 93].

3. Multiple edge clusters are all connected separately to the cloud [10, 23, 38, 54].

The endpoints consist of user-owned devices that are sensitive to latency. This includes IoT [4, 10, 23, 29, 33, 37, 39, 66, 73–76, 90], mobile devices [18, 23, 38, 40, 45–48, 54, 59, 93, 94], vehicles [16, 58], and UAVs [12]. The chosen endpoint affects the QoS requirements the scheduler needs to follow. Mobile devices are more likely to reside in a densely packed edge network where they often switch to another edge server. IoT has a lot of consistent requirements but needs to process large quantities of data. Vehicles move around quickly and could even move out of reach of the base stations. UAVs could suffer from uneven distribution over edge servers due to their location.

Another important aspect is where the scheduling decision is made. This scheduling decision is often made in the edge [48], but can also be made by the mobile device [45, 47], or by the distant cloud [46, 58, 66, 69]. Another entity is sometimes introduced to do the decision-making in the broker [29, 37], which is an observing entity that keeps track of the state of nodes in the edge. Where the decision is made decides whether the network is centralized or decentralized, where centralized happens in case of the broker and cloud, and decentralized when the edge and mobile devices need to make a decision based on the limited available information.

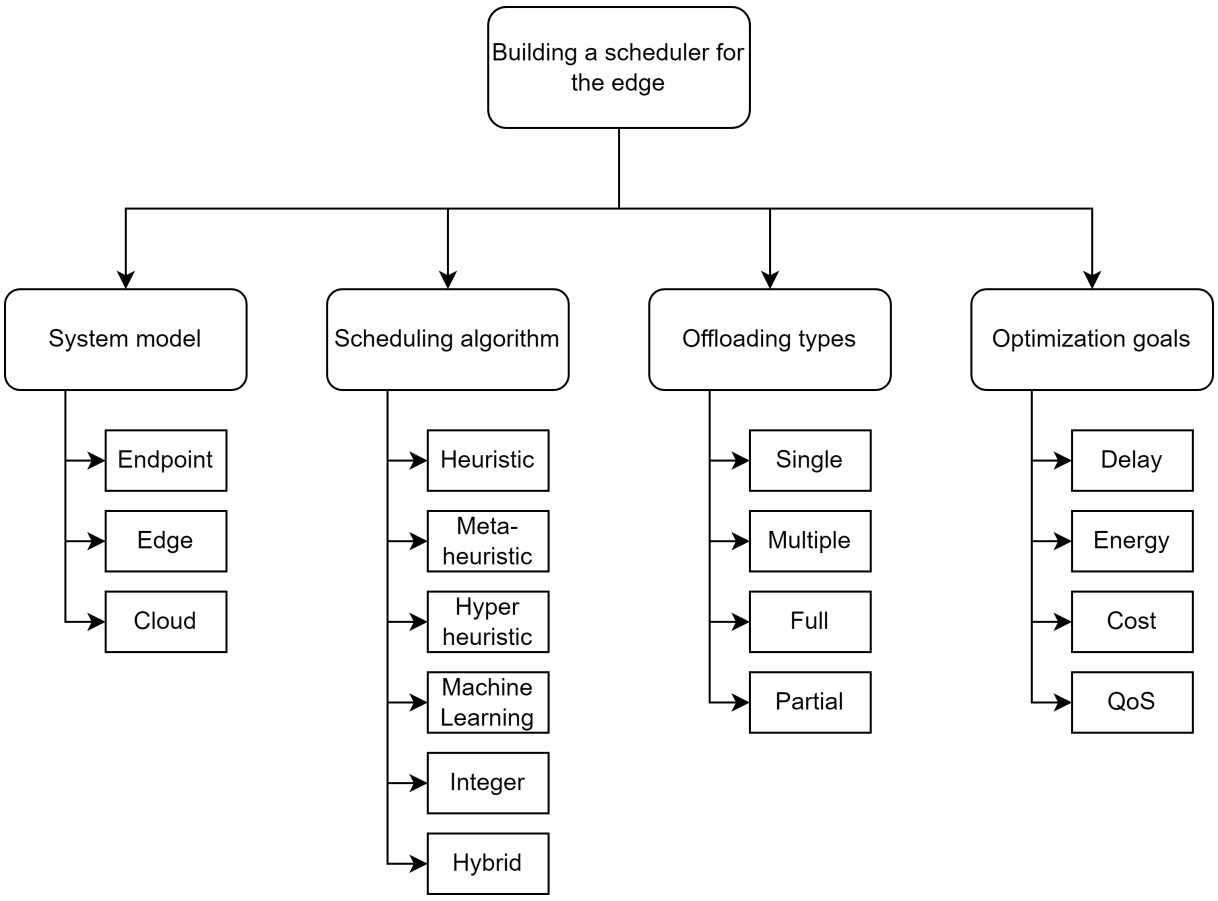

Figure 2: A taxonomy of the scheduling design space

6 Algorithmic approaches

The algorithmic approaches can be categorized into 6 major categories as seen in Figure 6. By far the most scheduling algorithms are based on metaheuristics as discussed in Section 6.2. This is followed by machine learning in Section 6.4 with an even split between reinforcement learning in Section 6.4.1 and deep learning in Section 6.4.2. The last major group is heuristics in Section 6.1. Both integer programming in Section 6.5 and hyperheuristics in Section 6.3 only have one algorithm instance in the reviewed set of papers. Concluding our overview, we discuss hybrid algorithms in Section 6.6, which are algorithms composed of 2 algorithms that work together in stages.

This section aims to give a rundown of the potential approaches. How the models on which the algorithms were built differ is explained in Section 5 and Section 7. The goal(s) an algorithm optimizes for is further discussed in Section 8.

6.1 Heuristic

A heuristic is an approximation of an optimal solution based on the available information. It is a static strategy, unlike metaheuristics which move towards an optimum.

The heuristics can be divided based on the data and theory used in the algorithm. One notable approach is the use of game theory [13, 85, 91]. A two-stage offloading approach [85] can be used for energy minimization by modeling the game as a decision to offload game participants with an associated cost. The offloading decision is balanced by adjusting the energy consumption ratio to reach an equilibrium. MUCRS [91] is the multi-user computing resource scheduling algorithm based on the big.LITTLE processor. In MUCRS the game is non-cooperative so each user tries to minimize their own energy usage. However energy efficiency will be low if all users were to use the same approach, so there is an equilibrium of how many users can offload and how many cannot based on the time and energy consumption requirements. Another non-cooperative game is presented which is solved by a dynamic (QCOG-DG) and static (QCOG-SG) algorithm [13]. In QCOG each player can choose a computation offloading policy combination that uses the same principal of cost as MUCRS to reach an equilibrium. What makes QCOG [13] interesting is that it simplifies a multiple node full offloading problem using game theory.

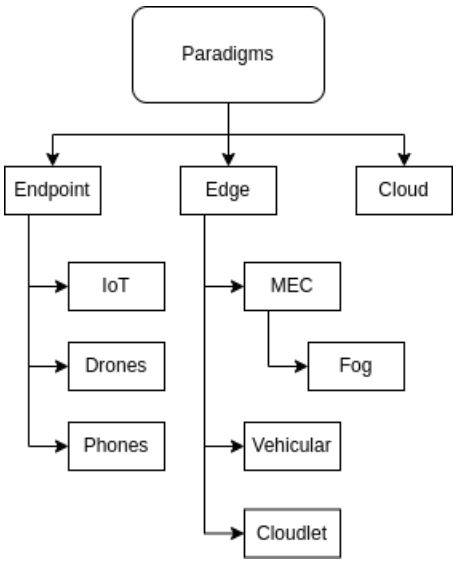

Figure 3: A taxonomy of the different paradigms in the design space

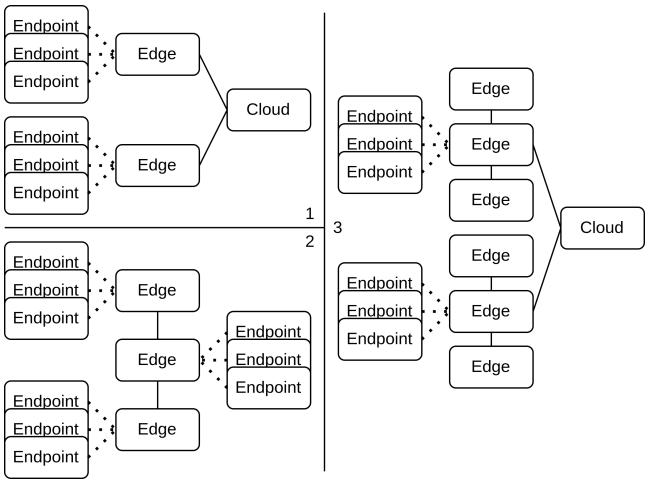

Figure 4: 3 potential system model edge setups, endpoints to edge to cloud, endpoints to edge cluster to cloud, and endpoints to edges to edges

Heterogeneous Earliest Finish Time (HEFT) is another strategy that has been used on multiple occasions [45,87]. One approach calculates a min-cut to minimize the overall cost and then use HEFT to schedule the unassigned tasks after the application partitioning [45]. Another approach uses HEFT as an initial solution for the population of a fireworks algorithm [87]. However, HEFT doesn't consider the non-crossover technique [56]. Crossover is the additional cost of transfering a task between processors. A Cross-Threshold value can be used together with EFT to overcome this crossover penalty [56]. Two phases. In the first phase, the tasks are ordered based on the heuristic method, heterogeneous earliest finish time (HEFT) and the ordered tasks are scheduled by applying the improved gaining sharing knowledge (IGSK) based algorithm [61].

Deadline and Energy-Efficient Task Scheduling (DEETS) [46] is aimed at reducing energy consumption while meeting the given deadlines. It does so via a priority queue. The scheduling is done by earliest deadline first, smallest slack time first, and smallest workload first, in that order. The slack time is defined as the deadline minus the time to execute the task.

Earliest Deadline First (EDF) has been proposed for autonomous driving [16]. EDF is shown to benefit urgent task while also being able to schedule more task compared to Best Fit Replacement Scheduling (BFRS).

Multi-queue priority-based algorithm uses Shortest Job First (SJF) with additional turn based queue for long task to prevent starvation [20].

Novel multi-objective Fog scheduler for latency-sensitive applications that focuses on service response time [33].

Three heuristic algorithms: minimum distance, minimum load, and minimum hop distance and load (MHDL) [73].

QoS-aware downlink-scheduling algorithm, QuAS [80], uses Proportional Fair (PF) algorithm to assign resources based on the possible throughput, and the average current resources for the users.

The last heuristic is Edge Cover Scheduling Algorithm (ECSA) with greedy search [14]. It uses the edge cover queue of a DAG. It uses a greedy task allocation method.

6.2 Metaheuristic

A metaheuristic can be seen as an optimized heuristic. It can continuously run small variations of a heuristic to move towards an acceptable solution. There are 3 main categories as seen in Figure 7, Genetic Algorithms, Hill climbing, and Swarm Intelligence.

Many schedulers are Genetic Algorithms (GA) [9, 10, 19, 24, 29, 40, 48, 49, 58]. Dynamic Multi-Objective Evolutionary Optimization based on Decomposition (DMOEA/D) [9] is a GA that uses relocation of promising individuals and a memory strategy to respond to new MEC environments. MOEA/D-TLS [49] uses three-stage local search to enhance its local search ability. An Evolutionary GA (EGA) with adaptive fitness [58] was proposed for vehicular ad-hoc networks. The adaptive fitness of the EGA ensures that the Service Level Agreement (SLA) is never violated [58]. One GA discusses how to deal with dependant task and the additional energy and latency associated with it [10], even going as far as to allow dependencies on multiple edge servers. A GA can also be enhanced using a hybrid approach with another metaheuristic as done using a Flamingo Search Optimization (FSO) [29]. The best candidate is chosen each round and a crossover is performed with the solution of the FSO over the remaining candidates [29]. Learning Interactive Genetic Algorithm

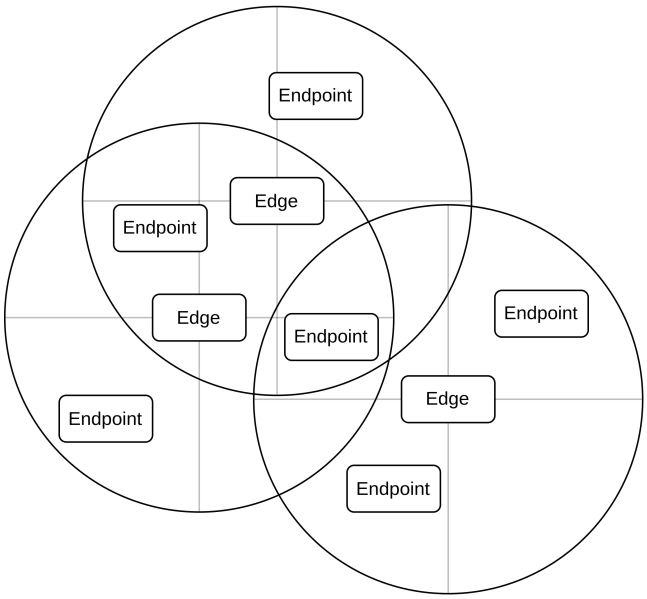

Figure 5: Potential overlap in densely distributed edge network

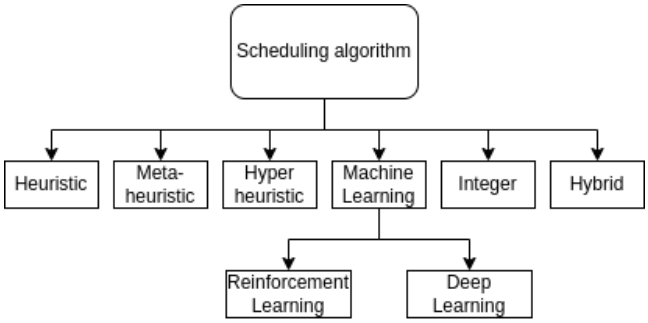

Figure 6: A taxonomy of the algorithms used for scheduling

based on Edge Selection Encoding (LIGA-ESE) [24], uses 2 competing populations. The main contribution of LIGA-ESE is the convergence efficiency from ESE. Another paper focuses on Multi-User-to-Multi-Servers (MUMS) and solves it by Genetic Algorithm Based Distributed Offloading Strategy (GABDOS) [48]. The selection phase groups users to a base station with the highest preference which considers the physical distance and workloads [48]. Grefenstette bias based Genetic Algorithm for MultiSite Offloading (GGA-MSO) [19] uses Grefenstette bias to achieve faster convergence and optimal partitioning. Non-dominated Sorting Genetic Algorithm (NSGA-II) [40] uses a crowding distance to estimate the density of solutions. Hybrid ACO and GA [66]. Improved elitism genetic algorithm (IEGA) [4], by dynamically altering the mutations and crossover to help exploration and prevent local minima.

Hill climbing is a method by which a local optimum can be found by exploring the solution space with 1 particle. 17%

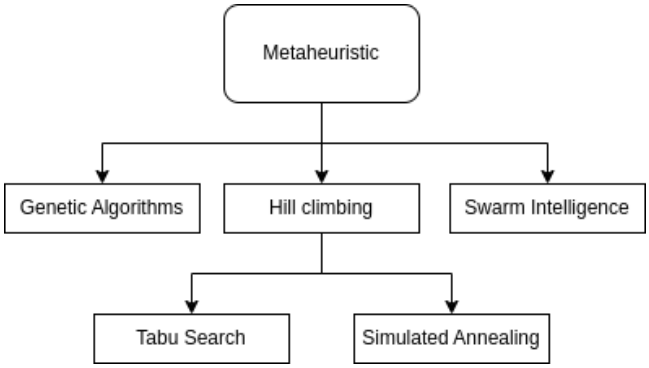

Figure 7: An overview of the reviewed metaheuristic types

of solutions do not support a one-climb policy [17], thus motivating the need for Tabu Search (TA) [59] and Simulated Annealing (SA) [5]. TS is combined with a Fruitfly Optimization Algorithm after the TS has finished to further improve the solution [59].

The remaining approaches can be grouped under the swarm intelligence optimizations [6, 7, 15, 21, 27, 29, 34, 37, 40, 41, 51, 53, 54, 69, 72, 74, 75, 87, 88, 90, 93]. The different optimizations can be seen in Figure 8. A Flamingo Search Optimization (FSO) can be used to enhance a GA [29]. The crossover was enhanced by selecting the second best candidate using FSO. An Ant Colony Optimization (ACO) for real-time offloading (RTACO) [15] to achieve lower latency, minimize makespan and optimize system load. Hybrid ACO and GA [66]. Resource scheduling in mobile edge computing using improved ant colony algorithm for space information network [82]. Improved Krill Herd and Earliest Finish Time (IKH-EFT) [41], it uses movement toward food, toward krill population, and stochastic movement for the discovery of new solutions. Multi-objective Artificial Algae (MAA) [72], it preprocesses tasks into separate task-lists based on the number of offspring so that the algorithm can focus on the dependencies. An improved Firefly Algorithm (FA) outperforms many metaheuristics, like EHOI, EHO, SCA, GOA, WOA, BBO, MFO, and PSO. Opposition-Based Differential Fireworks Algorithm (OBDFA) [88] for scheduling in a heterogeneous edge. Fractional mayfly optimization, combination of GA and firefly algo [52]. The main contributions of OBDFA are the use of Opposition-Based Learning (OBL) and Differential Evolution (DE). The FA can also use HEFT to create an initial population [87]. Energy-Aware Double-fitness Particle Swarm Optimization (EA-DFPSO) [53], uses a dual fitness function and optimized inertia weight. EA-DFPSO aims to reduce task completion time and energy consumption. The first PSO in edge literature is in the context of IaaS [69]. The PSO focuses on the heterogeneous nature of cloudlets and aims at reducing execution time and cost. Slow-movement PSO [93] uses slow-movement particle updates to make solution exploration more efficient. Industrial IoT

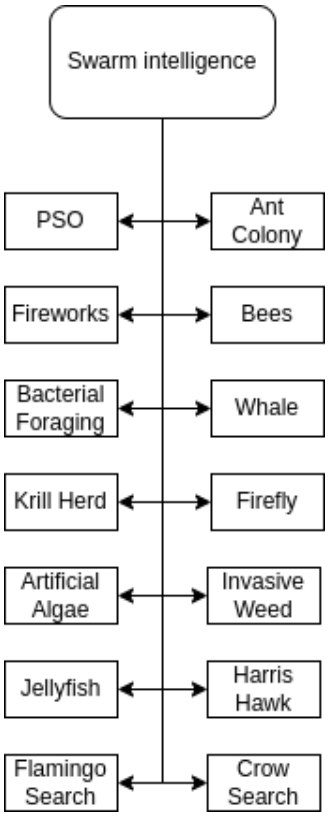

Figure 8: An overview of all reviewed swarm intelligence optimizations

using discrete PSO [90] to obtain an task offloading solution with the smallest total energy consumption. Queue-based Improved Multi-Objective PSO (IMOPSOQ) to target the OMDT-TC problem [54]. Hybrid genetic algorithm and particle swarm optimization (GA-PSO) [81]. FPFTS: A joint fuzzy particle swarm optimization mobility-aware approach to fog task scheduling algorithm for Internet of Things devices [38]. The Bees Algorithm (BA) uses local and random global search to get to a good solution [40]. Hybrid Artificial Bee Colony Optimization (HAWO) is optimized by integrating the Whales Optimization Algorithm (WAO) to overcome the search process [7]. Grey Wolf Optimizer Whale Optimization Algorithm (GWO-WOA) [21], GWO is used to find optima while the WOA prevents converging on local optima. A WOA was proposed that preserves privacy by aiming at the problem of location privacy leakage caused by offloading [51]. Hybrid Bacteria Foraging Optimisation (HBFA), aiming at minimizing completion time and maximizing resource utilization in the edge [74]. A Hybrid Invasive Weed Optimization (IWO) along with the Cultural Evolution Algorithm (CEA) was proposed [27]. Two-step scheduling algorithm focusing on deadline and priority using an enhanced artificial Jellyfish Search Optimizer (JS) as an Improved Jellyfish Algorithm (IJFA) [34]. Crow search algorithm to find a global solution [75]. Discrete Opposition Harris Hawk Optimization (DO-HHO) [37], where each round increases the convergence speed. Task scheduling method using population-based MFO (Moth Flame Optimization) algorithm [23] to assign the optimal number of tasks.

6.3 Hyperheuristic

Hyperheuristics are a higher level heuristic. It doesn't operate on the problem domain directly but rather works on a search space of heuristics. It acts like a portfolio scheduler where it aims to find the best heuristic or metaheuristic for the current state of the edge.

One such hyperheuristic algorithm searches the metaheuristic space by using a test and select phase [39]. The metaheuristic space is composed of a Genetic Algorithm, Particle Swarm Optimization, Ant Colony Optimization, and Simulated Annealing.

6.4 Machine Learning

Machine learning is often used to find a good ordering for a set of incoming requests, and sometimes to determine the best scheduler for the current situation (portfolio scheduler).

Multi-Aerial Base Station Assisted Joint Computational Offload algorithm based on D3QN in Edge VANETs (MA-JVD3) [11], uses SDN Controllers to sense global information and enable efficient scheduling.

6.4.1 Reinforcement Learning

In reinforcement learning the agent learns how to behave by performing actions and receiving rewards or penalties in return. This can be done in simulation using different workloads.

One approach uses 2 levels of reinforcement learning, 1 for local execution and 1 for remote execution [64]. Another uses Double Reinforcement Learning Computation Offloading (DRLCO) [47], which is an actor critic model with replay memory. DRLCO uses an evaluate model to update the target model which is used in the loss function. Asynchronous Update Reinforcement Learning-based Offloading (ARLO) [50] is another actor critic model that uses 5 sub-networks to interact with the environment simultaneously. Brain-Inspired Rescheduling Decision-making (BIRD) algorithm [60] is also an actor critic model, but mimics the decision making model of the hu7man brain to control voluntary motor activity to ensure convergence. Q Learning [18] using a reward that includes the time and energy using the offloading and caching decision.

6.4.2 Deep Learning

Deep learning is a form of machine learning inspired by how the brain functions. It contains many layers, and also requiring

more training.

Deep reinforcement learning in 2 tier MEC with heterogeneous edge [79] uses a trust mechanism to prevent criminals from attacking the node with a virus. The trusted behaviour is based on historical feedback and freshness of requests [79]. DRL PPO-based called IRATS [31] tries to minimize delay while penalizing tasks that couldn't finish before the deadline. DRL for cloudlets (IDRL) [5] alters the policy using previous experience, converting DNN's output to bianry actions using a Roulette Wheel Selection. DRL, DQN_GA [86] which is similar to Q-learning, but approximates the Q-value using a neural network. Deep Reinforcement learning based offloading decision (DROD) for Vehicular Edge Computing (VEC) [28] using an improved deep deterministic policy gradient to obtain the optimal decision for the optimal problem. Multi-agent load balancing distribution based on deep reinforcement learning, Distributed Task Offloading Multi-Agent Load Balancing (DTOMALB) [94], by adressing the competition and selfishness between users using a global load balancing penalty. Deep-Q-Network [18] using a reward that includes the time and energy using the offloading and caching decision. Deep Monte Calor Tree Search (MCTS) splitting Deep Neural Network (sDNN), intelligent Task Offloading Algorithm (iTOA) [12], deciding the offloading action by perceiving the network's environment. Self-learning task offloading and resource allocation algorithm (SLRTA) [78], uses a weighted reward to balance the response time and energy consumption trade-off. Deep Deterministic Policy Gradient [92] to optimize the phase shioft and amplitude of Reconfigurable Intelligent Surface (RIS).

6.5 Integer programming

Integer programming is a mathematical optimization technique where the solution space is restricted to integer values. It differs from linear programming in that it doesn't use continuous variables, but rather discrete variables.

One scheduler that makes use of integer programming is the Okita and Okita* schedulers [63]. Okita(*) first decomposes the scheduling problem on all currently uncompleted jobs using integer programming into a series of one-shot problems. These one-shot problems are then used to pick a worker and transmit the data. Both Okita and Okita* are online schedulers. Okita is a preemptive scheduler, while Okita* is a non-preemptive scheduler to target the overhead introduced by a preemptive scheduler. The proposed schedulers are specifically aimed at unbiased distributed learning to reduce training time [63].

ILP-based algorithm called load balancing aware scheduling ILP (LASILP) [73].

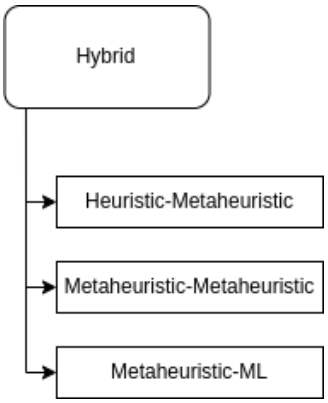

Figure 9: An overview of the types of hybrid algorithms in the reviewed papers

6.6 Hybrid

Hybrid algorithms are the combination of two (or more) algorithmic approaches in the same decision-making. It differs from hyperheuristics as seen in Section 6.3 in that it uses multiple algorithmic approaches simultaneously as opposed to picking a currently optimal heuristic. There are 3 hybrid combinations as seen in Figure 9, Heuristic-Metaheuristic, Metaheuristic-Metaheuristic, and Metaheuristic-ML.

Two approaches use a Heuristic-Metaheuristic combination. One such hybrid algorithm fireworks algorithm (FWA) uses an initial solution using HEFT for its population [87]. Another uses the first improvement type pivoting rule to find a local solution and crow search algorithm to find a global solution [75]. This local-global search combination achieves the benefits of both exploration and exploitation [75].

Four approaches use a Metaheuristic-Metaheuristic combination. One approach is a GA that uses Flamingo Search Optimization (FSO) to enhance candidates who were not chosen [29]. This best candidate of the not chosen candidates after going through FSO is then combined with the fittest solution from the GA which results in more optimal solutions. Another approach uses TS first and once no improvement is found uses a FOA to further improve the solution [59]. The last approach is the Grey Wolf Optimizer Whale Optimization Algorithm (GWO-WOA) [21], GWO is used to find optima while the WOA prevents converging on local optima. Hybrid ACO and GA [66].

Two approaches use a Metaheuristic-ML combination. One approach combines Q-learning with a hybridization of Artificial Ecosystem-based Optimization (AEO) and Arithmetic Optimization Algorithm (AOA) [89]. The Q-learning is used to hybridize the AEO-AOA [89]. Another approach uses a CNN to classify servers as suitable, after which a modified PSO picks the best server [76].

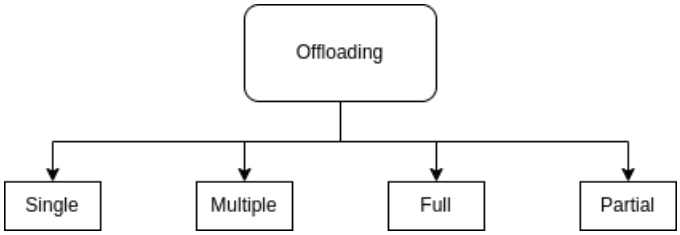

Figure 10: A taxonomy of the types of offloading

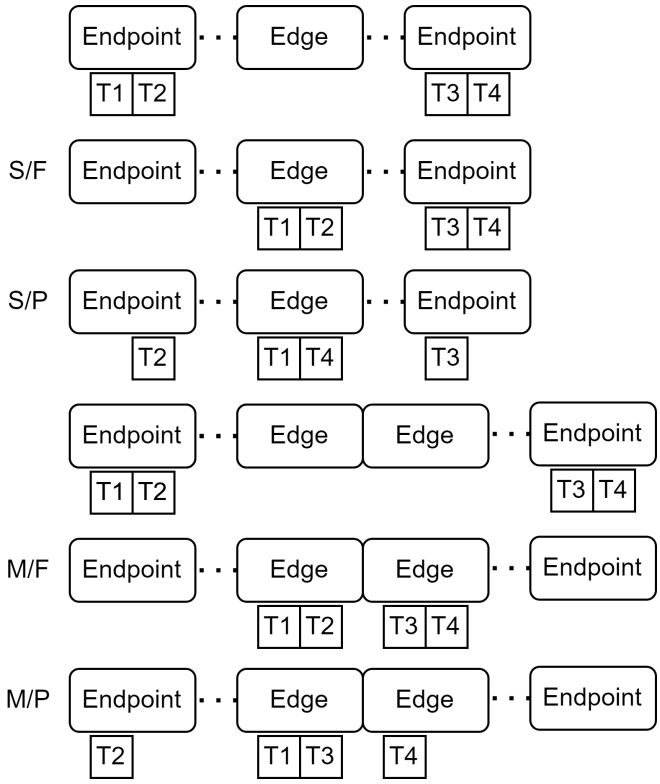

Figure 11: The 4 identified types of offloading

7 Offloading types

Offloading strategies in edge computing are essential for optimizing resource utilization, enhancing performance, and ensuring energy efficiency. The type of offloading depends on the problem model. The problem model lays out the dimensionality of the edge layer, the number of users at the endpoints, how many tasks can be offloaded to the edge per endpoint, and whether dependencies between tasks exist. Using the task flow from the endpoint layer to the edge layer lets us categorize the offloading using 4 types as seen in Figure 11, made of 2 groups as seen in Figure 10. The first group is where a task can be offloaded. A task can either only be offloaded to a single node (Single), or to multiple potential nodes (Multiple). The second group is the granularity of the tasks. A task is either a fully independent and separate chunk that isn't further decomposed into smaller subtasks (Full), or a task is dependent and part of a bigger application where a task can be offloaded to any node but might need to wait for other nodes to finish if a dependency isn't fulfilled (Partial). The combination of the aforementioned 2 groups leaves us with the 4 types: Single/Full (S/F), Single/Partial (S/P), Multiple/Full (M/F), and Multiple/Partial (M/P).

Single/Full (S/F) offloading is when the full task is either computed locally or on a singular edge node11. One paper focuses on multiple drones that all can offload their data to a single node [12]. UAVs upload their data to the edge server only based on the base station they are connected to. A set of base stations is connected to one edge server [12]. Another paper focuses on multiple mobile devices that pick one edge server [40]. The mobile device can then pick the mode, to either send the task in full or partial. If a fog node is too congested it propagates all tasks from a single user up to the cloud [40].

Single/Partial (S/P) offloading is when a task is decomposed and computed partially locally and/or partially remotely on a singular edge node11. One paper using S/F offloading also supports partial offloading [40], but on congestion of the edge server will propagate the entire task sequence up to the cloud. The rest of the papers using S/P offloading decide the offloading to the edge per task. One such paper looks at a problem with multiple users, sending potentially multiple tasks [18], each task is individually evaluated for offloading to the edge. All users connect to a single base station that is connected to one edge server where offloaded tasks can be cached for efficiency [18]. Mobile devices can connect to multiple base stations if these base stations overlap in coverage area. However, this overlap doesn't need a device to choose between multiple base stations. One paper considers the overlap of base stations, but lets the offloading destination be dependent on the base stations the mobile devices happens to connect to [48]. Offloading to a single node prevents the communication overhead and simplifies the offloading cost model [48], and if an edge server becomes overloaded it can use the vaster resources of the cloud. The last paper of this class has vehicles decompose tasks depending on their criticality and originating service [58]. Tasks from the vehicle can then be assigned to one edge server or propagated up to the cloud depending on QoS requirements [58] as discussed in Section 8.

Multiple/Full (M/F) offloading is when a dependent task sequence is fully offloaded in the edge at one node (or locally) but can be scheduled at multiple nodes11. One paper always performs offloading, but the decision is rather where to offload to [76]. Multiple servers are considered and the edge server to offload to is chosen based on current storage, computing, and RAM limitations [76]. Other papers also consider this load balancing of resources to maximize the utilization [33, 73, 74, 90], but do this via delay optimization with resource constraints. Some approaches are driven by the QoS, like

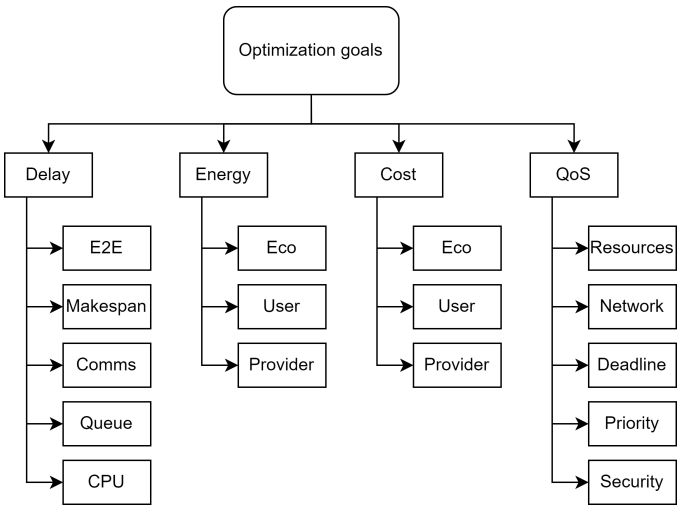

Figure 12: A taxonomy of optimization goals

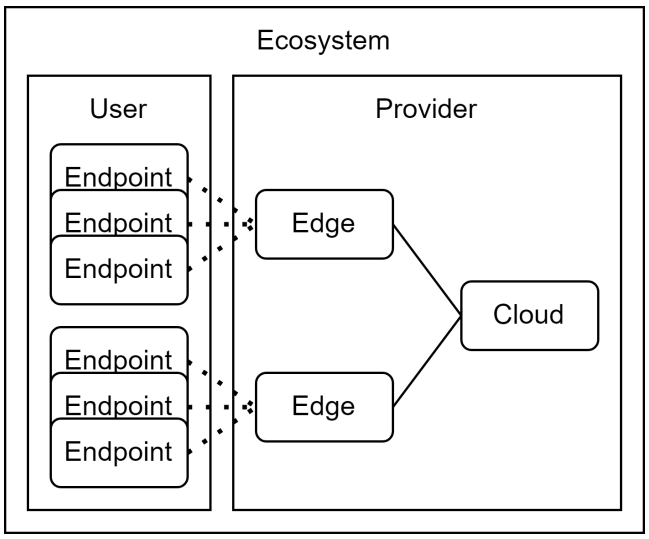

Figure 13: Measurement scopes for energy and cost optimization goals

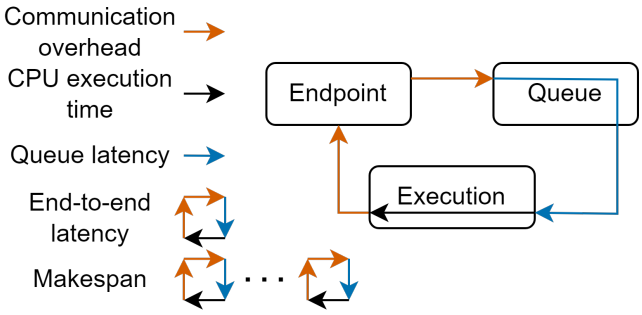

Figure 14: Differences in delay definitions

determining the offload location according to a task deadline with an expanded set of offload locations when the assigned location is over-constrained [16]. Others consider multiple edge servers and try to get an optimal global result by trying multiple solutions of task distributions [4, 29, 66, 69]. Another approach can be taken for mobile devices where the sequence of task is summarized using its cost in terms of computation and communication latency to select an appropriate edge node [46].

Multiple/Partial (M/P) offloading is when dependent tasks can be scheduled at several different edge nodes and the scheduler takes the amount of tasks coming from an endpoint into account11. The most trivial case is when scheduling a DAG of dependent tasks [6, 27, 45, 54, 56, 87], where varying cuts can be made to distribute the DAG over multiple edge nodes, this practice is often called workflow scheduling. When multiple fog nodes compute dependent tasks it also becomes important to calculate how long a task must wait for its dependent part to finish. The bandwidth between fog nodes is often taken into account to account for wait time [10, 23, 47, 94]. Another approach performs the placement of tasks entirely in the edge, where the endpoint sends its task to its closest edge server and the edge server can then offload it further using a central controller [88, 93]. When edge nodes are more sparse and the location of the base station becomes more important it could be done to divide the edge resources into regions directly connected via a broker [37, 38].

8 Optimization goals

An edge scheduling algorithm is designed with a certain optimization goal in mind. These optimization goals are multi-faceted, aiming to address the unique challenges and requirements of the edge. We will discuss the 4 major groups, and talk about the combination of the optimization goals. The groups are as seen in Figure 12.

- Delay: a time/latency measure in seconds or cycles as seen in Figure 14.
  - End-to-end (E2E) latency: total time to execute a single task as measured from the endpoint [18, 23, 33, 40, 48, 56, 94].
  - Makespan: total time to execute a sequence of tasks as measured from the endpoint [4, 6, 23, 29, 37–39, 45, 47, 74, 76, 87, 88, 90].
  - Communication overhead: cumulative transmission time between endpoint-edge, edge-edge, and edge-cloud [12, 23, 93, 94].
  - Queue latency: time waited in the scheduling queue [40, 74, 94].
  - CPU execution time: time spent on CPU [12, 93].
- Energy: a power usage measure in kWh or emissions defined over different scopes as seen in Figure 13.

  - Ecosystem energy usage: energy usage for communication and computation of endpoints, edge, and cloud [47].
  - User energy usage: energy usage for communication and computation of endpoints [10, 12, 40, 45, 46, 48].
  - Provider energy usage: energy usage for communication and computation of edge, and cloud [4, 39, 90].
- Cost: a monetary measure derived from hardware or operational cost defined over different scopes as seen in Figure 13.
  - Ecosystem cost: cost of hardware, infrastructure, and maintenance for endpoints, edge, and cloud [18].
  - User cost: cost of hardware for endpoints [59] or cost of offloading [29, 87, 88].
  - Provider cost: cost of hardware, infrastructure, and maintenance for edge, and cloud [6, 39, 56, 58, 69].
- QoS: requirements for a task or edge node. ghobaei2020efficient
  - Resource utilization: load balancing of endpoint and edge resources [27, 33, 48, 73, 74, 76, 94].
  - Network utilization: utilization of bandwidth [33, 38, 39, 80].
  - Deadline: a set time before which the task [16, 23, 46, 58, 74, 80], or group of tasks [6, 29, 69, 75] must be finished.
  - Priority: based on assigned level or deadline requirement [38, 56, 69].
  - Security/Privacy: ratio of tasks that can be scheduled according to their security level [75, 93], or tagging every task with a privacy level of public, semiprivate, and private [73].

9 Threats to validity

There are 3 main threats to the validity of the survey.

First, we used the most recent release of the dblp dataset as of writing the literature study seen in [2]. Other search engines were disregarded as the datasets were older, hard to crawl, or behind an API key. Semantic Scholar, was among other available sources. Future surveys could focus on a broader set of search engines also facilitating additional meta-analysis and highlighting differences between search engines. (maybe do a comparison of a subset as a way to justify, approach, or motivate future research).

Second, grey literature was excluded as dblp primarily focuses on peer-reviewed publications.

Third, there are several different communities with different perspectives which are all valid, but each field is biased towards certain goals. This could cause problems generalizing.

10 Conclusion

Through a systematic literature review, we have presented a taxonomy that spans paradigms, algorithmic approaches, offloading types, and optimization goals, offering a view of the current state of the art.

The challenges posed by the edge, such as mobility, computational constraints, and power limitations, require new innovative schedulers. One such scheduler type that is underrepresented but showed great promise is the hyperheuristics. This could be extended into a portfolio scheduler that also includes machine learning amongst the possible choices to best adapt to the current state of the edge and jobs. Metaheuristics were by far the most common algorithm type but recent schedulers use machine learning almost as often. Another major challenge is the generalizability of the tested scenarios. Many schedulers have a set amount of experiments which differ greatly from paper to paper.

Such a diverse range of algorithmic approaches, from heuristics and metaheuristics to machine learning techniques, offer many solutions to specific scenarios and requirements. The exploration of offloading strategies, whether single, multiple, full, or partial, provides avenues for optimizing resource utilization and performance. Moreover, the focus on optimization goals, including delay, energy, cost, and QoS constraints ensures that the solutions are aligned with real-world demands. Delay is usually included as a measurement but recent schedulers focus more on energy savings or balancing energy usage over the entire system.

In conclusion, the edge presents a promising future and will establish itself as a mainstay in the computing landscape. This will only be accelerated as more IoT devices, vehicles, drones, and smartphones start requiring real-time high-bandwidth applications. This systematic literature review has brought a taxonomy on scheduling in the edge and expanded upon this with further taxonomies on the paradigms, algorithmic approaches, offloading, and optimization goals. The presented separation of the system model, problem model (offloading and optimization goal), and algorithm serves as a future blueprint to establish a framework for a new scheduling algorithm.

References

[1] dblp search api for publication queries, Jul 2023.

[2] Monthly snapshot release of july 2023, Jul 2023.

[3] Aazam, M., Zeadally, S., and Harras, K. A. Offloading in fog computing for iot: Review, enabling technologies, and research opportunities. *Future Generation Computer Systems 87* (2018), 278–289.

[4] ABDEL-BASSET, M., MOHAMED, R., CHAKRABORTTY, R. K., AND RYAN, M. J. Iega: an improved elitism-based genetic algorithm for task scheduling problem in fog computing. *International Journal of Intelligent Systems 36*, 9 (2021), 4592–4631.

[5] AGHAPOUR, Z., SHARIFIAN, S., AND TAHERI, H. Task offloading and resource allocation algorithm based on deep reinforcement learning for distributed ai execution tasks in iot edge computing environments. *Computer Networks 223* (2023), 109577.

[6] BACANIN, N., ZIVKOVIC, M., BEZDAN, T., VENKATACHALAM, K., AND ABOUHAWWASH, M. Modified firefly algorithm for workflow scheduling in cloud-edge environment. *Neural computing and applications 34*, 11 (2022), 9043–9068.

[7] BALI, M. S., ALROOBAEA, R., ALGARNI, S., ALSAFYANI, M., MOHIUDDIN, K., GUPTA, K., AND GUPTA, D. An efficient task allocation framework for scheduled data in edge based internet of things using hybrid optimization algorithm approach. *Physical Communication 58* (2023), 102047.

[8] BELMAHDI, R., MECHTA, D., AND HAROUS, S. A survey on various methods and algorithms of scheduling in fog computing. *Ingénierie des systèmes d'information 26*, 2 (2021).

[9] CHAI, Z.-Y., LIU, X., AND LI, Y.-L. A computation offloading algorithm based on multi-objective evolutionary optimization in mobile edge computing. *Engineering Applications of Artificial Intelligence 121* (2023), 105966.

[10] CHAKRABORTY, S., AND MAZUMDAR, K. Sustainable task offloading decision using genetic algorithm in sensor mobile edge computing. *Journal of King Saud University-Computer and Information Sciences 34*, 4 (2022), 1552–1568.

[11] CHEN, G., ZHOU, Y., XU, X., ZENG, Q., AND ZHANG, Y.-D. A multi-aerial base station assisted joint computation offloading algorithm based on d3qn in edge vanets. *Ad Hoc Networks 142* (2023), 103098.

[12] CHEN, J., CHEN, S., LUO, S., WANG, Q., CAO, B., AND LI, X. An intelligent task offloading algorithm (itoa) for uav edge computing network. *Digital Communications and Networks 6*, 4 (2020), 433–443.

[13] CHEN, J., DENG, Q., AND YANG, X. Non-cooperative game algorithms for computation offloading in mobile edge computing environments. *Journal of Parallel and Distributed Computing 172* (2023), 18–31.

[14] CHEN, Y.-M., LIU, S.-L., CHEN, Y.-J., AND LING, X. A scheduling algorithm for heterogeneous computing systems by edge cover queue. *Knowledge-Based Systems 265* (2023), 110369.

[15] CHUANG, Y.-T., AND HUNG, Y.-T. A real-time and aco-based offloading algorithm in edge computing. *Journal of Parallel and Distributed Computing 179* (2023), 104703.

[16] DAI, H., ZENG, X., YU, Z., AND WANG, T. A scheduling algorithm for autonomous driving tasks on mobile edge computing servers. *Journal of Systems Architecture 94* (2019), 14–23.

[17] DE QUEIROZ, G. F., DE REZENDE, J. F., AND BARBOSA, V. C. A flexible algorithm to offload dag applications for edge computing. *arXiv preprint arXiv:2306.09458* (2023).

[18] ELGENDY, I. A., ZHANG, W.-Z., HE, H., GUPTA, B. B., AND ABD EL-LATIF, A. A. Joint computation offloading and task caching for multi-user and multi-task mec systems: reinforcement learning-based algorithms. *Wireless Networks 27*, 3 (2021), 2023–2038.

[19] EZHILARASIE, R., UMAMAKESWARI, A., REDDY, M. S., AND BALAKRISHNAN, P. Grefenstette bias based genetic algorithm for multi-site offloading using docker container in edge computing. *Journal of Intelligent & Fuzzy Systems 36*, 3 (2019), 2419–2429.

[20] FAHAD, M., SHOJAFAR, M., ABBAS, M., AHMED, I., AND IJAZ, H. A multi-queue priority-based task scheduling algorithm in fog computing environment. *Concurrency and Computation: Practice and Experience 34*, 28 (2022), e7376.

[21] FENG, S., CHEN, Y., ZHAI, Q., HUANG, M., AND SHU, F. Optimizing computation offloading strategy in mobile edge computing based on swarm intelligence algorithms. *EURASIP Journal on Advances in Signal Processing 2021*, 1 (2021), 1–15.

[22] GHOBAEI-ARANI, M., SOURI, A., AND RAHMANIAN, A. A. Resource management approaches in fog computing: a comprehensive review. *Journal of Grid Computing 18*, 1 (2020), 1–42.

[23] GHOBAEI-ARANI, M., SOURI, A., SAFARA, F., AND NOROUZI, M. An efficient task scheduling approach using moth-flame optimization algorithm for cyber-physical system applications in fog computing. *Transactions on Emerging Telecommunications Technologies 31*, 2 (2020), e3770.

[24] GUO, W., LEI, Q., SONG, Y., AND LYU, X. A learning interactive genetic algorithm based on edge selection encoding for assembly job shop scheduling problem. *Computers & Industrial Engineering 159* (2021), 107455.

[25] HASHEM, I. A. T., YAQOOB, I., ANUAR, N. B., MOKHTAR, S., GANI, A., AND KHAN, S. U. The rise of "big data" on cloud computing: Review and open research issues. *Information systems 47* (2015), 98–115.

[26] HONG, C.-H., AND VARGHESE, B. Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms. *ACM Computing Surveys (CSUR) 52*, 5 (2019), 1–37.

[27] HOSSEINIOUN, P., KHEIRABADI, M., TABBAKH, S. R. K., AND GHAEMI, R. A new energy-aware tasks scheduling approach in fog computing using hybrid meta-heuristic algorithm. *Journal of Parallel and Distributed Computing 143* (2020), 88–96.

[28] HU, X., AND HUANG, Y. Deep reinforcement learning based offloading decision algorithm for vehicular edge computing. *PeerJ Computer Science 8* (2022), e1126.

[29] HUSSAIN, S. M., AND BEGH, G. R. Hybrid heuristic algorithm for cost-efficient qos aware task scheduling in fog–cloud environment. *Journal of Computational Science 64* (2022), 101828.

[30] ISLAM, A., DEBNATH, A., GHOSE, M., AND CHAKRABORTY, S. A survey on task offloading in multi-access edge computing. *Journal of Systems Architecture 118* (2021), 102225.

[31] JAMIL, B., IJAZ, H., SHOJAFAR, M., AND MUNIR, K. Irats: A drl-based intelligent priority and deadline-aware online resource allocation and task scheduling algorithm in a vehicular fog network. *Ad hoc networks 141* (2023), 103090.

[32] JAMIL, B., IJAZ, H., SHOJAFAR, M., MUNIR, K., AND BUYYA, R. Resource allocation and task scheduling in fog computing and internet of everything environments: A taxonomy, review, and future directions. *ACM Computing Surveys (CSUR) 54*, 11s (2022), 1–38.

[33] JAMIL, B., SHOJAFAR, M., AHMED, I., ULLAH, A., MUNIR, K., AND IJAZ, H. A job scheduling algorithm for delay and performance optimization in fog computing. *Concurrency and Computation: Practice and Experience 32*, 7 (2020), e5581.

[34] JANGU, N., AND RAZA, Z. Improved jellyfish algorithm-based multi-aspect task scheduling model for iot tasks over fog integrated cloud environment. *Journal of Cloud Computing 11*, 1 (2022), 1–21.

[35] JANSEN, M., AL-DULAIMY, A., PAPADOPOULOS, A. V., TRIVEDI, A., AND IOSUP, A. The spec-rg reference architecture for the edge continuum. *arXiv preprint arXiv:2207.04159* (2022).

[36] JANSEN, M., AL-DULAIMY, A., PAPADOPOULOS, A. V., TRIVEDI, A. K., AND IOSUP, A. The spec-rg reference architecture for the compute continuum.

[37] JAVAHERI, D., GORGIN, S., LEE, J.-A., AND MASDARI, M. An improved discrete harris hawk optimization algorithm for efficient workflow scheduling in multi-fog computing. *Sustainable Computing: Informatics and Systems 36* (2022), 100787.

[38] JAVANMARDI, S., SHOJAFAR, M., PERSICO, V., AND PESCAPÈ, A. Fpfts: A joint fuzzy particle swarm optimization mobility-aware approach to fog task scheduling algorithm for internet of things devices. *Software: practice and experience 51*, 12 (2021), 2519–2539.

[39] KABIRZADEH, S., RAHBARI, D., AND NICKRAY, M. A hyper heuristic algorithm for scheduling of fog networks. In *2017 21st Conference of Open Innovations Association (FRUCT)* (2017), IEEE, pp. 148–155.

[40] KESHAVARZNEJAD, M., REZVANI, M. H., AND ADABI, S. Delay-aware optimization of energy consumption for task offloading in fog environments using metaheuristic algorithms. *Cluster Computing* (2021), 1–29.

[41] KHALEDIAN, N., KHAMFOROOSH, K., AZIZI, S., AND MAIHAMI, V. Ikh-eft: An improved method of workflow scheduling using the krill herd algorithm in the fog-cloud environment. *Sustainable Computing: Informatics and Systems 37* (2023), 100834.

[42] KINNEY, R. M., ANASTASIADES, C., AUTHUR, R., BELTAGY, I., BRAGG, J., BURACZYNSKI, A., CACHOLA, I., CANDRA, S., CHANDRASEKHAR, Y., COHAN, A., CRAWFORD, M., DOWNEY, D., DUNKELBERGER, J., ETZIONI, O., EVANS, R., FELDMAN, S., GORNEY, J., GRAHAM, D. W., HU, F., HUFF, R., KING, D., KOHLMEIER, S., KUEHL, B., LANGAN, M., LIN, D., LIU, H., LO, K., LOCHNER, J., MACMILLAN, K., MURRAY, T., NEWELL, C., RAO, S. R., ROHATGI, S., SAYRE, P. L., SHEN, Z., SINGH, A., SOLDAINI, L., SUBRAMANIAN, S., TANAKA, A., WADE, A. D., WAGNER, L. M., WANG, L. L., WILHELM, C., WU, C., YANG, J., ZAMARRON, A., VAN ZUYLEN, M., AND WELD, D. S. The semantic scholar open data platform. *ArXiv abs/2301.10140* (2023).

[43] KITCHENHAM, B. A., DYBA, T., AND JORGENSEN, M. Evidence-based software engineering. In *Proceedings. 26th International Conference on Software Engineering* (2004), IEEE, pp. 273–281.

[44] KUMAR, M., SHARMA, S. C., GOEL, A., AND SINGH, S. P. A comprehensive survey for scheduling techniques in cloud computing. *Journal of Network and Computer Applications 143* (2019), 1–33.

[45] LAKHAN, A., AND LI, X. Transient fault aware application partitioning computational offloading algorithm in microservices based mobile cloudlet networks. *Computing 102*, 1 (2020), 105–139.

[46] LAKHAN, A., MOHAMMED, M. A., RASHID, A. N., KADRY, S., AND ABDULKAREEM, K. H. Deadline aware and energy-efficient scheduling algorithm for fine-grained tasks in mobile edge computing. *International Journal of Web and Grid Services 18*, 2 (2022), 168–193.

[47] LIAO, L., LAI, Y., YANG, F., AND ZENG, W. Online computation offloading with double reinforcement learning algorithm in mobile edge computing. *Journal of Parallel and Distributed Computing 171* (2023), 28–39.

[48] LIAO, Z., PENG, J., XIONG, B., AND HUANG, J. Adaptive offloading in mobile-edge computing for ultra-dense cellular networks based on genetic algorithm. *Journal of Cloud Computing 10*, 1 (2021), 1–16.

[49] LIU, L., AND DU, Y. An improved multi-objective evolutionary algorithm for computation offloading in the multi-cloudlet environment. *Frontiers of Computer Science 15*, 5 (2021), 155503.

[50] LIU, Z., LIU, Y., LEI, Y., ZHOU, Z., AND WANG, X. Arlo: An asynchronous update reinforcement learning-based offloading algorithm for mobile edge computing. *Peer-to-Peer Networking and Applications* (2023), 1–13.

[51] LIU, Z., WANG, J., GAO, Z., AND WEI, J. Privacy-preserving edge computing offloading scheme based on whale optimization algorithm. *The Journal of Supercomputing 79*, 3 (2023), 3005–3023.

[52] LOHAT, S., JAIN, S., AND KUMAR, R. Fractional mayfly optimization algorithm-based infrastructure-to-vehicle and vehicle-to-vehicle scheduling for service message transmission in iov-fog. *International Journal of Communication Systems* (2023), e5479.

[53] LU, Y., LIU, L., GU, J., PANNEERSELVAM, J., AND YUAN, B. Eadfpso: An intelligent energy-efficient scheduling algorithm for mobile edge networks. *Digital Communications and Networks 8*, 3 (2022), 237–246.

[54] MA, S., SONG, S., YANG, L., ZHAO, J., YANG, F., AND ZHAI, L. Dependent tasks offloading based on particle swarm optimization algorithm in multi-access edge computing. *Applied Soft Computing 112* (2021), 107790.

[55] MACH, P., AND BECVAR, Z. Mobile edge computing: A survey on architecture and computation offloading. *IEEE communications surveys & tutorials 19*, 3 (2017), 1628–1656.

[56] MADHURA, R., ELIZABETH, B. L., AND UTHARIARAJ, V. R. An improved list-based task scheduling algorithm for fog computing environment. *Computing 103* (2021), 1353–1389.

[57] MAO, Y., YOU, C., ZHANG, J., HUANG, K., AND LETAIEF, K. B. A survey on mobile edge computing: The communication perspective. *IEEE communications surveys & tutorials 19*, 4 (2017), 2322–2358.

[58] MATERWALA, H., ISMAIL, L., SHUBAIR, R. M., AND BUYYA, R. Energy-sla-aware genetic algorithm for edge–cloud integrated computation offloading in vehicular networks. *Future Generation Computer Systems 135* (2022), 205–222.

[59] MEMARI, P., MOHAMMADI, S. S., JOLAI, F., AND TAVAKKOLI-MOGHADDAM, R. A latency-aware task scheduling algorithm for allocating virtual machines in a cost-effective and time-sensitive fog-cloud architecture. *The Journal of Supercomputing 78*, 1 (2022), 93–122.

[60] NAIR, B., AND BHANU, S. M. S. A reinforcement learning algorithm for rescheduling preempted tasks in fog nodes. *Journal of Scheduling 25*, 5 (2022), 547–565.

[61] NAVANEETHA KRISHNAN, M., AND THIYAGARAJAN, R. Multi-objective task scheduling in fog computing using improved gaining sharing knowledge based algorithm. *Concurrency and Computation: Practice and Experience 34*, 24 (2022), e7227.

[62] OKOLI, C. A guide to conducting a standalone systematic literature review. *Communications of the Association for Information Systems 37* (2015).

[63] PANG, J., HAN, Z., ZHOU, R., TAN, H., AND CAO, Y. Online scheduling algorithms for unbiased distributed learning over wireless edge networks. *Journal of Systems Architecture 131* (2022), 102673.

[64] RAMEZANI SHAHIDANI, F., GHASEMI, A., TOROGHI HAGHIGHAT, A., AND KESHAVARZI, A. Task scheduling in edge-fog-cloud architecture: a multi-objective load balancing approach using reinforcement learning algorithm. *Computing 105*, 6 (2023), 1337–1359.

[65] REN, J., ZHANG, D., HE, S., ZHANG, Y., AND LI, T. A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet. *ACM Computing Surveys (CSUR) 52*, 6 (2019), 1–36.

[66] REN, X., ZHANG, Z., AND AREFZADEH, S. M. An energy-aware approach for resource managing in the fog-based internet of things using a hybrid algorithm. *International Journal of Communication Systems 34*, 1 (2021), e4652.

[67] RODRIGUEZ, M. A., AND BUYYA, R. A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments. *Concurrency and Computation: Practice and Experience 29*, 8 (2017), e4041.

[68] SATYANARAYANAN, M. The emergence of edge computing. *Computer 50*, 1 (2017), 30–39.

[69] SAXENA, D., AND SAXENA, S. Highly advanced cloudlet scheduling algorithm based on particle swarm optimization. In *2015 Eighth International Conference on Contemporary Computing (IC3)* (2015), IEEE, pp. 111–116.

[70] SHAKARAMI, A., GHOBAEI-ARANI, M., AND SHAHIDINEJAD, A. A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective. *Computer Networks 182* (2020), 107496.

[71] SHI, W., CAO, J., ZHANG, Q., LI, Y., AND XU, L. Edge computing: Vision and challenges. *IEEE internet of things journal 3*, 5 (2016), 637–646.

[72] SHUKLA, P., AND PANDEY, S. Maa: multi-objective artificial algae algorithm for workflow scheduling in heterogeneous fog-cloud environment. *The Journal of Supercomputing* (2023), 1–43.

[73] SINGH, A., AND AULUCK, N. Load balancing aware scheduling algorithms for fog networks. *Software: Practice and Experience 50*, 11 (2020), 2012–2030.

[74] SOBHANAYAK, S., JAISWAL, K., TURUK, A. K., SAHOO, B., MOHANTA, B. K., AND JENA, D. Container-based task scheduling for edge computing in iot-cloud environment using improved hbf optimisation algorithm. *International Journal of Embedded Systems 13*, 1 (2020), 85–100.

[75] SUBBARAJ, S., THIYAGARAJAN, R., AND RENGARAJ, M. A smart fog computing based real-time secure resource allocation and scheduling strategy using multi-objective crow search algorithm. *Journal of Ambient Intelligence and Humanized Computing* (2021), 1–13.

[76] TALAAT, F. M., ALI, H. A., SARAYA, M. S., AND SALEH, A. I. Effective scheduling algorithm for load balancing in fog environment using cnn and mpso. *Knowledge and Information Systems 64*, 3 (2022), 773–797.

[77] TALEB, T., SAMDANIS, K., MADA, B., FLINCK, H., DUTTA, S., AND SABELLA, D. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials 19*, 3 (2017), 1657–1681.

[78] TONG, Z., DENG, X., MEI, J., LIU, B., AND LI, K. Response time and energy consumption co-offloading with slrta algorithm in cloud–edge collaborative computing. *Future Generation Computer Systems 129* (2022), 64–76.

[79] TONG, Z., YE, F., MEI, J., LIU, B., AND LI, K. A novel task offloading algorithm based on an integrated trust mechanism in mobile edge computing. *Journal of Parallel and Distributed Computing 169* (2022), 185–198.

[80] UYAN, O. G., AND GUNGOR, V. C. Qos-aware lte-a downlink scheduling algorithm: A case study on edge users. *International Journal of Communication Systems 32*, 15 (2019), e4066.

[81] WANG, B., WU, P., AND AREFZAEH, M. A new method for task scheduling in fog-based medical healthcare systems using a hybrid nature-inspired algorithm. *Concurrency and Computation: Practice and Experience 34*, 22 (2022), e7155.

[82] WANG, Y., LIU, J., TONG, Y., YANG, Q., LIU, Y., AND MOU, H. Resource scheduling in mobile edge computing using improved ant colony algorithm for space information network. *International Journal of Satellite Communications and Networking 41*, 4 (2023), 331–356.

[83] WOHLIN, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering* (2014), pp. 1–10.

[84] XIAO, Y., AND WATSON, M. Guidance on conducting a systematic literature review. *Journal of planning education and research 39*, 1 (2019), 93–112.

[85] XU, F., XIE, Y., SUN, Y., QIN, Z., LI, G., AND ZHANG, Z. Two-stage computing offloading algorithm in cloud-edge collaborative scenarios based on game theory. *Computers & Electrical Engineering 97* (2022), 107624.

[86] XUE, F., HAI, Q., DONG, T., CUI, Z., AND GONG, Y. A deep reinforcement learning based hybrid algorithm for efficient resource scheduling in edge computing environment. *Information Sciences 608* (2022), 362–374.

[87] YADAV, A. M., TRIPATHI, K. N., AND SHARMA, S. A bi-objective task scheduling approach in fog computing using hybrid fireworks algorithm. *The Journal of Supercomputing 78*, 3 (2022), 4236–4260.

[88] YADAV, A. M., TRIPATHI, K. N., AND SHARMA, S. An enhanced multi-objective fireworks algorithm for task scheduling in fog computing environment. *Cluster Computing* (2022), 1–16.

[89] YEGANEH, S., SANGAR, A. B., AND AZIZI, S. A novel q-learning-based hybrid algorithm for the optimal offloading and scheduling in mobile edge computing environments. *Journal of Network and Computer Applications 214* (2023), 103617.

[90] YOU, Q., AND TANG, B. Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things. *Journal of Cloud Computing 10* (2021), 1–11.

[91] ZHANG, J., ZHENG, R., ZHAO, X., ZHU, J., XU, J., AND WU, Q. A computational resources scheduling algorithm in edge cloud computing: from the energy efficiency of users' perspective. *The Journal of Supercomputing* (2022), 1–22.

[92] ZHANG, X., WU, W., LIU, S., AND WANG, J. An efficient computation offloading and resource allocation algorithm in ris empowered mec. *Computer Communications 197* (2023), 113–123.

[93] ZHANG, Y., LIU, Y., ZHOU, J., SUN, J., AND LI, K. Slow-movement particle swarm optimization algorithms for scheduling security-critical tasks in resource-limited mobile edge computing. *Future Generation Computer Systems 112* (2020), 148–161.

[94] ZHANG, Z., LI, C., PENG, S., AND PEI, X. A new task offloading algorithm in edge computing. *EURASIP Journal on Wireless Communications and Networking 2021* (2021), 1–21.