UNIVERSITY OF SOUTHAMPTON

Faculty of Physical Engineering and Science
School of Electronics and Computer Science

A project report submitted for the award of
Masters of Engineering

Supervisor: Dr Timothy Norman
Examiner: Dr Nicholas Harris

Reinforcement Learning agents for Online Elastic Resource allocation in Mobile Edge Computing

by **Mark Towers**

May 8, 2020

University of Southampton

Abstract

Faculty of Physical Engineering and Science
School of Electronics and Computer Science

A project report submitted for the award of Masters of Engineering

Reinforcement Learning agents for Online Elastic Resource allocation in Mobile Edge Computing

by Mark Towers

Mobile Edge clouds enable computational tasks to be completed at the edge of the network, without relying on access to remote data centres. A key challenge in these settings is that servers have limited computational resources that often need to be allocated to many self-interested users. Existing resource allocation approaches usually assume that tasks have inelastic resource requirements (i.e., a fixed amount of computation, bandwidth and storage), which may result in inefficient resource use and even bottlenecks. In this project, an elastic resource requirement mechanism is expanded upon to an online setting, such that tasks arrive over time with the prices and resource allocation determined by agents trained using reinforcement learning.

Contents

List of Figures

List of Tables

Declaration of Authorship

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a degree at this University;
2. Where any part of this thesis has previously been submitted for any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as: S.R. Gunn. Pdflatex instructions, 2001. URL http://www.ecs.soton.ac.uk/~srg/softwaretools/document/
   C. J. Lovell. Updated templates, 2011
   S.R. Gunn and C. J. Lovell. Updated templates reference 2, 2011

Signed:..... Date:.....

Acknowledgements

I want to thank my parents for all of the support they have given me because I would not be where I am now without them. Also to my housemates for surviving with me pestering them about proof reading badly written papers and for dealing with me stressing about this project.

This project wouldn't have started without Dr Sebastian Stein, Professor Tim Norman and a team of Pennsylvania State University that has produced a paper investigating the static case of this problem. Thank to all of them for sharing ideas and support for that paper and this project.

Also to Professor Tim Norman for this constant guidance over the project and the wisdom to know what for me to investigate and implement.

Chapter 1

Introduction

Cloud computing is a rapidly growing technology with competition from Google, Amazon, Microsoft and others that aims to allow users to run computer programs that are too large, difficult or time consuming to run locally. These services provide the computational resources, e.g. CPU cores, RAM, hard drive space, bandwidth, etc to be able to run such programs. However, as these resources are limited, if users request an unbalance quantity of resources, bottlenecks can occur limiting the number of tasks [1] that can be run on servers simultaneously.

For Google Cloud Services (GCP), Microsoft Azure or Amazon Web Services, their cloud computing facilities contain huge server nodes limiting the probability that such a bottleneck occurs. But if such an event does occur, users have a range of data centres across the global to use if a single data centre does becomes overloaded with requests. Therefore this work considers a developing paradigm (Mao et al., 2017) called Mobile Edge Computing (Hu et al., 2015) referred to as MEC. MEC aim to provide users with the ability to run their tasks closer to them in the network, reducing latency, network congestion and providing better application performance.

Currently Disaster Response (Guerdan et al., 2017), Smart Cities (Alazawi et al., 2014) and the Internet-of-Things (Corcoran and Datta , 2016) are all areas that utilise MECs due to its ability to process computationally small tasks locally with low latency. For example, in Smart Cities, this allows for smart intersection systems using road-side sensors or smart traffic lights to minimise cars waiting times at traffic lights and reduce traffic congestion (Mustapha et al.,

[1] Tasks, Programs and Jobs will be used interchangeable to refer to the same idea of a computer programs that has a fixed amount of resources required to compute.

2018). Or for the police to analyse CCTV footage to spot suspicious behaviour and to track people between cameras (Sreenu and Saleem Durai, 2019). In the case of Disaster Response, maps can be produced using data from autonomous vehicles' sensors that can support in the search for potential victims and support responders in planning rescues (Alazawi et al., 2014).

With MECs, the problem of bottlenecks is of particular relevant as instead of large server farms that can be geographically distant from the users. Servers are significantly smaller, possibly just high powered desktop computers or single server nodes. This results in greater demand on individual server resources, meaning that efficient allocation of these resources is of growing importance as the technology continues to grow and be utilised by new technologies.

However it is believed that there are shortcomings in existing research about resource allocation within MEC (Farhadi et al., 2019; Bi et al., 2019) due to the nature of how task resource usage is determined. Traditionally, a user would submit a request for a fixed amount of resources, i.e. 2 CPU cores, 8GB of RAM, 20GB of storage, that would be allocated for the user. As a result, these resources can't be redistributed until the user finishes with them. The reason that this form of resource allocation is used and effective within cloud computing is due to its simplicity for the user to decide resource requirements. The utilisation of simple linear pricing mechanisms and it being rare for servers with large resource capacity to have bottlenecks. However it is believed that the problem of bottlenecks within MEC systems, warrant the investigation of an alternative resource allocation mechanisms.

In previous work a novel resource allocation mechanism was proposed (Towers et al., 2020) to allow for significantly more flexibility in determining resource usage with the aims of reducing possible bottlenecks. The mechanism is based on the principle that the time taken for an operation to complete is generally proportional to the resources provided for the operation. An example for this is downloading an image, the time taken is proportional to the bandwidth allocated. This sort of flexibility is similarly true for computing of most tasks [2] or sending back results to the user.
Based on this principle, a modified resource allocation mechanism can be reconstructed such that the users provide the task's total resource usage over its lifetime instead of the task's requested resource usage. This allows for each task's resource

[2]It is well known that some algorithm are not linearly scalable making this principle incompatible with those tasks. Therefore in this work consider the case for algorithms that can be scalable linearly and leaves case of non-linearly scalable tasks to future research.

usage to be determined by the server rather than the user increasing a server's flexibility and control. Using this flexible resource allocation mechanism, algorithms proposed achieved 20% better social welfare than a fixed inflexible resource allocation mechanisms in one-shot cases investigated by Towers et al. (2020). This is due to the ability of the algorithms to properly balance task resources, preventing bottlenecks occurring as often, which in turn allowed for more tasks to run simultaneously and to reduce the price.

However that work only considered the proposed mechanism within a one-shot case where all tasks were presented at the first time step, where in all tasks would be auctioned and resource allocated. As a result, practically the proposed algorithms would require tasks to be processed in batches, such that servers would bid on all tasks submitted every 5 minutes for example. This also meant that while resources could be dynamically allocated at the first time step, they would not change during the next batches until the task was completed. This work aims to address these problems.

This was achieved by introducing time into the optimisation problem (outlined in section 3.1). As a result, task now arrive over time and servers can redistribute resources at each time step. However, all previous mechanisms proposed in Towers et al. (2020) are incompatible with this modified online flexible optimisation problem. Therefore this work investigates Reinforcement Learning methods that train agents to optimally bid on tasks based on their resource requirements and efficiently allocate resources to tasks running on a server.

This report is set out in the following chapters. Chapter 2 investigates previous research that this project builds upon within both resource allocation in Cloud Computing and Reinforcement Learning. Chapter 3 proposes a solution to the problem outline in Chapter 1. The solution is implemented in chapter 4 with testing and evaluation in Chapter 5. Chapter 6 presents the conclusion along with future work for the project.

In addition to this report, the paper referred to as Towers et al. (2020) was written within this academic year and thus considered part of this project's work. A copy of the paper can be found in Appendix A. The paper was also presented at SPIE Defense and Commercial Sensing 2020 as a recorded digital presentation. A copy of the slides can be found in Appendix B with a link to the recording.

Chapter 2

Literature Review

There is a considerable amount of research in the area of resource allocation and task pricing in cloud computing, where auction mechanisms are used to deal with competition. Section 2.1 presents the different approaches to resource allocation and pricing mechanisms in Cloud Computing.

The proposed solution of the project (presented in chapter 3) uses a form of machine learning, called Reinforcement Learning to train agents. Section 2.2 covers the current state-of-the-art algorithms in Q learning and policy gradient research.

2.1 Resource allocation and pricing in Cloud Computing

A majority of approaches taken for task pricing and resource allocation in Cloud Computing uses a fixed resource allocation mechanism, such that each user requests a fixed amount of resources for a task from a server. However this mechanism, as previously explained, provides no control for the server over the quantity of resource allocated to a task, only determining the task's price. As a result, a majority of approaches don't consider the server's management of resource allocation. Thus research has focused on designing efficient and strategyproof auction mechanisms.

Work by Kumar et al. (2017) provides a systematic study of double auction mechanisms that are suitable for a range of distributed systems like Grid computing, Cloud computing, Inter-Cloud systems. The work reviewed 21 different proposed auction mechanisms over a range of important auction properties like Economic

Efficiency, Incentive Compatibility and Budget-Balance. In a majority of the proposed auction mechanisms, truthfulness was only considered for the user, thus a Truthful Multi-Unit Double auction mechanism was presented as such that both users and server should act truthfully.

Deep Reinforcement Learning was implemented by Bingqian Du (2019) to learn resource placement and pricing in order to maximise cloud profits. Deep neural network models with Long/Short Term Memory units enabled state-of-the-art online cloud resource allocation and task pricing algorithms that had significantly better results than traditionally online mechanisms with that profit made and number of users accepted. The system considered both the pricing and placement of virtual machines in the system to maximise the profits of cloud providers through the use of Deep Deterministic Policy Gradient (Silver et al., 2014) to train agents. Users would request a type of virtual machine from the system that a server would allocate to a user where the price and placement of the virtual machine with a known fixed resource requirements where the price and server to be allocated to chosen by a neural network agents. The deep Reinforcement Learning models were trained using real-world cloud workloads and achieved significantly high profit even in worst-case scenarios.

Some approaches have been taken to increase flexibility within Fog Cloud Computing (Bi et al., 2019) through efficient distribution of data centers and connections to maximise social welfare. A truthful online mechanism was proposed that was incentive compatible and individually rational, to allow tasks to arrive over time by solving a integer programming optimisation problem. Similar research in Farhadi et al. (2019), considers the placement of code/data needed to run specific tasks over time where the demands change over time while also considering the operational costs and system stability. An approximation algorithm achieved 90% of the optimal social welfare by converting the problem to a set function optimisation problem.

Previous work proposed the novel resource allocation mechanism and optimisation problem that this project works to expand (Towers et al., 2020). The paper presents three mechanisms for the optimisation problem, one to maximise the social welfare and two auction mechanisms for self-interested users. The Greedy algorithm presented allows for quick approximation of a solution through the use of several heuristics in order to maximise the social welfare. Results found that the algorithm achieved over 90% of the optimal solution given certain heuristics compared to a fixed resource allocation solution that achieved 70%.

The algorithm has polynomial time complexity with a lower bound of $\frac{1}{n}$ however in practice achieves significantly better results.
The work also presented a novel decentralised iterative auction mechanism inspired by the VCG mechanism (Vickrey, 1961; Clarke, 1971; Groves, 1973) in order to iteratively increase a task's price. As a result, a task doesn't reveal its private task value that is believed to be particularly interest within military tactical network where countries do not need to reveal the important of a task to another coalition country but allow them to run the task. The auction mechanism achieves over 90% of the optimal solution due to iteratively solving of a specialised server optimisation problem. The third algorithm is an implementation of a single parameter auction (Nisan et al., 2007) using the greedy algorithm to find the critical value for each task. Using the greedy algorithm with a monotonic value density function means the auction is incentive compatible and inherits the social welfare performance and polynomial time complexity of the greedy mechanism.

2.2 Reinforcement Learning

Computer scientists have always been interested in comparing computers against humans (Turing, 1950) with a key characteristic of humans is the ability to learn from experience. An ability Computers must have this programed in and so researchers have invented a variety of ways for computers to do this. These are broadly grouped into three categories: Supervised, Unsupervised and Reinforcement Learning. Supervised learning uses inputs that are mapped to known outputs, an example is image classifications. Unsupervised learning in comparison doesn't have a known output for a set of inputs, instead algorithms try to find links between similar data, for example data clustering.

However both of these techniques are not applicable for cases where agents must interact with an environment making a series of actions that result in rewards over time. Algorithms designed for these problems fall into the category of Reinforcement Learning where agents select actions based on an environment state that generate the result state and reward from the action as shown in figure 2.1.

Q-learning (Watkins and Dayan, 1992) is a learning method used for estimating an action-value function, called the Q value, that forms the basis of modern

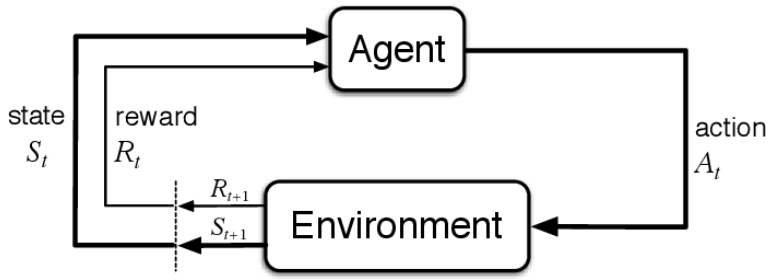


FIGURE 2.1: Reinforcement learning model (Source: Sutton and Barto (2018))

Reinforcement Learning algorithms. The Q value represents the estimated discounted reward in the future given an action in a particular state. Equation (2.1) gives a mathematically description where rewards in the future are discounted. A recursive version of this equation can be formulated as equation (2.2) where the next state is the max Q value of the next state-action. Agents can be trained to approximate the Q value using equation (2.3) through the used of a table of state action pairs.

$$Q(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \cdots] \tag{2.1}$$

$$Q(s_t, a_t) = r_t + \gamma \max_a Q(s_{t+1}, a) \tag{2.2}$$

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{2.3}$$

However the curse of dimensionally was found to be a major problem for using Q learning as the number of states or actions increased, the table grows exponentially in length as well. This made the method impractical for problems with large state spaces due to both the table size and the required training time for agents.

Therefore function approximators are used to circumvent this problem, typically done using neural networks due to their ability to approximate any function (Csáji, 2001) and to be trained using gradient descent. Work by Mnih et al. (2013), implemented a deep Q network (DQN) to achieve state-of-the-art performance in six of seven games tested as part of the Atari game engine, with three of these scores being superhuman. This was done through using of two different neural network, a model and target network in which the target network is slowly updated by the model network to act as a slowly updating target Q value. An experience replay buffer was also implemented to enable the agent to learn from previous actions.
Follow up work by Mnih et al. (2015) found that with no modifications to the

hyperparameters, neural network or training method; state-of-the-art results were achieved in almost all 49 Atari games with superhuman results in 29 of these games. The work showed that deep neural networks could be trained through observing just the raw game pixels and of the game score over time to achieve scores better than those humanly possibly.

Due to this research, a large number of heuristics have been proposed to the loss function (van Hasselt et al., 2015), network architecture (Wang et al., 2015), experience replay buffer (Schaul et al., 2015) and more to improve the algorithm. A combined agent (Hessel et al., 2017) applying a range of heuristics enabling it to achieved over 200% of the original DQN algorithm in score.

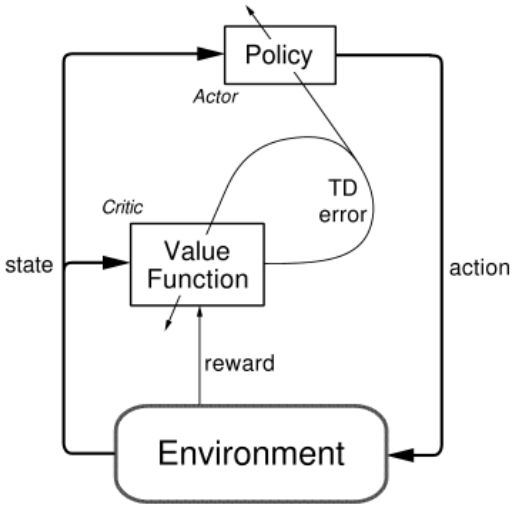


FIGURE 2.2: Actor Critic model (Source: Sutton and Barto (2018))

Policy gradient agents, shown in figure 2.2, using the base of Q-learning separate the action selection policy from the Q-value function known as the critic. In deep Q networks, actions are selected on the maximum Q-value for all of the actions, however this requires actions to be discretized. By splitting the actions from the Q values, an actor chooses an action based on the environment state allowing for both discrete and continuous action space to be utilised. The critic network is trained almost identically to the DQN agent except fro the use of a soft target update. While the actor network is trained through gradient ascent and the critic evaluation to increase the value. This has the advantage of the action policy being trained directly compare to DQN agents where used an epsilon greedy action selection policy however agents can also get stuck in local maxima's. As a result policy gradient has been used to master the game of Go (Silver et al., 2017) and achieve top 1% in both Dota 2 (OpenAI, 2018) and Starcraft 2 (Vinyals et al., 2017) video games.

Chapter 3

Optimising resource allocation in MEC

In chapter 1, the problem that this project aims to address was outlined along with a short description of the proposed solution. This chapter builds upon that, giving a formal mathematical model for the problem in section 3.1. Section 3.2 proposes an auction mechanism in order to pay servers for their resources in order to deal with self-interested users and as server are normally payed for use of their services.
Using the optimisation problem and auction mechanism from the previous sections, agents for both auction and resource allocation are proposed, in section 3.3, that learns together to maximise a server's profits over time.

3.1 Resource allocation optimisation problem

Using the flexible resource principle, the time taken for a operation to occur, e.g. loading of data, computing a program and sending back results, etc, is proportional to the amount of resources allocated to complete the operation. A modified version of a resource allocation optimisation model can be designed by building upon the formulation in Towers et al. (2020).

A sketch of the whole system is shown in figure 3.1. The system is assumed to contain a set of $I = \{1, 2, \ldots, |I|\}$ servers that are heterogeneous in all characteristics. Each server has a fixed resource capacity: storage for the code/data needed to run a task (e.g., measured in GB), computation capacity in terms of CPU cycles

per time interval (e.g., measured in GHz), and communication bandwidth to receive the data and to send back the results of the task after execution (e.g., measured in Mbit/s). An example of task could be the analyse of CCTV cameras with 2GB of data, 5GHz of CPU cycles and 5MB of results. The resources for server i are denoted: S_i for storage capacity, W_i for computation capacity, and R_i for communication capacity. The system occurs over discrete time steps that are defined as the set $T = \{1, 2, \ldots, |T|\}$.

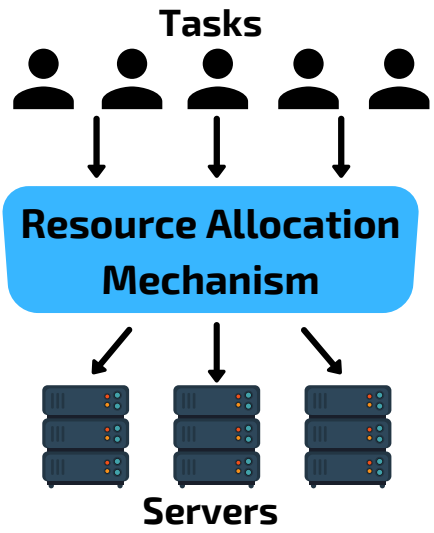


FIGURE 3.1: System model

The system is also assumed to contain a set of $J = \{1, 2, \ldots, |J|\}$ heterogeneous tasks that require services from one of the servers in set I. To run any of these tasks on a server requires storing the appropriate code/data on the same server. This could be, for example, a set of images, videos or Convolutional neural network layers used in identification tasks.

The storage size of task j is denoted as s_j with the rate at which the program is transferred to a server at time t being $s'_{j,t}$. For a task to be computed successfully, it must fetch and execute instructions on a CPU. We consider the total number of CPU cycles required for the program to be w_j, where the number of CPU cycles assigned to the task at time t is $w'_{j,t}$. Finally, after the task is run and the results obtained, the latter needs to be sent back to the user. The size of the results for task j is denoted with r_j, and the rate at which they are sent back to the user is $r'_{j,t}$ on a server at time t.

The allocation of tasks to server is denoted by $x_{i,j}$ for each task, $j \in J$, and server, $i \in I$. This is constrained by equation (3.1) meaning that a task can only be allocated to a single server at any point in time.

$$\sum_{i \in I} x_{i,j} \leq 1 \qquad \forall j \in J \tag{3.1}$$

$$x_{i,j} \in \{0, 1\} \qquad \forall i \in I, j \in J \tag{3.2}$$

As the task must complete each stage in series, additional variables are required to track the progress of each task stage. $\hat{s}_{j,t}$ denotes the loading progress (equation (3.3)), $\hat{w}_{j,t}$ denotes the compute progress (equation (3.4)) and $\hat{r}_{j,t}$ denotes the sending progress (equation (3.5)) of the task. Each of these variables are updated recursively depending on the progress at the previous time step with the resources allocation. In the compute and sending progress constraints, the resources allocated are clipped due to requiring the previous stage to finish before the next stage is started. Progress is limited for each task stage to the total resources required to prevent unneeded allocation in constraints (3.6), (3.7) and (3.8).

$$\hat{s}_{j,t+1} = \hat{s}_{j,t} + s^{'}_{j,t} \qquad \forall j \in J, t \in T \tag{3.3}$$

$$\hat{w}_{j,t+1} = \hat{w}_{j,t} + w^{'}_{j,t} \lfloor \frac{\hat{s}_{j,t}}{s_j} \rfloor \qquad \forall j \in J, t \in T \tag{3.4}$$

$$\hat{r}_{j,t+1} = \hat{r}_{j,t} + r^{'}_{j,t} \lfloor \frac{\hat{w}_{j,t}}{w_j} \rfloor \qquad \forall j \in J, t \in T \tag{3.5}$$

$$\hat{s}_{j,t} \leq s_j \qquad \forall j \in J, t \in T \tag{3.6}$$

$$\hat{w}_{j,t} \leq w_j \qquad \forall j \in J, t \in T \tag{3.7}$$

$$\hat{r}_{j,t} \leq r_j \qquad \forall j \in J, t \in T \tag{3.8}$$

Every task has an auction time, denoted by a_j and a deadline, denoted by d_j. This is the time step when the task is auctioned and the last time for which the task can be completed successfully. Using this deadline constraint can be constructed such that the sending results progress is finished by at least the deadline time step (equation (3.9)).

$$\hat{r}_{j,d_j} = r_j \qquad \forall j \in J \tag{3.9}$$

As servers have limited capacity, the total resource usage for all tasks running on a server must be capped. The storage constraint (equation (3.10)) is unique as the sum of the loading progress for each task allocated to the server. While the computation capacity (equation (3.11)) is the sum of compute resources used by all of the tasks on a server i at time t and the bandwidth capacity (equation (3.12)) being less than the sum of resources used to load and send results back by all allocated tasks.

$$\sum_{j \in J} \hat{s}_{j,t} x_{i,j} \leq S_i \qquad \forall i \in I, t \in T \tag{3.10}$$

$$\sum_{j \in J} w'_{j,t} x_{i,j} \leq W_i \qquad \forall i \in I, t \in T \tag{3.11}$$

$$\sum_{j \in J} (s'_{j,t} + r'_{j,t}) x_{i,j} \leq R_i \qquad \forall i \in I, t \in T \tag{3.12}$$

3.2 Auctioning of Tasks

While the mathematically description of the problem presented in the previous section doesn't consider any auctions occurring. In real life servers are normally paid for the use of their resources through auctions as discussed in Section 2.1. However due to the modifications that this project has to make to the optimisation problems, all of the auction mechanisms discussed are incompatible due to users not requesting a fixed amount of resources. Nor can the available resources be easily computed as this is dynamic, depending on the different stages of tasks allocated to a server. Also the modifications effect the algorithms presented in Towers et al. (2020) as they assume that all of the task stages can occur concurrently. This means that a novel or modified auction mechanism must be used to deal with these changes. Due to the complexities of devising a new auction mechanism and the large corpus of research on auctions already, an outline of the most common auctions is presented in table 3.1 with their respective properties in table 3.2.

Auction type	**Description**
English auction	A traditional auction where all participants can bid on a single item with the price slowly ascending till only a single participant is left who pays the final bid price. Due to the number of rounds, this requires a large amount of communication.

Dutch auction	The reverse of the English auction, where the starting price is higher than anyone is willing to pay with the price slowly dropping till the first participant "jumps in". This can result in sub-optimal pricing if the starting price is not high enough or possibly a large number of rounds until anyone bids. Plus due the auctions occurring over the internet, latency can have a large effect on the winner.
Japanese auction	Similar to the English auction, the Japanese auction is instead over a set period of time lets bids increasing with the last bid being the winner. Because of this, there is no guarantee that the price will converge to the maximum price like the English but the auction has a fixed time length. However factors like latency can have a large effect on the winner and resulting price like the Dutch auction.
Blind auction	Also known as a First-price sealed-bid auction, all participants submit a single secret bid for an item with the highest bid winning. As a result, no dominant strategy exist as an agent would wish to bid only a small amount more than the next highest price in order to not overpay for an item. But due to there being only a single round of biding, latency doesn't affect the winner and can occur over a fixed period of time.
Vickrey auction (Vickrey, 1961)	Also known as a second-price sealed-bid auction, participants each submit a single secret bid for an item with the highest bid winning like the blind auction. However the winner only pays the price of the second highest bid. Because of this, the dominant strategy (referred to as incentive compatibility) for an agent to bid its true value as even if the bid is much higher than all other participants its doesn't matter as they pay the minimum required for them to win.

TABLE 3.1: Descriptions of feasible auctions for the project: English, Dutch, Japanese, Blind and Vickrey auction

The auction properties that this project considers most important in the auction are if it a fixed time length as it is an Online Auction and Incentive Compatibility such that an optimal strategy actually exists. Because of these two properties,

Auction	Incentive compatible	Fixed time length
English	False	False
Dutch	False	False
Japanese	False	True
Blind	False	True
Vickrey	True	True

TABLE 3.2: Properties of the auctions described in Table 3.1

the Vickrey auction (Vickrey, 1961) has been chosen. An additional advantage of incentive compatibility, is that agents don't need to learn how to outbid another agents, they only needs learn to accurately evaluate each task allowing agents to learn purely through self-play.

However a modification must be made to the auction due to servers generate the prices for tasks rather than task suggesting a price to the servers. Because of this, the auction is reversed, such that the bid with the minimum price wins the task instead of the maximum price. The auction therefore works by allowing all servers to submit their bids for a task with the winner being the server with the lowest price, but as this is a Vickrey auction, the server actually gains second lowest price.

3.3 Server agents

Using the optimisation formulation and auction mechanism from the previous two sections, the problem can be modelled as Markov Decision Process (Bellman, 1957) in figure 3.2. This has the advantage of separating out the auction and resource allocation parts of the problem with separate agents acting for each. Subsection 3.3.1 and 3.3.2 proposes agents for each of the auction and resource allocation environments respectively.

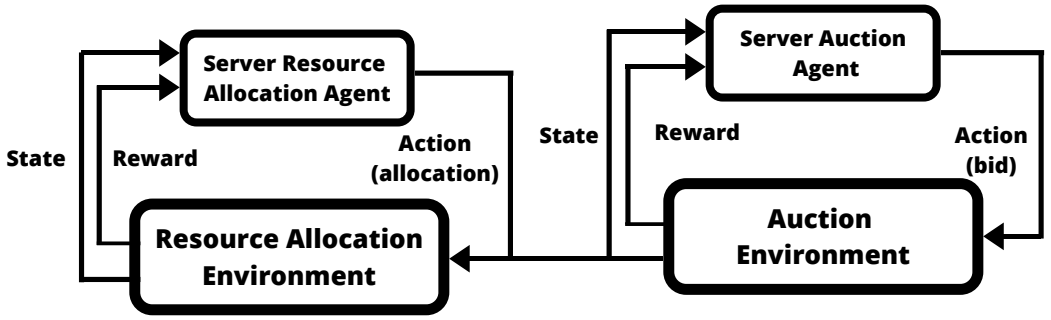


FIGURE 3.2: Markov Decision process system model

Due to the exploratory nature of the project, a range of reinforcement learning algorithm and neural network architectures will be implemented and compared to evaluate the effectiveness of each. These are outlined in Table 3.3 for proposed algorithms and Table 3.4 for networks layers.

Algorithm	**Description**
Dqn (Mnih et al., 2015)	A standard deep Q network (Dqn) agent that discretizes the action space with a target neural network and experience replay buffer.
Double Dqn (van Hasselt et al., 2015)	A heuristic for the Dqn that modifies the loss function using the target network actions index but the model network value.
Dueling Dqn (Wang et al., 2015)	A heuristic for the Dqn that modifies the network to separates the state value and action advantages that can help understand the environment.
Categorical Dqn (Bellemare et al., 2017)	DQN represents the Q value as a scalar value, the Categorical DQN agent outputs probability distribution over values which is helpful for environments that are stochastic.
Deep Deterministic Policy Gradient (Silver et al., 2014)	Deep Deterministic Policy Gradient (DDPG) allows for continuous actions space, compared to DQN agents that must discretize the action space. This is done through the use of an actor and critic network with target networks for each that are updated with a soft target update method.

Twin delay DDPG (Fujimoto et al., 2018)	Like the Double Dueling DQN agents, Twin deplay DDPG (TD3) includes a couple heuristics for the DDPG algorithm. A critic twin is used to prevent the actor network from tricking the critic network in evaluating a state's Q value. Another heuristic is the delaying the updates for actor network compared to the critic network to allow the critic to out learn the actor to prevent being tricked.
Td3 Central critic	As there are multiple agents working together, the critic used for all of the agents can be same, this is inspired by Lowe et al. (2017) which the critic only evaluates the agents private observation not the complete information.

TABLE 3.3: Table of the Reinforcement Learning algorithms

Neural Network	**Description**
Artificial neural networks (McCulloch and Pitts, 1943)	Originally developed as a theoretically approximation for the brain, it was found that for networks with at least one hidden layer, networks could approximate any function (Csáji, 2001). This made neural networks extremely helpful for cases where it would normally be too difficult for a human to specify the exact function, neural networks can be used to find a close approximation to the true function through gradient descent.
Recurrent neural network (Elman, 1990)	A major weakness of artificial neural networks is its use of a fixed number of inputs and outputs making it unusable with text, sound or video where previous inputs are important for understanding a current input. Therefore recurrent neural network's (RNN) extend artificial neural networks to allow for connections to previous neurons to "pass on" information. However this was found to struggle from vanishing or exploding gradient during training such that gradients would tended either to zero or infinity over long sequences.

Long/Short Term Memory (Hochreiter and Schmidhuber, 1997)	Long/Short Term memory (LSTM) aims to remedy RNNs problem of vanishing and exploding gradient problems. This is done by using forget gates that determine how much information from the last state would get, allowing for more complex information to be remembered over time compared to RNNs.
Gated Recurrent unit (Chung et al., 2014)	Gated recurrent unit (GRU) are very similar to LSTMs, except for the use of different wiring mechanisms with one less gate, an update gate instead of two forget gates. These changes mean that GRUs run faster and are easier to code than LSTMs. However are not as expressive meaning that less complex functions can be encoded.
Bidirectional (Schuster and Paliwal, 1997)	With RNNs, LSTMs and GRUs, inputs are passed through forward however in understanding an input the subsequent inputs are also required. Bidirectional neural network fixed this by passes in an input twice, once forwards normally and then a second time in reverse. This allows networks to understand the context around an input from both before it and after it.
Neural Turing Machine (Graves et al., 2014)	Inspired by computers, Neural Turing Machines build on long/short term memory by using an external memory module instead of memory being inbuilt to the network. This allows for external observers to understand what is going on much better than other networks due to their black-box nature.
Differentiable neural computer (Graves et al., 2016)	An expansion to the Neural Turing Machine that allows the memory module to be scalable in size allowing for additional memory to be added if needed.
Sequence to Sequence networks (Sutskever et al., 2014)	All networks considered above allow for sequences to be passed in while outputting a single output vector. Sequence to sequence network utilise two sub-networks; an encoder and decoder. The encoder takes a sequence that is encoded which is outputted to the decoder that then outputs another sequence by continually passing in the decoders last input.

TABLE 3.4: Neural network layer descriptions

3.3.1 Auction agents

Traditionally pricing mechanisms (Al-Roomi et al., 2013) rely on mixture of metrics: resource availability, resource demand, quality of service, task resource allocation quantity, etc to determine a price. However these values are difficult to approximate during the auction for this project. So, due to the complexity of deriving this function, reinforcement learning will be used to learn an optimal policy for a server to maximise its profits over time.

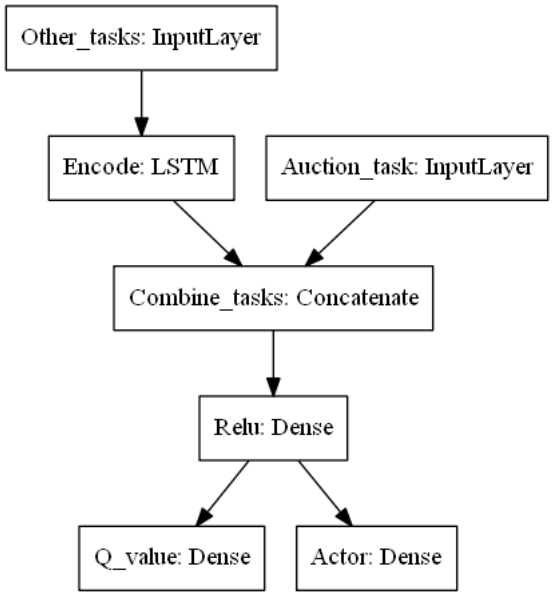


FIGURE 3.3: Task pricing network architecture

The network used for the auction must take into account a server's current tasks and the attributes of the task being auctioned. Because of this, a recurrent network must be used like the RNN, GRU, LSTM and Bidirectional due to the number of tasks allocated to a server is unknown. For the DDPG, a single ReLU output will be used while a linear Q value output for DQN agent must be used as shown in Figure 3.3.

3.3.2 Resource allocation agents

When a new task is allocated to the server or a task completes a stage, server resource need to be redistributed to the task. While the problem of how to allocation resources isn't as complex as the agent pricing in section 3.3.1, heuristics are not obvious due knowing how to balance resources usage between all allocated

tasks. Because of this, reinforcement learning agents are proposed to learn this with two different network structures with a heuristic to simplify the optimal function for agents.

The heuristic is proposed for the network output, such that the network doesn't output the raw resources allocated for a task but rather a weighted value for the resources needing to allocated for the task. This has the advantage of being a simpler function to approximate for agents as it is a single positive value but with a similar expressiveness as an exact resource usage function due to the other values be able to known by the server. This also avoids the problem of the network either over allocating the resources to tasks or severely under allocating resources.

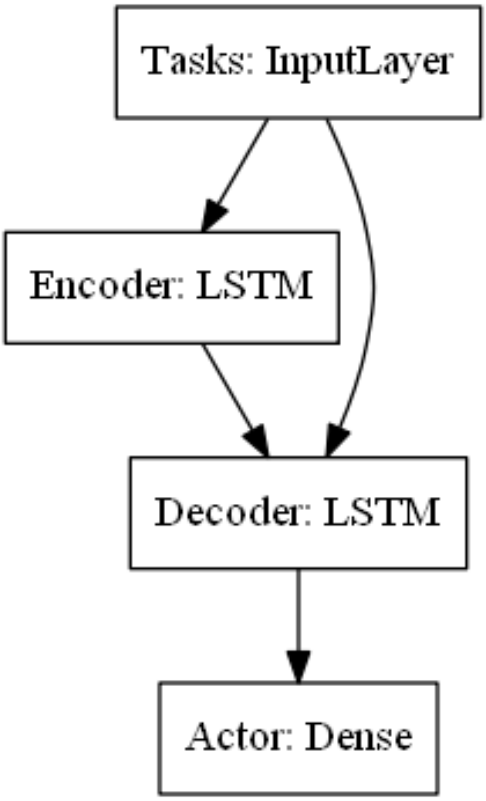


FIGURE 3.4: Multi-task Seq2Seq actor network architecture

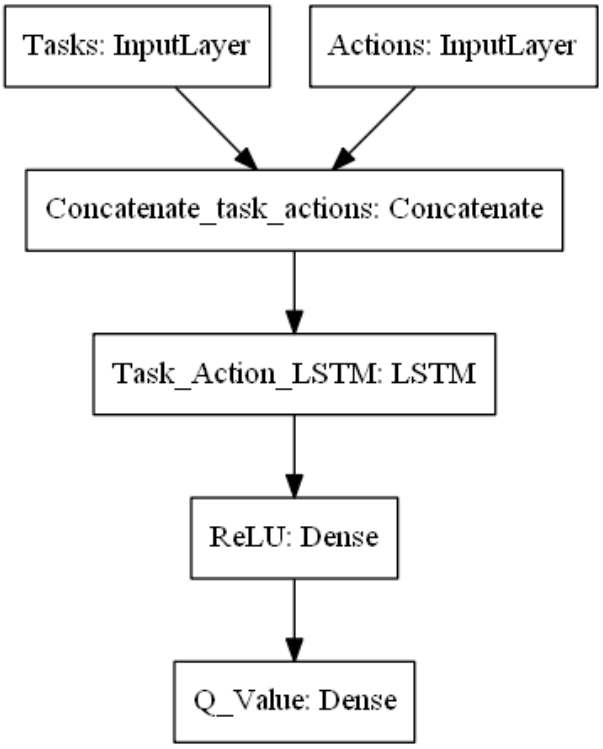


FIGURE 3.5: Multi-task Seq2Seq critic network architecture

Due to allocating resources to a single task requires being aware of the resource requirements of other tasks, this effects the network input shape. Two different network structures will be compared, one using a Seq2Seq network such all tasks are passed in to be encoded with the decode having all of the tasks passed

in again generating a sequence of task weightings. This network architecture is shown in figure 3.4 for the actor network and figure 3.5 for the critic network. The other network is similar such that a single task is weighted at a time through passing the single task and the other allocated tasks through the network as shown in figure 3.6. But this requires for each allocated task for the task to passed through the network however the network can be trained with both deep Q learning and policy gradient algorithms while the Seq2Seq network must be trained through policy gradient algorithms.

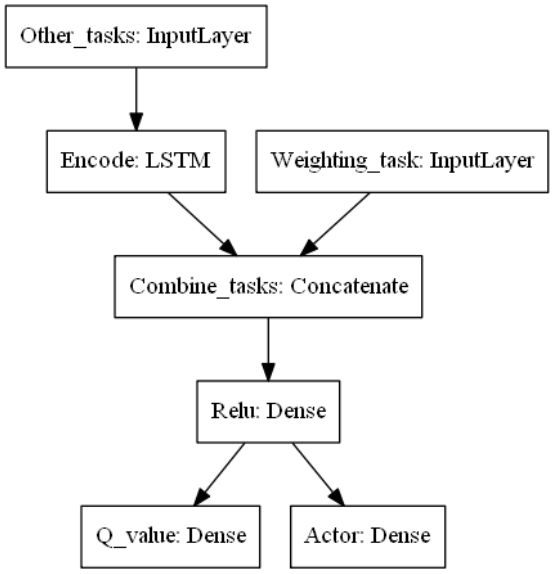


FIGURE 3.6: Single task resource weighting network architecture

Chapter 4

Implementing Flexible Resource Allocation Environment and Server Agents

In order to implement the solution from chapter **??**, a Mobile Edge Cloud (MEC) computing network must be simulated for both training and evaluation of agents. Due to the impractically of physically setting up such a network and to train agents offline and in parallel. This chapter splits the implementation into three sections: simulation of MEC networks (section 4.1), server auction and resource allocation agents (section **??**) and the training of agents (section 4.3).

The implementation discussed below as written in Python and available to download from Github [1].

4.1 Simulating MEC networks

While the aim of the environment is to accurately simulate MEC servers, the implementation of the environment must allow agents to interact and train on the environment efficiently. Therefore it has been implemented as an OpenAI gym (Brockman et al., 2016), the de facto standard for implementing reinforcement learning environments by researchers. However the standard specification must be modified due to the problem being multi-agent and multi-step.

[1] https://github.com/stringtheorys/Online-Flexible-Resource-Allocation

An example of running the environment is in listing 4.1.

```
# Load the environment with a setting
env = OnlineFlexibleResourceAllocationEnv('settings.env')

# Generate the environment state
server_state = env.reset()

for _ in range(1000):
    # Generate actions
    if server_state.auction_task:
        actions = {
            server: auction_agent.bid(state)
            for server, state in server_state
        }
    else:
        actions = {
            server: resource_allocation_agent.weights(state)
            for server, state in server_state
        }

    # Take environment step
    server_state, reward, done, info = env.step(actions)

    # If the environment is finished then reset it
    if done:
        server_state = env.reset()
```

LISTING 4.1: Example code for running the environment

4.1.1 Weighted server resource allocation

A particular complication of the environment is to distribute server resources due to the fact that the resource allocation agents provide a resource weighting rather than the actual task resource usage. Because of this, a novel algorithm was implemented to convert the weighting to the actual resources of each task.

To allocate the computational resources is relatively simple compared to allocating resources for both storage and bandwidth. For computational resources, the algorithm checks first if the weighted resources is greater than the quantity required for the task to finish the compute stage of the task. If this is true then the resources needed for the task to complete the compute stage are allocated. However, this also means that the weighted resources available for each task is increased due to a task not using all of its resources. Because of this, the checks are repeated until no task can be finished with its weighted resource within this time step. For the remaining task, they are just allocated their weighted compute resources.

For allocating storage and bandwidth, this is more difficult due to the fact that when the server is still loading the task, the server is allocating both storage and bandwidth resources while also allocating bandwidth resources to tasks to send results back to users. Because of this, a tension exists between allocating bandwidth and storage resources for all of the tasks fairly. The algorithm was therefore chosen to gives priority when allocating resources to the tasks sending results as these tasks that are more likely to finish and to not penalise the server by failing to complete the task within its deadline.

To allocate resources, a similar function to used to the one for allocating compute resources. First a check if done using the weighted bandwidth resources to see is any task sending results will be finished with the resources. This process is also repeated for the tasks loaded onto a server with the additional check that there is enough available storage for the new data to be added. For any remaining task, this process is repeated until all of the available resources are allocated in the time step. As a results, using this algorithm, the converting between weightings to the actual resources usage allows for near maximum allocation of a server's available resources.

4.2 Server auction and resource allocation agents

For the server, they require two policies, one for bidding on auction tasks and the other being allocating resources. Subsections 3.3.1 and 3.3.2 propose server agents that use any of reinforcement learning algorithms (Table 3.3) and with neural networks to learn the policy.

These policies were originally attempted to be implemented using Tf-Agent (Hafner et al., 2017) and a range of other frameworks with algorithm pre-programmed however due to numerous issues that was not possible to do. Therefore all of the algorithm have been handcrafted using tensorflow (Abadi et al., 2015), a Python module developed by Google, that gives the ability to construct neural networks, backpropagation with custom loss functions and more.

Both deep Q networks and policy gradient algorithms are based on the Q function (explained in section 2.2)) which tries to approximate the reward at the next time step. To do this requires a reward function and a next observation to compare to in order to train these agents. The agent's reward function and agent training observations are detailed in subsection 4.2.1 and 4.2.2.

A particular problem that this project encounter was with using recurrent neural network. This was because the number of inputs were not fixed which means that during training, a minibatch was not possible as most of the inputs all had different input lengths and tensorflow required all inputs to have a known, fixed length. Originally this was sidestepped by calculating the loss for each input individually then finding the mean loss and gradient to update the networks. However this was found to be computationally impractical requiring over two days to train an single agent over 500 episodes. Because of this, a solution was found by padding all the inputs to be the same size using the tensorflow preprocessing module with the sequence.pad_sequence function. As a result, training became significantly faster making large scale testing practical within a more reasonable time period of 16 hours for 600 episodes.

4.2.1 Agent Rewards Functions

As explained in the background review for reinforcement learning (section 2.2), the Q values is the estimated discounted reward in the future for an action given a particular state. Therefore the rewards that an agent receives for taking an action is extremely important to enable the agent to learn a predictable reward function. This problem of complex reward functions are a known problem for DQN agent to deal with (Mnih et al., 2013) that policy gradients agents can deal with this better due its ability directly learning the action policy (Sutton and Barto, 2018).

For the auction, the reward is based on the winning price of the task which is award for the winning. If the task fails, the reward is instead multiplied by a negative constant in order to discourage the auction agent from bidding on tasks that wouldn't be able to complete. As the price of zero is treated as a non-bid in the auction, the agent gets a reward of zero in order to not penalise the agent. However, if the agent does bid on a task however doesn't win, the agent's reward is set just below zero at -0.05 as a way of encouraging the agent to change their bid but not large enough to force it to do so.
This reward is awarded at the time step of the auction instead of when the task fails or is completed as this makes the function harder to learn as the auction agent has no control or observations over the resource allocation for a task at that point.

For resource allocation, the reward function is much simpler than the auction agent's reward function, as it only needs to consider the task being weighted at the time and rewards from other tasks allocated at the same time step. This is because a task must consider its actions in conjunction with the resource requirements of other allocated tasks.
For successfully finishing a task, the reward is 1 while the reward if the task has failed is -1.5. This makes failing a task more costly than completing a task. But when a task's action is not under consideration, this reward is multiplied by 0.4 as while this rewards impact the task, their value is not as impactful as the reward for the action on a particular task. These rewards don't consider the price payed for the task instead valuing each task equally with the aim of finish all tasks not just the valuable ones. Using this information, the reward function is simply the sum of the rewards of the finished tasks in the next time step.

4.2.2 Agent Training Observations

In order to update both DQN and DDPG critic networks, the Q learning equation is used (equation (2.3)) requires the next observation to evaluate the next Q value. However the next observation for an agent is not clear as shown in figure 4.1. For both the auction and resource allocation agents, there is an unknown number of actions taken by the other agents before its next observation is known.

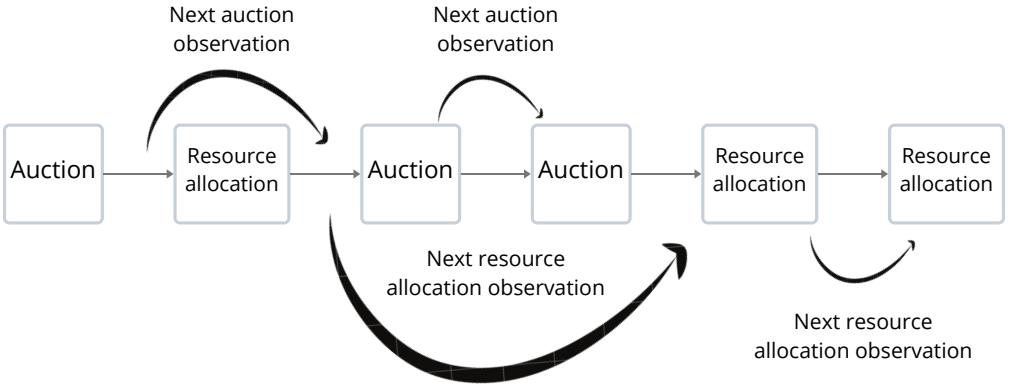


FIGURE 4.1: Environment server agents observations

For the resource allocation agent, a trick is implemented such that the next observation for the agent is not the actual next resource allocation observation as shown in figure 4.1 but a generated observation from the resulting server

state due to agent's actions. This is identical to the last case in the figure where no auction occurs between resource allocation steps. Because of this, the resource allocation Q value is able to approximate the reward for the results state of its actions directly making it appear to the agent during training that there is no auction steps between observations.

For the auction agent, as the agent observations require an auction task to select an action (the task bidding price). A trick like the one implemented for the resource allocation agent can't be implemented. Therefore during training, each server's last observation is recorded, such that when the next auction occurs, this new task observation can be used as the server's next observation. This is a suboptimal solution for the agent as the next observation has an unknown number of resource allocation steps resulting in changes to the current allocated tasks. A possible solution that has not been implement in this project is n step prediction (Sutton, 1988). Where the agent doesn't predict the Q value of the next environment, but the value in n steps time. This approach could help reduce the amount of randomness in the server's observations and improve bidding performance.

4.3 Training agents

The first section of this chapter allowed for simulating MEC servers (section 4.1) as a reinforcement learning environment. While the second section implements auction and resource allocation agents that can interact with the proposed environment in order to allow for the training of agents using a range of algorithms outlined in Table 3.3.

Neural networks, the basis of the reinforcement learning agents implemented, often require huge amounts of data and high powered GPUs to run efficiently. Because of this, Iridis 5, University of Southampton's supercomputer was utilised with GTX1050 GPUs to train these agents for long periods of time and en mass. During training, for each episode a random environment was generated from a list of possible settings so that the agents would be allocated to random servers. The environment was run until a fixed set time step, with the agent observation being added to their replay buffers after each actions that were chosen epsilon greedily.

After every 5 episodes, the agents would be evaluated using a set of environment that were pre-generated and saved at the beginning of training. This allowed the same environments to be used over training in order to have a consistent basis in order to compare the agents over time.

To ensure that agents explored the state space as much as possible some of the actions were taken randomly to allow the agents to ”explore”. This is a key problem within Reinforcement learning of Exploration vs Exploitation dilemma (Sutton and Barto, 2018). For the Dqn agents, including Double Dqn, Dueling Dqn and Categorical Dqn, epsilon greedy actions were taken such that epsilon percent of the actions taken were random with epsilon initially set at 1 and ending at 0.1 over 100 thousand actions. For the Ddpg and Td3 agents, originally the agents used Gaussian distribution to generate actions which has been used in Silver et al. (2014). However this was found to be ineffective, due to half of the actions added a negative value to the action while the action space is of positive number. Therefore Gamma distribution (Stacy, 1962) was used with a constant shape parameter of 1 with the scale parameter changing over time from 4.5 to 0.5 over 100k actions.

A table of agent hyperparameters used in training can be found in Appendix C.

Chapter 5

Testing and evaluation

Using the implemented solution from Chapter 4 to test and evaluate its effectiveness, both functional unit tests and agent training evaluation have been designed. To confirm that the environment and agents implemented in the previous chapter (section 4.2) works as intended, unit testing has been added that detailed in Section 5.1. While to evaluate the effectiveness of the solution in Chapter 3, a range of metric have been measured during training in order to test and compare implemented agents, neural network architectures and training parameters. These results are explained in Section 5.2.

5.1 Functional testing

To confirm that the implementation of the agents and environment correctly, PyTest a module within Python has been used to design to check functions. These tests are split into three families: agent, environment and training that are explained in their respective tables 5.1, 5.2 and 5.3.

Test name	**Description**
Building agents	Constructs all of the agents with any arguments to confirm agents can accept of all its attributes due to multi-inheritance.
Saving agents	Confirms that agents can successfully save their neural networks and can successfully load the network again which is equal to the original agent's neural network.

Agent actions	Confirms that all agents can generate valid actions for both bidding and weighting of tasks during both training and evaluation.
Gin config file	Gin is used to set the arguments used during training, so to confirm that the file can be successfully parsed.
Building networks	Constructs all of the neural networks to confirm that the network return a valid output and can accept valid inputs.
Agent epsilon policy	While training, agent randomly select actions in order to explore the state space. This tests that the random actions selected are valid and reduce a valid linear rate.

TABLE 5.1: Table of test functions of the agents

Testing name	**Description**
Saving and loading an environment	The environment allows for it to be saved to a file at its current state, server allocations and future task auctions. This tests that the environment can save and reload an identical environment successfully.
Loading environment settings	Tests that environment settings can be loaded correctly generating a new random environment.
Random action environment steps	Tests that inputs to the auction and resource allocation steps work, random actions are generated to check for environment edge cases.
Auction step	To confirm the Vickrey auction mechanism is completely implemented, a range of all edges cases are tested to confirm that right price and server that the task is allocated.
Resource allocation step	To confirm that servers allocate their resources correct given some inputs given all of the edge cases.
Allocation of computational resources	Checks that the server correctly allocates computational resources to allocated tasks.
Allocation of storage and bandwidth resources	Checks that the server correctly allocates storage and bandwidth resources to allocated tasks.

Allocation of all resources	Checks that resources are allocated by the server correctly for all of the resources.

TABLE 5.2: Table of test functions of the environment

Testing name	**Description**
Task pricing training	Tests that the task pricing reinforcement learning agents can correctly learning and train from different auction observations.
Resource allocation training	Tests that resource allocation reinforcement learning agents can correctly learn and train from different resource allocation observations.
Agent evaluation	Tests that the agent evaluation function during training correctly captures the correct information due to the actions taken.
Agent training	Tests that agents can be correctly trained over an environment with different actions and observations.
Random actions training	Tests random actions agent to quickly using the environment training function to confirm that the function work as intended.

TABLE 5.3: Table of test functions of agent training

5.2 Agent evaluation

In order to compare the implemented agents from Chapter 4, a range of metric are recorded each time the agents are evaluated during training. These metrics are: number of failed tasks, number of completed tasks, percentage of tasks attempted and a histogram of actions taken which together are used to compare the performance between different agents. These agents are compared in Subsection 5.2.1 to fixed resource heuristics in order to evaluate the effectiveness of the proposed optimisation problem compared to the fixed resource allocation optimisation problem.
The evaluation also analyses three different families of solutions that agents could take: environment settings and agent num, algorithms and network architecture that are analysed in Subsections 5.2.2 , 5.2.3 and 5.2.4 respectively.

5.2.1 Fixed resource Heuristics

5.2.2 Environment and Agent number training

The analysis of the different Reinforcement Learning algorithms and neural network architectures in Subsection 5.2.3 and 5.2.4 respectively assume two qualities that this subsection analyses. These are that the training and evaluation environments used and the number of agents used during training.

As there are huge ranges of possible environment settings that agents could be trained, investigating agent environment generality is a challenge and an important measure within machine learning. This is of particular importance for this work, as in real-life, the environment that agents experiences can be unpredictable and different from those trained on. This means that agents should learn to generalise and not overfit to particular environments used during training.

The other assume quality is due an advantage of the Vickrey auction over alternative auctions, explored in Section 3.2, that is it's Incentive Compatibility. This auction property means that the dominant strategy for all agents is to bid truthfully, that is the agent's true evaluation of the task. Therefore due to agents not needing to learn to "out bid" each other, this allows for the possibility to learn through self-play.

Therefore this subsection compare the results of agents that are trained on a single environment setting and those trained on multiple environment settings and when multiple or single agents are trained together. These are compared using a set of pre-generated environments, 5 from the single environment setting and 20 from the multiple environment settings to evaluate agents.

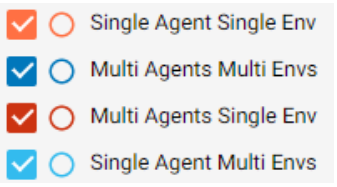


FIGURE 5.1: Environment and number of Agents Legend

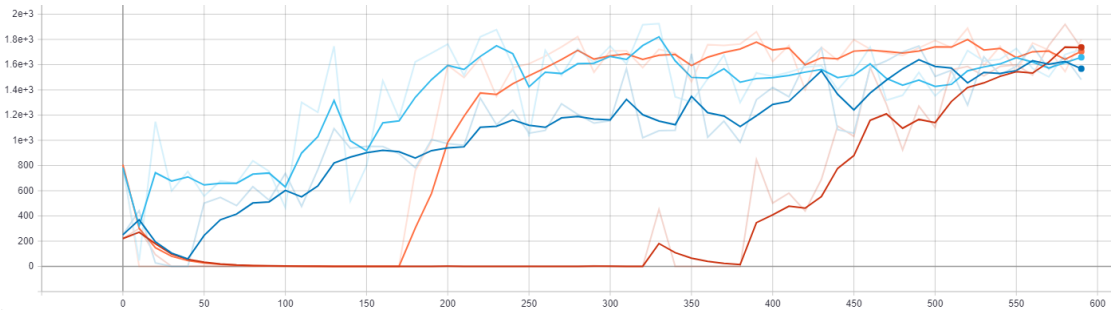


FIGURE 5.2: Environment and number of Agents - Number of completed tasks (Multi-env settings)

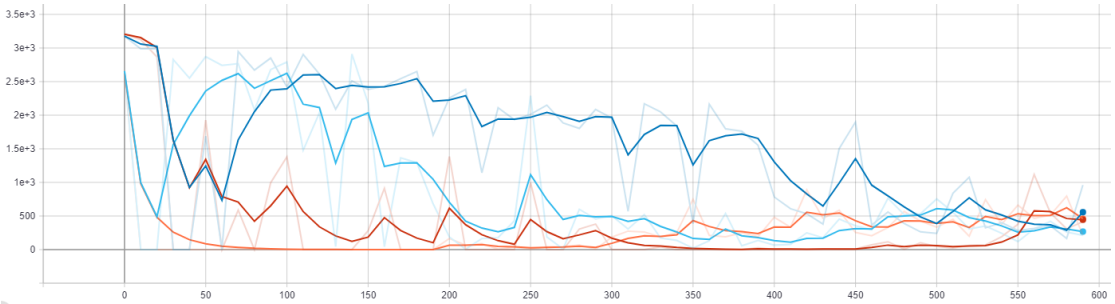


FIGURE 5.3: Environment and number of Agents - Number of failed tasks (Multi-env settings)

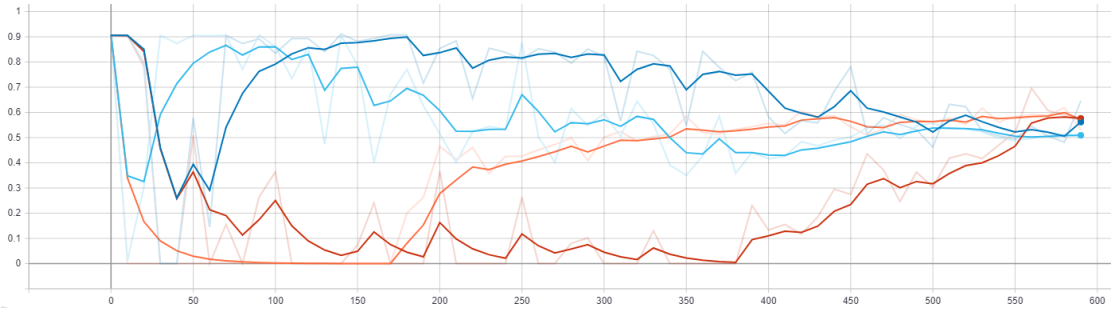


FIGURE 5.4: Environment and number of Agents - Percentage of tasks attempted (Multi-env settings)

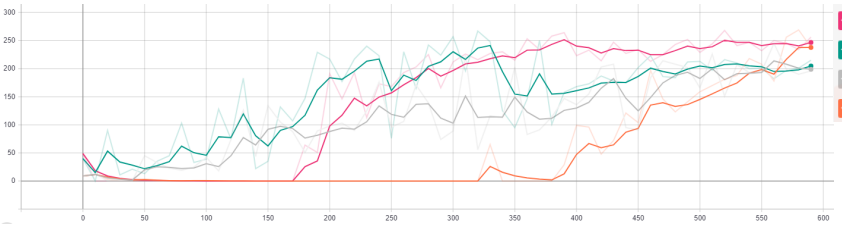


FIGURE 5.5: Environment and number of Agents - Number of completed tasks (Single env settings)

FIGURE 5.6: Single environment results legend

Figure 5.2 shows that for the agents trained using only
a single environment, took over 150 episodes for the single agent and 350 episodes for the multiple agents to achieve over 25% of the final total tasks allocated. This is in comparison to both the single and multiple agents that were trained with multiple environments that were able to achieve better results faster than the single environment agents.
Surprisingly, the number of tasks allocated by all agents after 600 episodes are within 6% of each other. This being despite the single environment agents not being trained on the multi-environment settings that it is being evaluated on. This means agents are able to generalised efficiently to unknown environments however this is difficult to confirm in real-life due to the multi-environment trained are only a small subset of the possible environment will experience in real-life.

The agents were simultaneously evaluated on the single environment setting with the single environment agents achieving 10% higher than the multiple environment agents. This is understandable due to single environment being more ”specialised” in the single environment allowing the agent to maximise profit compared to the multi-environment agent that have learnt to maximise profit over more environments.

5.2.3 Reinforcement learning algorithm training

In table 3.3, a range of reinforcement learning algorithms implemented, to compare the effectiveness of the algorithms, each were trained. This was done using three task pricing agents with a single resource weighting on multiple environment settings for training due the reasoning given in Subsection 5.2.2. Due to time limits, the policy gradients agents were only trained for 24 hours however during this time were only about to complete 300 to 400 episodes of training while the Dqn agents were able to train in 15 hours for 600 episodes.

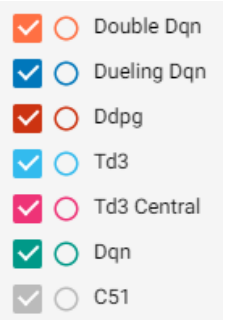


FIGURE 5.7: Reinforcement learning algorithm Legend

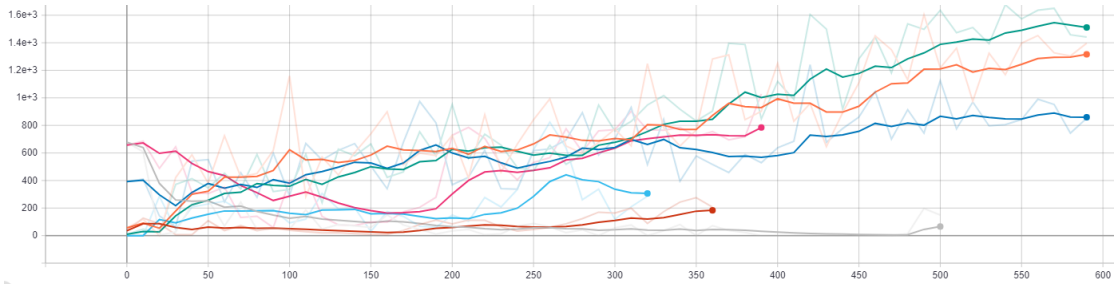


FIGURE 5.8: Reinforcement learning algorithms - Number of completed tasks

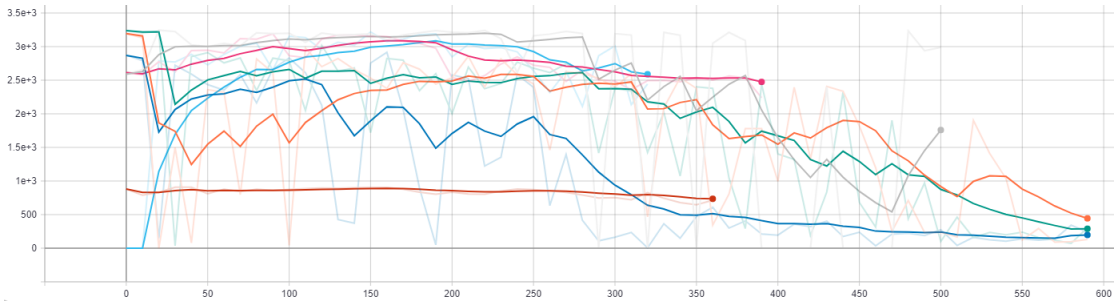


FIGURE 5.9: Reinforcement learning algorithms - Number of failed tasks

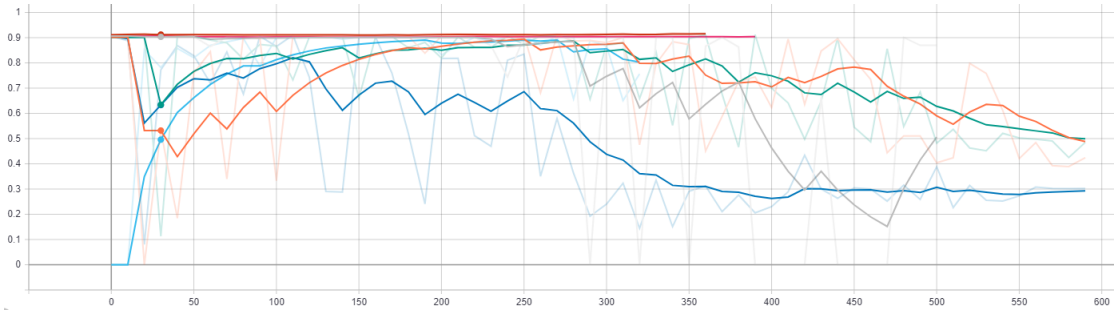


FIGURE 5.10: Reinforcement learning algorithms - Percentage of tasks attempted

The results did not match expectation due to the heuristic Dqn agents (Double Dqn and Dueling Dqn) achieved a fewer number of completed tasks (figure 5.8) than the simpler Dqn agents despite no changes being made to the agents except to the heuristic modifications. Another surprise were how badly the Ddpg agents did in comparison to the Dqn agents, particularly with regards to the percentage of tasks attempted (figure 5.10) where it appears that Ddpg agents attempted every task resulting in the agents being unable successfully complete as many tasks. However the Td3 Central critic (where the critic was the same network shared by all task pricing agents) did achieve results that are on par with the

Dqn agent but due to training time limits this can't be known if this relationship will continue. The Categorical Dqn agent implemented, while could completed much simpler reinforcement learning problem, the agent struggles with the environment however this problem is believed to be a problem with the implementation rather than the algorithm not being effective.

5.2.4 Neural network architecture training

There are a wide-range of compatible neural network architectures that agents can use, as outlined in table 3.4. To compare these architectures, four different network architectures are trained: RNN (Elman, 1990), LSTM (Hochreiter and Schmidhuber, 1997), GRU (Chung et al., 2014) and Bidirectional (Schuster and Paliwal, 1997) using an LSTM network. These networks are trained using the DQN algorithm due to it constant performance compared to the DDPG agents that can get stuck in a local maxima making the results probably inconstant as shown in the previous section. A Seq2Seq agent was also implemented as an experimental agent with a LSTM Dqn Agent for the task pricing agent and the Seq2Seq Policy Gradient agent acting only for the resource weighting.

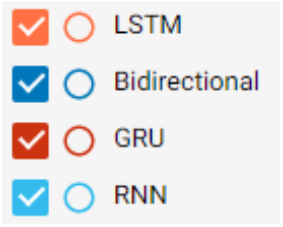


FIGURE 5.11: Network architecture legend

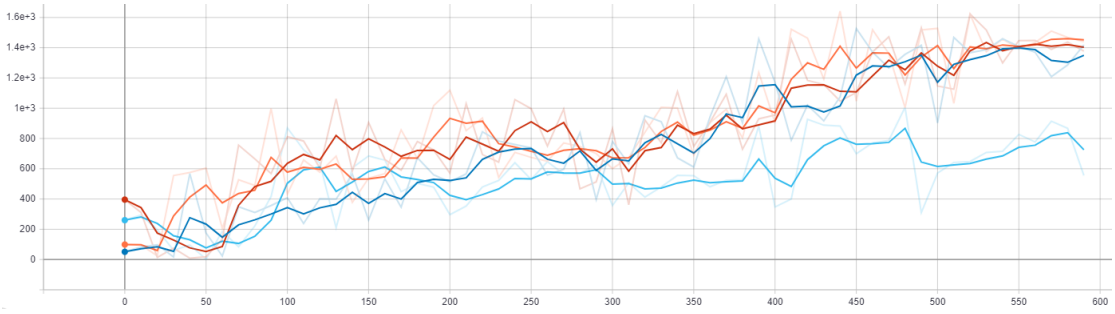


FIGURE 5.12: Network Architecture - Number of completed tasks

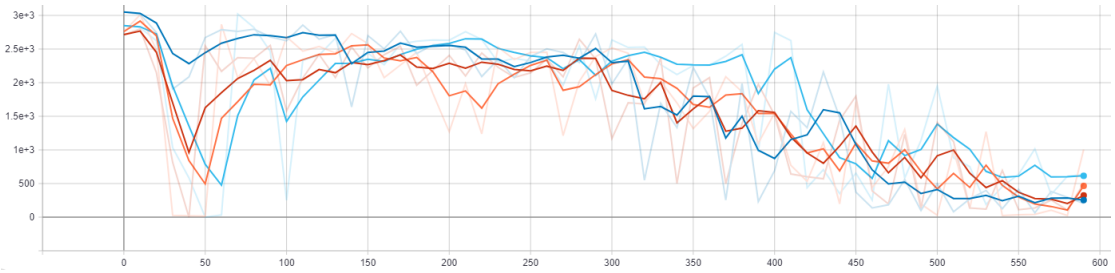


FIGURE 5.13: Network Architecture - Number of failed tasks

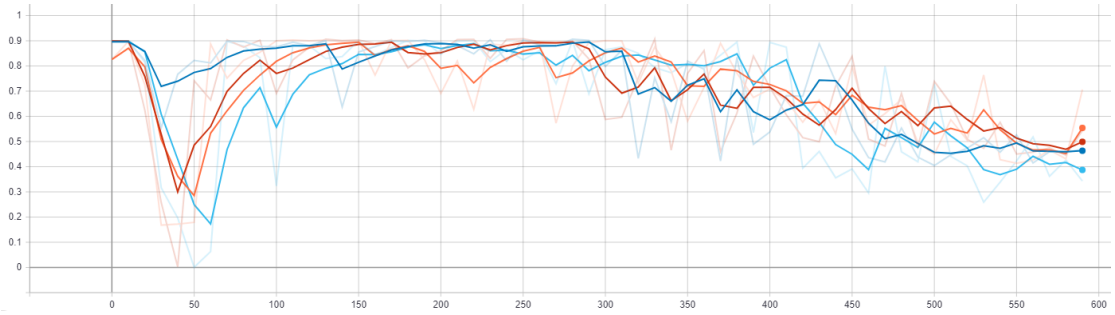


FIGURE 5.14: Network Architecture - Percent of tasks attempted

As shown in figure 5.12, the LSTM, GRU and Bidirectional networks all achieved a similar score in the number of tasks completed while the RNN network achieves 30% less. This is understandable due to the known of problem for RNNs of vanishing or exploding gradients as explained in Table 3.4. The Bidirectional neural network doesn't achieve a greater score that the other networks despite the network architecture allowing all inputs to be passed in is understandable due to the network providing a single output. Because of this, all tasks to the network are considered before any action is taken which is why the Bidirectional network didn't achieve a better score. However the Bidirectional seems more resilient to changes during training as shown in figure 5.13 and 5.14. At episode 30 to 70, all of the network's (except Bidirectional), the number of failed tasks and number of attempted tasks suddenly drop, most likely to an over evaluating to not bid on any tasks by the task pricing agents.

Chapter 6

Conclusion and future work

The aim of this project was to expand previous research to fix perceived flaws in the formulation by introducing the notion of time into the resource allocation optimisation model. As a result, a new optimisation problem was presented in Section 3.1 with an auction mechanism proposed as well to deal with self-interested users and to distribute tasks to self-interested servers. To know how to efficiently bid and allocation resources to tasks, reinforcement learning agents were proposed that aimed to learn these policies. An implementation of an MEC environment was developed and numerous reinforcement learning algorithms were used to train both auction and resource weighting agent. These agents were found to efficiently learn an optimal policy however were found to not produce optimal policies such that $\tilde{5}\%$ of all tasks were no completed within there time frame. A range of reasons why this may have occurred in Chapter 5 as a well as policy gradient agents being unable to escape local maxima prevent them from achieving results close to that of the deep Q learning agents. Therefore this project has been viewed as a success however this author believes have more research and analysis of agents is required before such agents can be implemented into real-life systems.

For future work into this project, this author believes that several additions to the agents proposed could greatly improve their performance like n-step rewards (Sutton, 1988) and distributional agents (Bellemare et al., 2017) that would improve Q value estimation within stochastic environment. An additional heuristic for the policy gradient, would be use a centralised critic (Lowe et al., 2017) that has been proposed in mix competitive-cooperative environment to help agents work together.

The word count of the Project can be found in Appendix D.

Bibliography

P. Corcoran and S. K. Datta . Mobile-edge computing and the internet of things for consumers: Extending cloud computing and services to the edge of the network. *IEEE Consumer Electronics Magazine*, 5(4), 2016.

V. Farhadi , F. Mehmeti , T. He , T. L. Porta , H. Khamfroush , S. Wang , and K. S. Chan . Service placement and request scheduling for data-intensive applications in edge clouds. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1279–1287, April 2019. . URL https://ieeexplore.ieee.org/document/8737368.

L. Guerdan , O. Apperson , and P. Calyam . Augmented resource allocation framework for disaster response coordination in mobile cloud environments. In *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, 2017.

Y. Mao , C. You , J. Zhang , K. Huang , and K. B. Letaief . A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys Tutorials*, 19(4), 2017.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

May Al-Roomi, Shaikha Al-Ebrahim, Sabika Buqrais, and Imtiaz Ahmad. Cloud computing pricing models: a survey. *International Journal of Grid and Distributed Computing*, 6(5):93–106, 2013.

Zubaida Alazawi, Omar Alani, Mohmmad B. Abdljabar, Saleh Altowaijri, and Rashid Mehmood. A smart disaster management system for future cities. In *Proceedings of the 2014 ACM International Workshop on Wireless and Mobile Technologies for Smart Cities*, WiMobCity '14, pages 1–10, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3036-7. . URL http://doi.acm.org/10.1145/2633661.2633670.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *CoRR*, abs/1707.06887, 2017. URL http://arxiv.org/abs/1707.06887.

Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6: 679–684, 1957. ISSN 0022-2518.

Fan Bi, Sebastian Stein, Enrico Gerding, Nick Jennings, and Thomas La Porta. A truthful online mechanism for resource allocation in fog computing. In A. Nayak and A. Sharma, editors, *PRICAI 2019: Trends in Artificial Intelligence. PRICAI 2019*, volume 11672, pages 363–376. Springer, Cham, August 2019. URL https://eprints.soton.ac.uk/431819/.

Zhiyi Huang Bingqian Du, Chuan Wu. Learning resource allocation and pricing for cloud profit maximization. In *The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, pages 7570–7577, 2019.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL http://arxiv.org/abs/1606.01540.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

Edward H. Clarke. Multipart pricing of public goods. *Public Choice*, 11(1):17–33, Sep 1971. ISSN 1573-7101. . URL https://doi.org/10.1007/BF01726210.

Balázs Csanád Csáji. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24:48, 2001.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2): 179–211, 1990. . URL https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018. URL http://arxiv.org/abs/1802.09477.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014. URL http://arxiv.org/abs/1410.5401.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, October 2016. ISSN 00280836. URL http://dx.doi.org/10.1038/nature20101.

Theodore Groves. Incentives in teams. *Econometrica*, 41(4):617–631, 1973. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1914085.

S.R. Gunn. Pdflatex instructions, 2001. URL http://www.ecs.soton.ac.uk/~srg/softwaretools/document/.

S.R. Gunn and C. J. Lovell. Updated templates reference 2, 2011.

Danijar Hafner, James Davidson, and Vincent Vanhoucke. Tensorflow agents: Efficient batched reinforcement learning in tensorflow. *CoRR*, abs/1709.02878, 2017. URL http://arxiv.org/abs/1709.02878.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017. URL http://arxiv.org/abs/1710.02298.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. . URL https://doi.org/10.1162/neco.1997.9.8.1735.

Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing—a key technology towards 5g. *ETSI white paper*, 11 (11):1–16, 2015.

Dinesh Kumar, Gaurav Baranwal, Zahid Raza, and Deo Prakash Vidyarthi. A systematic study of double auction mechanisms in cloud computing. *Journal of Systems and Software*, 125:234 – 255, 2017. ISSN 0164-1212. . URL http://www.sciencedirect.com/science/article/pii/S0164121216302540.

C. J. Lovell. Updated templates, 2011.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017. URL http://arxiv.org/abs/1706.02275.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943. ISSN 1522-9602. . URL https://doi.org/10.1007/BF02478259.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Kabrane Mustapha, Krit Salah-ddine, and L. Elmaimouni. Smart cities: Study and comparison of traffic light optimization in modern urban areas using artificial intelligence. *International Journal of Advanced Research in Computer Science and Software Engineering*, 8:2277–128, 02 2018. .

Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. *Algorithmic game theory*. Cambridge university press, 2007. URL https://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf.

OpenAI. Openai five. https://blog.openai.com/openai-five/, 2018.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2015. URL http://arxiv.org/abs/1511.05952. cite arxiv:1511.05952Comment: Published at ICLR 2016.

Mike Schuster and Kuldip Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45:2673 – 2681, 12 1997. .

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR. URL http://proceedings.mlr.press/v32/silver14.html.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–, October 2017. URL http://dx.doi.org/10.1038/nature24270.

G. Sreenu and M. A. Saleem Durai. Intelligent video surveillance: a review through deep learning techniques for crowd analysis. *Journal of Big Data*, 6(1):48, Jun 2019. ISSN 2196-1115. . URL https://doi.org/10.1186/s40537-019-0212-5.

E. W. Stacy. A generalization of the gamma distribution. *Ann. Math. Statist.*, 33 (3):1187–1192, 09 1962. . URL https://doi.org/10.1214/aoms/1177704481.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL http://arxiv.org/abs/1409.3215.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Mach. Learn.*, 3(1):9–44, August 1988. ISSN 0885-6125. . URL https://doi.org/10.1023/A:1022633531479.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Mark Towers, Sebastian Stein, Fidan Mehmeti, Caroline Rubeun, Tim Norman, Tom La Porta, and Geeth Demel. Auction-based mechanisms for allocating elastic resources in edge clouds. Unpublished, 2020.

Alan M Turing. Computing machinery and intelligence-am turing. *Mind*, 59 (236):433, 1950.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL http://arxiv.org/abs/1509.06461.

William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of Finance*, 16(1):8–37, 1961. ISSN 00221082, 15406261. URL http://www.jstor.org/stable/2977633.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy P. Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft II: A new challenge for reinforcement learning. *CoRR*, abs/1708.04782, 2017. URL http://arxiv.org/abs/1708.04782.

Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015. URL http://arxiv.org/abs/1511.06581.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8 (3-4):279–292, 1992.

Appendix A: Paper

This paper was been produced with the authors being myself, Dr Sebastian Stein, Professor Tim Norman from Southampton University, Dr Fidan Mehmeti, Professor Tom La Porta, Caroline Rubein from Pennsylvania State University and Dr Geeth Demel from IBM and within this project is referred to as Towers et al. (2020).

Auction-based Mechanisms for Allocating Elastic Resources in Edge Clouds

Paper #1263

ABSTRACT

Edge clouds enable computational tasks to be completed at the edge of the network, without relying on access to remote data centres. A key challenge in these settings is the limited computational resources that need to be allocated to many self-interested users. Here, existing resource allocation approaches usually assume that tasks have inelastic resource requirements (i.e., a fixed amount of compute time, bandwidth and storage), that may result in inefficient resource use due to unbalanced requirements. To address this, we propose a novel approach that takes advantage of the elastic nature of some of the resources, e.g., to trade-off computation speed with bandwidth allowing a server to execute more tasks by their deadlines. We describe this problem formally, show that it is NP-hard and then propose a scalable approximation algorithm. To deal with the self-interested nature of users, we show how to design a centralized auction that incentives truthful reporting of task requirements and values. Moreover, we propose novel auction-based decentralized approaches that are not always truthful, but that limit the information required from users and that can be adjusted to trade off convergence speed with solution quality. In extensive simulations, we show that considering the elasticity of resources leads to a gain in utility of around 20% compared to existing fixed approaches and that our novel auction-based approaches typically achieve 95% of the theoretical optimal.

KEYWORDS

Edge clouds; elastic resources; auctions

1 INTRODUCTION

In the last few years, cloud computing [2] has become a popular solution to run data-intensive applications remotely. However, in some application domains, it is not feasible to rely a remote cloud, for example when running highly delay-sensitive and computationally-intensive tasks, or when connectivity to the cloud is intermittent. To deal with such domains, *mobile edge computing* [13] has emerged as a complementary paradigm, where computational tasks are executed at the edge of mobile networks at small data-centers, known as *edge clouds*.

Mobile edge computing is a key enabling technology for the Internet-of-Things (IoT) [6] and in particular applications in smart cities [19] and disaster response scenarios [9]. In these applications, low-powered devices generate computational tasks and data that have to be processed quickly on local edge cloud servers. More specifically, in smart cities, these devices could be smart intersections that collect data from road-side sensors and vehicles to produce an efficient traffic light sequence to minimize waiting times [14]; or it could be CCTV cameras that analyse video feeds for suspicious behaviour, e.g., to detect a stabbing or other crime in progress [20]. In disaster response, sensor data from autonomous vehicles (including video, sonar and LIDAR) can be aggregated in real time to produce maps of a devastated area, search for potential victims and help first responders in focusing their efforts to save lives [1].

To accomplish these tasks, there are typically several types of resources that are needed, including communication bandwidth, computational power and data storage resources [7], and tasks are generally delay-sensitive, i.e., have a specific completion deadline. When accomplished, different tasks carry different values for their owners (e.g., the users of IoT devices or other stakeholders such as the police or traffic authority). This value will depend on the importance of the task, e.g., analysing current levels of air pollution may be less important than preventing a large-scale traffic jam at peak times or tracking a terrorist on the run. Given that edge clouds are often highly constrained in their resources [12], we are interested in allocating tasks to edge cloud servers to maximize the overall social welfare achieved (i.e., the sum of completed task values). This is particularly challenging, because users in edge clouds are typically self-interested and may behave strategically [3] or may prefer not to reveal private information about their values to a central allocation mechanism [18].

An important shortcoming of existing work of resource allocation in edge clouds, e.g., [3, 7], is that it assumes tasks have strict resource requirements — that is, each task consumes a fixed amount of computation (CPU cycles per time), takes up a fixed amount of bandwidth to transfer data and uses up a fixed amount of storage on the server. However, in practice, edge cloud servers have some flexibility in how they allocate limited resources to each task. In more detail, to execute a task, the corresponding data and/or code first has to be transferred to the server it is assigned to, requiring some bandwidth. This then takes up storage on the server. Next, the task needs computing power from the server in terms of CPU cycles per time. Once computation is complete, the results have to be transferred back to the user, requiring further bandwidth. Now, while the the storage capacity at the server for every task is *strict*, since the task cannot be run unless all the data is stored, the bandwidth and compute speed allocated to the task can be *elastic*. This allows flexibility in the resource allocation process enabling resources to be shared evenly, prevent resource self-interested users and for more task to receive service simultaneously.

Against this background, we make the following novel contributions to the state of the art:

- We formulate an optimization problem for assigning the tasks to the servers, whose objective is to maximize total

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 2020, Auckland, New Zealand

https://doi.org/doi

social welfare, taking into account resource limitations and elastic allocation of resources.
- We prove that the problem is NP-hard and propose an approximation algorithm with a performance guarantee of $\frac{1}{n}$, where n is the number of tasks, and a linearithmic computational complexity, i.e., $O(n \log(n))$.
- We propose a range of auction-based mechanisms to deal with the self-interested nature of users. These offer various trade-offs regarding truthfulness, optimality, scalability, information requirements from users, communication overheads and decentralization.
- Using extensive realistic simulations, we compare the performance of our algorithm against other benchmark algorithms, and show that our algorithm outperforms all of them, while at the same time being within 95% to the optimal solution.

The paper is organized as follows. In the next section we discuss related work. This is followed by the problem formulation in Section 3. Our novel resource allocation mechanisms are presented in Section 4. In Section 5, we evaluate the performance of our mechanisms and compare them against the optimal solution and other benchmarks. Finally, Section 6 concludes the work.

2 RELATED WORK

There is a considerable amount of research in the area of resource allocation and pricing in cloud computing, some of which use auction mechanisms to deal with competition [3, 4, 11, 22]. However, these approaches assume that users request a fixed amount of resources system resources and processing rates, with the cloud provider having no control over the speeds, only the servers that the task was allocated to. In our work, tasks' owners report deadlines and overall data and computation requirements, allowing the edge cloud server to distribute its resources more efficiently based on each task's requirements.

Our problem is related to multidimensional knapsack problems. In particular, Nip et al. [15] consider flexibility in the allocation, with linear constraints that are used for elastic weights. The paper provides a pseudo-polynomial time complexity algorithm for solving this problem to maximize the values in the knapsack. Our problem case is similar to their problem, but our problem has non-linear constraints due to the deadline constraint, so their algorithm cannot be applied here.

Other closely related work on resource allocation in edge clouds [7] considers both the placement of code/data needed to run a specific task, as well as the scheduling of tasks to different edge clouds. The goal there is to maximize the expected rate of successfully accomplished tasks over time. Our work is different both in the setup and the objective function. Our objective is to maximize the value over all tasks. In terms of the setup, they assume that data/code can be shared and they do not consider the elasticity of resources.

3 PROBLEM FORMULATION

In this section we first describe the system model. Then, we present the optimization problem and prove its NP-hardness.

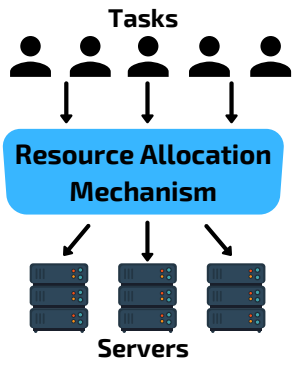


Figure 1: System Model

3.1 System model

A sketch of the system is shown in Fig. 1. We assume that in the system there is a set of servers $I = \{1, 2, \ldots, |I|\}$ servers, which could be edge clouds that can be accessed either through cellular base stations or WiFi access points (APs). Servers have different types of resources: storage for the code/data needed to run a task (e.g., measured in GB), computation capacity in terms of CPU cycles per time interval (e.g., measured in FLOP/s), and communication bandwidth to receive the data and to send back the results of the task after execution (e.g., measured in Mbit/s). We assume that the servers are heterogeneous in all their characteristics. More formally, we denote the storage capacity of server i with S_i, computation capacity with W_i, and the communication capacity with R_i.

There is a set $J = \{1, 2, \ldots, |J|\}$ of different tasks that require service from one of the servers.[1] Every task $j \in J$ has a value v_j that represents the value of running the task to its owner. To run any of these tasks on a server requires storing the appropriate code/data on the same server. These could be, for example, a set of images, videos or CNN layers in identification tasks. The storage size of task j is denoted as s_j with the rate at which the program is transferred to the server being $s_j^{'}$. For a task to be computed successfully, it must fetch and execute instructions on a CPU. We consider the total number of CPU cycles required for the program to be w_j, where the rate at which the CPU cycles are assigned to the task per unit of time is $w_j^{'}$. Finally, after the task is run and the results obtained, the latter need to be sent back to the user. The size of the results for task j is denoted with r_j, and the rate at which they are sent back to the user is $r_j^{'}$. Every task has its deadline, denoted by d_j. This is the maximum time for the task to be completed in order for the user to derive its value. This time includes: the time required to send the data/code to the server, run it on the server, and get back the results. We assume that there is an *all* or *nothing* task execution reward scheme, meaning that for the task value to be awarded the entire task must be run and the results sent back within the deadline.

[1] We focus on a single-shot setting in this paper. In practice, an allocation mechanism would repeat the allocation decisions described here over regular time intervals, with longer-running tasks re-appearing on consecutive time intervals. We leave a detailed study of this to future work.

3.2 Optimization problem

Given the aforementioned assumptions, the optimal assignment of tasks to servers and optimal allocation of resources in a server to the tasks assigned to that server is obtained as a solution to the following optimization problem. Here, the decision variables are $x_{i,j} \in \{0,1\}$ (whether to run task j on server i) as well as $s_j^{'}$, $r_j^{'}$ and $w_j^{'}$ (indicating the bandwidth rates for transferring the code, for returning the results and the CPU cycles per unit of time, respectively).

$$\max \sum_{\forall j \in J} v_j \left(\sum_{\forall i \in I} x_{i,j} \right) \tag{1}$$

s.t.

$$\sum_{\forall j \in J} s_j x_{i,j} \leq S_i, \qquad \forall i \in I, \tag{2}$$

$$\sum_{\forall j \in J} w_j^{'} x_{i,j} \leq W_i, \qquad \forall i \in I, \tag{3}$$

$$\sum_{\forall j \in J} (r_j^{'} + s_j^{'}) \cdot x_{i,j} \leq R_i, \qquad \forall i \in I, \tag{4}$$

$$\frac{s_j}{s_j^{'}} + \frac{w_j}{w_j^{'}} + \frac{r_j}{r_j^{'}} \leq d_j, \qquad \forall j \in J, \tag{5}$$

$$0 \leq s_j^{'} \leq \infty, \qquad \forall j \in J, \tag{6}$$

$$0 \leq w_j^{'} \leq \infty, \qquad \forall j \in J, \tag{7}$$

$$0 \leq r_j^{'} \leq \infty, \qquad \forall j \in J, \tag{8}$$

$$\sum_{\forall i \in I} x_{i,j} \leq 1, \qquad \forall j \in J, \tag{9}$$

$$x_{i,j} \in \{0,1\}, \qquad \forall i \in I, \forall j \in J. \tag{10}$$

The objective (Eq.(1)) is to maximize the total value over all tasks (i.e., the social welfare). Task j will receive the full value v_j only if it is executed entirely and the results are obtained within the deadline for that task. Constraint (Eq.(2)) relates to the finite storage capacity of every server to store code/data for the tasks that are to be run. The finite computation capacity of every server is expressed through Eq.(3), whereas Eq.(4) denotes the constraint on the communication capacity of the servers. As can be seen, the communication bandwidth comprises two parts: one part to send the data/code or request to the server, and the other part to get the results back to the user.[2] Constraint Eq.(5) is the deadline associated with every task, where the total time of the task in the system is the sum of the time to send the request and code/data to the server, time to run the task, and the time it takes the server to send all the results to the user. Note that if a task is not run on any server, this constraint can be satisfied by choosing arbitrarily high bandwidth and CPU rates (without being constrained by the resource limits of any server). The rates at which the code is sent, run and the results are sent back are all positive and finite (Eqs. (6), (7), (8)). Further, every task is served by at most one server (Eq.(9)). Finally, a task is either served or not (Eq.(10)).

[2]Not that sending and receiving data will not always overlap, but for tractability we assume they deplete a common limited bandwidth resource per time step. This ensures that the bandwidth constraint is always satisfied in practice.

Complexity: In the following we show that this optimization problem is NP-hard.

Theorem 3.1. *The optimization problem* (1)-(10) *is NP-hard.*

Proof. The optimization problem without constraint (5) is a 0-1 multidimensional knapsack problem [10], which is a generalization of a simple 0-1 knapsack problem. The latter is an NP-hard problem [10]. Given this, it follows that the 0-1 multidimensional knapsack problem is also NP-hard. Since optimization problem (1)-(10) is a generalization of a 0-1 multidimensional knapsack problem, it follows that it is NP-hard as well. □

Before we propose our novel allocation mechanisms for the allocation problem with elastic resources, we briefly outline an example that illustrates why considering this elasticity is important. In this example, there are 12 potential tasks and 3 servers (the exact settings can be found in table 2 for the tasks and table 1 for the servers).

Figure 2 shows the best possible allocation if tasks have fixed resource requirements. The resource speeds were chosen such to the minimum total resource usage that the task would require from the deadline. Here, 9 of the tasks are run, resulting in a total social welfare of 980 due to the limitation of the server's computation and the task requirement not being balanced.

In contrast to this, Figure 3 depicts the optimal allocation if elastic resources are considered. Here, it is evident that all of the resources are used by the servers whereas the fixed (in figure 2) cant do this. In total, the elastic approach manages to schedule all 12 tasks within the resource constraints, achieving a total social welfare of 1200 (an 19% improvement over the fixed approach).

The figures represent resource usage of the servers by the three bars relating to each of this resources (storage, CPU and bandwidth). For each task that is allocated to the server, the percentage of the resource's used is bar size. Then, for the tasks that are assigned to corresponding servers, the percentage of used resources are also depicted.

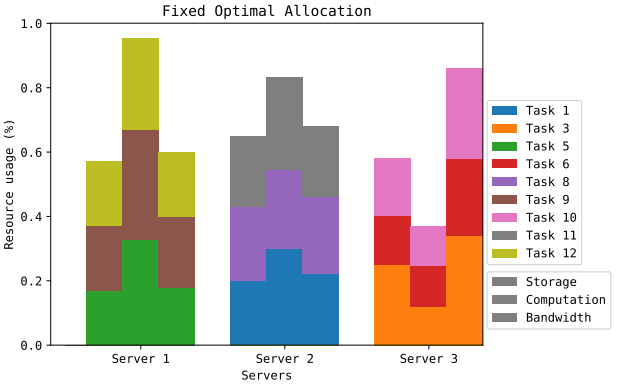


Figure 2: Optimal solution with fixed resources. Due to not being able to balance out the resources, bottlenecks on the server 1 and 2's computation have occurred

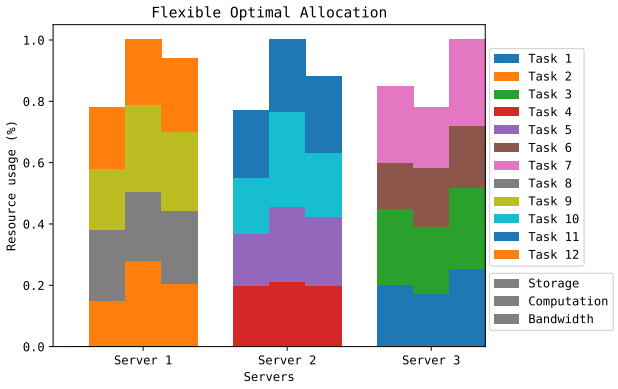


Figure 3: Optimal solution with elastic resources. Compared to the fixed allocation, the elastic allocation is able to fully use all of its resources

Name	S_i	W_i	R_i
Server 1	400	100	220
Server 2	450	100	210
Server 3	375	90	250

Table 1: Servers - Table of server attributes

Name	v_j	s_j	w_j	r_j	d_j	$s_j^{'}$	$w_j^{'}$	$r_j^{'}$
Task 1	100	100	100	50	10	30	27	17
Task 2	90	75	125	40	10	22	32	15
Task 3	110	125	110	45	10	34	30	17
Task 4	75	100	75	35	10	27	21	13
Task 5	125	85	90	55	10	24	28	17
Task 6	100	75	120	40	10	20	32	16
Task 7	80	125	100	50	10	31	30	19
Task 8	110	115	75	55	10	30	22	20
Task 9	120	100	110	60	10	27	29	24
Task 10	90	90	120	40	10	25	30	17
Task 11	100	110	90	45	10	30	26	16
Task 12	100	100	80	55	10	24	24	22

Table 2: Tasks - Table of task attributes, the columns for resource speeds ($s_j^{'}, w_j^{'}, r_j^{'}$) is for fixed speeds which the flexible allocation does not take into account. The fixed speeds is the minimum required resources to complete the task within the deadline constraint.

4 FLEXIBLE RESOURCE ALLOCATION MECHANISMS

In this section, we propose several mechanisms for solving the resource allocation problem with elastic resources. First, we discuss a centralized greedy algorithm (detailed in Section 4.1) with a $\frac{1}{|J|}$ performance guarantee and polynomial run-time. Then, we consider settings where task users are self-interested and may either report their task values and requirements strategically or may wish to limit the information they reveal to the mechanism. To deal with such cases, we propose two auction-based mechanisms, one of which can be executed in a decentralized manner (in Sections 4.2 and 4.3).

4.1 Greedy Mechanism

As solving the allocation problem with elastic resources is NP-hard, we here propose a greedy algorithm (Algorithm 1) that considers tasks individually, based on an appropriate prioritisation function.

More specifically, the greedy algorithm does this in two stages; the first sorts the tasks and the second allocates them to servers. A value density function is applied to each of the task based on its attributes: value, required resources and deadlines. Stage one uses this function to sort the list of tasks. The second stage then iterates through the tasks in the given order, applying two heuristics to each task: one to select the server and another to allocate resources. The first of these heuristics, called the server selection heuristic, works by checking if a server could run the task if all of its resources were to be used for meeting the deadline constraint (eq 5) then calculating how good it would be for the job to be allocated to the server. The second heuristic, called the resource allocation heuristic, finds the best permutations of resources to minimise a formula, i.e., the total percentage of server resources used by the task.

In this paper we prove that the lower bound of the algorithm is $\frac{1}{|J|}$ (where $|J|$ is the number of jobs) using the value of a task as the value density function and using any feasible server selection and resource allocation heuristic. However we found that the task value heuristic is not the best heuristic as it does not consider the effect of the deadline or resources used for a job. In practice, the following heuristic often works better: $\frac{v_j \cdot (s_j + w_j + r_j)}{d_j}$. For the server selection heuristic we use $argmin_{\forall i \in I} S_i^{'} + W_i^{'} + R_i^{'}$, where $S_i^{'}$, $W_i^{'}$, $R_i^{'}$ are the server's available storage, computation and bandwidth resources respectively. While for the resource allocation heuristic we use $min \frac{W_i^{'}}{w_j} + \frac{R_i^{'}}{s_j^{'} + r_j^{'}}$.

Theorem 4.1. *The lower bound of the greedy mechanism is $\frac{1}{n}$ of the optimal social welfare*

Proof. Taking the value of a task as the value density function, the first task allocated will have a value of at least $\frac{1}{n}$ total values of all jobs. As the allocation of resources for a task is not optimal, allocation of subsequent tasks is not guaranteed. Therefore, as the optimal social welfare must be the total values of all jobs or lower then the lower bound of the mechanism must be $\frac{1}{n}$ of the optimal social welfare. □

In figure 4, an example allocation using the algorithm is shown using the model from tables 1 and 2. The algorithm uses the recommend heuristic proposed above and allows for all tasks to be allocated achieving 100% of the flexible optimal in figure 3.

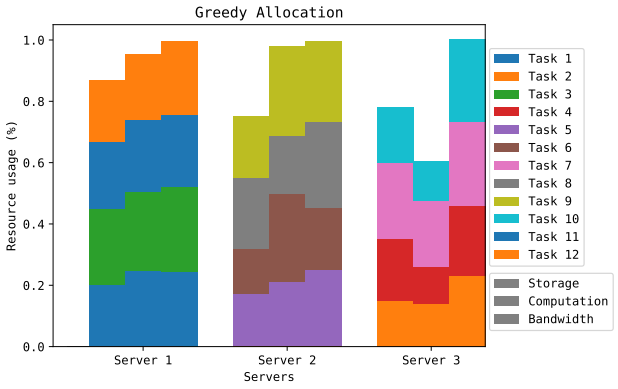


Figure 4: Example Greedy allocation using model from table 2 and 1

Algorithm 1 Greedy Mechanism

Require: J is the set of tasks and I is the set of servers
Require: $S_i^{'}, W_i^{'}$ and $R_i^{'}$ is the available resources (storage, computation and bandwidth respectively) for server i.
Require: $\alpha(j)$ is the value density function of a task
Require: $\beta(j, I)$ is the server selection function of a task and set of servers returning the best server, or $\emptyset$ if the task is not able to be run on any server
Require: $\gamma(j, i)$ is the resource allocation function of a task and server returning the loading, compute and sending speeds
Require: $sort(X, f)$ is a function that returns a sorted list of elements in descending order, based on a set of elements and a function for comparing elements
$J^{'} \leftarrow sort(J, \alpha)$
for all $j \in J^{'}$ **do**
  $i \leftarrow \beta(j, I)$
  if $i \neq \emptyset$ **then**
    $s_j^{'}, w_j^{'}, r_j^{'} \leftarrow \gamma(j, i)$
    $x_{i,j} \leftarrow 1$
  end if
end for

Theorem 4.2. *The time complexity of the greedy algorithm is $O(|J|\,|I|)$, where $|J|$ is the number of tasks and $|I|$ is the number of servers. Assuming that the value density and resource allocation heuristics have constant time complexity and the server selection function is $O(|I|)$.*

Proof. The time complexity of the stage 1 of the mechanism is $O(|J| \log(|J|))$ due to sorting the tasks and stage 2 has complexity $O(|J|\,|I|)$ due to looping over all of the tasks and applying the server selection and resource allocation heuristics. Therefore the overall time complexity is $O(|J|\,|I| + |J| \log(|J|)) = O(|J|\,|I|)$. □

4.2 Critical Value Auction

Due to the problem case being non-cooperative, if the greedy mechanism was used to allocate resources such that the value is the price paid. This is open to manipulation and misreporting of task attributes like the value, deadline or resource requirements. Therefore in this section we propose an auction that is weakly-dominant for tasks to truthfully report it attributes.

Single-Parameter domain auctions are extensively studied in mechanism design [16] and are used where an agent's valuation function can be represented as single value. The task price is calculated by finding the task's value such that if the value were any smaller, the task could not be allocated. This value is called the critical value. This has been shown to be a strategyproof [17] (weakly-dominant incentive compatible) auction so it is a weakly-dominant strategy for a task to honestly reveal its value.

The auction is implemented using the greedy mechanism from section 4.1 to find an allocation of tasks using the reported value. Then for each task allocated, the last position in the ordered the task list such that the task would still allocated is found. The critical value of the task is then equal to the inverse of the value density function where the density is the density of the next task in the list after that position.

In order that the auction is strategyproof, the value density function is required to be monotonic so that misreporting of any task attributes will result in the value density decreasing. Therefore a value density function of the form $\frac{v_j d_j}{\alpha(s_j, w_j, r_j)}$ must be used so that the auction is strategyproof.

Theorem 4.3. *The value density function $\frac{v_j d_j}{\alpha(s_j, w_j, r_j)}$ is monotonic for task j assuming the function $\alpha(s_j, w_j, r_j)$ is monotonic decreasing.*

Proof. In order to misreport the task private value and deadline must be less than the true value. The opposite is true for the required resources (storage, compute and result data) with the misreported value being greater than the true value. Therefore the α function will increase as the resource requirements increase as well, meaning that density will decrease. □

4.3 Decentralised Iterative Auction

VCG (Vickrey-Clark-Grove) auction [21] [5] [8] is proven to be economically efficient, budget balanced and incentive compatible. A task's price is found by the difference of the social welfare for when the task exists compared to the social welfare when the task doesn't exist. Our auction uses the same principle for pricing by finding the difference between the current server revenue and the revenue when the task is allocated (at £0).

The auction iteratively lets a task advertise its requirements to all of the servers who respond with their price for the task. This price is equal to the server's current revenue minus the solution to the the problem in section 4.3.1 plus a small value called the price change variable. Being the reverse of the VCG mechanism, such that the price is found for when the task exists rather than when it doesn't exist. The price change variable allows for the increase in the revenue of the server and is can be chosen by the server. Once all of the server have responded, the task can compare the minimum server price to its private value. If the price is less then the task will accept the servers with the minimum price offer, otherwise the task will stop looking as the price for the task to run on any server is greater than its reserve price.

To find the optimal revenue for a server m given a new task p and set of currently allocated tasks N has a similar formulation to section 3.2. With an additional variable is considered, a task's price being p_n for task n.

4.3.1 Server problem case.

$$\max \sum_{\forall n \in N} p_n x_n \tag{11}$$

$$\text{s.t.} \tag{12}$$

$$\sum_{\forall n \in N} s_n x_n + s_p \leq S_m, \tag{13}$$

$$\sum_{\forall n \in N} w_n^{'} x_n + w_p \leq W_m, \tag{14}$$

$$\sum_{\forall n \in N} (r_n^{'} + s_n^{'}) \cdot x_n + (r_p^{'} + s_p^{'}) \leq R_m, \tag{15}$$

$$\frac{s_n}{s_n^{'}} + \frac{w_n}{w_n^{'}} + \frac{r_n}{r_n^{'}} \leq d_n, \qquad \forall n \in N \cup \{p\}, \tag{16}$$

$$0 \leq s_n^{'} \leq \infty, \qquad \forall n \in N \cup \{p\} \tag{17}$$

$$0 \leq w_n^{'} \leq \infty, \qquad \forall n \in N \cup \{p\} \tag{18}$$

$$0 \leq r_n^{'} \leq \infty, \qquad \forall n \in N \cup \{p\} \tag{19}$$

$$x_n \in \{0, 1\}, \qquad \forall n \in N \tag{20}$$

The objective (Eq.(11)) is to maximize the price of all tasks (not including the new task as the price is zero). The server resource capacity constraints are similar to the constraints in the standard model set out in section 3.2 however with the assumption that the task k is running so there is no need to consider if the task is running or not. The deadline and non-negative resource speeds constraints (5, 6, 7 and 8) are all the same equation with the new task included with all of the other tasks. The equation to check that a task is only allocated to a single server is not included as only server i considers the task k's price.

In auction theory, four properties are considered: Incentive compatible, budget balanced, economically efficient and individual rationality.

- Budget balanced - Since the auction is run without an auctioneer, this allows for the auction to be run in a decentralised way resulting in no "middlemen" taking some money so all revenue goes straight to the servers from the tasks
- Individually Rational - As the server need to confirm with the task if it is willing to pay an amount to be allocated, the task can check this against its secret reserved price preventing the task from ever paying more than it is willing
- Incentive Compatible - Misreporting can give a task as if the task can predict the allocation of resources from server to tasks then tasks can misreport so to be allocate to a certain server that otherwise would result in the task being unallocated.
- Economic efficiency - At the begin then task are almost randomly assigned in till server become full and require kicking tasks off, this means that allocation can fall into a local price maxima meaning that the server will sometime not be 100% economically efficient.

Algorithm 2 Decentralised Iterative Auction

Require: I is the set of servers
Require: J is the set of unallocated tasks, which initial is the set of all tasks to be allocated
Require: $P(i,k)$ is solution to the problem in section 4.3.1 using the server i and new task k. The server's current tasks is known to itself and its current revenue from tasks so not passed as arguments.
Require: $R(i,k)$ is a function returning the list of tasks not able to run if task k is allocated to server i
Require: $\leftarrow_R$ will randomly select an element from a set

```
while |J| > 0 do
    j ←_R J
    p, i ← argmin_{i∈I} P(i, j)
    if p ≤ v_j then
        p_j ← p
        x_{i,j} ← 1
        for all j' ∈ R(i, j) do
            x_{i,j'} ← 0
            p_j' ← 0
            J ← J ∪ j'
        end for
    end if
    J ← J \ {j}
end while
```

The algorithm 2 is a centralised version of the decentralised iterative auction. It works through iteratively checking a currently unallocated job to find the price if the job was currently allocated on a server. This is done through first solving the program in section 4.3.1 which calculates the new revenue if the task was forced to be allocated with a price of zero. The task price is equal to the current server revenue − new revenue with the task allocated + a price change variable to increase the revenue of the server. The minimum price returned by $P(i,k)$ is then compared to the job's maximum reserve price (that would be private in the equivalent decentralised algorithm) to confirm if the job is willing to pay at that price. If the job is willing then the job is allocated to the minimum price server and the job price set to the agreed price. However in the process of allocating a job then the currently allocated jobs on the server could be unallocated so these jobs allocation's and price's are reset then appended to the set of unallocated jobs.

4.4 Attributes of proposed algorithms

In table 3, the important attributes for the proposed algorithm

Attribute	GM	CVA	DIA
Truthfulness		Yes	No
Optimality	No	No	No
Scalability	Yes	Yes	No
Information requirements from users	All	All	Not the reserve value
Communication over heads	Low	Low	High
Decentralisation	No	No	Yes

Table 3: Attributes of the proposed algorithms: Greedy mechanism (GM), Critical Value auction(CVA) and Decentralised Iterative auction (DIA)

5 EMPIRICAL EVALUATION

To test the algorithms presented in section 4, synthetic models have been used to generate a list of tasks and servers.

The synthetic models have been handcrafted with each attribute being generated from a gaussian distribution with a mean and standard deviation.

To compare the greedy algorithm to the optimal elastic allocation, a branch and bound was implemented to solve the problem in section 3.2. In order to compare to fixed speed equivalent models, the minimum total resource required to run the job is found and set as the resource speeds for all of the tasks, with the optimal solution for running the job with the fixed speeds is found as well. To implement the greedy mechanism, the value density function was $\frac{v_j}{s_j+w_j+r_j}$, server selection was $argmin_{\forall i \in I} S_i^{'} + W_i^{'} + R_i^{'}$ and the resource allocation was $mins_j^{'} + w_j^{'} + r_j^{'}$ for job j and servers I.

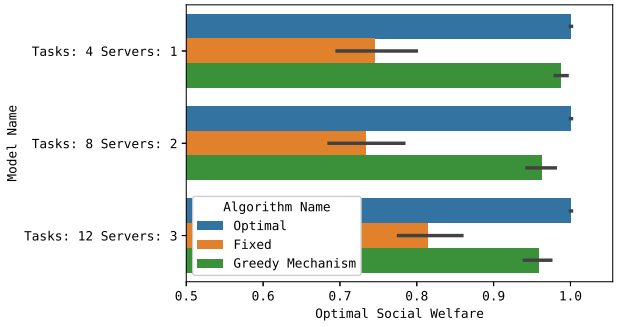


Figure 5: Comparison of the social welfare for the greedy mechanism, optimal, relaxed problem, time limited branch and bound

As figure 5 shows, the greedy mechanism achieves 98% of the optimal solution for the small models, the mechanism achieves within 95% for larger models. In comparison, the fixed allocation achieves 80% of the optimal solution and always does worse than the social welfare of the greedy mechanism.

Figure 6 compares the social welfare of the auction mechanisms: vcg, fixed resource speed vcg, critical value auction and the decentralised iterative auction with different price change variables.

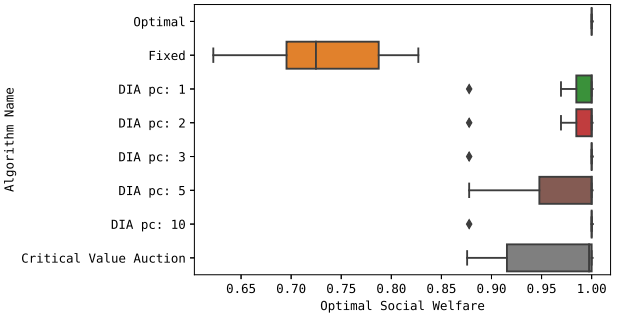


Figure 6: Comparison of the social welfare for the auction mechanisms

VCG is an economically efficient auction that requires the optimal solution to the problem in section 3.2.

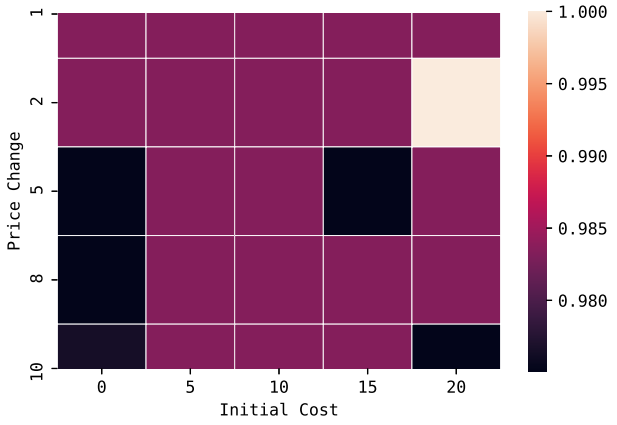


Figure 7: Average number of rounds with a price change variables and task initial cost

Within the context of edge cloud computing, the number of rounds for the decentralised iterative auction is important to making it a feasible auction as it is proportional to the time required to run. We investigated the effect of two heuristic on the number of rounds and social welfare of the auction; the price change variable and initial cost heuristic. With an auction using as minimum heuristic values for the price change and initial cost, figure 7, on average 400 rounds were required for the price to converge while an auction using a price change of 10 and initial cost of 20 means that only on average 80 rounds are required, 5x less. But by using high initial cost and price change heuristics, this can prevent tasks from being allocated, figure 8, shows that the difference in social welfare is only 2% from minimum to maximum heuristics.

6 CONCLUSIONS

In this paper, we studied a resource allocation problem in edge clouds, where resources are elastic and can be allocated to tasks at varying speeds to satisfy heterogeneous requirements and deadlines.

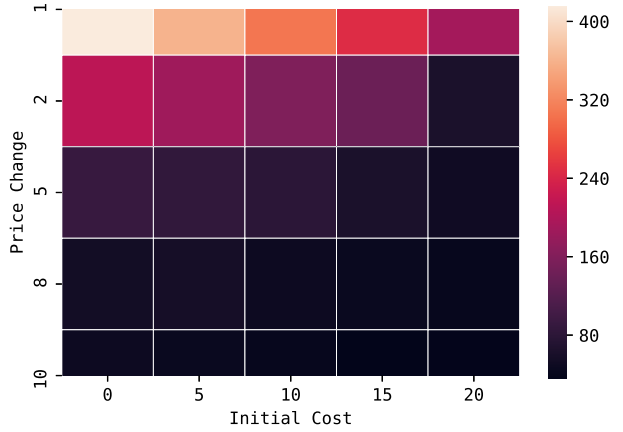


Figure 8: Average social welfare with a price change variables and task initial cost

To solve the problem, we proposed a centralized greedy mechanism with a guaranteed performance bound, and a number of auction-based mechanisms that also consider the elasticity of resources and limit the potential for strategic manipulation. We show that explicitly taking advantage of resource elasticity leads to significantly better performance than current approaches that assume fixed resources.

In future work, we plan to consider the dynamic scenario where tasks arrive and depart from the system over time, and to also consider the case where task preemption is allowed.

REFERENCES

[1] Zubaida Alazawi, Omar Alani, Mohmmad B. Abdljabar, Saleh Altowaijri, and Rashid Mehmood. 2014. A Smart Disaster Management System for Future Cities. In *Proceedings of the 2014 ACM International Workshop on Wireless and Mobile Technologies for Smart Cities (WiMobCity '14)*. ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/2633661.2633670

[2] M. Bahrami. 2015. Cloud Computing for Emerging Mobile Cloud Apps. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. 4–5. https://doi.org/10.1109/MobileCloud.2015.40

[3] Fan Bi, Sebastian Stein, Enrico Gerding, Nick Jennings, and Thomas La Porta. 2019. A truthful online mechanism for resource allocation in fog computing. In *PRICAI 2019: Trends in Artificial Intelligence. PRICAI 2019*, A. Nayak and A. Sharma (Eds.), Vol. 11672. Springer, Cham, 363–376. https://eprints.soton.ac.uk/431819/

[4] Zhiyi Huang Bingqian Du, Chuan Wu. 2019. Learning Resource Allocation and Pricing for Cloud Profit Maximization. In *The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*. 7570–7577.

[5] Edward H. Clarke. 1971. Multipart pricing of public goods. *Public Choice* 11, 1 (01 Sep 1971), 17–33. https://doi.org/10.1007/BF01726210

[6] P. Corcoran and S. K. Datta. 2016. Mobile-Edge Computing and the Internet of Things for Consumers: Extending cloud computing and services to the edge of the network. *IEEE Consumer Electronics Magazine* 5, 4 (2016).

[7] V. Farhadi, F. Mehmeti, T. He, T. L. Porta, H. Khamfroush, S. Wang, and K. S. Chan. 2019. Service Placement and Request Scheduling for Data-intensive Applications in Edge Clouds. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 1279–1287. https://doi.org/10.1109/INFOCOM.2019.8737368

[8] Theodore Groves. 1973. Incentives in Teams. *Econometrica* 41, 4 (1973), 617–631. http://www.jstor.org/stable/1914085

[9] L. Guerdan, O. Apperson, and P. Calyam. 2017. Augmented Resource Allocation Framework for Disaster Response Coordination in Mobile Cloud Environments. In *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*.

[10] Hans Kellere, Ulrich Pferschy, and David Pisinger. 2004. *Knapsack problems*. Springer.

[11] Dinesh Kumar, Gaurav Baranwal, Zahid Raza, and Deo Prakash Vidyarthi. 2017. A systematic study of double auction mechanisms in cloud computing. *Journal of Systems and Software* 125 (2017), 234 – 255. https://doi.org/10.1016/j.jss.2016.12.009

[12] Y. Liu, F. R. Yu, X. Li, H. Ji, and V. C. M. Leung. 2018. Distributed Resource Allocation and Computation Offloading in Fog and Cloud Networks With Non-Orthogonal Multiple Access. *IEEE Transactions on Vehicular Technology* 67, 12 (2018).

[13] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief. 2017. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys Tutorials* 19, 4 (2017).

[14] Kabrane Mustapha, Krit Salah-ddine, and L. Elmaimouni. 2018. Smart Cities: Study and Comparison of Traffic Light Optimization in Modern Urban Areas Using Artificial Intelligence. *International Journal of Advanced Research in Computer Science and Software Engineering* 8 (02 2018), 2277–128. https://doi.org/10.23956/ijarcsse.v8i2.570

[15] Kameng Nip, Zhenbo Wang, and Zizhuo Wang. 2017. Knapsack with variable weights satisfying linear constraints. *Journal of Global Optimization* 69, 3 (01 Nov 2017), 713–725. https://doi.org/10.1007/s10898-017-0540-y

[16] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. 2007. *Algorithmic game theory*. Cambridge university press. 229 pages. https://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf

[17] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. 2007. *Algorithmic game theory*. Cambridge university press. 229–230 pages. https://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf

[18] Mallesh M. Pai and Aaron Roth. 2013. Privacy and Mechanism Design. *SIGecom Exch.* 12, 1 (June 2013), 8–29. https://doi.org/10.1145/2509013.2509016

[19] M. Sapienza, E. Guardo, M. Cavallo, G. La Torre, G. Leombruno, and O. Tomarchio. 2016. Solving Critical Events through Mobile Edge Computing: An Approach for Smart Cities. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*.

[20] G. Sreenu and M. A. Saleem Durai. 2019. Intelligent video surveillance: a review through deep learning techniques for crowd analysis. *Journal of Big Data* 6, 1 (06 Jun 2019), 48. https://doi.org/10.1186/s40537-019-0212-5

[21] William Vickrey. 1961. Counterspeculation, Auctions, and Competitive Sealed Tenders. *The Journal of Finance* 16, 1 (1961), 8–37. http://www.jstor.org/stable/2977633

[22] X. Zhang, Z. Huang, C. Wu, Z. Li, and F. C. M. Lau. 2017. Online Auctions in IaaS Clouds: Welfare and Profit Maximization With Server Costs. *IEEE/ACM Transactions on Networking* 25, 2 (April 2017), 1034–1047. https://doi.org/10.1109/TNET.2016.2619743

Appendix B: SPIE Presentation

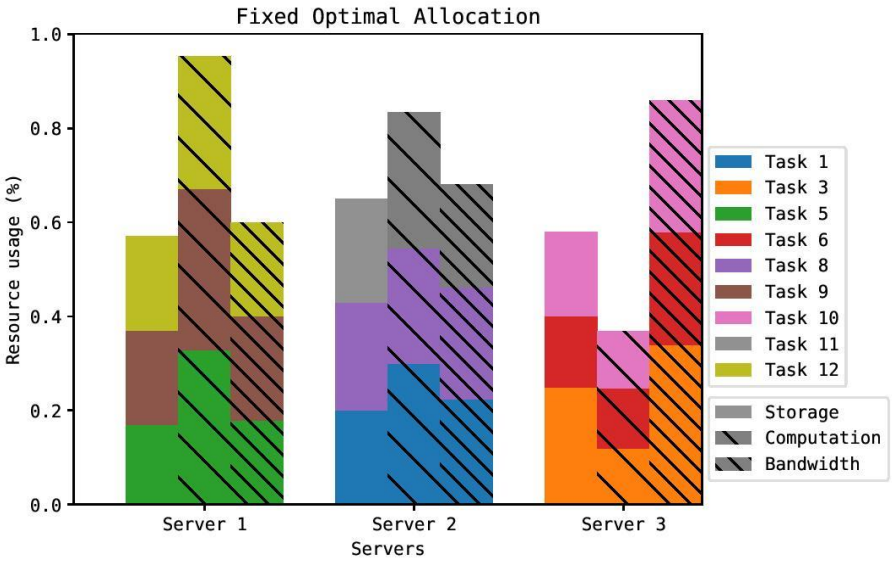
The research of Towers et al. (2020) was presented at SPIE Defense + Commercial Sensing to the conference on Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications II under the title ”Analytical agility at the edge of the network through auction mechanisms”. The link to the recorded presentation.

Analytical agility at the edge of the network through auction mechanisms

By Mark Towers, Fidan Mehmeti, Sebastian Stein, Tim Norman, Tom La Porta, Caroline Rublein and Geeth De Mel

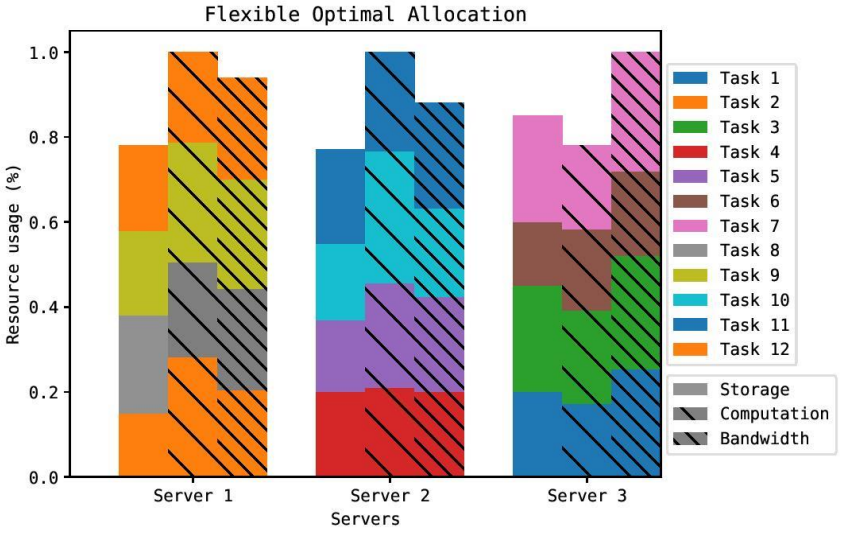
Motivation

- Edge cloud computing allows coalitions to run computationally demanding analytical tasks in military tactical networks that couldn't be run locally by the user.
- However, edge cloud computing servers have significantly fewer resources than traditional cloud computing servers.
- They therefore require efficient and effective allocation of these resources to maximise the number of tasks that can be run concurrently
- Previous research considered a fixed allocation scheme where task requested a fixed resource usage
- However, resource bottleneck can easy occur when numerous users over request particular resources, limit the number of tasks that can be run



- Here servers are described with three resources (storage, computation and bandwidth).
- Each task uses a percentage of these resources in order to run being the coloured bars.

Flexible resource allocation mechanism



- Principle – That the time taken for an operation to complete is proportional to the amount of resources allocated
- Instead task submit their resource requirement over their lifetime and a deadline to be computed by
- This is used by servers to determine how resources are allocated to individual tasks
- Algorithm aims is to maximise the social welfare (sum of task values' that are computed within the deadline)
- This proposes research challenges as no previous algorithms exist that are compatible with this mechanism and to deal with self-interested task owners who may wish to misreport their task values and requested resources.

Deadline Constraint

$$\frac{s_j}{s_j'} + \frac{w_j}{w_j'} + \frac{r_j}{r_j'} \leq d_j$$

s_j	Required Storage	s_j'	Loading task speed
w_j	Required computation	s_j'	Compute speed
r_j	Required results data	r_j'	Sending results speed
d_j	Deadline		

Optimisation problem

$$\max \sum_{\forall j \in J} v_j \left(\sum_{\forall i \in I} x_{i,j} \right) \quad (1)$$

s.t.

$$\sum_{\forall j \in J} s_j x_{i,j} \leq S_i, \quad \forall i \in I, \quad (2)$$

$$\sum_{\forall j \in J} w_j' x_{i,j} \leq W_i, \quad \forall i \in I, \quad (3)$$

$$\sum_{\forall j \in J} (r_j' + s_j') \cdot x_{i,j} \leq R_i, \quad \forall i \in I, \quad (4)$$

$$\frac{s_j}{s_j'} + \frac{w_j}{w_j'} + \frac{r_j}{r_j'} \leq d_j, \quad \forall j \in J, \quad (5)$$

$$0 \leq s_j' \leq \infty, \quad \forall j \in J, \quad (6)$$

$$0 \leq w_j' \leq \infty, \quad \forall j \in J, \quad (7)$$

$$0 \leq r_j' \leq \infty, \quad \forall j \in J, \quad (8)$$

$$\sum_{\forall i \in I} x_{i,j} \leq 1, \quad \forall j \in J, \quad (9)$$

$$x_{i,j} \in \{0, 1\}, \quad \forall i \in I, \forall j \in J. \quad (10)$$

- Using this flexibility principle, an optimisation problem can be mathematically described
- The aim is to maximise the sum of task value that can be computed within the task deadline (equation 1)
- Constraints 2-4 limit the resource usage to be within server capacity
- Constraint 5 forces the task to be completed within its deadline
- Constraints 6-8 force the resource speeds to be positive
- Constraints 9-10 limit a task be allocated to only a single server
- This problem is NP-Hard due to being a knapsack problem

s_j	Required Storage for task j	s_j'	Loading task speed for task j
w_j	Required computation for task j	s_j'	Compute speed for task j
r_j	Required results data for task j	r_j'	Sending results speed for task j
d_j	Deadline for task j	v_j	Value of task j
S_i	Storage capacity of server i	W_i	Computational capacity of server i
R_i	Bandwidth capacity of server i	$X_{i,j}$	Allocation of task j to server i

Approaches

Properties	Greedy mechanism	Critical value auction	Decentralised Iterative auction
Strategyproof	No	Yes	No
Optimal	No	No	No (however this does occur a majority of the time)
Scalability	High	High	Medium
Information requirements from users	All information	All information	All information except the reserved private value
Communications overhead	Low	Low	High
Decentralisation	No	No	Yes

Greedy mechanism

- The algorithm has a lower-bound of 1/n of the optimal welfare. This is as it unknown after the first task is allocated if any subsequent tasks can be allocated. However in practice, this case almost never occurs.
- Algorithm steps:
    1. The list of tasks are sorted in descending order by a value density function.
    2. For each task in the sorted list, a server is selected from the list of available servers using the server selection function.
    3. Then the resource allocated is determined by a resource allocation function for the task on the server
- As a results, the algorithm has polynomial time complexity
- The algorithm however assumes that tasks are not lying about their attributes like value or required resources. This problem is addressed by the critical value auction.

Algorithm 1 Greedy Mechanism

Require: J is the set of tasks and I is the set of servers
Require: $S_i^{'}$, $W_i^{'}$ and $R_i^{'}$ is the available resources (storage, compu
tation and bandwidth respectively) for server i.
Require: $\alpha(j)$ is the value density function of a task
Require: $\beta(j, I)$ is the server selection function of a task and se
of servers returning the best server, or $\emptyset$ if the task is not able t
be run on any server
Require: $\gamma(j, i)$ is the resource allocation function of a task an
server returning the loading, compute and sending speeds
Require: $sort(X, f)$ is a function that returns a sorted list of e
ements in descending order, based on a set of elements and
function for comparing elements

$J^{'} \leftarrow sort(J, \alpha)$
for all $j \in J^{'}$ **do**
  $i \leftarrow \beta(j, I)$
  if $i \neq \emptyset$ **then**
    $s_j^{'}, w_j^{'}, r_j^{'} \leftarrow \gamma(j, i)$
    $x_{i,j} \leftarrow 1$
  end if
end for

Critical value auction

- Single parameter domain auctions are a well-researched area in mechanism design, with the critical value being the minimum value a buyer must report such that the item is still sold to them.
- Using the critical value, strategyproof (weakly-dominant incentive compatible) auctions can be created using the greedy mechanism previously explained as the method of calculating each task's critical value.
- Auction steps:
    1. The greedy mechanism is run with the reported task values to find the task that would be allocated
    2. For each task that would be allocated, we find the critical value for the task, the user then pays this value instead of the reported value as in the greedy mechanism. The critical value is found by:
        a. Removing the task from the list of sorted tasks (greedy mechanism step 1)
        b. Running the greedy mechanism normally except after each task is allocated checking if the task can still be allocated on any server
        c. Once the task is unable to be allocated to any server, the critical value density is equal to the value density of last task allocated
        d. The critical value is equal to the inverse of the value density function using the critical value density
- Due to the use of the greedy mechanism, the auction inherits the same social welfare performance as the greedy mechanism
- The auction's time complexity is still polynomial as well, as it just computes the greedy mechanism multiple times, up to number of tasks + 1.
- While any value density function can be used, in order for the auction to be strategyproof, the value density function used must be monotonic meaning that if the user misreports a task attribute, the value density must decrease.

$$\frac{v_j d_j}{s_j + w_j + r_j}$$

Decentralised iterative auction

- While the critical value auction is incentive compatible, it requires the revelation of a task's value.
- However, for some users, e.g., in tactical networks, they may not wish to reveal this information to other coalition partners.
- The VCG mechanism works by calculating the price of an auction item by finding the difference in social welfare when the task exists and doesn't exist
- We modify this mechanism to be decentralised instead of centralised such that each server calculates the difference. To do this requires a modified optimisation problem for the server to maximise task prices instead of task values.
- Auction steps:
    1. Unallocated tasks is equal to a initial list of tasks
    2. While unallocated tasks has tasks
        A. Select a random task from the list of unallocated tasks
        B. The task price is the minimum price from all of the servers which calculates its price using the modified optimisation problem
        C. If the task price is greater than the minimum price then the task is disregard otherwise allocate the task to the server
        D. However by allocated the task to the server, other tasks may be kicked off. These task are added back to the unallocated tasks list

Algorithm 2 Decentralised Iterative Auction

Require: I is the set of servers
Require: J is the set of unallocated tasks, which initial is the set of all tasks to be allocated
Require: $P(i,k)$ is solution to the problem in section 4.3.1 using the server i and new task k. The server's current tasks is known to itself and its current revenue from tasks so not passed as arguments.
Require: $R(i,k)$ is a function returning the list of tasks not able to run if task k is allocated to server i
Require: $\leftarrow_R$ will randomly select an element from a set

```
while |J| > 0 do
    j ←_R J
    p, i ← argmin_{i∈I} P(i, j)
    if p ≤ v_j then
        p_j ← p
        x_{i,j} ← 1
        for all j' ∈ R(i, j) do
            x_{i,j'} ← 0
            p'_j ← 0
            J ← J ∪ j'
        end for
    end if
    J ← J \ {j}
end while
```

Modified server optimisation problem

- A task's price is equal to the difference in the server revenue (the task doesn't exist) to when the task is included with a price of zero (plus a small value)
- This modifications to the general optimisation problem is that it is only for single server with the new task referred to as n'.
- The resulting constraints are extremely similar except that the new task is forced to be allocated to the server.
  - Objective function is to maximise the sum of computed task prices (Equation 11)
  - Constraints 12-14 limit the resource usage to be within server capacity
  - Constraint 15 forces the task to be completed within its deadline including the new task.
  - Constraints 16-18 force the resource allocation to be positive for all tasks
  - Constraints 19 limits task allocation to be binary

$$\max \sum_{\forall n \in N} p_n x_n \tag{11}$$

s.t.

$$\sum_{\forall n \in N} s_n x_n + s_{n'} \leq S_m, \tag{12}$$

$$\sum_{\forall n \in N} w_n^{'} x_n + w_{n'} \leq W_m, \tag{13}$$

$$\sum_{\forall n \in N} (r_n^{'} + s_n^{'}) \cdot x_n + (r_{n'}^{'} + s_{n'}^{'}) \leq R_m, \tag{14}$$

$$\frac{s_n}{s_n^{'}} + \frac{w_n}{w_n^{'}} + \frac{r_n}{r_n^{'}} \leq d_n, \qquad \forall n \in N \cup \{n^{'}\} \tag{15}$$

$$0 < s_n^{'} < \infty, \qquad \forall n \in N \cup \{n^{'}\} \tag{16}$$

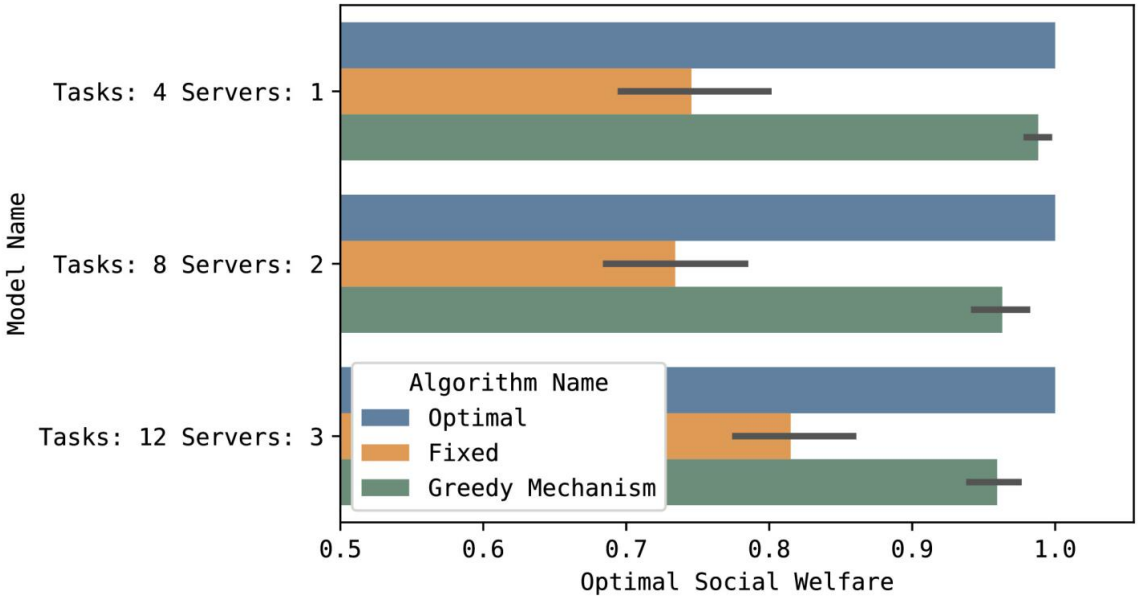
$$0 < w_n^{'} < \infty, \qquad \forall n \in N \cup \{n^{'}\} \tag{17}$$

$$0 < r_n^{'} < \infty, \qquad \forall n \in N \cup \{n^{'}\} \tag{18}$$

$$x_n \in \{0, 1\} \qquad \forall n \in N \tag{19}$$

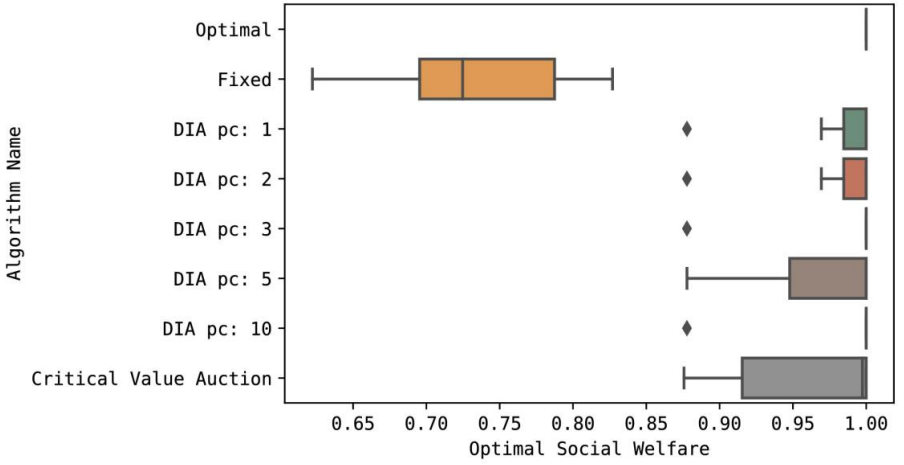
Greedy mechanism and critical value auction results

- To compare our algorithm, we implemented a time-limited optimal solver and a fixed resource allocation mechanism where the task resource usage is fixed between the server preventing any flexibility.
- These results compare the proportion of the optimal social welfare (sum of the computed task values).
- We compared results over a range of environments with different levels of supply and demand.
- With environment that have extremely high demand over a single resources, our system is extremely advantageous as it can deal with this type of pressure that normally is not possible. In cases where server resources are well distributed then this mechanism achieves similar results as the fixed version.



Decentralised Iterative auction

- The VCG auction is an economically efficient auction that finds the optimal allocation to calculate prices. We therefore use the VCG for the optimal flexible and fixed results.
- We found that a majority of the time, the DIA achieves the optimal social welfare however sometimes falls into a local optima
- We also found that the value of the price change doesn't effect the social welfare of the solution much.



Conclusion and future work

- In this work, we have presented a novel resource allocation optimisation problem along with a greedy mechanism to maximise social welfare. Along with two auction mechanisms: critical value auction for strategyproof auctions and a novel decentralised iterative auctions for users who don't wish to reveal their private task value.
- Future work is to consider an online case of this work as tasks arrive over time instead resources being allocated in batches.

Appendix C: Agent hyperparameters

During training there are a range of hyperparameters for each agent, table 1 provides an explanation for the values for each hyperparameter used in the project.

Agent	**Properties name**	**Value**	**Explanation**
RL Agent	batch_size	32	The number of trajectories from the experience replay buffer that are used each time to train an agent.
RL Agent	error_loss_fn	tf.losses. huber_loss	The loss function used in calculating a network's error.
RL Agent	initial_training replay_size	5000	The number of trajectories in the experience replay buffer required before the agent begins training.
RL Agent	training_freq	2	For every trajectory added to the experience replay buffer, for each 2, the agent tries to be trained.
RL Agent	discount_factor	0.9	Within the Q learning function (equation (2.3)), the discount factor determines how important the rewards in the future impact the Q value.
RL Agent	replay_buffer length	25000	The length of the circular experience replay buffer.
RL Agent	save_frequency	25000	The agent networks are saved after 25000 time that agent has been trained
RL Agent	training_loss log_freq	250	Tensorboard allows for data to be saved using training, after every the agent has been trained 250 time, the agents loss is logged for future analysis.

Task Pricing RL Agent	reward_scaling	1	
Task Pricing RL Agent	failed_auction reward	-0.05	The reward for when the agent bids on a task but fails to win the auctioned task.
Task Pricing RL Agent	failed_multiplier	-1.5	A multiplier applied to the winning price is the task fails to be computed within its deadline.
Resource weighting RL Agent	other_task_discount	0.4	The multiplier to tasks not under consideration for a weighting action.
Resource weighting RL Agent	success_reward	1	The reward when the agent successfully completes a task
Resource weighting RL Agent	failed_reward	-1.5	The reward when the agent fails to complete a task within its deadline.
Dqn Agent	target_update_tau	1.0	The update tau value for use in the target update frequency.
Dqn Agent	target_update_freq	2500	The target network in the DQN agent is updated after the agent has been updated 2500 times.
Dqn Agent	initial_epsilon	1	The initial exploration factor during training
Dqn Agent	final_epsilon	0.1	The final exploration factor during training
Dqn Agent	epsilon_steps	10000	The number of training step for linear exploration to move between the initial_epsilon and the final_epsilon factor.
Dueling Dqn Agent	double_loss	True	If to use the double dqn loss function

Categorical Dqn Agent	max_value	-20.0	The maximum value for the value distribution
Categorical Dqn Agent	min_value	25.0	The minimum value for the value distribution
Categorical Dqn Agent	num_atoms	21	The number of atoms for each actions.
Ddpg Agent	actor_learning_rate	0.0001	The learning rate for the optimiser for the actor network.
Ddpg Agent	critic_learning_rate	0.0005	The learning rate for the optimiser for the critic network.
Ddpg Agent	initial_epsilon_std	0.8	The initial exploration standard deviation of the normal distribution used during training
Ddpg Agent	final_epsilon_std	0.05	The final exploration standard deviation of the normal distribution used during training
Ddpg Agent	actor_target update_freq	3000	The actor target network update frequency
Ddpg Agent	critic_target update_freq	1500	The critic target network update frequency
Ddpg Agent	upper_action bound	30.0	The upper action bound for the actor network
Task pricing Ddpg Agent	min_value	-100.0	The minimum value for the critic network to estimate for an action
Task pricing Ddpg Agent	max_value	100.0	The maximum value for the critic network to estimate for an action

Resource allocation Ddpg Agent	min_value	-20	The minimum value for the critic network to estimate for an action
Resource allocation Ddpg Agent	max_value	15	The maximum value for the critic network to estimate for an action
TD3 Agent	actor_update_freq	3	The actor network update frequency for each critic network update.

TABLE 1: Agent hyperparameters

Appendix D: Project management

To management this project has been difficult due to having to split my time between writing the academic paper at the start of the year (plus creating the recorded presentation) and the programming and write up the expanded research. Because of this, a risk assessment (Table 2) to understand possible risks during the project. The project was also planned using a grantt chart (figure 1) to understand the progress of the project over time and to plan how the project would run. The project brief is submitted in October 2019 is included at the end of this report as well. The word count of the project generated by texcount for chapters 1 to 6 is included.

Risk assessment

Risk	Severity	Possibility	Explanation
Rejected paper submission	1	4	The paper (Appendix A) was submitted to the AAMAS 2020 conference in December that could be rejected. While this was be disappointing if this happened, the paper can still be submitted to another conference at a later time.
Iridis failure	5	1	In order to train the agent in this project, the University of Southampton supercomputer, Iridis 5 was utilised so if this could not be used anymore. This was cause massive issues to be able to train all of the agents for both long enough.
Agent Training time	4	2	Reinforcement learning agents can take several million actions to achieve effective results which takes days or weeks to learn. This project doesn't have such resources or time to spend training agents to do this therefore making it a large problem.
Personal Illness or Injury	4	1	Physical illness or injury could prevent the project from being both programmed or written.
Pandemic	4	1	While Pandemic will not prevent the project from being completed due to all work being digital, it would prevent talking with my supervisor in person or for University to be open.

TABLE 2: Risk Assessment

Progress Grantt Chart

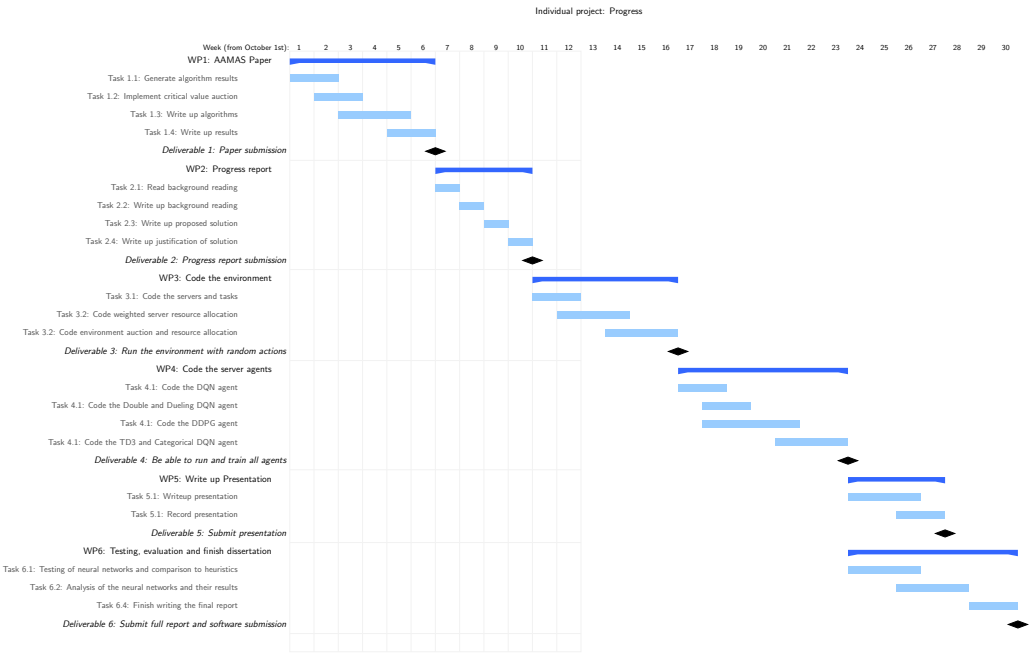


FIGURE 1: Progress Grantt chart

Project Brief

Mark Towers

October 2019

1 Problem

When computer programs are too large, difficult or time consuming to be run on a normal computer, these programs are often offloaded to cloud providers like Google Cloud Platform, Amazon Web Service, Microsoft Azure and many more. These providers all allow customers to individually request a set of resources to compute their program with. However, this can create a bottleneck on certain resource preventing other jobs from running with this fixed requirement model. This project considers the case where users don't request a set amount of resources but rather the user details the total requirements for the program and a deadline for when the program must be finished. This then means that the cloud provider can effectively balances resource demands as it has complete knowledge of its different user's requirements, allowing more jobs to run simultaneously and lower price as there can be a lower overall demand of individual resources.

This research is being done in conjunction with the DAIS ITA project with previous work done by myself, Dr Sebastian Stein and a team from Pennsylvania State University. The difference between the prior work done and this proposed work is to introduce the idea of time more fully into the problem case. Where previously, all jobs would arrive at the beginning of the program and the resources speeds allocated would be fixed for the rest of the program, in this work, jobs will arrive over time and at each time step then the resource speeds allocated can change.

To do this, I believe that I will have to create two algorithms, the first to evaluate a price to charge jobs for running on the cloud provider and second to know how to allocate resources for each jobs currently allocated to the cloud provider at a given time step.

2 Goals

The goal of the project is to apply reinforcement learning to solve the two problem stated above. This is because to the two functions above are too complex for humans to describe effectively and I believe that a universal function approximation will be able to learn them over time through the use of a reward function based on the revenue caused by finishing a job.

3 Scope

The scope of this project is to investigate the use of reinforcement learning in solving this program and if time allows compare to greedy or handcrafted algorithms. We will test the effectiveness of these algorithms through synthetic data and real world data from a google data center.

Word count

```
File: chapters/1_introduction.tex
Encoding: ascii
Sum count: 1074
Words in text: 1003
Words in headers: 1
Words outside text (captions, etc.): 70
Number of headers: 1
Number of floats/tables/figures: 0
Number of math inlines: 0
Number of math displayed: 0

File: chapters/2_background_lit.tex
Encoding: ascii
Sum count: 1493
Words in text: 1472
Words in headers: 11
Words outside text (captions, etc.): 8
Number of headers: 3
Number of floats/tables/figures: 1
Number of math inlines: 1
Number of math displayed: 1
Subcounts:
  text+headers+captions (#headers/#floats/#inlines/#displayed)
  76+2+0 (1/0/0/0) Chapter: Literature Review
  676+7+0 (1/0/1/0) Section: Resource allocation and pricing in Cloud Computing
  720+2+8 (1/1/0/1) Section: Reinforcement Learning

File: chapters/3_solution.tex
Encoding: ascii
Sum count: 2856
Words in text: 2746
Words in headers: 19
Words outside text (captions, etc.): 58
Number of headers: 6
Number of floats/tables/figures: 4
Number of math inlines: 29
```

```
Number of math displayed: 4
Subcounts:
  text+headers+captions (#headers/#floats/#inlines/#displayed)
  99+5+0 (1/0/0/0) Chapter: Optimising resource allocation in MEC
  650+4+2 (1/0/29/4) Section: Resource allocation optimisation problem
  695+3+21 (1/0/0/0) Section: Auctioning of Tasks
  829+2+15 (1/1/0/0) Section: Server agents
  141+2+4 (1/1/0/0) Subsection: Auction agents
  332+3+16 (1/2/0/0) Subsection: Resource allocation agents

File: chapters/4_implementation.tex
Encoding: ascii
Sum count: 2082
Words in text: 2045
Words in headers: 29
Words outside text (captions, etc.): 8
Number of headers: 7
Number of floats/tables/figures: 1
Number of math inlines: 0
Number of math displayed: 0
Subcounts:
  text+headers+captions (#headers/#floats/#inlines/#displayed)
  84+8+4 (1/0/0/0) Chapter: Implementing Flexible Resource Allocation Environmer
  164+3+0 (1/0/0/0) Section: Simulating MEC networks
  384+4+0 (1/0/0/0) Subsection: Weighted server resource allocation
  310+6+0 (1/0/0/0) Section: Server auction and resource allocation agents
  427+3+0 (1/0/0/0) Subsection: Agent Rewards Functions
  306+3+4 (1/1/0/0) Subsection: Agent Training Observations
  370+2+0 (1/0/0/0) Section: Training agents

File: chapters/5_evaluation.tex
Encoding: ascii
Sum count: 1798
Words in text: 1653
Words in headers: 23
Words outside text (captions, etc.): 122
Number of headers: 7
Number of floats/tables/figures: 10
```

```
Number of math inlines: 0
Number of math displayed: 0
Subcounts:
  text+headers+captions (#headers/#floats/#inlines/#displayed)
  87+3+0 (1/0/0/0) Chapter: Testing and evaluation
  487+2+21 (1/0/0/0) Section: Functional testing
  113+2+0 (1/0/0/0) Section: Agent evaluation
  0+3+0 (1/0/0/0) Subsection: Fixed resource Heuristics
  437+5+55 (1/4/0/0) Subsection: Environment and Agent number training
  259+4+25 (1/3/0/0) Subsection: Reinforcement learning algorithm training
  270+4+21 (1/3/0/0) Subsection: Neural network architecture training

File: chapters/6_conclusion.tex
Encoding: ascii
Sum count: 282
Words in text: 278
Words in headers: 4
Words outside text (captions, etc.): 0
Number of headers: 1
Number of floats/tables/figures: 0
Number of math inlines: 0
Number of math displayed: 0

Total
Sum count: 9585
Words in text: 9197
Words in headers: 87
Words outside text (captions, etc.): 266
Number of headers: 25
Number of floats/tables/figures: 16
Number of math inlines: 30
Number of math displayed: 5
Files: 6
Subcounts:
  text+headers+captions (#headers/#floats/#inlines/#displayed)
  1003+1+70 (1/0/0/0) File(s) total: chapters/1_introduction.tex
  1472+11+8 (3/1/1/1) File(s) total: chapters/2_background_lit.tex
  2746+19+58 (6/4/29/4) File(s) total: chapters/3_solution.tex
```

```
2045+29+8 (7/1/0/0) File(s) total: chapters/4_implementation.tex
1653+23+122 (7/10/0/0) File(s) total: chapters/5_evaluation.tex
278+4+0 (1/0/0/0) File(s) total: chapters/6_conclusion.tex
```