MSci Software Engineering Thesis on

Load-Aware Task Offloading in Mobile Edge Computing: A House Selling Problem Approach

Odysseas Polycarpou (2210049p)

February 11, 2021

University of Glasgow | School of Computing Science

Abstract - Mobile Edge Computing has emerged as a new computing paradigm to provide computing resources and storing applications closer to the end-users (node devices) at the operator network boundary (edge server). This computing paradigm is also often used by vehicles such as trains, planes and private cars. One of the main challenges of MEC is task offloading. Task offloading is the transfer of computational tasks to a separate processor or external platforms such as a grid of servers or the Cloud. Task offloading mainly faces when and where is best to offload tasks to mitigate a smart device's energy consumption and workload. This paper tackles this challenge by adopting the Optimal Stopping Theory principles with three time-optimised sequential decision-making models. A performance evaluation is provided with 3? Real data-sets on which our proposed models are applied and compared to the Optimal model. The results showcase how close our models can be to the Optimal one based on probability and scaling factors. Moreover, in our performance evaluation section, we conclude that one of the applied sequential models can be extremely close to the Optimal one making it suitable in single-user and competitive user scenarios.

Index Terms - Mobile Edge Computing, Task Offloading, Optimal Stopping Theory, Sequential decision making.

1 Introduction

In recent years, Mobile Edge Computing architecture has been proposed as a new network concept that enables Cloud computing capabilities and information technology service environment at the *edge* of any network. MEC technology is being developed and improved rapidly. It is designed to be used by machines such as mobile devices, trains, planes, private cars or even enterprise premises such as factory buildings and homes. These are called edge nodes. Edge nodes, store and perform applications/tasks on the network closer to the end-users. The emergence of MEC has seen the development of several applications being launched on the Internet-of-Things. Because of this, the network is heavily loaded, and devices require fast and substantial processing to handle all these applications. The basic idea behind MEC is that by bringing these tasks closer to the cellular customer, usually the Cloud or servers connected on the Cloud (edge servers), the congestion on the network is reduced and applications perform better and more efficiently.

One of the most notable mobile edge computing applications is computational offloading or task offloading in cloud computing. Cloud computing refers to the applications being moved over the internet hosting data centres providing application execution and computing services. One can think of the Cloud as a large data storage space for applications with many edge servers connected to it and distributed over a large area. Whenever the Cloud is busy, which is the case, applications will be transferred to one of the deployed edge servers for faster hosting and execution. Task offloading, as the term states is the transfer of computational tasks from a local edge node to a separate external processor such as a server or grid of servers for execution. A central processor on a local edge node such as a mobile device will process tasks by executing rudimentary arithmetic, control logic and operations. The efficiency of this processing is depended on the instructions per second, a CPU can execute, and because of the wide range of CPU types, the processing power and efficiency varies. Moving applications to an external faster and more powerful processor such as an edge server can accelerate processing and improve application execution efficiency and latency.

1.1 Motivation and Challenge

When various things are connected to a network, it is referred to as the Internet of Things (IoT). Vast amounts of incomplete data generated by the IoT needs to be processed and responded to in a short time. The Cloud has become essential part of this process. However, since Cloud has been centrally deployed globally, it needs to process an enormous amount of data. Also, as the physical distance between the user and the cloud increases, data needs to travel more. This increases the transmission latency, and together with it, the response time increases and stresses out the user a lot. If we consider smartphones alone which are part of the IoT, a person actively uses a smartphone for about 5hrs/day. This means that competing resources will be wasted for the rest 19hrs!

On top of that, the processing speed depends on the performance of the user's device. This means that applications waiting to be executed can stay idle for 19hrs/day due to the heavy queuing of applications. Therefore, Task Offloading's challenge is to find the *best possible* server at *best possible time* to offload the task that is to be executed while the user is on the move. The output of the decision-making model adopted for task offloading will significantly impact application performance quality, such as the latency and execution time. The two factors that directly affect this is the *current* load on the MEC server and the *transmission status* between mobile device nodes and the MEC server.

Another challenge that has emerged in task offloading is when a mobile node moves among many MEC servers. In the optimal scenario, the mobile node should find the best server to connect to at the best time based on the load on the MEC server at that time instance. The load on a deployed MEC server among a group of servers can broadly vary. For example, at time instance t=0, many users are connected on the MEC server waiting to execute their tasks. However, at t=1, only a few users may be connected to the same MEC server making it ideal for offloading and executing tasks. Some of the questions that arise from this scenario are: *When a decision for offloading is made, should we offload now or keep it for later if a better server may be found*? *What are the chances that we can find a better server if we keep looking*?

In this paper, we examine a scenario where a mobile node device moves between multiple MEC servers. The computation offloading decision is made by three sequential decision-making models that can be managed and optimised by applying the principles of Optimal Stopping Theory[12]. We will monitor the behaviour on real data sets of varying CPU load values at different time points. The models applied have to pick the *best* server to offload the data at *best* possible time in a sequential manner to achieve the *best* execution of tasks possible on the MEC server. We also assume that in all these three decision-making models, when a server is discarded, we cannot return to pick it up.

1.2 Contribution

Our House Selling Optimal Stopping Time (OST) [5] method model can be applied in different case scenarios to be integrated with decision-making applications. Our proposed model can be used efficiently on node devices and with real data. Unlike other related work done on Task Offloading, our model explicitly focuses on the CPU load value on the MEC servers. Our model can efficiently determine the best possible server to offload the tasks in most of the simulations performed in our Performance Evaluation section. The contributions of our work are:

We propose three OST-based models that are implemented and applied on a real data set to maximise the chance to find the optimal MEC server to offload the tasks assuming a uniform network status between the node and the server.

- We optimise our House Selling model to outperform our Random and Secretary models [5] and perform almost as efficiently as the problem's optimal solution.
- We created a time series analysis tool and a simulation executable that can allow the user to set different parameters and produce a visual representation of the results.
- Using those results, we provide a comparative evaluation of the models against the Optimal model and individually.

2 Related Work

The vast majority of the work being done on task offloading focuses on whether the task should be performed locally or offloaded to the Cloud. The approaches' main goal is to be as close as possible to the Optimal Stopping result by minimising the execution delay and energy consumption. The work in [1] proposes an OST model that aims to address Task Offloading's problem in Mobile Edge Computing and proposes two baseline deterministic and stochastic models that will improve the challenge node mobile devices face. That is when and where they will get connected (to an edge server) to perform computing tasks. i.e. the problem addressed is to find and offload on the best candidate among a group of edge server and the best one to connect based on the load traffic and latency of each one. The OST-models proposed by the authors are compared to the House Selling (HS), Random and p-stochastic models. In [8] motivated by the increasing demand for computation resources of IoT mobile devices caused by the creation of diverse IoT applications, the paper focuses on addressing the problem of the task offloading between Internet Moblile Devices and the Unmanned Aerial Vehicles which aims to minimise the overall energy consumption for accomplishing the tasks. In [3], the work focuses on the problem of which tasks should be offloaded and not on where the tasks will be offloaded. Therefore, the paper's main challenge is to select the appropriate tasks to be offloaded to peers or Cloud. The model proposed in this paper to solve the problem targets two characteristics. To maximise the performance and minimise the consumption of resources. The significance of the approach is the use of Machine Learning in a new proposed model on a Long Short Term Memory (LSTM) network that will be able to indicate which tasks should be offloaded. The intelligent scheme makes this decision based on a deep learning scheme and a rewarding mechanism. In [2] given the movement of mobile nodes between MEC servers, the paper aims to propose a model that gives the connection of a mobile node to the best edge server at the best time in order to optimise the quality of service. The problem tackled is the offloading decision making by adopting the OST principles using real data in the experiment. Also, the optimal server/time is unknown and not provided meaning that the OST-based model can achieve a delay close to the optimal. In [7] the work focuses on the problem of moving mobile nodes such as crewless aerial vehicles, vehicular networks, data analytics and augmented reality being able to choose the ideal time and server candidate, using principles of the Optimal Stopping Theory to minimise the execution delay in a sequential decision manner. This study's significance is reducing execution delay for the decision-making process and how the foundations are used to reach the optimal time and server for the moving node to connect. The work in [4] tackles the optimal mulit decision movile computation offlloading by setting hard task deadlines. The paper uses a Markovian optimal stopping theory model which is computed using dynamic programming. The proposed model is proven to be energy optimal. Unlike our approach this study aims to optimize the model in terms of energy also guaranteeing hard task deadlines. The authors in [6] assume that the offloading decisions are given and derive the closed-form expressions of the optimal transmit power and local CPU frequencies. Their algorithm is based on a reduced-complexity Gibbs Sampling algorithms to take the optimal offloading decisions. Their aim is to minimize complexity and the model achieves this efficiently. Almost all of the works above, focus on reducing energy consumption, reducing complexity and introduce new approaches to perform the decision making. Unlike these, we focus on applying three OST-based Models, sequentially and prove that we can perform as close as possible to the optimal solution of finding the best server at the best time to offload our tasks.

3 Preliminaries

In mathematics, the Optimal Stopping Theory (OST) or early stopping theory refers to the problem of choosing a time to stop and take a particular decision so that we can maximise an expected reward or minimise an expected cost. Optimal Stopping Time problems can be found in many areas such as statistics, economics and computing science. Stopping Time rule problems are associated with a sequence of random variables [X_1, X_2,…X_n] and a sequence of reward functions that depend on the observed values in random variables. Given these, while observing the sequence of random variables at each step, we choose between stopping or continuing observing. If we stop observing at a given step, we receive the reward function sequence's corresponding reward. The aim is to stop at an observation where the expected reward will be as high as possible (or equivalently, expected cost as low as possible). Optimal Stopping Time models can be applied either sequentially or non-sequentially on a large set of variable observations. Sequential means that we divide our data into chunks and observe from the next position we chose to stop. Non-sequential means that we start to observe from the first position of each chunk of observation variables no matter where we chose to stop in the sequence. For our study, the models were implemented non-sequentially as we assumed uniformity of data. Some of the most common Optimal Stopping Time problems are the Secretary Problem, the Random Probability and House Selling which are the 3 models we applied and optimised in our evaluation.[11]

Definition [12]:The definition of the problem is given by two objects, as mentioned before.

1. A sequence of variables, X_1, X_2,…, where joint distribution is assumed known
2. A sequence of real-valued reward functions given by

$$y_0,\ y_1(x_1),\ y_2(x_1,x_2),\ ...,\ y\infty(x_1,x_2,...)$$

Based on these two objects, the associated stopping rule problem can be described as we may

- Observe the sequence of variables as long as we want and for each n =1,2,…, after observing the sequence, we may stop and receive the known reward.
- Continue and observe with the hope of maximising the reward later on.

4 Server Load-based & Time-optimized Task Offloading

In this paper, we examine three optimal stopping time problems by adopting principles of the Optimal Stopping Theory. These are the Random(P) Problem, the Secretary Problem and the House Selling Problem. In the performance evaluation at the end of this paper, each model's results are compared to the Optimal Solution to see how far we are from it by tweaking several parameters. The Optimal Solution of the problem is the best combination of stopping decisions that can be taken. i.e. We get the maximum reward every time we choose to stop observing. In practice and theory, we can never achieve a better solution than the Optimal solution. Instead, we can be as close as possible.

4.1 Problem Formulation

The scenario envisioned in this paper is a case of Vehicular Network or Internet of Vehicles, moving along the road and where many MEC servers are distributed, similar to the way studied and presented in [9] as shown in Figure 1. The MEC servers along the road provide computing resources for vehicles on the move, to offload and perform computing tasks. Such tasks involve map rendering, image recognition or data analytic tasks. Similarly, tasks can be generated by smartphones from the passengers in the cars. The connection between a MEC server and a vehicle on the road is established wirelessly over the Cellular Network. At each time instance, there is a load value L on each server, which is a random variable in our contexr. In simpler words, what L means is how crowded a server is, of users connected, waiting to execute their tasks. The higher the load, the more crowded a server is. When a MEC server is heavily packed and loaded with users, the transmission delay from the user to the server and back increases. One can think of the transmission delay as the expected time for a user to offload the task and receive the back results. Our approach to this problem in this paper aims to *minimise this transmission delay by adopting models that can pick the best possible server with the least load (least crowded) to minimise the transmission delay*. Hereafter, the time needed to upload the data on the server, process and return the result as fast as possible. This is achieved by finding the best possible server with the lowest load CPU load L to offload our tasks.

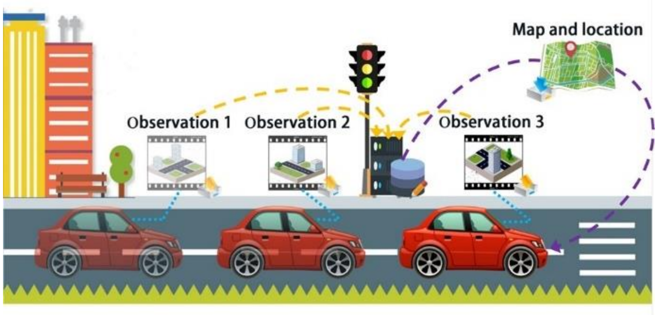


Figure 1: Car passing through MEC servers

Once a task is generated and needs to be offloaded on the MEC server, the node user passes through several servers ready to access the computation task. To do so, the current load of the server needs to be checked. When a server is picked up, then the observation stops and the rest of the servers are discarded (are not considered by the mobile node anymore). However, if the model chooses to discard a server while observing and move to the next one, it cannot return to it (e.g., imaging the scenario in Figure 1 where the car/vehicle cannot abruptly do the reverse while driving in one specific lane/direction). Therefore, our approach's main objective is to maximise the probability of offloading the tasks to the best possible server with the least load in every run. To formulate our problem and apply our proposed solution, we made the two following critical assumptions:

A1: The network status between all the servers and the node users is uniformly distirbuted.

A2: When a server is discarded, it cannot be rechecked, that is, no recall is allowed.

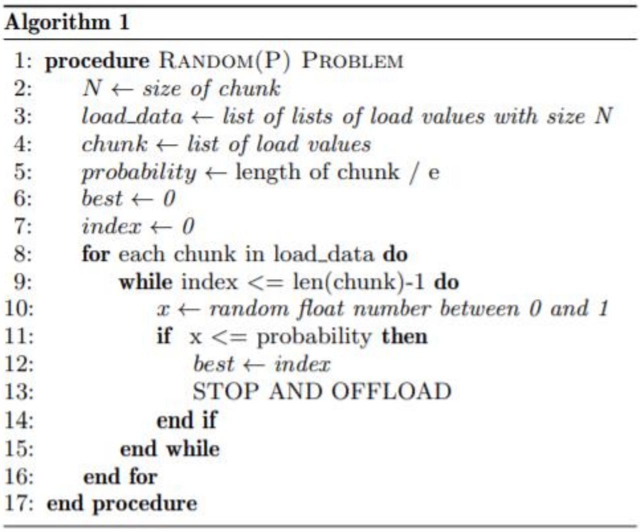
4.2 OST-based Decision-Making Models

4.2.1 Random (P) Model

The random probability model or Random(P) relies on a probability P and a randomly generated float number x to decide whether to offload on a MEC server or keep observing. Unlike the other two Optimal Stopping models we implement, the random probability model does not consider the MEC servers' load value to make this decision. Whether the model will stop and offload on a MEC server or keep looking is based on how high the simulation's probability has been set and the random variable x's value.

The random probability model takes a sequence of N>0 variable observations, which are the MEC servers available. Each server holds a load that does not affect the model's decision-making in implementing this model. Before applying the model on the data and start the simulation, we set a fixed probability P and a function which generates a random floating number x in a given range. In the first run, the model will look at the first entry of N, that is N1. If the probability P is greater than the random number x, then the model will stop and offload on that server. If the above condition is not satisfied, the model will discard the server and move to the next one, generate a new random number x and then make a decision again. As we can see, the load on the MEC server currently under consideration is not taken into account when deciding whether to offload or not. Mathematically speaking, if we suppose that the random floating number x will be set in the range [0,1], then the higher we set the probability P means that there is also a higher probability that the model will satisfy p>x. Thus stop and offload, i.e., early stopping when looking at server. The same occurs when we make the range of the random floating number x smaller. The opposite happens when we make the probability smaller or widen the range of the random floating number. The best approach is to set a reasonable range for the floating number x and a challenging probability of equal chances of being greater or less than x.

In our experiments, we take chunks of N = 100 servers each. We set the range between 0 and 1 and provide several probability values P to examine the model's behaviour as the load is not involved in our decision-making models. We set the values of off-loading probability P = [0.05, 0.1, 0.2, 0.3, 0.5]. Our algorithm in Algorithm 1 uses the value of P and the random floating number x to pick the server on which the tasks will be offloaded.



Algorithm 1

```
procedure RANDOM(P) PROBLEM
    $N \leftarrow$ size of chunk
    load_data $\leftarrow$ list of lists of load values with size $N$
    chunk $\leftarrow$ list of load values
    probability $\leftarrow$ length of chunk / e
    best $\leftarrow$ 0
    index $\leftarrow$ 0
    for each chunk in load_data do
        while index <= len(chunk)-1 do
            $x \leftarrow$ random float number between 0 and 1
            if  x <= probability then
                best $\leftarrow$ index
                STOP AND OFFLOAD
            end if
        end while
    end for
end procedure
```

4.2.2 Secretary Model

The Secretary OST problem [16] is the most common example of Optimal Stopping Time problem. As the name of the problem states, an administrator wants to hire the best secretary out of N applicants' sample. The order the applicants are interviewed is one at a time, and when an applicant is discarded cannot be reviewed. This means a decision for each applicant is taken immediately after interviewed. In this scenario, the administrator can evaluate and rank each seen secretary's quality and ability, but not those of the yet unseen applicants. The challenge here is to adapt an optimal strategy to find the best time to stop reviewing applicants to maximise selecting the best. This approach's difficulty is that the decision needs to be made immediately after interviewing a secretary applicant. Using this context, we can apply this scenario to our case, taking candidates to be a group of N MEC servers available for task offloading. The algorithm needs to find the best possible server candidate to offload the tasks to achieve optimality of task execution based on the current load of the specific server chosen.[14]

4.2.2.1 Solving the Secretary Problem

In terms of solving the Secretary Problem and discovering which approach to follow, the example in [16], and scan through the first r MEC servers and then choose the first option that is better than any of the MEC servers in [1, r]. Assume that *i* is the greatest integer and occurs at n+1. For this approach to return the best option in the sequence, the following two conditions must hold.

1. The maximum integer cannot be in [1,r]. If yes, then we lose because we are missing the best option.
2. max([1,r] == max([1,n]) must be true.

Thus, to calculate our approach's effectiveness, we need to know the probability that both will hold. Given some n, the probability is:

$$\frac{r}{n}\frac{1}{N} \qquad (1)$$

where:

- 1/N = probability that i occurs at n+1
- r/n = the probability the condition in condition 2 is true

To calculate the probability for some r, P(R), not for arbitrary n but everything, we need to sum over n>=r:

$$P(\mathrm{R}) = \frac{1}{N}\left(\frac{r}{r} + \frac{r}{r+1} + \frac{r}{r+2} + \cdots + \frac{r}{N-1}\right) \qquad (2)$$

$$= \frac{r}{N}\ \sum_{n=r}^{N-1}\frac{1}{n} \qquad (3)$$

The above is a Riemann approximation of an integral so we can rewrite it by letting $\lim_{N\to\infty}\frac{r}{N} = x$ and $\lim\ _{N\to\infty}\frac{n}{N} = t$ we get that:

$$P(\mathrm{R}) = \lim_{N\to\infty}\frac{r}{N}\sum_{n=r}^{N-1}\frac{N}{n}\frac{1}{N} = x\int_x^1\frac{1}{t}dt = -x\ln x \quad (4)$$

Now, we can find the optimal r by solving for $P'(\mathrm{R}) = 0$ back plugging r_{optimal} back into $P(\mathrm{R})$, can find the probability of success which is:

$$P'(\mathrm{R}) = -\ln x - 1 = 0 \Rightarrow x = \frac{1}{e} \qquad (5)$$

$$P\left(\frac{1}{e}\right) = \frac{1}{e} \approx .37 \qquad (6)$$

Therefore, this strategy selects the best server, about 37% of the time. We cannot select the first observed server as we have no other observation to compare with. A much better approach is to have fewer observations as a sample and set a benchmark for the remaining candidate servers. Therefore the sample will only be used for setting a benchmark. On the one hand, if we take a small sample, we do not have enough information for setting a benchmark.

On the other hand, if we take a large sample, we might burn many potential candidates trying to set the benchmark. The best approach is to choose an optimal sample size which can be done by calculating N/e. The probability that the applicant selected is also the best is given by:

$$P(R) = \sum_{n=R+1}^{N} P(n\ is\ selected\ and\ is\ best) \qquad (7)$$

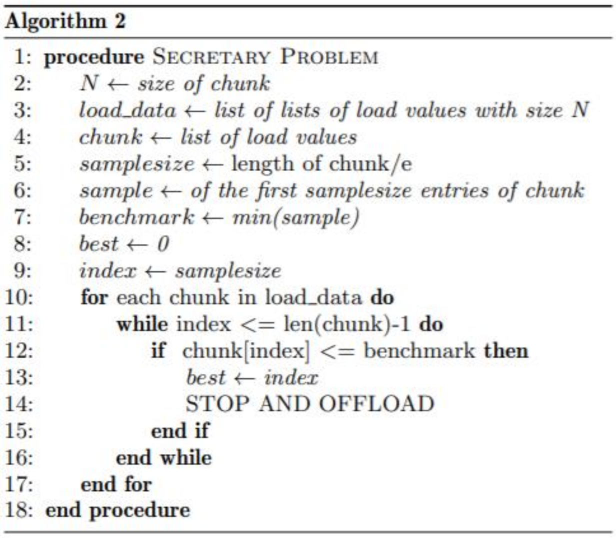
$$= \sum_{n=R+1}^{N} P(n\ best)\, P(next\ in\ R\ out\ of\ n-1) \qquad (8)$$

$$= \sum_{n=R+1}^{N}\frac{1}{N}\frac{R}{n-1} \qquad (9)$$

$$= \frac{R}{N}\sum_{n=R+1}^{N}\frac{1}{n-1} \qquad (10)$$

To help us apply the Secretary Problem in terms of our implementation, we assume that we take chunks of 100 candidate servers N each.

Therefore, our sample with N = 100 will be 100/e = 37. After discarding the first 37 servers, the model will pick the next server candidate with less load than the first 37. Our tool also offers the availability to change the number of servers N in each chunk to observe and decide which one is closest to the optimal as in Algorithm 2 below.



Algorithm 2

```
procedure SECRETARY PROBLEM
    $N \leftarrow$ size of chunk
    load_data $\leftarrow$ list of lists of load values with size $N$
    chunk $\leftarrow$ list of load values
    samplesize $\leftarrow$ length of chunk/e
    sample $\leftarrow$ of the first samplesize entries of chunk
    benchmark $\leftarrow$ min(sample)
    best $\leftarrow$ 0
    index $\leftarrow$ samplesize
    for each chunk in load_data do
        while index <= len(chunk)-1 do
            if chunk[index] <= benchmark then
                best $\leftarrow$ index
                STOP AND OFFLOAD
            end if
        end while
    end for
end procedure
```

4.2.3 House Selling Model

A famous Optimal Stopping Ttime problem is the House Selling problem [5]. The definition of the problem is simple. As the term explains, the problem occurs when a seller has a house they want to sell. The sequence of observations N represents the price which the seller is offered for their house. In order to advertise it, the seller has to pay an amount each day (discounting factor). The House Selling problem's basic approach is to maximise the amount we earn by choosing a stopping rule. If the seller sells the house on the n day, the amount earned is given by:

$$y_k = (1+r)^{N-k} X_k \qquad (11)$$

where r in [0,1] is a discount factor. There are many approaches to the House Selling problem, depending on the situation. In our case, one can think of the load values in the sequence of N available MEC servers as *availability* values. The aim is to sell our house (mapping to off-load our tasks) or stop observing and offload the tasks on a MEC server when the availability is highest, which means *better* opportunity. To do this, we scale the availability values to be in the [0,1] range in the sequence using:

$$A = [A_1, A_2, \dots, A_n]$$

$$s_k = \frac{max(A) - A_k}{max(A) - min(A)} \qquad (12)$$

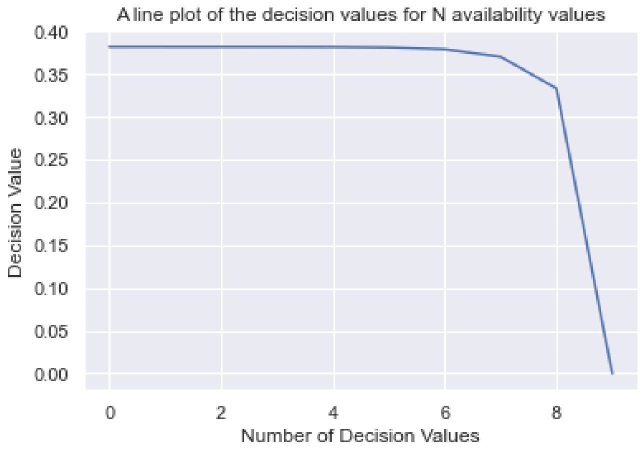


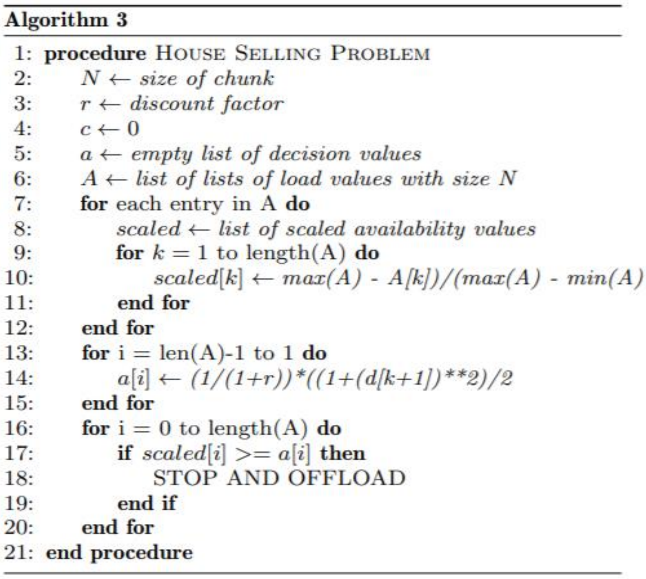
Figure 2: Decision Values plot

In Figure 2 we can see a graphical representation of the decision values. The higher the decision value is, the less likely to stop and offload at an observed server. This graph tells us that we start with a small possibility to find the best at the beginning of the sequence. As time passes and we keep observing the load values sequence, it is more likely to hit the best server. Please note that the graph below takes a case scenario where we have n = 10 observations/time instances.

The target is to stop at a time instance 1<= k <=n, where we potentially have the highest availability. We also introduce a discount factor r which again is in the range [0,1]. Before proceeding to the offloading decision process, we need to calculate the decision values to be compared with the scaled availability values, which will lead to the decision of whether to offload or not. The decision values are calculated as in (2) below with backward induction which is part of dynamic programming. A more detailed calculation of the proof for the scalar decision values can be found in Section 4.2.3.1

$$a_k = \big(1/(1+r)\big) * \big((1 + (a_{k+1})^2)/2\big) \qquad (13)$$

for k = 0, …, n-1, with initial value: a_{n-1}=0.5/(r+1). When the decision-making process is started, our model will decide whether to offload on a specific server or keep looking on the rest of the server, as shown in Algorithm 3. The scaled availability values are compared with the decision values. If the availability value at index i in the sequence of N MEC server loads is greater or equal to the corresponding decision value at the same index i then the model will stop and offload the tasks on that server. If the above condition is not satisfied, then the model will go to the next availability and decision value and compare it until the simulation is finished.



Algorithm 3

```
procedure HOUSE SELLING PROBLEM
    $N \leftarrow$ size of chunk
    $r \leftarrow$ discount factor
    $c \leftarrow 0$
    $a \leftarrow$ empty list of decision values
    $A \leftarrow$ list of lists of load values with size N
    for each entry in A do
        scaled ← list of scaled availability values
        for $k = 1$ to length(A) do
            scaled[k] ← max(A) - A[k])/(max(A) - min(A)
        end for
    end for
    for i = len(A)-1 to 1 do
        a[i] ← (1/(1+r))*((1+(d[k+1])**2)/2
    end for
    for i = 0 to length(A) do
        if scaled[i] >= a[i] then
            STOP AND OFFLOAD
        end if
    end for
end procedure
```

4.2.3.1 *Solving the House Selling Model*

Starting with the dynamic programming objective function:

$$J_n(x_n) = max\left[(1+r)^{N-k} x_n, E[J_{n+1}(w)]\right]$$

and $a_n = \frac{E[J_{n+1}(w)]}{(1+r)^{N-k}}$ (14)

The decision is taken as:

- if $s_n >= a_n$ then send task
- if $s_n < a_n$ then continue looking

Let:

$$V_k(x_n) = max(x_n, (1+r)^{-1} E[V_{k+1}(w)]) \qquad (15)$$

And then:

$$a_k = E[V_k + 1(w) * (1+r)^{-1}] \qquad (16)$$

Hence:

$$V_k(x_n) = max(x_k, a_k) \qquad (17)$$

And

$$V_{k+1}(x_{n+1}) = max(x_{k+1}, a_{k+1}) \text{ or} \qquad (18)$$

$$E[V_{k+1}(x)] = E[max(x, a_{k+1})] \quad \text{or} \qquad (19)$$

$$E[V_{k+1}(x)] * (1+r)^{-1} \qquad (20)$$

$$= E[max(x, a_{k+1})](1+r)^{-1} \qquad (21)$$

Substituting

$max(x, a_{k+1})$ *from* (17)

Into

$E[V_{k+1}(x)] * (1+r)^{-1}$ (20)

Gives

$$a_k = E[max(x, a_{k+1})] * (1+r)^{-1} \qquad (22)$$

$$a_k = (1+r)^{-1}\left[\int_{a_{k+1}}^{1} x\, df(x) + \int_0^{ak+1} a_{k+1} df(x)\right] \qquad (23)$$

$$a_k = (1+r)^{-1}\left[1 - \int_0^{a_{k+1}} x\, df(x) + \int_0^{ak+1} a_{k+1} df(x)\right] \qquad (24)$$

$$a_k = (1+r)^{-1}\left[1 - \frac{{a_{k+1}}^2}{2} + {a_{k+1}}^2\right] \qquad (25)$$

$$a_k = (1+r)^{-1}\left[1 + \frac{{a_{k+1}}^2}{2}\right] \qquad (26)$$

$$a_k = (1+r)^{-1}\left[1 + \frac{{a_{k+1}}^2}{2}\right] \qquad (27)$$

$* \ k = 1,2,3, \ldots., N-1$

$* \ df(x) = dx$ and $P(x) = x$ and $E[x] = \frac{1}{2}$

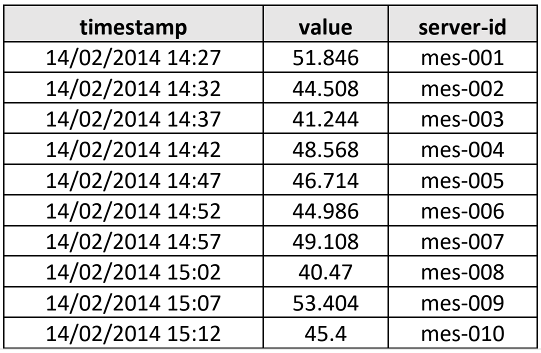
Gives:

$$a_{N-1} = E[V_N(w)](1+r)^{-1} = \frac{E[x]}{(1+r)} \qquad (28)$$

5 Implementation & Experiments

5.1 Data Sets

For this paper, we chose to work with time-series data. To simulate the MEC server candidates distributed along the path where the node device will be moving, we used the real data sets of EC2 CPU utilisation load values collected over 2 weeks. The data comes from a real Amazon Web Services Cloud Watch [10]. In the data set, the CPU load value is being recorded every 5 minutes. The data set contains about 4000 records of CPU loads, separated in chunks of N = 100 values each as we are applying our models non-sequentially. Time instances are replaced to be represented by scalar values instead of the real timestamps to allow us to illustrate how far we are from the optimal solution. Therefore, we have 4000 real values at 4000 points in time, and every time our model looks through these records will process and store the best possible point in time based on the load value on the MEC servers. The idea behind our chosen real data set is that our proposed solution can be used in a real-life scenario. From the performance evaluation, we will see which model operates best and under what circumstances. A sample of our time series data is given below and a graphical representation of this sample, shown in Figure 3.



timestamp	value	server-id
14/02/2014 14:27	51.846	mes-001
14/02/2014 14:32	44.508	mes-002
14/02/2014 14:37	41.244	mes-003
14/02/2014 14:42	48.568	mes-004
14/02/2014 14:47	46.714	mes-005
14/02/2014 14:52	44.986	mes-006
14/02/2014 14:57	49.108	mes-007
14/02/2014 15:02	40.47	mes-008
14/02/2014 15:07	53.404	mes-009
14/02/2014 15:12	45.4	mes-010

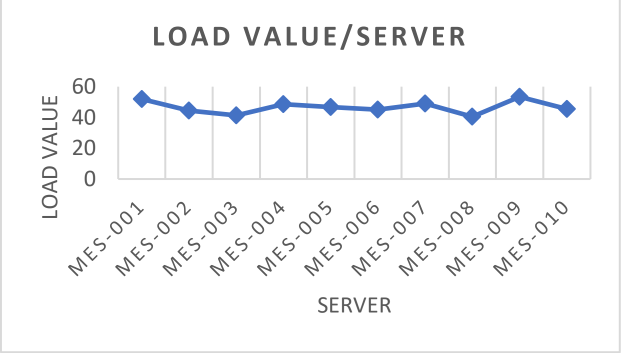
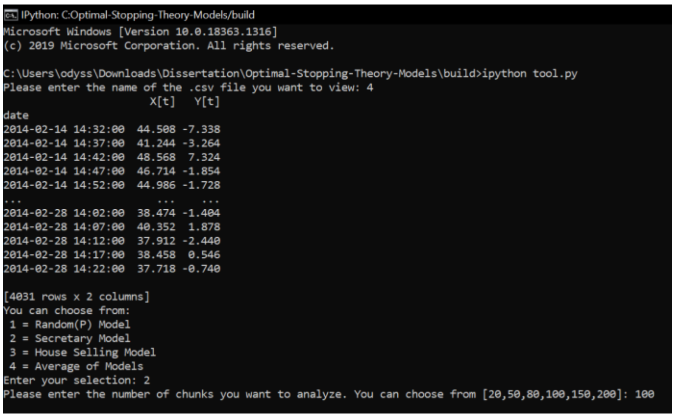


Figure 3: Plotted Sample of data [10]

5.2 Framework and Methods

Our tool comes in two parts. The first one is the *Time Series Analysis* tool used to analyse the real-time series data inserted and find any trends. This is done, so the user is aware of the data nature before applying the models. The second one is the *Simulation* tool for our constructed models. To implement our Optimal Stopping models, we had to choose an appropriate tool that will allow us to carry out the simulations and produce a graphical representation of the results. For this, we chose to work with Python 3.x Programming Language, Jupyter Notebook and Numpy and Pandas libraries. We used these to manage our data, carry out a time-series analysis on the records, implement the proposed solution models and apply them to the data set. Our tool's development was done to work through the command line console and allow the user to input their preferences regarding what analysis they want to carry out and for which models. Finally, our tool's competence is to take in any correctly defined CSV file, apply the models on it, manage and process it and return results. The first screen the user sees when running the tool is a prompt to enter the CSV name that will be analysed. Then the user will be guided by the instructions on the screen. A screenshot of our tool's screen can be seen below:



```
Microsoft Windows [Version 10.0.18363.1316]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\odyss\Downloads\Dissertation\Optimal-Stopping-Theory-Models\build>ipython tool.py
Please enter the name of the .csv file you want to view: 4
                       X[t]   Y[t]
date
2014-02-14 14:32:00  44.508 -7.338
2014-02-14 14:37:00  41.244 -3.264
2014-02-14 14:42:00  48.568  7.324
2014-02-14 14:47:00  46.714 -1.854
2014-02-14 14:52:00  44.986 -1.728
...                     ...    ...
2014-02-28 14:02:00  38.474 -1.404
2014-02-28 14:07:00  40.352  1.878
2014-02-28 14:12:00  37.912 -2.440
2014-02-28 14:17:00  38.458  0.546
2014-02-28 14:22:00  37.718 -0.740

[4031 rows x 2 columns]
You can choose from:
 1 = Random(P) Model
 2 = Secretary Model
 3 = House Selling Model
 4 = Average of Models
Enter your selection: 2
Please enter the number of chunks you want to analyze. You can choose from [20,50,80,100,150,200]: 100
```

Finally, it is worth mentioning that the source code's implementation is developed in functions to give the user the availability to choose between the models or compare them all together—the code as shown in [17], displays the functions created with commenting to assist in using each one. When the user makes a selection, the specific function is called and returns graphs and data frames as a visual representation of the results. The tool is also capable of saving the output files for future reference of analysis.

The notations adopted in this papers are provided in Table 3.

Remark 1: ***A GitHub repository has also been created to host our files and can be found in [15].***

6 Performance Assessment

6.1 Time Series Analysis

A time series is a series of data points indexed in a timed order. Most specifically, a time series sequence is a sequence of observed values taken at successively equally spaced points over a fixed period. Therefore, we can think about such data as discrete-time data. Time series analysis comprises methods for analysing time series data to extract meaningful statistics and other trends in our sequence of values. It can be applied to real-valued, continuous and discrete numeric data as in [13]. We can divide the methods for time series analysis is two groups, frequency-domain and time-domain methods. For this study, we will focus on the latter since our data is defined over time. In the time-domain, we can use correlation and autocorrelation techniques to extract the patterns in a filter-like manner.

The basic functionality of our Times Series Analysis tool is to identify the trends in our series of data. There are several ways to think about identifying trends in a time series sample. One of the most popular ways to identify a trend is to take our data's rolling average [13]. This means that, for each time point, we calculate the average of the points on either side of it.

The number of those points is specified by a window size, which we can set using the rolling() function. What this causes is to smooth out noise and seasonality. When it comes to determining the window size, we can set this directly in the function. It makes sense to choose 14 as we are talking about a 2-week period of time. We can see trend smoothed out in Figure 4. After finding the data trend, it is time to think about seasonality, which is a repetitive nature on the time series.

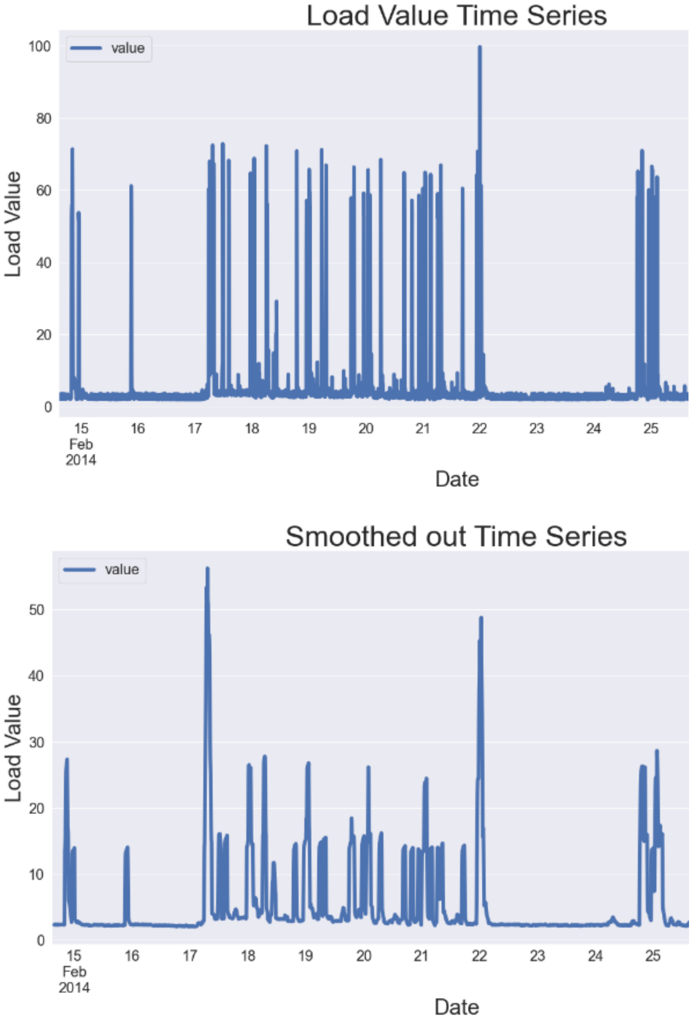


Figure 4: Time Series before and after smoothing

One common and effective way to remove the trend from the time series is "differencing" [13]. First-order differencing refers to the calculation of the difference between successive points. For this, we used the diff() function. Figure 5 shows that most of the trend is removed and where the data spikes. Differencing turns the time series into a stationary time series. This series, as shown in the graph, repeats itself but not in a constant rate. We can see a weak seasonality in the beginning weeks (15-17), denser in the middle weeks (17-22) and falls during weeks 22-25 and becomes dense again during the last 3 weeks (25-28). This is incredibly tellings as removing the trend can reveal correlation in seasonality.

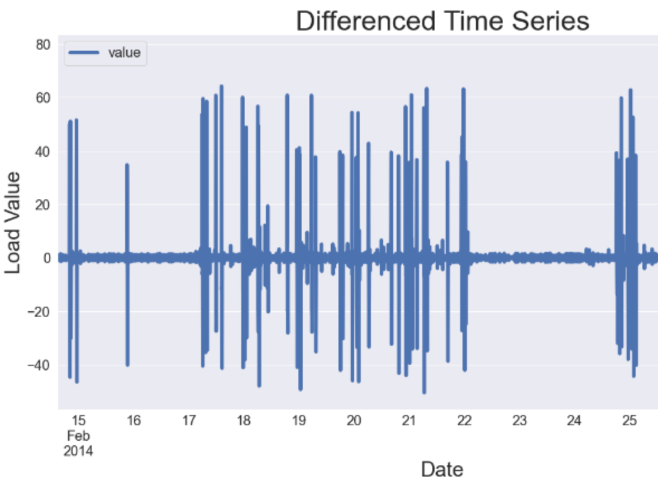


Figure 5: Differenced Plot

A series is periodic, when it repeats itself at equally spaced time intervals, let us say, every 5 minutes or every day. To check if our data is indeed periodic, we plotted the autocorrelation of the series in Figure 6. This means that we expect to see some spikes in the autocorrelation graph if the original repeats itself. Indeed in the graph below, we can see small spikes every few observations among 4000. This is very good as it means that our data has a constant uniformity thoughout. There are no unexpected ‘highs’ or ‘lows’ among the series. Therefore, this makes it suitable for working with during our simulations and performance assessment.

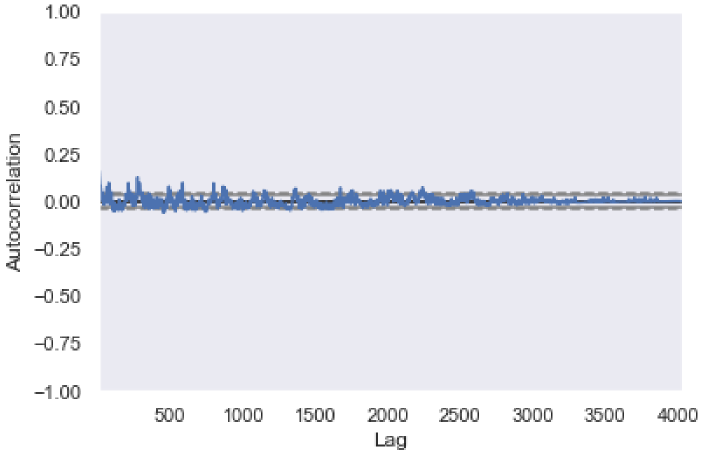


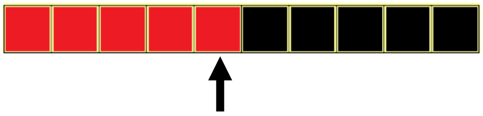
Figure 6: Autocorrelation

6.2 Experimental Evaluation

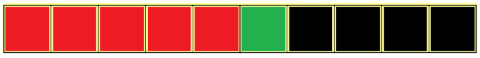
We compare each of our 3 models, Random(P), Secretary and House Selling with the Optimal Solution individually in our performance evaluation. The difference between the optimal offloading value and the one we achieved and how far we were from the optimal stopping instance. Also, to get a clearer picture of our models' performance, we compare each model with average load values with the optimal average values. In this way, we can see which model behaves best in different scenarios. Our tool can take in parameters giving the user the ability to change each simulation's desired chunk sizes. The sizes that can be passed are N = [20, 50, 80, 100, 150, 200]. This makes the tool dynamic and applicable in many real case situations as we deal with real data.

To illustrate the process of our experimental simulations, we first need to analyse our data and separate it accordingly. The simulations take chunks of 100 records each, and in this sequence, they need to observe each MEC server and make a decision to offload to the best possible server. To help visualise how the models carry out the decision-making process, we provide a part of our simulation process below. The squares represent the servers in the sequence. The red ones denote the seen servers that we checked and discarded while the green denote the server on which we decide to offload.

1. We start with the first batch of servers, apply the model and start observing:



2. We discard servers based on the model logic and decide where to stop. We offload and move to the next batch:



3. First run is completed, and we repeat the process with the next sequence of servers.

Application of Random (P) Model: For the Random(P), we assign a probability value p for offloading each server's task under observation. This means that the server has a probability p to be selected, equally not selected with 1-p. The probability of offloading depends on the value of the randomly generated floating number x in our code. If p>x then we stop and offload. Otherwise, we keep observing. Whenever a server is selected, we stop and offload the task storing the load value and the point in time. For example, for a probability p=0.7, then it is more likely that the user will offload the task early in the sequence. It will not be efficient to offload at the start of the period though we miss many servers. Similarly, if the probability is too small, then the offloading decision will be delayed, and many servers will be discarded. Interestingly, each time the probability is changed, a significant difference is noted. A probability of between 0.1 and 0.2 produces and most promising results. i.e. the stopping does not happen too early or too late. In case we do not stop anywhere, we offload to the very last server. We combine all load values, and we calculate the average.

Application of Secretary Model: Applying the Secretary Model, since we directly consider each server's load value, we increase performance as we do not rely on luck or probabilities. For each chunk of code, we take the first 37 observations and set a benchmark. We call these first 37 observations of the sample. The benchmark is the minimum value of the sample. The sample of the data is calculated as N/e. N is the chunk size, which the user can adjust in the console interface at the beginning of the simulation. Then, we start from the 38th server observation and compare every next value with the benchmark. At the point where value observed is <= benchmark, we stop and offload the task, and we again store the load value and the point in time when the offload happened.

Application of House Selling Model: Finally, the House Selling model scales the load values at each server based on a discount factor r in the range [0,1] and each chunk N's size, both defined by the user when choosing to apply the House Selling Model. For example, when the discount factor is high close 1, then the model's performance falls. On the other hand, as we bring r closer to 0, we can see an improvement in the model's performance. This is because as r increases, the decision values increase. Since those decision values are responsible for the offloading decision, it is more likely that the model will stop early at the start of the sequence. However, when we make r smaller decision values decrease, the observation and decision making will be more effective. Since when r=0 there is no discount, it is more likely that we will hit the best MEC server.

6.3 Experimental Results

In this section, we apply the Random Probability, the Secretary Model and the House Selling Model on our set of data as we explained before. We pass in the chunk size we want to observe, and the several parameters we want to give each model for our decision-making process. For these simulations, we assigned the following parameter values as shown in Table 1:

Csv Number under testing	Sequence Size, N	Random(P) Probability, p	House Selling Discount factor, r
4	200	0.1	0.015

Table 1: Values used in simulations

The sequence's size is set to N = 200 to see how the models behave with a large set of MEC servers. The group's a figures below show a bar chart of the optimal values (red) compared with the loads on the MEC servers we achieved when offloaded (black). Figures of group b, show the difference in time instances of the Optimal points in time that we could have stopped, i.e. how far we were from the optimal stopping point in time.

We used bar charts to display our result. In the group a figures, the y-axis denotes value at each server we decided to offload. This is represented by the black bars. The red bars show the value on the optimal servers. They are placed side by side so we can see the comparison. Since we compare with optimal, we can never beat it; therefore, black bars can always be equal to or greater than the red bars. The group b charts represent the time instances at which we decided to stop and offload. The negative/positive values in the time graphs represent whether we decided to stop before or after the point when we observed the optimal server. For example, if we see a value of -5 on the chart, this means that we stopped 5 steps ahead of the optimal point in time. If we see a value of 3 then we stopped 3 steps before the optimal stopping point—a value of 0 means we have offloaded to the best server in the sequence.

As we can see, for the Random(P) model in Figure 7a, never achieves to offload to the optimal server in any of the runs performed. Not only that, the difference between the optimal and achieved MEC server is vast in most of the runs even with a small probability like the one we have set p=0.1. Besides, if we use a higher probability, the chance to offload too early increases, minimising the probability that we hit the best server. Early or delayed stopping is not a good idea. The Figure 7b shows that our model's offset from the optimal stopping times is also significant, with most of the stopping times being far ahead of the optimal point.

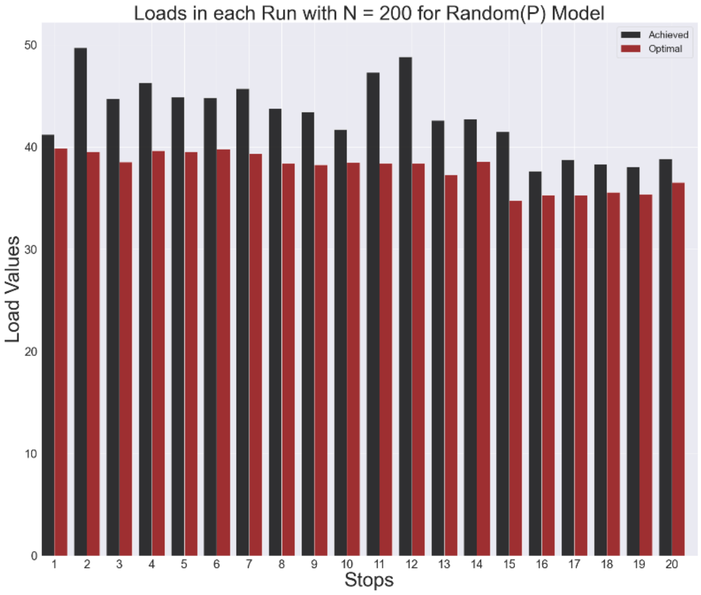


Figure 7a: Random(P) load value/stop chart

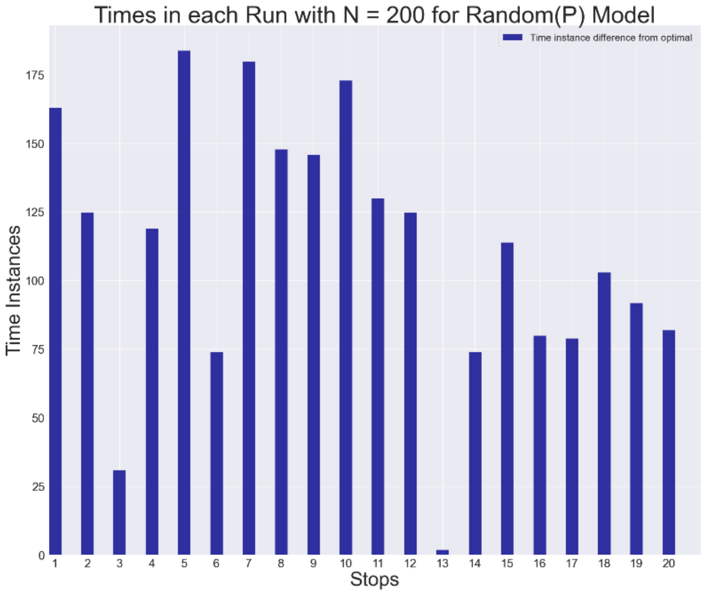


Figure 7b: Stop time instances (Random(P)) chart

Remark 2: ***Implementing the Secretary Model can significantly increase efficiency compared to the Random(P) Model. This is because the Secretary Model's implementation algorithm looks at the MEC servers' actual values and does not rely on a probability.***

From Figure 8a, we can see that the model can find the optimal server to offload most of the runs. The model is far from optimal in *only* 4 runs. The emptiness on the time instances graph confirms that we have found the optimal stopping time in most of the runs. Additionally, in Figure 8b, it is essential to mention that, even though the times achieved show that we have missed the optimal instance by a lot, the server we chose to offload is the next best server after the optimal. ***This proves the efficiency of our implemented Secretary Model.***

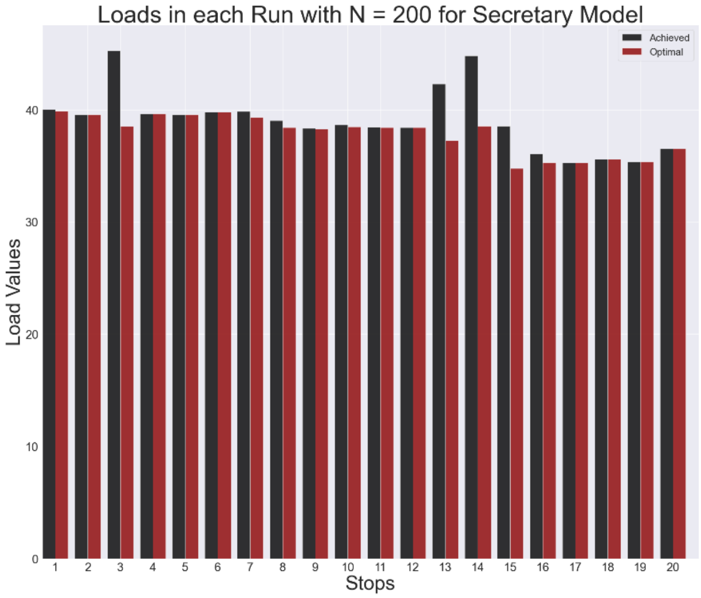


Figure 8a: Secretary load value/stop chart

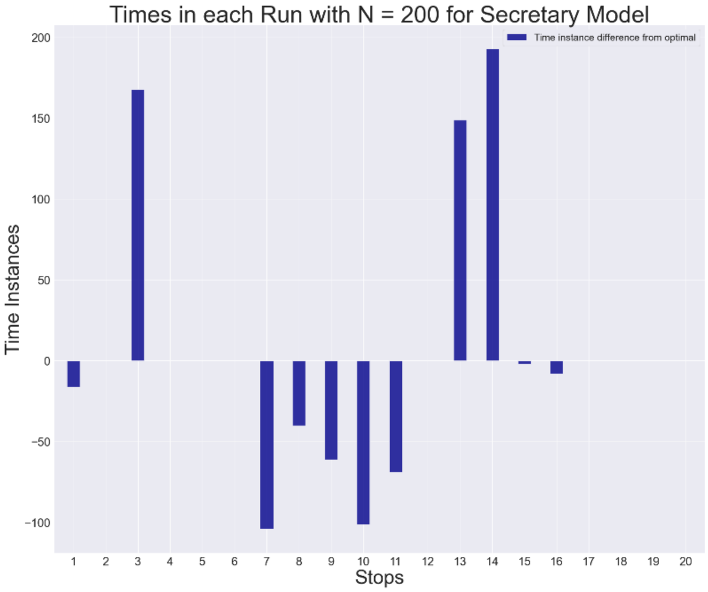


Figure 8b: Stop time instances (Secretary) chart

The House Selling Model relies on a discount factor to execute the offloading decision-making process. As explained in the paper's methodology, the closer this factor is to 0, our model's performance is best. Since using a factor of 0 is not realistic we can bring it down to a low of 0.015, which is the highest value of r at which our House Selling model performs better than the Secretary model.

From the simulation results in Figure 9a, we can see that our implemented model chooses the *best* or the *second-best server* to offload the runs' tasks. ***Our House Selling model solution is proven to be too close to the Optimal Solution***. Also, making r even smaller brings our model closer and closer to the Optimal Solution. The House Selling model outperforms the other two OST-based models by a lot.

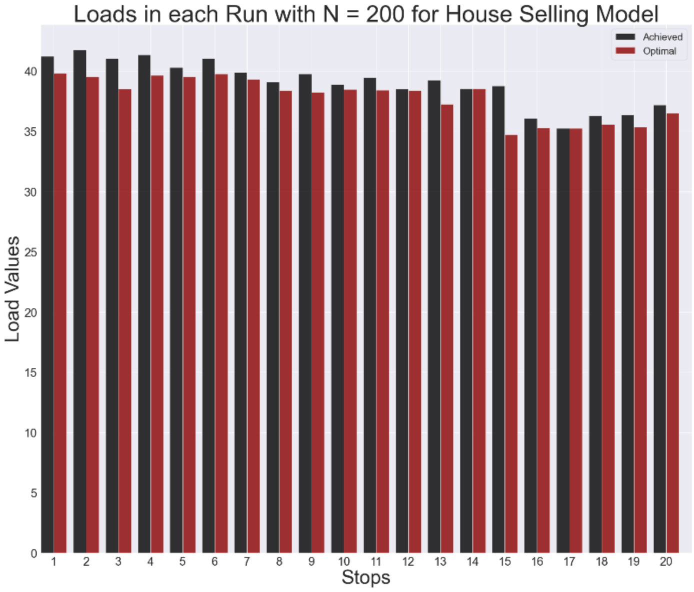


Figure 9a: House Selling load value/stop chart

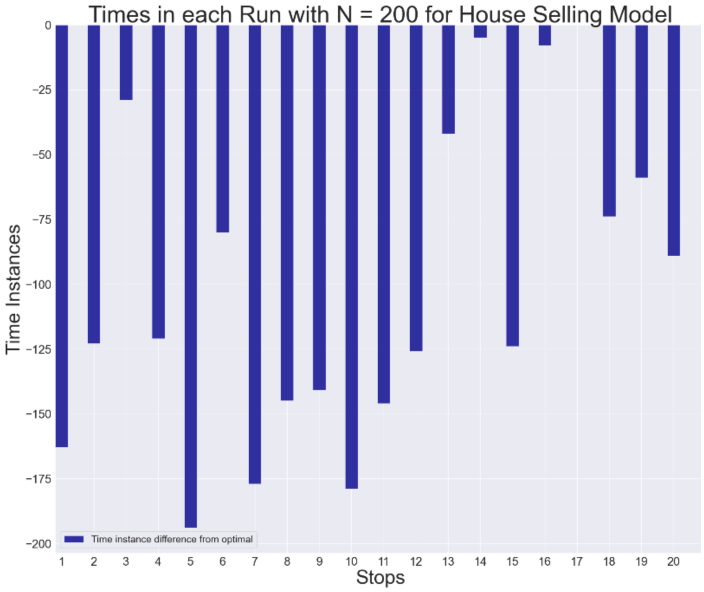
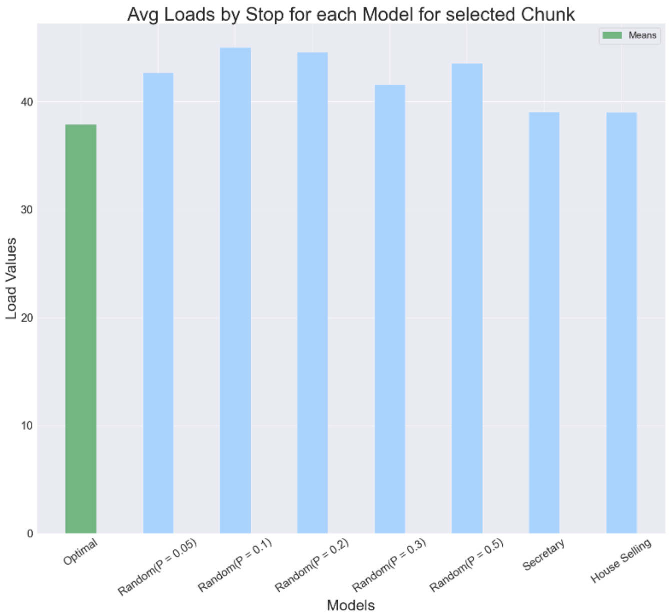


Figure 9b: Stop time instances (House Selling) chart

As mentioned above, the way we developed our analysis tool, offers the user the option to change all the parameters N, p and r to their preference. Same applies for the data set; the tool accepts any CSV formatted file. To test our tool's functionality for this study, we used five different data sets included in the /data/ folder with all other code scripts. After plotting the individual graphs of our designed models and examining performance, we put them together to compare against the Optimal Solution. In this experiment, we have calculated all models' averages for the load values each time we stopped and offloaded. Keeping N = 200 and r the same as we have set for the individual simulations, Figures 10&11 show the Optimal Solution average in green colour, together with our models' average.

Remark 3: ***For the Random(P) model, we have used different values of P to examine how its performance varies compared to the Optimal. To illustrate which model is closest to the Optimal Solution, we have calculated the <u>difference</u> between each models' and the optimal average as shown in Figure 10&11.***



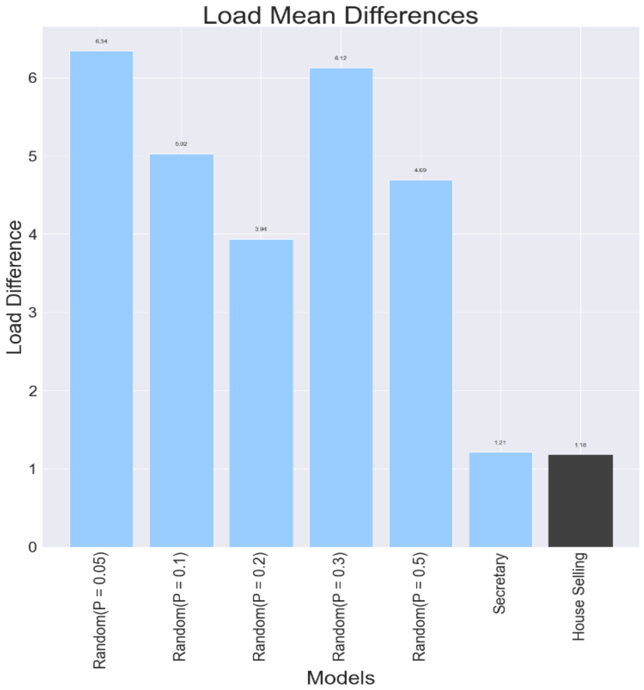
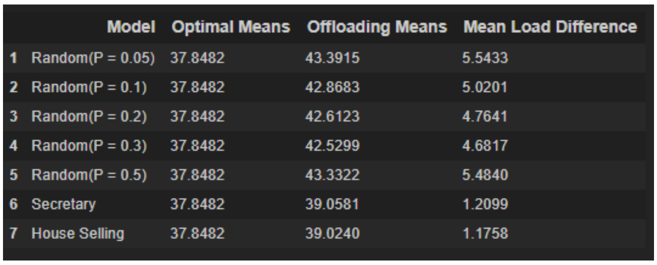


Figure 10 & 11:Average Mean load values of models Vs Optimal Solution

Interestingly, from the graph, we can see that when probability P is set to 0.5, we have a better performance than lower values of P. However, since the Random(P) model relies on randomly generated numbers and a probability, the analysis will produce different results every time we run it. Overall, this Figure suggests that, at r=0.015 for server sequences of N = 200, our implemented House Selling Model <u>*outperforms*</u> all other models with the Secretary model being the next best model. The 4th column in the dataframe from Pandas in Figure 12 below shows the difference of all models from the Optimal.



	Model	Optimal Means	Offloading Means	Mean Load Difference
1	Random(P = 0.05)	37.8482	43.3915	5.5433
2	Random(P = 0.1)	37.8482	42.8683	5.0201
3	Random(P = 0.2)	37.8482	42.6123	4.7641
4	Random(P = 0.3)	37.8482	42.5299	4.6817
5	Random(P = 0.5)	37.8482	43.3322	5.4840
6	Secretary	37.8482	39.0581	1.2099
7	House Selling	37.8482	39.0240	1.1758

Figure 12: Dataframe with Optimal Solutions vs all models average

6.4 The Optimal Value of the Factor *r*

We have proved that the Secretary and the House Selling models have the best performance overall out of our three implemented models. Since our tool gives the user the ability to change the parameter values of N and r, we compared different r factor values (combined values) with several N sizes to examine the House Selling Model against the Secretary approach. This is done to find a threshold of r, for which any value below that gives better performance to the House Selling Model. We already know from our previous simulations that making r smaller improves the decision making of the model. Thus, after examining, we discovered that the threshold values of r factor for each chunk are shown in Table 2.

Sequence Size	Threshold r factor value	Average Load	Difference from Secretary
50	0.052	39.77	0.01
100	0.064	40.14	0.03
150	0.046	39.59	0.07
200	0.015	39.02	0.04

Table 2: Threshold of discount factor r for different N values

7 Conclusions

In this paper, we experiment with load-aware time-series analysis and propose Optimal Stopping Theory-based models capable of taking non-sequential offloading decisions for task offloading between users and MECs. Node devices such as mobiles, determine *when* and on *which* server to offload the tasks considering the current load on each MEC server. Our implemented models can use this information to determine when is the best time to offload tasks and which server. The three models were compared *against* the Optimal Solution in our performance assessment with the House Selling Model being the best and the Secretary Model coming next. The House Selling Model we constructed can take in a discount factor parameter r. Based on this value and the chunk size of the sequence we are observing we outperform other baseline solutions in terms of CPU load. For each chunk size the optimal value of r differs. We examine all cases to find these optimal values so we know which is the threshold perform better than the Secretary model.

In our future research agenda, we plan to investigate and further to extend and evolve this model to look in network conditions and other parameters e.g., experimenting with the degree at which data turn obsolete and othger contextual parameters to depart from the load-aware model to the context-aware task-offloading model.

Notation Table

N	Number of MEC servers in each run
p	Probability set for Random (P) problem
x	Randomly generated variable used in Random (P) problem used for comparison
sample	Sample used in the Secretary Problem
benchmark	Minimum value in the sample
R	Discount factor set for House Selling Problem
K	Number of instances in the
P(R)	Probability selecting the best candidate in Secretary model (2)
$\mathbf{y_k}$	Reward from HS (11)
A	Availability values for HS
$\mathbf{s_k}$	Scaled Availability Values for HS (12)
$\mathbf{a_k}$	Decision Values for HS (13)

8 References

1. Alghamdi, C. Anagnostopoulos and D. P. Pezaros, "On the Optimality of Task Offloading in Mobile Edge Computing Environments," 2019 IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 2019, pp. 1-6, doi: 10.1109/GLOBECOM38437.2019.9014081
2. Alghamdi, Ibrahim; Anagnostopoulos, Christos; P. Pezaros, Dimitrios. 2019. "Delay-Tolerant Sequential Decision Making for Task Offloading in Mobile Edge Computing Environments" *Information* 10, no. 10: 312. https://doi.org/10.3390/info10100312
3. Kolomvatsos, Kostas; Anagnostopoulos, Christos. 2020. "A Deep Learning Model for Demand-Driven, Proactive Tasks Management in Pervasive Computing" *IoT* 1, no. 2: 240-258. https://doi.org/10.3390/iot1020015
4. A. Hekmati, P. Teymoori, T. D. Todd, D. Zhao and G. Karakostasy, "Optimal Multi-Decision Mobile Computation Offloading With Hard Task Deadlines," 2019 IEEE Symposium on Computers and Communications (ISCC), Barcelona, Spain, 2019, pp. 1-8, doi: 10.1109/ISCC47284.2019.8969696.
5. D. Bertsekas.Dynamic Programming and OptimalControl. Athena Scientific, 2017.
6. J. Yan, S. Bi, Y. J. Zhang and M. Tao, "Optimal Task Offloading and Resource Allocation in Mobile-Edge Computing With Inter-User Task Dependency," in IEEE Transactions on Wireless Communications, vol. 19, no. 1, pp. 235-250, Jan. 2020, doi: 10.1109/TWC.2019.2943563.
7. I. Alghamdi, C. Anagnostopoulos and D. P. Pezaros, "Time-Optimized Task Offloading Decision Making in Mobile Edge Computing," 2019 Wireless Days (WD), Manchester, United Kingdom, 2019, pp. 1-8, doi: 10.1109/WD.2019.8734210.
8. D. Callegaro and M. Levorato, "Optimal Computation Offloading in Edge-Assisted UAV Systems," 2018 IEEE Global Communications Conference

(GLOBECOM), Abu Dhabi, United Arab Emirates, 2018, pp. 1-6, doi: 10.1109/GLOCOM.2018.8648099.

9. J. Zhang and K. B. Letaief, "Mobile Edge Intelligence and Computing for the Internet of Vehicles," in Proceedings of the IEEE, vol. 108, no. 2, pp. 246-261, Feb. 2020, doi: 10.1109/JPROC.2019.2947490.

10. Numenta. (n.d.). Numenta/nab. Retrieved February 11, 2021, from https://github.com/numenta/NAB/tree/master/data/realAWSCloudwatch

11. Knowing when to stop. (2018, March 05). Retrieved February 11, 2021, from https://www.americanscientist.org/article/knowing-when-to-stop

12. Optimal Stopping and Applications, UCLA, https://www.math.ucla.edu/

13. Python time series Analysis Tutorial. (n.d.). Retrieved February 11, 2021, from https://www.datacamp.com/community/tutorials/time-series-analysis-tutorial

14. Rs.io. (n.d.). Retrieved February 12, 2021, from https://rs.io/the-secretary-problem-explained-dating/

15. https://github.com/podyssea/Optimal-Stopping-Theory-Models

16. Duso, L. (2020, September 10). Math-based decision making: The secretary problem. Retrieved February 12, 2021, from https://medium.com/cantors-paradise/math-based-decision-making-the-secretary-problem-a30e301d8489

Appendix (Implemeted Functions)

def random_prob_model(counter, probability)	#Apply random probability model on our data
def secretary_model(counter)	#Apply secretary model on our data
def house_selling_model(counter, r)	#Apply house selling model on our data
def randomP_simulation_run(chunk_func, N)	#Produce the simulation results for random_prob_model() #Returns graphs and dataframe
def secretary_simulation_run(chunks)	#Produce the simulation results for secretary_model() #Returns graphs and dataframe
def house_selling_simulation_run(chunks, r)	#Produce the simulation results for house_selling_model() #Returns graphs and dataframe
def avg_loads_by_stop(rpb_model, secretary_model, house_selling_model)	#Produce the results of the comparison results Vs Optimal Solution #Returns grap
def main()	#Calls the main body of the tool