Learning Augmented Algorithms and their applications to queues

Bryce McHugh

3rd year Discrete Mathematics

Supervised by

Prof. Artur Czumaj

Abstract:

Applications to queuing theory are everywhere. Its uses in computing may be immediately obvious, with examples including how a processer prioritises requests, calculating the number of servers needed to keep delays low for a website and how that may vary with time. The uses of queuing theory are not restricted to just to computing with applications including logistics of business and transport. This results in the field being largely studied. However, with the rising prevalence of machine learning, there are now more approaches that were previously unavailable. Learning Augmented Algorithms are one of these approaches. They use a machine learning approach to study some characteristic of the input and use it adapt to its nuances. They aim to combine the speed of a machine learning approach with the guarantees of correctness found in more traditional algorithm design.

Keywords:

Learning-augmented algorithms, Queuing theory, M/G/1 Queue, M/M/1 Queue

CONTENTS

1) INTRODUCTION

Applications to queuing theory are everywhere. Its uses in computing may be immediately obvious, with examples including how a processer prioritises requests, calculating the number of servers needed to keep delays low for a website and how that may vary with time. The uses of queuing theory are not restricted to just to computing with applications including logistics of business and transport. This results in the field being largely studied. However, with the rising prevalence of machine learning, there are now more approaches that were previously unavailable. Learning Augmented Algorithms are one of these approaches. They use a machine learning approach to study some characteristic of the input and use it adapt to its nuances. They aim to combine the speed of a machine learning approach with the guarantees of correctness found in more traditional algorithm design.

This project will primarily focus on M/G/1 queues and the effects that utilising a prediction has on the waiting time of a job in the queue. When a specific distribution is needed for job service time, we will use an exponential distribution reducing to M/M/1 queues. To guide our discussion, we will study the work of Mitzenmacher [1] whilst developing all of the necessary background for such discussion. In doing so, we present the queuing policies, Shortest Predicted Job First (SPJF) and Shortest Predicted Remaining Processing Time (SPRPT) as well as the notion of the price of misprediction which allows us to compare how each policy fares compared to its corresponding version which uses the true processing time of the job. We shall also run simulations to ensure that the analysis we have applies in a practical setting. To conclude, we analyse the simulation results and propose possible solutions for any shortcomings that we have identified.

Literature review:

To create this project, multiple sources of literature were reviewed to build a clear understanding of the topics presented. The following are some brief remarks regarding the contents of the most notable sources.

The work presented at the 14th IGAFIT colloquium by Vassilvitskii [2] helped provide a strong basis of understanding with regards to Learning Augmented Algorithms. The ideas discussed were clear and concise. It also helped provide some understanding of the common real-world instances to which Learning Augmented Algorithms may be applied. It also showcased the two key types of learning augmented algorithms, ones which use a prediction to decern a hidden characteristic that is then used during the run of the algorithm and ones that use a prediction for a “warm start”. The presented example of the first type is the Secretaries Problem with a letter of recommendation, whilst the two examples of the second type are searching and minimum cost matchings.

The work by Mitzenmacher [1] provided insight to the learning augmented policies on M/G/1 queues, especially SPRPT. As this work assumes substantial familiarity with queues, there are several claims left as statements. There is also a methodology for simulating the queues with the different policies.

An integral work for this project was that of Harchol-Balter [3]. This book was primarily example driven which resulted in it being particularly easy to follow. The order in which the material was presented was extremely coherent and allowed for a quick understanding of the topics presented. This material was also instrumental for understanding the SJF, SPJF and SRPT policies.

The work of Gautam [4] provides some useful knowledge regarding the optimal policies for different instances of M/G/1 queuing. As such, this was useful in the selection of policies to focus on when predictions were not involved.

The joint work of Conway, Maxwell and Miller [5] provided insight into commonly assumed theorems and claims as well as other topics such as the residence times of jobs under the SJF policy. Many proofs focused on the use of Laplace transforms which results in them being substantially complex. As such, other literature was preferred when they contained alternative proofs. Due the focus on applications in business and manufacturing, the notations and notions used were different, with the use of deadlines of jobs and flow time being included.

2) LEARNING AUGMENTED ALGORITHMS

To full discuss learning augmented algorithms, we must first define them.

Definition: A Learning Augmented Algorithm is an algorithm that utilises a noisy oracle, which computes some property of the input, as part of its computation. The uses of this oracle should be robust, as in should that oracle give information that is either useless or detrimental to achieving the optimal result, the algorithm should not perform significantly worse. More succinctly, the use of this oracle should not make the algorithm worse.

The common assumption is that this oracle comes from a machine learning approach, which may depend on the problem and other finer details. To that end, our discussion will not focus on how to develop such noisy oracles, we will assume that they exist and can be utilised in our algorithmic design.

It should be noted that there are two common uses for this oracle. One use is as a warm start for the solution. As such the oracle is used to predict the result of the algorithm which is then used as a starting point. This type of Learning Augmented Algorithm is particularly applicable to problems wherein there exists a known optimal algorithm which iteratively alters a current feasible solution. Common examples of these include maximal integral flow with the Ford-Fulkerson algorithm and the searching problem with the doubling binary search approach.

The other approach is wherein the oracle is used to calculate a property which may be hidden and can then be used in the algorithm to inform how is best to precede. The work in this project focuses on this type of approach.

To help understand the benefits of using Learning Augmented Algorithms, we will consider a simple example of how they can perform better than a traditional algorithm whilst being robust to noise from the oracle.

Example: The Secretaries Problem

This example was originally presented in [2].

We will consider a version of the Secretaries Problem wherein we have n candidates for a sole position. Each candidate has a unique score, which is unknown to us until we interview them, that tells us how well they will perform in the position. We assume that the candidates arrive for interviews in an arbitrary order. When we interview a candidate, we learn their score and must decide if we wish to hire them or reject them on the spot. All decisions are final, that is if we reject a candidate, we cannot hire them at a later point. Similarly, should we hire a candidate, all other candidates are rejected or equivalently we cannot consider any more candidates. We shall consider a success as hiring the candidate with the highest score out of all n possible candidates. Any other outcome is considered a failure. To this end, our goal with designing an algorithm for this problem is to maximise our probability of success.

As the score distribution is not known to us and our decisions are final, it should make intuitive sense that there is no algorithm which always selects the best possible candidate.

We will consider the $\frac{1}{e}$ stopping rule policy, defined as such:

- We reject the first $\frac{n}{e}$ candidates, keeping track of the highest score seen among them
- We hire the first candidate that exceeds the highest score that we have seen so far

Note that it does not matter if we end the hiring process without hiring a candidate as that is considered a failure. If we were required to have the position filled, we could hire the last possible candidate to fix this. We will now analyse the efficacy of this policy

Claim: The policy has a success probability of at least $\frac{1}{e}$ and achieves this value in the limit of n

Proof:

Let A_i be the event that candidate i is accepted, B_i the event that that candidate i is the best candidate and F_i be the event that no candidate has been accepted before the ith candidate is considered. We want to find the probability that the best candidate is hired under this algorithm which is the same as

$$\Pr\left[\bigcup_{i=1}^{n} A_i \cap B_i\right]$$

As we cannot accept more than one candidate and exactly one candidate can be the best, A_i and A_j are mutually independent and B_i and B_j are mutually independent for all $i \neq j,\ i,j \in \{1, \dots, n\}$. Due to their mutual independence, we have the following equality

$$\Pr\left[\bigcup_{i=1}^{n} A_i \cap B_i\right] = \sum_{i=1}^{n} \Pr[A_i \cap B_i]$$

We can simplify this further as we know that

$$A_i = \emptyset\ \forall\ i \in \left\{1, \dots, \frac{n}{e}\right\}$$

as our algorithm rejects the first $\frac{n}{e}$ candidates. This then gives

$$\Pr\left[\bigcup_{i=1}^{n} A_i \cap B_i\right] = \sum_{i=\frac{n}{e}+1}^{n} \Pr[A_i \cap B_i]$$

By the definition of conditional probability,

$$\Pr[A_i \cap B_i] = \Pr[A_i|B_i]\Pr[B_i]$$

As the candidate are in a random order,

$$\Pr[B_i] = \frac{1}{n} \ \forall\, i \in \left\{\frac{n}{e}+1, \dots, n\right\}$$

As the algorithm only accepted a candidate if we haven't accepted any candidates yet and they are the best candidate seen so far,

$$\Pr[A_i|B_i] = \Pr[F_i]$$

as i is the best candidate out of all n, they must be the best out of the first i. F_i only occurs when the second-best candidate, candidate j, out of the first i candidates is automatically rejected. Suppose otherwise. Then when candidate j is considered, they will be better than all other candidates considered so far and thus be accepted and prevent us from accepting candidate i. As there are $\frac{n}{e}$ positions for candidate j out of $i-1$ for F_i to occur,

$$\Pr[F_i] = \frac{n/e}{i-1} = \frac{n}{e(i-1)}$$

Putting it all together gives

$$\Pr\left[\bigcup_{i=1}^{n} A_i \cap B_i\right] = \sum_{i=\frac{n}{e}+1}^{n} \frac{1}{n} \times \frac{n}{e(i-1)} = \sum_{i=\frac{n}{e}+1}^{n} \frac{1}{n} \times \frac{n}{e(i-1)} = \frac{1}{e}\sum_{i=\frac{n}{e}+1}^{n} \frac{1}{i-1}$$

$$= \frac{1}{e}\left(\sum_{i=1}^{n}\frac{1}{i-1} - \sum_{i=1}^{\frac{n}{e}}\frac{1}{i-1}\right) = \frac{1}{e}\left(\sum_{i=2}^{n-1}\frac{1}{i} - \sum_{i=2}^{\frac{n}{e}-1}\frac{1}{i}\right)$$

By using the fact that

$$\int_{1}^{N+1}\frac{1}{x}dx < \sum_{i=1}^{N}\frac{1}{i} < \int_{1}^{N+1}\frac{1}{x}dx + 1$$

We get that

$$\Pr\left[\bigcup_{i=1}^{n} A_i \cap B_i\right] \geq \frac{1}{e}\left(\int_1^n \frac{1}{x}dx - \left(\int_1^{\frac{n}{e}} \frac{1}{x}dx + 1\right)\right)$$
$$= \frac{1}{e}\left(\ln(n) - \left(\ln\left(\frac{n}{e}\right) + 1\right)\right) = \frac{1}{e}(\ln(n) - (\ln(n) - \ln(e) + 1))$$
$$= \frac{1}{e}$$

■

Claim: The policy is optimal for this formulation of the Secretaries Problem

This claim is well know and studying the proof would not yield any noteworthy contributions to our discussion. As such, we will take this claim at face value, and shall omit the proof. Thus in our current formulation, we cannot exceed a success probability of $\frac{1}{e}$ in the limit. Now suppose that each candidate also has a letter of recommendation which states if they are the best candidate or not, which is accurate with probability p. In this situation, the letter of recommendation is our noisy oracle. We will now consider how can we utilise this oracle to improve our chance of success. First, it is noteworthy that we can restrict the range of values that p can take.

Claim: We can trivially construct a new oracle with accuracy $p' \in \left[\frac{1}{2}, 1\right]$ using our old oracle

Proof:

Case 1) If $p \in \left[\frac{1}{2}, 1\right]$, we shall just use the result given by the old oracle

Case 2) If $p \in \left[0, \frac{1}{2}\right)$, our new oracle will predict the opposite label to which our old oracle chose. As in if the old oracle says a candidate is not the best, the new oracle will say the candidate is the best and if the old oracle says a candidate is the best, the new oracle will say the candidate is the not best.

Thus

$$\Pr[New\ says\ best \mid Candidate\ is\ best] = \Pr[Old\ says\ not\ best \mid Candidate\ is\ best]$$
$$= 1 - p = p' \in \left[\frac{1}{2}, 1\right]$$

and

$$\Pr[New\ says\ not\ best|Candidate\ is\ not\ best] = \Pr[Old\ says\ best|Candidate\ is\ not\ best]$$
$$= 1 - p\ = p' \in \left[\frac{1}{2}, 1\right]$$

■

Thus, we will only consider oracles with an accuracy of $p \in \left[\frac{1}{2}, 1\right]$. We shall now consider the following policy:

- We reject the first τ candidates, keeping track of the highest score seen so far
- We then hire any candidate before the τ'th candidate if they both exceed the highest score so far and their letter of recommendation says they are the best candidate, rejecting them otherwise. If a candidate has a higher score than the highest score seen so far, we update that accordingly
- Afterwards the τ'th candidate, we hire the first candidate that exceeds the highest score seen so far, regardless of their letter of recommendation

With $\tau = n\left(\frac{1-p}{p}\right)^{1/p} e^{1-\frac{1}{p}}$ and $\tau' = n\, e^{1-\frac{1}{p}}$

Claim: When p = $\frac{1}{2}$, the above policy performs as the $\frac{1}{e}$ stopping rule policy.

This claim should be intuitive. This is because an oracle which is accurate with probability $\frac{1}{2}$ is akin to a coin toss, result assigned to each candidate which provides no additional information. This means if the policy is a good learning augmented policy, it should perform as well as the optimal policy with no information.

Proof of claim:

Subbing in the value for p gives:

$$\tau = n\left(\frac{1-p}{p}\right)^{1/p} e^{1-\frac{1}{p}} = n\left(\frac{1-0.5}{0.5}\right)^{1/0.5} e^{1-\frac{1}{0.5}} = n\ 1^2\ e^{-1} = \frac{n}{e}$$

and

$$\tau' = n\, e^{1-\frac{1}{p}} = n\, e^{1-\frac{1}{0.5}} = n\, e^{-1} = \frac{n}{e}$$

Thus we reject the first $\frac{n}{e}$ candidates and hire the first seen that exceeds the highest score seen so far, which is identical to the $\frac{1}{e}$ stopping rule policy.

■

To justify the work we have done, we need to check that for $p \in \left[\frac{1}{2}, 1\right)$, the naïve policy, which just listens to the oracle, is not an effective policy.

Claim: The naïve policy has an accuracy of $O\left(\frac{1}{n}\right)$ when $p \neq 1$

Proof:

Let A_i be the event that candidate i is accepted, B_i is the event that that candidate i is the best candidate and F_i be the event that no candidate has been accepted before the ith candidate is considered. We want to find the probability that the best candidate is hired under this algorithm which is the same as

$$\Pr\left[\bigcup_{i=1}^{n} A_i \cap B_i\right]$$

As we cannot accept more than one candidate and exactly one candidate can be the best, A_i and A_j are mutually independent and B_i and B_j are mutually independent for all $i \neq j,\ i,j \in \{1, \dots, n\}$. Due to their mutual independence, we have the following equality

$$\Pr\left[\bigcup_{i=1}^{n} A_i \cap B_i\right] = \sum_{i=1}^{n} \Pr[A_i \cap B_i]$$

As we can only accept the ith candidate if all candidates before the ith are rejected,

$$\Pr[A_i \cap B_i] = \Pr[A_i \cap B_i \cap F_i]$$

By multiple uses of the definition of conditional probability, we can write

$$\Pr[A_i \cap B_i \cap F_i] = \Pr[A_i | B_i \cap F_i] \Pr[F_i | B_i] \Pr[B_i]$$

As the candidate are in a random order,

$$\Pr[B_i] = \frac{1}{n} \ \forall\, i \in \{1, \dots, n\}$$

As we can only hire the i^{th} candidate if all prior candidates are rejected,

$$\Pr[F_i|B_i] = \Pr\left[\bigcap_{k=1}^{i-1} A_k{}^c|B_i\right] = \prod_{k=1}^{i-1}\Pr[A_k{}^c|B_i] = \prod_{k=1}^{i-1} p = p^{i-1}$$

As our algorithm hires when the oracle says the best candidate is the best, which occurs with probability p, and our algorithm can only hire said candidate when no one has been hired yet,

$$\Pr[A_i|B_i \cap F_i] = p$$

Thus, substituting in gives

$$\Pr\left[\bigcup_{i=1}^{n} A_i \cap B_i\right] = \sum_{i=1}^{n}\Pr[A_i \cap B_i \cap F_i] = \sum_{i=1}^{n}\Pr[A_i|B_i \cap F_i]\Pr[F_i|B_i]\Pr[B_i]$$
$$= \sum_{i=1}^{n} p \times p^{i-1} \times \frac{1}{n} = \sum_{i=1}^{n}\frac{p^i}{n} = \frac{p(1-p^n)}{n(1-p)} = O\left(\frac{1}{n}\right)$$

■

One thing to note is that the value of p is required to devise the policy. This can however be circumvented by using the rejected candidates to calculate the current accuracy of the oracle, $\tilde{p}$. It should be noted that whilst $\tilde{p} \rightarrow p$ as the number of rejected candidates tends to infinity, the number of candidates that we reject may be rather small in practice allowing for a large discrepancy between the two. This may reduce the accuracy of a policy that uses $\tilde{p}$ as opposed to p however we will not study the effect of using $\tilde{p}$. With this, we conclude our discussion of the Learning Augmented version of the secretaries problem and can start discussing queuing theory.

3) QUEUEING THEORY

3.1) Pre-requisite set up

Before we discuss any specific policies, we shall define all of the elements that we need.

Definition: A queue consists of two components; the queue and something that processes entities that reside in the queue, which we will call a server or servers. Entities that enter the queue will be called jobs. Each job has an amount of processing attached to it. Once a job has received that much processing, it is considered finished and will leave the queue. Each server has its own processing rate which may be different from other servers if there are multiple.

Kendall's notation is often used to denote the type of queue. The first letter corresponds to the type of distribution used for the time between job arrivals. The second letter corresponds to the type of distribution used for the amount of processing jobs take. The number at the end denotes the number of servers in the queue.

For this project, we will mainly consider a specific type of queue known as M/G/1 queues

Definition: The M/G/1 queue is a Markovian arrival, General service distribution and single server queue. M/G/1 queues satisfy the following conditions

- During a unit time interval, the number of jobs that arrive fits a Poisson distribution, with mean λ
    - Equivalently, the time between two jobs joining the queue is drawn from an exponential distribution with mean $\frac{1}{\lambda}$
- The amount of processing each job needs is drawn from an arbitrary distribution with mean $\frac{1}{\mu}$
- There is a single server for processing jobs with fixed processing power

As we are dealing with a single server with fixed processing power, we can refer to the amount of processing each job will need as the time it will require for the server to completely process it. We call this the service time of a job.

When a specific service distribution is needed, we will use an exponential distribution. Whilst this will restrict us to M/M/1 queues, a large amount of the analysis will not call for a specific service distribution. We will also assume that μ = 1 as this makes our expected service time for an arbitrary job equal to 1.

Notations: S is defined to be the distribution of service times. f_S is defined to be the probability density function of the service times.

Definition: The waiting time of a job is defined to be the amount of time it spends in the queue, not including the time it is being serviced. The residence time of a job is defined to be the total length of time in the queue including service time. $\mathbf{E}[W]$ denotes the expected waiting time of an arbitrary job whilst $\mathbf{E}[W(x)]$ denotes the waiting time of a job of length x. $\mathbf{E}[R]$ and $\mathbf{E}[R(x)]$ have similar definition but concern residence times instead. Trivially, $\mathbf{E}[R(x)] = \mathbf{E}[W(x)] + x$ and $\mathbf{E}[R] = \mathbf{E}[W] + \mathbf{E}[S]$.

It should be noted that residence time and waiting time are often interchanged somewhat freely in most literature. Whilst this can be done in most instances, we will keep these notions distinct to avoid confusion in the select cases where they are not freely interchangeable.

Definition: The load on a queue, denoted by ρ, is defined to be the expected amount of processing time that a comes from jobs joining the queue in each unit of time. Similarly, the load on a queue caused by jobs of length less than or equal to x, denoted by ρ_x, is defined to be the expected amount of processing time that a comes from jobs of length less than or equal to x joining the queue in each unit of time. Namely, we have

$$\rho = \lambda \int_{t=0}^{\infty} t\, f_S(t)\, dt$$

$$= \lambda\, \mathbf{E}[S] = \frac{\lambda}{\mu}$$

$$\rho_x = \lambda \int_{t=0}^{x} t\, f_S(t)\, dt$$

Another interpretation of ρ is that it is the expected number of jobs that join the queue during the time it takes to service an arbitrary job. This should be fairly obvious as the $\mathbf{E}[S]$ denotes the amount of time we have for new arrivals and λ is the expected number of arrivals in a unit time period. Multiplying the two gives the value we seek, which is ρ.

It should be noted that when $\rho > 1$, the amount of processing arriving per unit time exceeds the processing power of the server. This results in the queue filling up and the residence time tending to infinity, regardless of the queuing policy. This means we will only consider instances wherein $\rho < 1$.

It should also be noted that when ρ takes small values i.e. less than 0.5, the load on the system is not substantial. This results in low mean residence times regardless of the policy used. As such, we will not consider these cases as they yield little worth when analysing the performance of policies.

Now we will prove a useful theorem which shows the expected length only depends on the arrival rate and the expected residence time.

Theorem:(Little's law)

Let $\mathbf{E}[L]$ be the expected number of jobs in a queue at an arbitrary point. Then,

$$\mathbf{E}[L] = \lambda \mathbf{E}[R]$$

regardless of the queuing policy.

Proof:

This proof follows the style presented in [3]

Let $L(t)$ be the length of the queue at time t, $C(t)$ be the total amount of waiting time from completed jobs at time t, $R(t)$ be the sum of time spent in the queue across all jobs and $Q(t)$ to be the total amount of waiting time that all jobs arriving before or at time t. Then trivially

$$C(t) \leq R(t) \leq Q(t)$$

$R(t)$ can be expressed in terms of the length of the queue as such

$$R(t) = \int_{x=0}^{t} L(x)\, dx$$

as the length of the queue shows how many jobs are currently in it, and thus waiting to be completed. Once they are completed, they leave the queue and no longer contribute to either side. Substituting this in and dividing by t gives

$$\frac{C(t)}{t} \leq \frac{\int_{x=0}^{t} L(x)\, dx}{t} \leq \frac{P(t)}{t}$$

$C(t)$ and $Q(t)$ can be expressed as their averages

$$C(t) = c(t) \times Avg\left(R_{completed}(t)\right)$$

$$Q(t) = q(t) \times Avg\left(R_{queued}(t)\right)$$

where $c(t)$ denotes the number of completed jobs at time t, $q(t)$ denotes the number of jobs that have joined the queue at time t, $Avg(R_{completed}(t))$ denotes the average residence time of jobs completed before time t and $Avg(R_{queued}(t))$ denotes the average residence time of jobs that have join the queue. Subbing this in gives

$$\frac{c(t)}{t} Avg\left(R_{completed}(t)\right) \leq \frac{\int_{x=0}^{t} L(x)\,dx}{t} \leq \frac{q(t)}{t} Avg\left(R_{queued}(t)\right)$$

As $t \to \infty$,

$$\frac{\int_{x=0}^{t} L(x)\,dx}{t} \to \mathbf{E}[L]$$

$$Avg\left(R_{completed}(t)\right) \to \mathbf{E}[R]$$

$$Avg\left(R_{queued}(t)\right) \to \mathbf{E}[R]$$

$$\frac{q(t)}{t} \to \lambda$$

$$\frac{c(t)}{t} \to \gamma$$

The first three limits are trivial, as they are the definition of the values. $\frac{q(t)}{t}$ is the average rate of arrivals within before time t. Thus as $t \to \infty$, $\frac{q(t)}{t}$ must tend to the rate of arrivals, λ. Similarly, $\frac{c(t)}{t}$ tends to the rate of job completion, which we define as γ. Thus, it follows

$$\gamma\, \mathbf{E}[R] \leq \mathbf{E}[L] \leq \lambda\, \mathbf{E}[R]$$

This also gives us $\gamma \leq \lambda$ which makes sense as the rate of job completion cannot exceed the rate of job arrivals. As $\mathbf{E}[S]$ is the expected job service time of an arbitrary job, the expected number of jobs that leave in a given unit of time is $\frac{1}{\mathbf{E}[S]} = \mu$, provided there are a sufficient number of jobs. Then follows that

$$\gamma = \min\{\mu, \lambda\} = \lambda$$

as the rate of job completions is μ unless this exceeds the rate of arrivals, as there cannot be jobs that leave the queue without joining it. Thus

$$\lambda\ \mathbf{E}[R] \leq \mathbf{E}[L] \leq \lambda\, \mathbf{E}[R]$$

■

There is a relatively important distinction that should be made between queuing policies. These are the notions of a preemptive and a non-preemptive policy. They are defined as follows:

Definition: A preemptive policy is a policy that employs the pausing of the job currently being processed. This means that the job currently being processed can be changed without the completion of a job. A non-preemptive policy does not allow for any of these pauses of job processing.

This distinction is of note as there may be instances wherein jobs cannot be interrupted. One such example of this is a queue at a supermarket till. The interrupting of serving the current customer would add significant delays and will likely cause confusion for all the customers in the queue. As such, it makes sense that only non-preemptive policies are allowed. It should also be noted that when preemptive policies are allowed, a non-preemptive policy may still be employed.

Now that we have sufficient notions, we can discuss specific policies.

3.2) FIFO

The first policy we will consider is First-In-First-Out (FIFO). Some use the notation M/G/1/F, M/G/1/FIFO or M/G/1/FCFS to denote that a queue has this policy. FIFO is rather easy to state and understand.

Definition: The First-In-First-Out policy is a non-preemptive policy that organises the jobs in the queue in order that they arrive. Namely, if job x arrives before job y, stays in front of job y in the queue.

It should be noted that the FIFO policy does not need to know any information about each job. We will now analyse the expected residence time of a job under the FIFO policy. To do this, we will use the Pollaczek–Khinchine formula.

Theorem: (Pollaczek–Khinchine formula) A queue using the FIFO policy satisfies the following equality

$$\mathbf{E}[L] = \rho + \frac{\rho^2 + \lambda^2 Var(S)}{2(1-\rho)}$$

The proof of this is presented in [4]. From this, we can derive the residence time for a job under the FIFO policy.

Corollary: When the service times are exponentially distributed, the expected residence time is

$$\mathbf{E}[R] = \frac{1}{1-\rho}$$

Proof:

As S is an exponential distribution,

$$Var(S) = \frac{1}{\mu^2}$$

Therefore,

$$\mathbf{E}[L] = \rho + \frac{\rho^2 + \frac{\lambda^2}{\mu^2}}{2(1-\rho)} = \rho + \frac{2\rho^2}{2(1-\rho)} = \frac{\rho - \rho^2 + \rho^2}{1-\rho} = \frac{\rho}{1-\rho}$$

Using little’s law, we get

$$\mathbf{E}[R] = \frac{\mathbf{E}[L]}{\lambda} = \frac{1}{1-\rho}$$

■

This shows that the FIFO policy has particularly poor performance as $\rho \to 1$. For example, when $\rho = 0.99$, $\mathbf{E}[W] = 100$ which is a significant amount of time. This, in tandem with the following theorem, highlights how poorly queues that do not use any information can perform.

Theorem: All non-preemptive queuing policies that do not make use of job service time have the same distribution of the number of jobs in the queue

The proof of this will be omitted for brevity but can be found in [5]. This allows us to use the FIFO policy as a baseline for comparison to show how much of an improvement that using job service times can make to the expected waiting time of a policy. Whilst a preemptive policy that does not need to know any information about each job can be developed, the improvements to waiting time will be somewhat negligible when compared to policies that uses either full information or a prediction. As such, we will not develop one and use the FIFO policy as our basis of comparison for both non-preemptive and preemptive policies.

We require one last theorem which can be proved easily now that we have developed the sufficient machinery for FIFO

Theorem: The expected amount of work in the server given that it is currently processing a job, $\mathbf{E}[S_e]$, is

$$\mathbf{E}[S_e] = \frac{\mathbf{E}[S^2]}{2\mathbf{E}[S]}$$

Proof:

This proof follows the style presented in [3]

First, we need another formula for the expected waiting time of an arbitrary job

$$\begin{aligned}\mathbf{E}[W] &= \mathbf{E}[Work\ in\ the\ queue] + \mathbf{E}[Work\ remiaining\ in\ the\ server] \\ &= \mathbf{E}[S](\mathbf{E}[L]-1) + \Pr[Server\ processing\ a\ job]\,\mathbf{E}[S_e] \\ &= \lambda\mathbf{E}[S]\mathbf{E}[W] + \rho\mathbf{E}[S_e] = \rho\mathbf{E}[W] + \rho\mathbf{E}[S_e] \\ &= \frac{\rho}{1-\rho}\mathbf{E}[S_e]\end{aligned}$$

Rearranging gives

$$\mathbf{E}[S_e] = \frac{\mathbf{E}[W](1-\rho)}{\lambda\mathbf{E}[S]} = \frac{\left(\frac{\mathbf{E}[L]}{\lambda} - \mathbf{E}[S]\right)(1-\rho)}{\lambda\mathbf{E}[S]}$$

Using the Pollaczek–Khinchine formula we get

$$\begin{aligned}\mathbf{E}[S_e] &= \frac{\left(\frac{\rho}{\lambda} + \frac{\rho^2 + \lambda^2 Var(S)}{2\lambda(1-\rho)} - \mathbf{E}[S]\right)(1-\rho)}{\lambda\mathbf{E}[S]} = \frac{\rho^2 + \lambda^2 Var(S)}{2\lambda^2\mathbf{E}[S]} = \frac{\mathbf{E}[S]^2 + Var(S)}{2\mathbf{E}[S]} \\ &= \frac{\mathbf{E}[S^2]}{2\mathbf{E}[S]}\end{aligned}$$

■

It should be noted that the expected amount of work in the server given that it is currently processing a job does not depend on the queuing policy. This formula will be useful for calculating the waiting and residence times for the remaining policies we will develop. Because of this, we will now discuss SJF.

3.3) SJF

Definition: The Shortest-Job-First policy is a non-preemptive that policy organises the jobs in the queue in order of service time. Namely, if job x has less service time than job y, job x stays in front of job y in the queue until job x leaves the queue. The only exception to this rule is if job x arrives whilst job y is being processed as this policy is non-preemptive.

Another way of thinking about a SJF queue is wherein the server and the queue are somewhat separate entities. The queue sorts its contents of jobs by ascending service time with the queue plucking the first job in the queue once it has finished its current job. As such, the job currently being processed is “protected” from the reorganisation of the queue. We now wish to calculate the residence time of an arbitrary job under the SJF policy. To do this, we calculate the waiting time using the following theorem.

Theorem: The expected waiting time of a job with service time x under the SJF policy is

$$\mathbf{E}[W(x)] = \frac{\rho\ \mathbf{E}[S^2]}{2\mathbf{E}[S](1-\rho_x)^2}$$

And thus, the expected waiting time of a job of arbitrary length is

$$\mathbf{E}[W] = \int_{x=0}^{\infty} f_S(x)\ \mathbf{E}[W(x)]\ dx$$

Due to the similarities with a later proof, we will omit this proof. Instead, we will look at the intuition as to what each component of $\mathbf{E}[W(x)]$ corresponds to. The first part that we will look at is $\frac{\rho\ \mathbf{E}[S^2]}{2\mathbf{E}[S]}$. This is the probability is the expected amount of work that the server is processing when our job joins the queue and can be written as $\rho\mathbf{E}[S_e]$. The first $\frac{1}{1-\rho_x}$ term comes for the expected amount of work that our job has to wait for caused by jobs of length less than or equal to x which are in the queue before our job joins the queue. The other $\frac{1}{1-\rho_x}$ term comes from the jobs of length less than x which join the queue after our job joins the queue. Using this, we can now return to our specific M/M/1 instance. By substituting in the appropriate values, we get that

$$\mathbf{E}[W] = \int_{x=0}^{\infty} \frac{\lambda e^{-x}}{(1-\lambda(1-(1+x)e^{-x}))^2}\,dx$$

$$\mathbf{E}[R] = \int_{x=0}^{\infty} \frac{\lambda e^{-x}}{(1-\lambda(1-(1+x)e^{-x}))^2}\,dx \;+1$$

Whilst this equation does not simplify further, it can be numerically evaluated. The following results were generated using WolframAlpha.

λ	$\mathbf{E}[W]$	$\mathbf{E}[R]$
0.5	0.712686	1.712686
0.6	0.962457	1.962457
0.7	1.3122	2.3122
0.8	1.88222	2.88222
0.9	3.19691	4.19691
0.95	5.26396	6.26396
0.98	10.2849	11.2849
0.99	17.4507	18.4507
0.999	115.817	116.817

When compared to the residence time of jobs under the FIFO policy, we can see that SJF has significantly better average residence times than FIFO. This is especially noticeable as the load increases

λ	$\mathbf{E}[R]$ under SJF	$\mathbf{E}[R]$ under FIFO	% reduction in residence time
0.5	1.712686	2	14.4%
0.6	1.962457	2.5	21.5%
0.7	2.3122	3.333333	30.6%
0.8	2.88222	5	42.4%
0.9	4.19691	10	58.0%
0.95	6.26396	20	68.7%
0.98	11.2849	50	77.4%
0.99	18.4507	100	81.5%
0.999	116.817	1000	88.3%

However, we can do much better provided preemptive policies are allowed. For this, we turn to the SRPT policy.

3.4) SRPT

Definition: The Shortest-Remaining-Processing-Time policy is a preemptive policy that organises the jobs in the queue in order of their remaining processing time. Namely, suppose job x has service time s_x and has receive p_x amount of processing and job y has service time s_y and has receive p_y amount of processing. Then if $s_x - t_x$ is less than $s_y - t_y$, job x will be in front of job y in the queue.

Another way of thinking about the SRPT policy is that jobs are organised in order of ascending remaining processing time, where remaining processing time is the amount of time that the server takes to complete the job in its current state. Whilst at first glance this policy may seem like a preemptive variant of the SJF policy, it is in fact much more powerful. We will consider a small example to illustrate this.

Suppose we have an empty queue and two jobs, x and y, join time t and $t + 1.4$ respectively. Suppose that $s_x = 1.5$ and $s_y = 1$. Then when y arrives, $t_x = 1.4$ and trivially $t_y = 0$. It is clear to see that under a preemptive SJF that the residence times for x and y are

$$W_x = 1.4 + 1 + 0.1 = 2.5, W_y = 1$$

for a total waiting time of 3.5. Under SRPT, the waiting times are

$$W_x = 1.4 + 0.1 = 1.5, W_y = 0.1 + 1 = 1.6$$

for a total waiting time of 3.1. This example shows that SRPT is strictly better than preemptive SJF when a job of shorter length but longer remaining processing time than the current job being processed joins the queue. We will now calculate the residence time of a job under the SRPT

Theorem: The expected residence time of a job with service time x under the SRPT policy is

$$\mathbf{E}[R(x)] = \frac{\lambda\left(\int_{t=0}^{x} t^2 f_s(t)dt + x^2\left(1 - \int_{t=0}^{x} f_s(t)dt\right)\right)}{2(1-\rho_x)^2} + \int_{t=0}^{x} \frac{1}{1-\rho_t}\, dt$$

The proof of this is involved and would likely yield little that a discussion of the intuition behind each term would provide. As such we will take this approach. The two distinct sections of this equation have a clear interpretation. The first term equates to the time the job spends in the queue before it first receives service. The second term equates to the time that the job takes to be serviced once the queue begins servicing it, including interruptions. We shall first discuss the simpler section, the time it takes for a job to be serviced once it first reaches the server.

The integral being from 0 to x can be thought of as the job being serviced starting with remaining time x and decreasing down to 0. The term $\frac{1}{1-\rho_t}$ can be thought of as the expected amount of work that joins the queue whilst the job in question has remaining processing time t.

For the other term, we will break it down into its major components. This decomposition is

$$\frac{\lambda\left(\int_{t=0}^{x} t^2 f_s(t)dt + x^2\left(1-\int_{t=0}^{x} f_s(t)dt\right)\right)}{2} \times \frac{1}{1-\rho_x} \times \frac{1}{1-\rho_x}$$

Like with SJF, the two terms come from the jobs in the jobs in the queue that have service time less than or equal to x and from the jobs with service time less than x that join the queue after our job joins the queue respectively. The first term is similar to the term $\rho\mathbf{E}[S_e]$ in SJF but it must take a slightly different value due to the fact that SRPT is preemptive. The first term is caused by jobs of length less than or equal to x and as such, this can be calculated as if we were calculating the second moment. The second term is from the jobs of length greater than x. If this job has a remaining processing time of length greater than x, then our job joining the queue would interpret it, meaning it cannot contribute at all to the waiting time of our job. Thus that a job of processing time greater than x can contribute to this 'second moment' is x^2. Note that $\frac{\rho}{2\mathbf{E}[S]}$ can be simplified to $\frac{\lambda}{2}$. This concludes our discussion of the intuition behind the residence time of jobs under SRPT. We can use this equation and values under our specific M/M/1 instance to get that

$$\mathbf{E}[R(x)] = \frac{\lambda(2e^{-x}(x^2+x+1))}{2(1-\lambda(1-(1+x)e^{-x}))^2} + \int_{t=0}^{x} \frac{1}{1-\lambda(1-(1+t)e^{-t})}\,dt$$

and thus we also have

$$\mathbf{E}[R] = \int_{x=0}^{\infty} e^{-x}\,\mathbf{E}[R(x)]\,dx$$

Whilst this equation does not simplify further, it can be numerically evaluated. The following results were generated using WolframAlpha.

λ	$\mathbf{E}[W]$	$\mathbf{E}[R]$
0.5	0.425373	1.425373
0.6	0.604094	1.604094
0.7	0.874567	1.874567
0.8	1.352773	2.352773
0.9	2.552125	3.552125
0.95	4.541011	5.541011
0.98	9.494747	10.49475
0.99	16.62693	17.62693
0.999	114.9328	115.9328

We can do similar comparison to FIFO as we did with SJF. The following table contains the results for SJF as well for comparison purposes.

λ	$\mathbf{E}[R]$ under SJF	$\mathbf{E}[R]$ under FIFO	% reduction in residence time for SRPT	% reduction in residence time for SJF
0.5	1.712686	2	28.7%	14.4%
0.6	1.962457	2.5	35.8%	21.5%
0.7	2.3122	3.333333	43.8%	30.6%
0.8	2.88222	5	52.9%	42.4%
0.9	4.19691	10	64.5%	58.0%
0.95	6.26396	20	72.3%	68.7%
0.98	11.2849	50	79.0%	77.4%
0.99	18.4507	100	82.4%	81.5%
0.999	116.817	1000	88.4%	88.3%

Comparing these values with values from those of SJF, we can see that SRPT performs notably better. This is expected as we allow for jobs to be interrupted to allow for a shorter job to be processed. However, the performance guarantees of SRPT are much stronger than just being better than SJF.

Theorem: When the aim is to minimise the waiting time (and thus also residence time) of a job in a M/G/1 queue wherein the service times are known, SRPT is an optimal policy.

The proof of this can be found in [4]. From this, we will use SRPT as the goal we aim to achieve when preemptive policies are permitted.

Whilst SJF and SRPT grant significant improvements to waiting times, they require full knowledge of the job service times. This is a significant limitation as it is likely that they are unknown. To this end, should job service times be unknown, we would have to resort to FIFO which is less than desirable. We shall now study the effects of including a prediction with the aim of developing policies that have better performance.

4) QUEUES WITH PREDICTION

We shall first start with an example of when a queue with prediction may be relevant. Suppose we are running a call centre for a company. Naturally, we are aiming to minimise the time the average person spends waiting as large waiting times may make customers seek out a different company. Without any information, the best we can do is use the FIFO policy as knowing how long helping each customer will take before helping them is not known. This will result in some rather large waiting times as previously demonstrated. Now suppose that we ask each customer what they require before they join the queue. Depending on how the information is received, we can use an appropriate machine learning approach to process this and generate a prediction as to how long each call will take. Two examples of this are a multiple-choice form and asking for a verbal description of what they require.

For a small multiple-choice form, a classifier would be used which could be done with something as simple as a decision tree. This however can be reduced to scheduling with classes, including noise if you wish to model the fact that a customer can select a response that does not quite fit what they need. As such, this approach is not appropriate for the models we have seen so far. As a result, this type of approach will not be discussed further.

On the other hand, for a verbal response we may consider using a natural language processing method to assign a prediction of service time. This then would give us a continuous range of predicted service times allowing us to employ the type of queuing that we will be considered. Similarly, if the multiple-choice form is sufficiently large, it may be more convenient to model the range of predicated service time in a continuous fashion. This would then also allow us to employ the work that we will devise.

Naturally, the prediction will not always align with the actual time taken but this is preferable to knowing nothing. This may allow us to reduce the time that customers will spend waiting. Should we also wish to allow our call centre operators to put a customer on hold, we can do this by allowing preemptive policies. It should be noted that whilst it is highly unlikely that there would only be one person answering calls, resulting in the queue being an M/G/1 queue, the sentiment still stands.

Another possible example could be a teacher marking the work of their students. Suppose that the teacher wishes to return each student's work at the earliest possible point. Also suppose that within a negligible amount of time, the teacher can glance over their work to see how long it may take to make based on factors such as the student's handwriting, the frequency of incoherent sentences, how long it has taken to decipher each incoherent sentence as well as how many comments they need to write for a given student. Then whilst the true time it takes to mark each piece of work is unknown, a prediction can easily be made. This forms a queue with predictions.

Now we have idea as to how common instances wherein it is impossible to know the service time of a job, but it is easy to make a prediction regarding their length, we shall now devise the theory that allows us to deal with such instances. As we are dealing with a prediction, there are several new definitions and notations that are required to allow for meaningful discussion.

Notation: g is the joint distribution of jobs service time and predicted service time. We denote X as the job service domain and Y as the predicted service time domain. It can also be said that $g(x,y)$ is the joint probability that a job has service time equal to x and predicted service time equal to y.

As such, it holds that

$$f_S(x) = \int_{y=0}^{\infty} g(x,y)\,dy$$

$$f_P(y) = \int_{x=0}^{\infty} g(x,y)\,dx$$

where f_P denotes the probability density function of predicted services times.

It should be noted that $f_S(x)$ is the same as in section 3) for each given instance.

Definition: The load on a queue caused by jobs of predicted length less than or equal to y, denoted by ρ'_y, is defined to be the expected amount of processing time that a comes from jobs of predicted length less than or equal to y joining the queue in each unit of time. ρ'_y can be calculated as follows

$$\rho'_y = \lambda \int_{t=0}^{y} \int_{x=0}^{\infty} x\, g(x,t)\, dx\, dt$$

It should be note that introducing ρ' would be redundant as

$$\rho' = \lambda \int_{y=0}^{\infty} \int_{x=0}^{\infty} x\, g(x,y)\, dx\, dy = \lambda \int_{x=0}^{\infty} x \int_{y=0}^{\infty} g(x,y)\, dy\, dx = \lambda \int_{x=0}^{\infty} x\, f_s(x)\, dx = \rho$$

As for our M/M/1 instance, the values for each function are as follows. Each has been presented in its simplest form

$$g(x,y) = \frac{1}{x} e^{-x - \frac{y}{x}}$$

$$f_s(x) = e^{-x}$$

$$f_P(y) = \int_{x=0}^{\infty} \frac{e^{-x-\frac{y}{x}}}{x}\, dx$$

$$\rho_x = \lambda(1 - (1+x)e^{-x})$$

$$\rho'_y = \lambda \int_{t=0}^{y} \int_{x=0}^{\infty} e^{-x - \frac{t}{x}}\, dx\, dt$$

$f_s(x)$ follows from the case with full information. As it is an exponential distribution with mean 1, this is trivially known. From this, we can also easily calculate ρ_x. The nuance comes from finding a suitable equation for $g(x,y)$. We know that

$$\Pr[Y = y |\, X = x] = \frac{1}{x} e^{-\frac{y}{x}}$$

from the definition of our prediction and thus we can see that

$$\frac{1}{x} e^{-\frac{y}{x}} = \Pr[Y = y |\, X = x] = \frac{g(x,y)}{f_s(x)} = \frac{g(x,y)}{e^{-x}}$$

Rearranging gives our value of $g(x, y)$. From this, we simply substitute into the equations for $f_P(y)$ and ρ'_y. It should be noted that they lack closed forms and the double integral in ρ'_y can not be rearranged due to $e^{-x-\frac{t}{x}}$ being undefined, and thus not continuous, at $x = 0$.

To aid our comparison of the efficacy of the policies we present, we require a score for how well they perform. For this, we introduce the price of misprediction

Definition: The price of misprediction of a learning augmented policy is equal to

$$\frac{\mathbf{E}[R']}{\mathbf{E}[R]}$$

where $\mathbf{E}[R']$ is the expected residence time of a job under the learning augmented policy and $\mathbf{E}[R]$ is the expected residence time of a job under the equivalent policy using full information. An equivalent definition is where $\mathbf{E}[R]$ is the expected residence time of a job under the learning augmented policy but the prediction used is noiseless, i.e. $g(x, y) = 0\ \forall x \neq y$. This second definition should be avoided when implemented as it would assign $f_P(y) = f_s(x) = 0$. However, it is useful for understanding. Now that we have the notions we need, we shall consider our first Learning Augmented policy, SPJF.

4.1) SPJF

The Shortest-Predicted-Job-First policy is similar to the SJF policy but instead uses the predicted service time of a job when organising the queue. As such, its definition is rather similar.

Definition: The Shortest-Predicted-Job-First policy is a non-preemptive policy that organises the jobs in the queue in order of service time. Namely, if job x has a smaller predicted service time than job y, job x stays in front of job y in the queue until job x leaves the queue. The only exception to this rule is if job x arrives whilst job y is being processed as this policy is non-preemptive.

The same intuition as with SJF can also be used here, all be it with predicted service times instead. It should be no surprise that the waiting times of jobs under SPJF have a rather similar equation to those under the SJF, with the predicted waiting times being the main focus.

Theorem: The expected waiting time of a job with predicted service time y under the SPJF policy is

$$\mathbf{E}[W'(y)] = \frac{\rho\, \mathbf{E}[S^2]}{2\mathbf{E}[S](1-\rho'_y)^2}$$

And thus, the expected waiting time of a job of arbitrary length is

$$\mathbf{E}[W'] = \int_{y=0}^{\infty} f_P(y)\, \mathbf{E}[W'(y)]\, dy$$

Proof:

This proof follows the style presented in [3]

To do this, we will consider discrete priority queuing and considering what happens as the number of priority classes tend to infinity. We will consider n class priority queuing. Each class k has a job service time drawn from service distribution S_k, and takes priority over jobs of class $k+1, k+2, \dots, n$. We also require that class k has an arrival rate of λ_k. From this, when looking at the queue, we see that

$$\sum_{k=1}^{n} \lambda_k = \lambda, \rho_k = \lambda_k \mathbf{E}[S_k], \mathbf{E}[S] = \frac{1}{\lambda}\sum_{k=1}^{n} \lambda_k \mathbf{E}[S_k], \sum_{k=1}^{n} \rho_k = \rho$$

Where ρ_k denotes the load on the system caused by jobs of class k rather than our usual definition. Now we will consider the waiting times of a job of each class, $\mathbf{E}[W_k]$. For class k, we need to wait for

1) The expected amount of work left from the job the server is currently processing
2) All jobs of class 1 to k which have joined the queue before our job in question to be processed
3) All jobs of class 1 to $k-1$ which join the queue after our job joins the queue

The first term is trivially $\rho\mathbf{E}[S_e]$ which we have seen derived several times before. The second term can be expressed as

$$\sum_{i=1}^{k} \boldsymbol{E}[\#of\ jobs\ of\ class\ i\ in\ the\ queue]\, \mathbf{E}[S_k]$$

By using little's law, we get that this can be rewritten as

$$\sum_{i=1}^{k} \boldsymbol{E}[W_i]\,\lambda_i \mathbf{E}[S_i] = \sum_{i=1}^{k} \boldsymbol{E}[W_i]\,\rho_i$$

We will leave it in this form for now. As for the third term, we have $\mathbf{E}[W_k]$ time for new jobs to arrive. The expected amount for work from jobs joining the queue of class i each unit length of time is ρ_i. Combining this gives that our third term is equal to

$$\boldsymbol{E}[W_k]\sum_{i=1}^{k-1} \rho_i$$

Combining these terms together, we get that

$$\mathbf{E}[W_k] = \rho\mathbf{E}[S_e] + \sum_{i=1}^{k} \boldsymbol{E}[W_i]\,\rho_i + \boldsymbol{E}[W_k]\sum_{i=1}^{k-1} \rho_i \; = \rho\mathbf{E}[S_e] + \sum_{i=1}^{k-1} \boldsymbol{E}[W_i]\,\rho_i + \boldsymbol{E}[W_k]\sum_{i=1}^{k} \rho_i$$

Rearranging this gives

$$\mathbf{E}[W_k] = \frac{\rho\mathbf{E}[S_e] + \sum_{i=1}^{k-1} \boldsymbol{E}[W_i]\,\rho_i}{1 - \sum_{i=1}^{k} \rho_i}$$

We can then get that

$$\mathbf{E}[W_k] - \mathbf{E}[W_{k-1}] = \frac{\rho\mathbf{E}[S_e] + \sum_{i=1}^{k-1} \boldsymbol{E}[W_i]\,\rho_i}{1 - \sum_{i=1}^{k} \rho_i} - \frac{\rho\mathbf{E}[S_e] + \sum_{i=1}^{k-2} \boldsymbol{E}[W_i]\,\rho_i}{1 - \sum_{i=1}^{k-1} \rho_i}$$
$$= \frac{\boldsymbol{E}[W_{k-1}](\rho_{k-1} + \rho_k)}{1 - \sum_{i=1}^{k} \rho_i}$$

$$\mathbf{E}[W_k] = \boldsymbol{E}[W_{k-1}]\left(1 + \frac{(\rho_{k-1} + \rho_k)}{1 - \sum_{i=1}^{k} \rho_i}\right) = \boldsymbol{E}[W_{k-1}]\frac{1 - \sum_{i=1}^{k-2} \rho_i}{1 - \sum_{i=1}^{k} \rho_i}$$

Using proof by induction, it follows that

$$\mathbf{E}[W_k] = \frac{\rho\mathbf{E}[S_e]}{(1 - \sum_{i=1}^{k} \rho_i)(1 - \sum_{i=1}^{k-1} \rho_i)}$$

As for

$$\mathbf{E}[W_1] = \frac{\rho\mathbf{E}[S_e] + \sum_{i=1}^{0} \boldsymbol{E}[W_i]\,\rho_i}{1 - \sum_{i=1}^{1} \rho_i} = \frac{\rho\mathbf{E}[S_e]}{1 - \sum_{i=1}^{1} \rho_i} = \frac{\rho\mathbf{E}[S_e]}{(1 - \sum_{i=1}^{1} \rho_i)(1 - \sum_{i=1}^{0} \rho_i)}$$

And for $k > 1$,

$$\mathbf{E}[W_k] = \frac{\rho\mathbf{E}[S_e]}{(1-\sum_{i=1}^{k-1}\rho_i)(1-\sum_{i=1}^{k-2}\rho_i)}\frac{1-\sum_{i=1}^{k-2}\rho_i}{1-\sum_{i=1}^{k}\rho_i} = \frac{\rho\mathbf{E}[S_e]}{(1-\sum_{i=1}^{k}\rho_i)(1-\sum_{i=1}^{k-1}\rho_i)}$$

Thus, we have our formula for our discrete class case. For SPJF, we shall define the k^{th} class to be the jobs whose predicted service times lie in the region $\left[\frac{T(k-1)}{n}, \frac{Tk}{n}\right)$, for some sufficiently large arbitrary T. Thus as $n \to \infty$, this version of the discrete priority queuing tends to our formulation for SPJF queuing and

$$\sum_{i=1}^{k-1}\rho_i \to \rho'_{\frac{T(k-1)}{n}} \to \rho'_{\frac{Tk}{n}}, \sum_{i=1}^{k}\rho_i \to \rho'_{\frac{Tk}{n}}$$

From this, we can then calculate that for a job of predicted length y, we have that

$$\mathbf{E}[W'(y)] = \frac{\rho\,\mathbf{E}[S_e]}{(1-\rho'_y)^2} = \frac{\rho\,\mathbf{E}[S^2]}{2\mathbf{E}[S](1-\rho'_y)^2}$$

■

This is the furthest we can go without a specific distribution for service time. Going back to our specific M/M/1 instance, we can substitute values in to get that

$$\mathbf{E}[W'] = \int_{y=0}^{\infty} \frac{\lambda\int_{x=0}^{\infty}\frac{e^{-x-\frac{y}{x}}}{x}\,dx}{(1-\lambda\int_{t=0}^{y}\int_{x=0}^{\infty}e^{-x-\frac{t}{x}}\,dx\,dt)^2}\,dy$$

$$\mathbf{E}[R'] = \int_{y=0}^{\infty} \frac{\lambda\int_{x=0}^{\infty}\frac{e^{-x-\frac{y}{x}}}{x}\,dx}{(1-\lambda\int_{t=0}^{y}\int_{x=0}^{\infty}e^{-x-\frac{t}{x}}\,dx\,dt)^2}\,dy + 1$$

Again, this equation does not simplify further but it can be numerically evaluated. The following results were generated using WolframAlpha.

λ	$\mathbf{E}[W']$	$\mathbf{E}[R']$
0.5	0.794808	1.794808
0.6	1.10862	2.10862
0.7	1.57257	2.57257
0.8	2.37578	3.37578
0.9	4.361	5.361
0.95	7.65368	8.65368
0.98	15.9501	16.9501
0.99	28.0535	29.0535
0.999	199.222	200.222

Now using the equation for $\mathbf{E}[R]$ that we derived for SJF, we can calculate the price of misprediction for SPJF in the arbitrary setting

$$\frac{\mathbf{E}[R']}{\mathbf{E}[R]} = \frac{\int_{y=0}^{\infty} f_P(y)\,\mathbf{E}[W'(y)]\,dy + 1}{\int_{x=0}^{\infty} f_s(x)\,\mathbf{E}[W(x)]\,dx + 1}$$

$$= \frac{\int_{y=0}^{\infty} \frac{f_P(y)}{(1-\rho'_y)^2}\,dy + 1}{\int_{x=0}^{\infty} \frac{f_s(x)}{(1-\rho_x)^2}\,dx + 1}$$

Again, we can simplify no further. To calculate the price of misprediction for our M/M/1 instance, we can either sub the appropriate values into the equation, to get

$$\frac{\mathbf{E}[R']}{\mathbf{E}[R]} = \frac{\displaystyle\int_{y=0}^{\infty} \frac{\int_{x=0}^{\infty} \frac{e^{-x-\frac{y}{x}}}{x}\,dx}{\left(1-\lambda\int_{t=0}^{y}\int_{x=0}^{\infty} e^{-x-\frac{t}{x}}\,dx\,dt\right)^2}\,dy + 1}{\displaystyle\int_{x=0}^{\infty} \frac{e^{-x}}{(1-\lambda(1-(1+x)e^{-x}))^2}\,dx + 1}$$

and calculate numerically for given values of λ, or we can utilise the results from the tables computed prior. Follows is the price of misprediction calculated by using the later method

λ	**Price of misprediction**
0.5	1.047949
0.6	1.07448
0.7	1.112607
0.8	1.171243
0.9	1.277368
0.95	1.381503
0.98	1.502016
0.99	1.574656
0.999	1.71398

From this, we can see that the performance of the queue under SPJF does not cause a substantial increase in waiting times when compared to SJF. It is also not surprising that when the load on the system increases, the price of misprediction increases. As all policies perform roughly as well on low loads, we should expect the price of misprediction to be very close to 1 for low load. As the load increases, the differences in efficacy become more pronounced which was also seen when comparing SJF with FIFO. As such, since SPJF is a worse policy than SJF when full information is known, we should expect their values to grow further apart, resulting in a larger price of misprediction. However, from this, it is unclear as to how much of an improvement including a prediction provides. As such, we will now compare the residence time of jobs under both FIFO and SPJF.

λ	$\mathbf{E}[R']$ under SPJF	$\mathbf{E}[R]$ under FIFO	% reduction in residence time
0.5	1.712686	2	10.3%
0.6	1.962457	2.5	15.7%
0.7	2.3122	3.333333	22.8%
0.8	2.88222	5	32.5%
0.9	4.19691	10	46.4%
0.95	6.26396	20	56.7%
0.98	11.2849	50	66.1%
0.99	18.4507	100	70.9%
0.999	200.222	1000	80.0%

These results show that SPJF is substantially more effective than FIFO. We shall now adapt the SRPT policy to work with our prediction.

4.2) SPRPT

The Learning Augmented version of SRPT is SPRPT. It is defined as follows.

Definition: The Shortest-Predicted-Remaining-Processing-Time policy is a preemptive policy that organises the jobs in the queue in order of their predicted remaining processing time. Namely, suppose job x has predicted service time p_x and has receive t_x amount of processing and job y has service time p_y and has receive t_y amount of processing. Then if $\max\{p_x - t_x, 0\}$ is less than $\max\{p_y - t_y, 0\}$, job x will be in front of job y in the queue.

Like with SPJF, this definition and many that follow are rather like those for SRPT. This time, we require the introduction of $\max\{p - t, 0\}$ due to the possibility that the predicted processing time for job could be smaller than the processing time of the job. The introduction of this will allow us to keep integrating from 0 as opposed to $-\infty$. It should be noted that this issue was not present with SPJF as the ordering there was exclusively based on the predicted service time of a job.

Due to the more complex nature of the residence time of a job, we will decompose the residence time of a job as follows

$$R'(y) = Q'(y) + C'(y)$$

where $Q'(y)$ denotes the time that a job spends in the queue waiting to receive its first amount of processing and $C'(y)$ is the time between the job reaching the server and it being completed. As such, we shall calculate these values separately. We shall start with the easier of the two, $C'(y)$.

Theorem:

$$\mathbf{E}[C'(y)] = \int_{x=0}^{\infty} \frac{g(x,y)}{f_P(y)} \int_{t=0}^{x} \frac{1}{1 - \rho'_{\max\{y-t,0\}}}\, dt\, dx$$

Where y is the service time of the job.

Proof:

This proof follows the style presented in [1]

Fix a value of x. Then when the job has received t amount of service, the expected amount of work that should interrupt the job's processing is $\frac{1}{1-\rho'_{\max\{y-t,0\}}}$. This is similar to SRPT however we are considering the predicted remaining processing time instead. Because of this, we require the inclusion of $\max\{y-t,0\}$ as $y-t$ may take negative values. This is then integrated from 0 to x as this is the amount of work we need to do to compete the job. As we have considered a fixed value of x, we need to consider the probability that our job has that fixed value of x. This is trivially $\frac{g(x,y)}{f_P(y)}$. From this, we can integrate over all possible x to give us the value for $C'(y)$

■

Thus all we need now is the value for $Q'(y)$

Theorem:

$$\mathbf{E}[Q'(y)] = b(y)\left(\frac{a_2(y)}{2a_1(y)(1-\rho'_y)} + \lambda\frac{\int_{t=0}^{y}\int_{x=0}^{\infty} g(x,t)x^2\ dx\,dt}{(1-\rho'_y)^2}\right)$$

Where x is the service time of the job and

$$b(y) = \rho'_y + \lambda\int_{t=y}^{\infty}\int_{x=0}^{\infty} g(x,t)\max\{0, x-t+y\}\ dx\,dt$$

$$a_1(y) = \big(1-b(y)\big)\int_{t=0}^{y}\int_{x=0}^{\infty} g(x,t)x\,dx\,dt + \int_{t=y}^{\infty}\int_{x=t-y}^{\infty} g(x,t)(x-t+y)\ dx\,dt$$

$$a_2(y) = \big(1-b(y)\big)\int_{t=0}^{y}\int_{x=0}^{\infty} g(x,t)x^2\,dx\,dt + \int_{t=y}^{\infty}\int_{x=t-y}^{\infty} g(x,t)(x-t+y)^2\ dx\,dt$$

Proof:

This proof follows the style presented in [1]

We will rewrite $\mathbf{E}[Q'(y)]$ as

$$\mathbf{E}[Q'(y)] = P(y)\mathbf{E}[T(y)]$$

As where $P(y)$ is the probability that when a job with predicted processing time y joins the queue that we are processing a job with a shorter or equal remaining processing time and

$T(y)$ denotes the time taken for the queue to process all jobs before our job. We will start with the easier of the two, $P(y)$.There are two type of jobs that contribute to $P(y)$:

1. Jobs of predicted length less than or equal to y
2. Jobs who have a predicted processing time of greater than y but have received some amount of processing to cause their remaining processing time to be less than or equal to y.

The first of these is rather simple to calculate. This is simply ρ'_y. The second is slightly more complicated. As we have seen before, we can consider the load of a category of jobs as the probability that it is processing a job of that type at a given point in time. We will now calculate the load on the system caused by these second type of jobs. We start by fixing a remaining predicted processing time, t, and processing time x. The probability of this job occurring is $g(x,t)$. We know that the remaining processing time of the job is $\max\{0, x-t+y\}$ as the job must have a non-negative remaining processing time and the job has received $t-y$ service already. Collating this and integrating over all possible values for x and t, being 0 to ∞ and y to ∞ respectively, we get the expected service time of a job of the second type. Multiplying by the arrival rate of the system gives

$$\lambda \int_{t=y}^{\infty} \int_{x=0}^{\infty} g(x,t) \max\{0, x-t+y\} \; dx \, dt$$

This gives us that

$$P(y) = \rho'_y + \lambda \int_{t=y}^{\infty} \int_{x=0}^{\infty} g(x,t) \max\{0, x-t+y\} \; dx \, dt = b(y)$$

Now for $\mathbf{E}[T(y)]$. Again, we have two types of jobs that contribute to this value:

1) Jobs of predicted length less than or equal to y
2) Jobs who have a predicted processing time of greater than y but have received some amount of processing to cause their remaining processing time to be less than or equal to y.

It is worth noting that the number of jobs of type two cannot grow as there is no way for a job to receive processing whilst it is behind our job. We will calculate $\mathbf{E}[T(y)]$ by considering the expected busy period of our queue restricted to the jobs before our job, which we will denote as $B(y)$. We can assume that this queue is using the FIFO policy as busy periods are independent of queuing policy. We will consider the two distributions to do this. The first will be the distribution of jobs in the queue when our job arrives, which we will denote as $I(y)$. The second will be the distribution of jobs arriving to the queue after our job arrives, which we will denote as $A(y)$. The probably that a job starts a busy period for $I(y)$ is

$$\Pr[Busy\ period] = \big(1 - b(y)\big)\int_{t=0}^{y} f_P(t)dt + \int_{t=y}^{\infty}\int_{x=t-y}^{\infty} g(x,t)\ dx\,dt$$

With the first term coming from jobs of type 1 and the second coming from jobs of type 2. Calculating the first and second moments of these gives us

$$\mathbf{E}[I(y)] = \frac{a_1(y)}{\Pr[Busy\ period]}$$

$$\mathbf{E}[I(y)^2] = \frac{a_2(y)}{\Pr[Busy\ period]}$$

Where

$$a_1(y) = \big(1 - b(y)\big)\int_{t=0}^{y}\int_{x=0}^{\infty} g(x,t)x\,dx\,dt + \int_{t=y}^{\infty}\int_{x=t-y}^{\infty} g(x,t)(x-t+y)\ dx\,dt$$

$$a_2(y) = \big(1 - b(y)\big)\int_{t=0}^{y}\int_{x=0}^{\infty} g(x,t)x^2\,dx\,dt + \int_{t=y}^{\infty}\int_{x=t-y}^{\infty} g(x,t)(x-t+y)^2\ dx\,dt$$

For $A(y)$, we have that

$$\mathbf{E}[A(y)] = \frac{1}{\int_{t=0}^{y} f_P(t)dt}\int_{t=0}^{y}\int_{x=0}^{\infty} g(x,t)x\ dx\,dt$$

$$\mathbf{E}[A(y)^2] = \frac{1}{\int_{t=0}^{y} f_P(t)dt}\int_{t=0}^{y}\int_{x=0}^{\infty} g(x,t)x^2\ dx\,dt$$

which should be obvious as the distribution of job service times in $A(y)$ are that of S when restricted to jobs of predicted length less than or equal to y. Using problem 49 from [4], we get that

$$\mathbf{E}[B(y)] = \frac{\mathbf{E}[I(y)]}{1 - \rho'_y}$$

$$\mathbf{E}[B(y)^2] = \frac{\mathbf{E}[I(y)^2]}{\left(1-\rho'_y\right)^2} + \lambda\mathbf{E}[I(y)]\int_{t=0}^{y} f_P(t)dt\,\frac{\mathbf{E}[A(y)^2]}{\left(1-\rho'_y\right)^3}$$

From this, we can calculate $\mathbf{E}[T(y)]$ as

$$\mathbf{E}[T(y)] = \frac{\mathbf{E}[B(y)^2]}{2\mathbf{E}[B(y)]} = \frac{a_2(y)}{2a_1(y)(1-\rho'_y)} + \lambda\frac{\int_{t=0}^{y}\int_{x=0}^{\infty} g(x,t)x^2\ dx\,dt}{(1-\rho'_y)^2}$$

Which gives us

$$\mathbf{E}[Q'(y)] = b(y)\left(\frac{a_2(y)}{2a_1(y)(1-\rho'_y)} + \lambda\frac{\int_{t=0}^{y}\int_{x=0}^{\infty} g(x,t)x^2\ dx\,dt}{(1-\rho'_y)^2}\right)$$

$$\blacksquare$$

We can now compute values that correspond to our instance.

$$\mathbf{E}[C'(y)] = \int_{x=0}^{\infty}\frac{\frac{1}{x}e^{-x-\frac{y}{x}}}{\int_{t=0}^{\infty}\frac{e^{-t-\frac{y}{t}}}{t}\,dt}\int_{t=0}^{x}\frac{1}{1-\lambda\int_{u=0}^{\max\{y-t,0\}}\int_{v=0}^{\infty}e^{-v-\frac{u}{v}}\ dv\,du}\ dt\,dx$$

$$\mathbf{E}[Q'(y)] = b(y)\left(\frac{a_2(y)}{2a_1(y)(1-\ \lambda\int_{t=0}^{y}\int_{x=0}^{\infty}e^{-x-\frac{t}{x}}\ dx\,dt)} + \lambda\frac{\int_{t=0}^{y}\int_{x=0}^{\infty}xe^{-x-\frac{y}{x}}\ dx\,dt}{(1-\ \lambda\int_{t=0}^{y}\int_{x=0}^{\infty}e^{-x-\frac{t}{x}}\ dx\,dt)^2}\right)$$

$$b(y) = \ \lambda\int_{t=0}^{y}\int_{x=0}^{\infty}e^{-x-\frac{t}{x}}\ dx\,dt + \ \lambda\int_{t=y}^{\infty}\int_{x=0}^{\infty}\frac{1}{x}e^{-x-\frac{t}{x}}\max\{0,x-t+y\}\ dx\,dt$$

$$a_1(y) = \left(1-b(y)\right)\int_{t=0}^{y}\int_{x=0}^{\infty}e^{-x-\frac{t}{x}}dx\,dt + \int_{t=y}^{\infty}\int_{x=t-y}^{\infty}\frac{1}{x}e^{-x-\frac{t}{x}}(x-t+y)\ dx\,dt$$

$$a_2(y) = \left(1-b(y)\right)\int_{t=0}^{y}\int_{x=0}^{\infty}xe^{-x-\frac{t}{x}}dx\,dt + \int_{t=y}^{\infty}\int_{x=t-y}^{\infty}\frac{1}{x}e^{-x-\frac{t}{x}}(x-t+y)^2\ dx\,dt$$

These values can then be combined and used to calculate the average residence time of a job by

$$\mathbf{E}[R'] = \int_{y=0}^{\infty} \left(Q'(y) + C'(y)\right) \int_{x=0}^{\infty} \frac{e^{-x-\frac{y}{x}}}{x}\, dx\ dy$$

It is possible to calculate these values for given values of λ using numerical methods. Due to technical issues, the values presented below were taken from [1].

λ	$\mathbf{E}[W']$	$\mathbf{E}[R']$
0.5	0.6531	1.6531
0.6	0.9305	1.9305
0.7	1.3539	2.3539
0.8	2.1168	3.1168
0.9	4.04808	5.04808
0.95	7.3221	8.3221
0.98	15.6239	16.6239
0.99	27.7302	28.7302

Using these values, we can calculate the price of misprediction for SPRPT.

λ	**Price of misprediction**
0.5	1.15977
0.6	1.20348
0.7	1.25570
0.8	1.32473
0.9	1.42114
0.95	1.50191
0.98	1.58402
0.99	1.62990

From this, it may seem that SPRPT lags behind SPJF in efficacy due to having a higher cost of misprediction for given values of λ. However, this is not the case due to the significantly shorter residence times of SRPT. As such, we will now consider the percentage reduction in residence time compared to FIFO.

λ	$\mathbf{E}[R']$ under SPRPT	$\mathbf{E}[R]$ under FIFO	% reduction in residence time for SRPT	% time reduction for SPJF
0.5	1.6531	2	17.3%	10.3%
0.6	1.9305	2.5	22.8%	15.7%
0.7	2.3539	3.333333	29.4%	22.8%
0.8	3.1168	5	37.7%	32.5%
0.9	5.04808	10	49.5%	46.4%
0.95	8.3221	20	58.4%	56.7%
0.98	16.6239	50	66.8%	66.1%
0.99	28.7302	100	71.3%	70.9%

This shows that SPRPT is indeed more effective than SPJF, with their difference becoming less pronounced as the load on the system increase. This can partially be explained as the residence times for FIFO grow substantially large as λ tends to 1, causing the differences to be less pronounced. With this, we conclude our discussion regarding the theoretical efficacy of different Learning Augmented policies. We will now study the residence times of jobs under different policies when simulated.

5) SIMULATION RESULTS

Whilst the results that we have proved up until this point show that the learning-augmented policies are effective and are not substantially worse that the corresponding policies which require full knowledge, it is prudent to check that reality coincides with the theory we developed. For this, we shall develop our own simulation environment using Java. As what we require is rather simplistic, we do not require any external libraries other than the ExponentialDistribution from Apache Commons' Math Library. For each value of λ, 1,000 iterations were run. Each iteration ran for 1,000,000 time steps to generate sufficiently many jobs. Each iteration maintains five queues each with a different policy, namely FIFO, SJF, SPJF, SRPT and PSPRT. This allows for direct comparison as each queue is subject to the same input instance. We are then also able to track the service times of each given job and how their waiting time varies from policy to policy. This was initially stored, however due to the substantial number of jobs we are generating, this was not feasible to do for several values of λ. Instead, we opt to calculate the mean and variance of waiting times for each iteration. From this, we can then calculate the price of misprediction for each of the polices using predictions. We then calculate the average of the waiting times, variance of waiting times and the price of misprediction. We are also able to calculate the variance of the price of misprediction. This then gives us an idea as to how consistent the learning augmented versions perform in practice in relation to the algorithm using full knowledge.

To form a baseline of comparison, we will look at how FIFO performed.

λ	**Residence time under FIFO (Simulation)**	**Residence time under FIFO (Equation)**
0.5	2.000342	2
0.6	2.50038	2.5
0.7	3.334016	3.333333
0.8	4.998817	5
0.9	9.997418	10
0.95	19.99858	20
0.98	49.97307	50
0.99	99.76211	100
0.999	532.8161	1000

From this, we can see that the simulations aligned with the residence times of jobs, with the exception being when the load was 0.999. This is presumably due to the methodology used. There are two possible explanations to this, both of which concern the methodology used. The first is that the jobs that remain in the queue once the time limit has been exceeded have a substantial impact. This would not have a major effect for small values of λ but would grow as seen here. Another possibility is that the jobs serviced whilst the queue was otherwise empty have skewed the waiting times. This is somewhat understandable but would likely not occur substantially often or have the pronounced effect that we see here, especially across all the simulations. Regardless of the cause, it is obvious values obtained for $\lambda = 0.999$ may not be correct for the other policies. As such, the results for $\lambda = 0.999$ will be ignored in our further discussions.

Now we shall consider the results for SJF and SPJF.

λ	**Residence time under SJF (Simulation)**	**Residence time under SJF (Equation)**	**Residence time under SPJF (Simulation)**	**Residence time under SPJF (Equation)**
0.5	1.712897	1.712686	1.795038	1.794808
0.6	1.962618	1.962457	2.1088	2.10862
0.7	2.312455	2.3122	2.572895	2.57257
0.8	2.881786	2.88222	3.375034	3.37578
0.9	4.196275	4.19691	5.359738	5.361
0.95	6.262397	6.26396	8.647823	8.65368
0.98	11.27123	11.2849	16.90607	16.9501
0.99	18.34835	18.4507	28.80922	29.0535
0.999	57.67953	116.817	95.80074	200.222

Again, the values for the simulations align with those derived from the equations, with $\lambda = 0.999$ being the exception. There is also another trend that is noticeable. When λ takes values 0.5, 0.6 and 0.7, the expected residence time of a job is greater for the simulations than the equations. As for the other values, the reverse is true, with their disparity growing as λ grows. The latter of these trends can be explained as the same issue with the issues seen at $\lambda = 0.999$, only on a smaller scale. As for the former, it is unlikely to be caused by the same issue. A possible explanation is the jobs created for these had, on average, slightly longer service times than 1. This is feasible but unlikely due to the large number of jobs created. This would have also had to happen across 3 different simulations making it less likely to be a valid explanation. This then means it may be a nuance of the simulation causing this.

We will now consider the price of misprediction for SPJF

λ	**Price of misprediction (Equation)**	**Mean price of misprediction (Simulation)**	**Variance of price of misprediction (Simulation)**
0.5	1.047949	1.0479533407497446	4.409176255215641E-7
0.6	1.07448	1.07448095507294	8.39587134038311E-7
0.7	1.112607	1.1126212186568207	1.6081471536644187E-6
0.8	1.171243	1.171152000011721	3.840176440839116E-6
0.9	1.277368	1.2772290080348538	1.4003818090913E-5
0.95	1.381503	1.3807882621848946	4.3849747073387846E-5
0.98	1.502016	1.499220655292107	1.7854954394191935E-4
0.99	1.574656	1.567801240567334	4.119284944197332E-4
0.999	1.71398	1.654803818644473	0.0016207413177791175

It should be no surprise that the values for price of misprediction match, especially for the smaller values. We still have the value of $\lambda = 0.999$ deviating from its calculated value but it can also be seen somewhat that the same is true for 0.99, all be it to a lesser extent.

From the inclusion of the variance, we can see that the simulated values were consistent across the iterations. This implies that the price of misprediction is a robust measure for the performance for SJF. It can also be calculated that for all bar $\lambda = 0.999$, the difference between the simulation values and the value obtained in the equation were no more than 0.5 times the standard deviation for the simulation. The following table illustrates this.

λ	**Difference in price of misprediction (Equation – Simulation)**	**Difference divided by the standard deviation**
0.5	-4.3E-06	-0.00654
0.6	-9.6E-07	-0.00104
0.7	-1.4E-05	-0.01121
0.8	9.1E-05	0.046437
0.9	0.000139	0.037142
0.95	0.000715	0.107935
0.98	0.002795	0.209197
0.99	0.006855	0.337739
0.999	0.059176	1.469908

This shows that our simulations can be used to obtain an accurate price of misprediction for the SPJF policy. This implies that the issues with the waiting times are consistent regardless of the policy.

We shall now look at SRPT and SPRPT.

λ	**Residence time under SRPT (Simulation)**	**Residence time under SRPT (Equation)**	**Residence time under PSRPT (Simulation)**	**Residence time under PSRPT (Equation)**
0.5	1.425485	1.425373	1.658499	1.6531
0.6	1.604249	1.604094	1.939542	1.9305
0.7	1.874802	1.874567	2.369255	2.3539
0.8	2.352424	2.352773	3.137062	3.1168
0.9	3.551495	3.552125	5.097204	5.04808
0.95	5.539405	5.541011	8.398692	8.3221
0.98	10.48104	10.49475	16.71989	16.6239
0.99	17.52398	17.62693	28.69099	28.7302
0.999	56.78316	115.9328	95.7231	-

It is also little surprise to find that the values from the simulations align with those from the equations, with $\lambda = 0.999$ being the exception. It is plain to see that all the queues suffer from the same issue within the methodology. With this being the case, there is little more to discuss regarding these results.

Again, we can study the price of misprediction

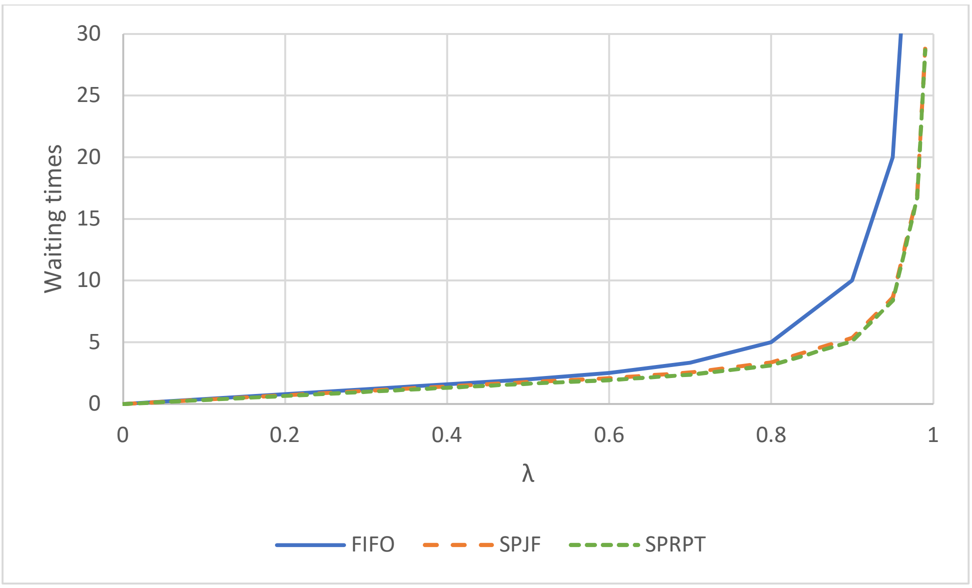
λ	**Price of misprediction (Equation)**	**Mean price of misprediction (Simulation)**	**Variance of price of misprediction (Simulation)**
0.5	1.15977	1.1634609426468545	1.4318321630568676E-6
0.6	1.20348	1.2090003819986241	2.1226335484847425E-6
0.7	1.25570	1.263731509120869	3.099388456551111E-6
0.8	1.32473	1.333535879013506	5.283786565390969E-6
0.9	1.42114	1.435201372410504	1.4067282693819294E-5
0.95	1.50191	1.5160862399229764	3.373390428551559E-5
0.98	1.58402	1.594827892797492	1.1326909516640171E-4
0.99	1.62990	1.635935367575643	2.4733223260664516E-4
0.999	-	1.6828198297113381	0.0014439363143408102

Again, the values from the equations and simulations align for all bar $\lambda = 0.999$. The simulated price of misprediction of SPRPT is also higher than of SPJF, which is also consistent with our analysis. Like before with SPJF, we will also express the difference of the two values for the price of misprediction in terms of its variance.

λ	**Difference in price of misprediction (Equation – Simulation)**	**Difference divided by the standard deviation**
0.5	-0.00369	-3.08715
0.6	-0.00552	-3.78703
0.7	-0.00803	-4.56004
0.8	-0.0088	-3.82892
0.9	-0.01406	-3.74809
0.95	-0.01418	-2.44077
0.98	-0.01081	-1.01541
0.99	-0.00603	-0.38352
0.999	-	-

These results imply that our simulations were less consistent that those of SJF. This may be caused by the issue in methodology manifesting in a substantially different manner for SRPT.

Finally, we will plot the expected waiting times generated by our simulations against λ. This will be done in a piecewise linear fashion. We will be omitting the results for $\lambda = 0.999$ due to the issues with them.



From this, we can see that the improvements of including a prediction are rather substantial. The improvements become more profound as the load increases. Again, this should be no surprise as we knew that this result held in theory and the simulations aligned with the other results. With this we have shown that via the inclusion of a prediction, we can significantly reduce the waiting time of jobs both in the theory and in practice. We have also shown that the theory and our simulations mostly agree.

6) CONCLUSION

In conclusion, this project was a success. We show that the inclusion of a prediction substantially reduces the expected residence time of a job in a queue, regardless of if the queue is preemptive or not. We also show that when a prediction is used instead of full information, the resulting policy is not substantially worse than the policy requiring full information. This result was done by considering the price of misprediction. We also highlighted some of the shortcomings of the price of misprediction as the sole metric for a Learning Augmented policy, namely that the improvement for the inclusion is not conveyed in the slightest. To counteract this, we also included the percentage reduction in residence time when no information was available. We also showed that the results from the theory align with those of the simulations and that the results presented also adhere to applied instances.

7) FURTHER WORK

When specific distributions were required, we restrict ourselves to M/M/1 queues. An obviously extension to this work would be to focus on other distributions for job service time. This would be relevant as a large number of distributions are fairly common when dealing with M/G/1 queues.

When deriving the optimal policy for when service times were unknown, no distinction was made between preemptive and non- preemptive queues. As such, the FIFO policy was used for both instances which need not be optimal for preemptive unknown service time queues, resulting in some of the discussion being skewed in favour of SPRPT due to the baseline of comparison not being as optimal. As such, development of such an optimal policy to be used as a basis of comparison would provide a better idea as to the amount of improvements that the prediction provides.

With there also being a large number of variations on queue structures, a further relevant extension would be to focus on these different styles of queue. For example, developing similar analysis for G/M/1 queue is one possible route. This then may help understanding G/G/1 queue allowing for the work to apply to a larger classification of queues. In a similar vein, the restriction of one server could also be lifted to give us M/G/k queues.

Another variation which may be relevant to Learning Augmented Queuing is queues with start-up times. This is particularly relevant as the prediction will take some amount of time which may detract from processing of jobs. Queuing with start-up times would allow us to model such a delay play the processing of jobs. This however would have to be done in the arbitrary sense as the delay caused by machine learning prediction would heavily vary depending on the specific instance that is being dealt with. For example, the prediction may have to be an exceedingly deep neural network which would add a larger relative delay, especially if the expected service time of jobs is rather small.

8) PROJECT MANAGEMENT

Overall, the management of this project could have been handled better. There was extremely slow progress in the first term due to a lack of familiarity with queueing theory and statistical methods. This resulted in the initial reading of the work by Mitzenmacher [1] taking substantially longer than expected. This in turn delayed the progress of other aspects of the project, resulting in compounding issues. This, in conjunction with external commitments being mitigated poorly, resulted in just the initial reading being completed. The approach adopted during the first term of attempting to implement understood elements when there were issues with understanding theory proved to be useful. This allowed for insight into the nuances of each competent of the theory, which in turn allowed for progress in understanding elements which had caused progress to stop.

During the second term, progress was substantially faster. This was due to growing confidence with queueing theory and a reduction in external commitments. However, the approach taken had to change. This was due to the more sequential structure of work needed to progress, with the appropriate theory being required before implementation could begin. It also became clear that the understanding gained in term one was more intuition driven, with a lack of concrete understanding of proofs.

With the benefit of hindsight, it has become clear that a more effective approach to the project would have instead been to focus on background reading into queuing theory before the first term. With this, the scale of the project would have been clearer from the onset and a more realistic initial objectives may have been proposed. Performing background reading first would also have fixed the issues that came with “learning to run before learning to walk” as it were.

REFERENCES

[1] M. Mitzenmacher, “Scheduling with Predictions and the Price of Misprediction,” 2019 . [Online]. Available: arXiv:1902.00732.

[2] S. Vassilvitskii, “Algorithms with Predictions: Simpler Analyses and Better Running Times,” IGAFIT Colloquia, 20 May 2021. [Online]. Available: https://youtu.be/0NpvsUgXiCY.

[3] M. Harchol-Balter, Performance Modeling and Design of Computer Systems, Cambridge University Press, 2013.

[4] N. Gautam, Analysis of Queues: Methods and Applications, CRC Press, 2012.

[5] Conway, Richard W.; Maxwell, William L.; Miller, Louis W, Theory of Scheduling, Dover Publications, 2003.