VRIJE UNIVERSITEIT AMSTERDAM

RESEARCH PAPER

Function Identification in Single Node Queuing Systems using Genetic Programming

Author
E.P. VAN DE BIJL

Supervisor
Prof. R.D. VAN DER MEI

March 8, 2019

Function Identification in Single Node Queuing Systems using Genetic Programming

E.P. van de Bijl

March 8, 2019

Vrije Universiteit Amsterdam, Faculty of Science, Business Analytics, de Boelelaan 1081a, 1081 HV Amsterdam

Abstract

In queuing theory, closed-form expressions for key performance metrics (such as the waiting time distribution, numbers of customers in the system, etc.) are useful as they show how the performance of a queuing systems depends on the system parameters. Unfortunately, many queuing systems prohibit derivation of closed-form expressions. Alternatively, mathematical approximations or simulation approaches are very useful, but they fail to give fundamental insight in the functional relationship between system parameters and the performance measures of a queuing system. This paper proposes a data-driven approach to obtain closed-form expressions for key performance metrics by symbolic regression. Searching the mathematical expressions space is performed by genetic programming, an evolutionary algorithm variant. Data sets are created by selecting system parameters for a variety of single node queuing systems and obtaining the key performance metrics by simulation when these metrics are not derivable. Three different sampling techniques are used for selecting parameters: single random sampling, stratified sampling and systematic sampling. This research shows that for the M/M/1, M/G/1 and M/M/s queuing systems, genetic programming is able to obtain exact performance metrics. Prior knowledge, such as the heavy-traffic behaviour, can improve the speed of convergence when for example this behaviour is implemented in the form of a explanatory variable. Furthermore, it is shown that none of the sampling techniques resulted in improving the speed of convergence. For the M/G/s queue, genetic programming is able to find accurate approximations for some performance metrics when using prior knowledge on the heavy traffic behavior and the probability on waiting.

Preface

This paper is written for the course Business Analytics Research Paper which is part of the Master Business Analytics. The aim of this project is to be able to analyse a specific issue or problem within the field of Business Analytics. The end results should be a written paper in which the problem is clearly described and well formulated.

My paper addresses the problem of obtaining closed form approximations for performance metrics for a set of single node queuing systems. The approach used in this research is a data-driven approach by means of symbolic regression. This symbolic regression task is performed by genetic programming which is a variant of evolutionary computation.

I would like to thank my supervisor Rob van der Mei from the Vrije Universiteit for his guidance during the project. Rob encouraged me to dive into the problem and try to obtain new findings. Furthermore, I would like to thank Asparuh Hristov for helping me at the start of the project by providing me some software that helped me using genetic programming in practice. I used his PhD thesis as guidance and inspiration.

Etienne Pieter van de Bijl
Amsterdam, February 2019

Contents

1 Introduction

Congestion is a common phenomenon in our urbanized society. One result of congestion is that we have to wait in a queue before services can be provided. Companies are interested in achieving a certain service level while also minimizing the corresponding costs of the service capacity. One example is a call centre with agents that provide customer services. This call centre wants to meet a certain service level, such as helping 80 percent within 20 seconds. How many agents are required to meet this service level? In the queuing theory, we try to answer this type of questions.

One issue in queuing theory is the unavailability of closed-form expressions for key performance metrics of certain queuing systems. This led to an increasing interest in approximation techniques and simulation approaches. Narayan Bhat, Shalaby, and J. Fischer (1979) distinguish three categories approximation techniques: system approximations (simplify the system), process approximations (diffusion/ fluid approximation and asymptotic/ limiting results) and numerical approximations (point/ interval approximation). These techniques are proven to be very accurate in practice (Narayan Bhat et al., 1979). In approximation techniques, results are analytically obtained by simplifying queuing systems. However, this is at the possible expense cost of losing generality and ignoring some of the complexity of reality. Another approach to get key performance measures for given queuing instances is by simulation. In Monte Carlo simulation, a queuing instance is simulated multiple times to get confidence intervals on the key performance measures. This approach, however, is computationally expensive if systems are heavily loaded. Simulation a system that is heavily loaded requires a longer period to obtain accurate results. While this method is useful to obtain results for complex queuing systems, it will only give results for the predefined queuing system instances.

The approximation techniques and the simulation approach have in common that they are not seeking for the closed-form expressions of key performance metrics of queuing instances. Instead, a machine is much better in searching the mathematical expressions space to approximate key performance metrics. This task is also called symbolic regression. The evolutionary computation variant genetic programming (GP) is an algorithm that is able to perform this task. GP is an algorithm that can be positioned in the machine learning area (Eiben & Smith, 2015). Using GP instead of other machine learning algorithms has the advantage that the solutions obtained by GP are functions and can be mathematically analysed. Furthermore, GP does not make assumptions on the form of the function and can uncover surprising relationships.

In this research paper, we are interested in finding the relationship between the input variables and the output variables for single node queuing systems by using GP. For these queuing systems, the input variables are the properties of the inter-arrival/service time distribution (e.g., first moment, second mo-

ment), and the output variables are the performance measures (e.g., expected waiting time). By using GP, we do not simplify reality and we can extract the complexity of queues which are not algebraically solvable. In section 2 we discuss the related literature on the subject. Section 3 outlines some of the fundamentals of the queuing theory. In section 4 the methodology to solve the symbolic regression task is discussed. Section 5 shows how the data is generated for this research. In section 6, two experiments are performed and their results discussed. Section 7 gives the discussion of this paper. At last, Section 8 gives the conclusions of this report and Section 9 states some further research.

2 Literature review

An introduction to queuing theory is given in the books (Gross & Harris, 1998) and (Koole, 2014). Some important distributions for queuing modelling can be read in (Koole, 2014). Symbolic regression performed by GP is discussed in the books (Koza, 1998) and (Poli, B. Langdon, & Mcphee, 2008). Other variants of evolutionary computation can be read in (Eiben & Smith, 2015). There are several studies in combining queuing theory and evolutionary computation. Raman, Nagalingam, and Gurd (2009) use genetic algorithms and queuing theory for a facility layout problem. Semen, Sergey, Viktor, and Sergey (2006) study a similar approach for a scheduling problem with switching times.

Onderwater, Bhulai, and van der Mei (2016) combine GP with Markov Decision Processes to learn value functions. The authors apply GP to a queuing system which has a single queue and two servers: a fast and a slow server. The goal here is to minimize the number of jobs in the system by determining when to assign jobs to the fast server. Another experiment performed here is using symbolic regression to obtain the value function for a single server single type queue. Hristov (2018) performs symbolic regression by GP to obtain a performance metric for a given queuing system. In this study, the analysed queuing system is a two-stream blending system with two classes of jobs with different priority. The class with the highest priority are helped by all servers while the class with inferior priority can only be helped by a subset of all servers. To be determined is how many servers will only help jobs with a higher priority. The searched performance metric is the mean waiting time for both job classes. This research paper is a follow up study of Hristov (2018) on searching closed-form expressions for performance metrics.

Another study that is close to this research paper is the research paper by Van der Stap (2016), who tries to approximate key performance measures with data mining techniques. By simulating queues, data such as the load of the queue and expected waiting in the queue are generated. Using simulated data, key performance measures are approximated with decision trees. It is shown in this research that only for certain cases it was possible to fit a function on the simulated data. Unfortunately, it was unknown if the approximation functions generalizes well.

Similar to the research paper of Van der Stap, the symbolic regression will result in approximations of performance measures. However, symbolic regression will discover closed-form expressions which can be analysed. Both Onderwater et al. (2016) and Hristov (2018) are focused on Markov Decision Processes, while this research is also focused on queues where the Markov property does not hold.

3 Basics of queuing theory

In this section, the fundamentals of the queuing theory are discussed. First, a representation of queues is given and the standard notation to describe these queues is stated. Secondly, the mathematical notation for all parameters will be discussed. At last, the closed-form expressions for algebraically solvable queuing models are given.

3.1 Classification

All queuing systems share the same characteristic: customers/jobs competing for certain services. Figure 1 gives a visual representation of a queue with six components that define it: 1) the arrival pattern, 2) the service pattern, 3) the number of servers and service channels, 4) the system capacity, 5) the queuing discipline (e.g. FCFS: First Come First Serve) and 6) the number of service stages.

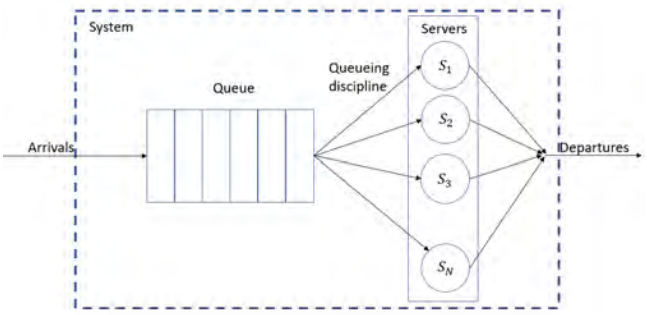


Figure 1: Example of a single node queuing model

The Kendall notation is the standard notation to describe the characteristics of a queuing system by a sequence of symbols and slashes of the form $A/B/c/d/e$. A determines the arrival distribution, B the service time distribution, c the number of parallel servers, d the total number of places in the system and e the queue discipline. In this notation, we assume one service stage. The symbols A and B are either M (Markovian, Poisson arrivals or exponential service times), D (deterministic) or G (general inter-arrival or service times). c and d are integer values with the constraint $c \leq d$. When d is not explicitly given, it is ∞ and e is by default FCFS.

3.2 Notation

This section will give the mathematical notation for each parameter.

- S is the service time distribution; A is the interarrival time distribution;

- λ denotes the parameter of the Poisson arrival process;
- μ denotes the parameter of the service time distribution when it is exponential;
- s is the number of servers;
- π is the stationary distribution of the number of customers in the system;
- L is the limiting number of customers in the system;
- W is the time an arbitrary customer spends in the system;
- L_Q and W_Q denote the same results when only considering the queue.

3.3 Closed-form expressions

In this section, we will give the analytic solutions for key performance indicators in time-stationary queuing models. It is assumed that the reader has some knowledge on the basics of queuing theory.

The steady state probabilities of a M/M/s queue can be obtained by modelling the M/M/s queue as a birth-death process. The performance metrics for this queue can be calculated using these steady state probabilities and the PASTA property. Furthermore, the *Pollaczek-Khintchine* formula is used to calculate the mean waiting time distribution for the M/G/1 queue. Proof on these results are given in (Gross & Harris, 1998). Table 1 shows how the offered load a and the offered load per server ρ is calculated.

load	M/M/s	M/G/1
a	$\frac{\lambda}{\mu}$	$\lambda * ES$
ρ	$\frac{a}{s}$	a

Table 1: Offered load per server

Using the information of Table 1, the performance metrics can be calculated in Table 2 for the given queues. EL is read as the expected steady-state mean number of customers in the system. The last row in Table 2 is interpreted as the probability on waiting for a service. The c^2 term is the squared coefficient of variation and it is defined by $c^2(X) = \frac{E(X-EX)^2}{(EX)^2}$ for random variable X. For the M/M/s queue, $\pi(0)$ can be calculated as follows:

$$\pi(0)^{-1} = \sum_{j=0}^{s-1} \frac{a^j}{j} + \frac{a^s}{(s-1)!(s-a)}$$

KPI	M/M/1	M/M/s	M/G/1
EL	$\frac{\rho}{1-\rho}$	$\frac{\rho P(W^Q>0)}{1-\rho} + a$	$\frac{\rho^2(1+c^2(S))}{2(1-\rho)} + \rho$
EL_Q	$\frac{\rho^2}{1-\rho}$	$\frac{\rho P(W^Q>0)}{1-\rho}$	$\frac{\rho^2(1+c^2(S))}{2(1-\rho)}$
EW	$\frac{1}{\mu*(1-\rho)}$	$\frac{P(W^Q>0)}{s\mu-\lambda} + \frac{1}{\mu}$	$\frac{\rho ES(1+c^2(S))}{2(1-\rho)} + ES$
EW_Q	$\frac{\rho}{\mu*(1-\rho)}$	$\frac{P(W^Q>0)}{s\mu-\lambda}$	$\frac{\rho ES(1+c^2(S))}{2(1-\rho)}$
$P(W_Q > 0)$	ρ	$\pi(0)\frac{a^s}{(s-1)!(s-a)}$	No expression available

Table 2: Exact equations for performance metrics of algebraically solvable queues

4 Methodology

This section will discuss the method of finding closed-form solutions for performance measures of complex queues by using Genetic Programming. Section 4.1 will give a introduction to evolutionary computing. In section **??** a description will be given of an evolutionary algorithm. Section 4.2 will discuss genetic programming. In section 4.3, we will discuss the Tree-based Genetic Programming algorithm and its components.

4.1 Evolutionary Computing

Evolutionary Computing is a research area that is concerned with problem solving using a trial-and-error method and is inspired by the process of natural evolution. One motivation for using evolution as a source of inspiration is that it is the most powerful natural problem solver. Engineers have always looked at natural problem solvers, such as the brain, to be able to develop automated problem solver. Another motivation is the fact that the growing demand for problem-solving automation outpaced the available research options and development capacity. As problems become more complex, obtaining the optimal solution is hard, if not impossible.

An evolutionary algorithm (EA) can be described as follows: a population of individuals (solutions) live in some environment. As there are limited resources, competition between these individuals causes natural selection, or let's say, survival of the fittest. As a consequence, the fitness of the population changes over time. Individuals with a higher fitness are more able to survive compared to individuals with a low fitness. New individuals arise in the environment by recombining individuals in the environment and mutating these new individuals. This process of creating new individuals and natural selection proceeds until a certain stopping criteria is met. Figure 2 shows a high-level scheme for the common basis for all variants of EA.

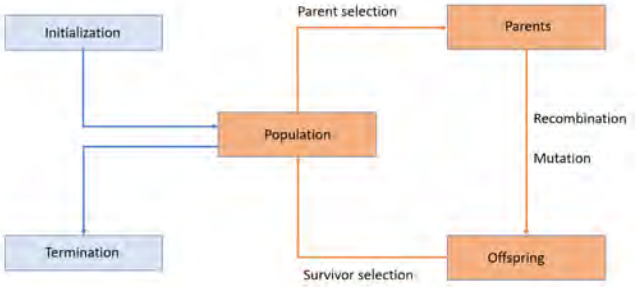


Figure 2: Flow chart of an evolutionary algorithm

The two main forces that form the basis of an evolutionary system are the *variation operators* (*recombination* and *mutation*) and the *selection mechanism*. By the variation operators, diversification of the population can be achieved, while selection increases the mean quality of the solutions in the population.

4.2 Genetic programming

In 1992, Koza introduced the evolutionary computation technique named genetic programming (GP). Koza shows that various problems, such as symbolic regression, can be automatically solved by evolving computer programs. By letting computer programs evolve, it is not required in GP to know or specify the form or structure of the solution in advance (Poli et al., 2008). Initially, Koza expresses programs as syntax trees rather than lines of code (1998). GP computer programs are also described by linear data structures (Brameier & Banzhaf, 2007), graphs (Miller, 2011) or data stacks of instructions (Perkis, 1994). As the interest of this paper is to obtain closed-form solutions, the tree-based syntax has the preference above other representations.

4.3 Tree-based genetic programming

In this section, we will describe the main components of a Tree-based Genetic Programming. Poli et al. (2008) is used as guidance to discuss all components.

4.3.1 Representation

In GP, programs are represented as a tree. One example is shown in Figure 3. This is a tree representation of the function $(x + y) - 5 * (x/y)$. A GP tree consists of two components: the leaves and the internal node. The highest internal node is called the root of a tree. Internal nodes can only contain the arithmetic operations (e.g., $+, -, *, /$). These operations are called the *functions* of the tree. At the leaves are the constants and the variables, which are the *terminals* of the tree. Together the *terminals* and the *functions* form the *primitive set* of a Genetic Programming system. A property of a GP tree is its depth. The

depth of a node is the minimal number of edges that needs to be traversed between the starting node and the root of the tree. The depth of a tree is the depth of its deepest leaf. For example the tree in Figure 3 has a depth of 3.

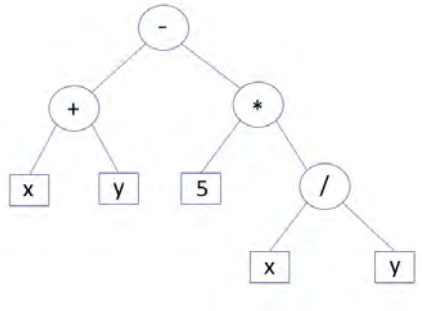
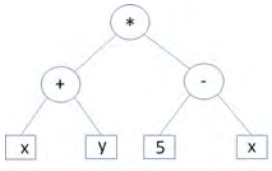


Figure 3: GP tree representing $(x + y) - 5 * (x/y)$

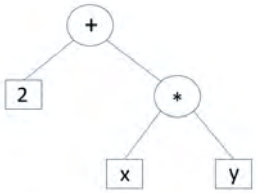
4.3.2 Initialization

The first step in an evolutionary algorithm is generating a population of individuals. These individuals need to be randomly generated. An initialization method that ensures diversity in the initial population is the Ramped half-and-half. This method combines the *full* and *grow* method, which are both insufficient to create diversification in the initial population.

Both the *full* and *grow* methods require a user defined maximum tree depth. At this depth, only terminals can be randomly chosen by both methods. In the full method, trees are created in which all leaves have the same depth. On the contrary, the *grow* does not require all leaves to have the same depth. The grow method allows each node to be of the primitive set (terminals and functions). Figure 4 gives two examples of both initialization methods with a maximum tree depth 2.



(a) Full method



(b) Grow method

Figure 4: Full and grow initialization method with max depth 2

As both grow and full methods do not provide a very wide array of sizes and

shapes, Koza (1998) proposed the *Ramped half-and-half method*. In this method, half of the population is constructed by the grow method, while the other half is constructed by the full method. By setting a range of maximum depth limits, the population can differ in sizes and shapes. In this range of maximum depth limits, a maximum tree depth can be chosen randomly and for both methods a tree can be constructed.

Poli et al. (2008) state that it is hard to control statistical distributions such as the sizes and shapes of generated trees. Both methods seem sensitive to the size of the function and terminal sets. Changing the ratio between the two sets can significantly change the shapes and sizes of trees generated.

4.3.3 Selection

In comparison to other evolutionary computation variants, GP has only one selection mechanism. This mechanism selects individuals on which variation operators are performed. The population with off springs will form the next generation. Individuals with a higher fitness are more likely to have more child programs compared to inferior individuals. A frequently used selection mechanism in GP is *tournament* selection.

In *tournament* selection, we randomly select a subset of the population and let these individuals compete for survival. The individual with the highest fitness will win a tournament and recombination or mutation operators are applied on this individual. *Tournament* selection has the useful property that it does not require any global knowledge of the population. It only relies on the ordering relation of multiple individuals. Therefore, this selector mechanism is simple and fast to implement. The only parameter used in *tournament* selection is the tournament size. This is denoted with the parameter k. The larger the tournament, the greater the chance that the tournament contains members of above-average fitness. Therefore, as k increases, the selection pressure increases. This is the case when the population is not converging. When converging occurs, most programs will have the same fitness and therefore the tournament selection becomes random (Gustafson, 2004). Xie (2008) studies the relationship between population size, tournament size and selection pressure. One finding in this study is that tournament size has a large impact on the selection pressure while population size does not have any impact unless the population size is very small. Choosing k is still open for experimentation. Hancock (1994) compares multiple selection mechanisms and states that each selection mechanism has errors and that only by experimentation the best selection mechanism can be obtained. Although there are other selection techniques, we choose for the tournament mechanism. We consider comparing other techniques beyond the scope of this study.

4.3.4 Recombination and mutation

In contrast to other evolutionary computation fields are crossover and mutation in GP not sequential. Winners of tournament selection are changed by either crossover or mutation operators. We will now discuss the crossover operator and the mutation operators which are commonly used in GP. Other operators are possible, but as the research is focused on finding closed-form expressions, comparing different operators is out of scope.

Subtree crossover is a recombination mechanism. Each winner of a tournament is changed by this operator with probability p_{sc}. When performing this operator, a donor parent is required. By selecting another winner of a tournament, the crossover operator is performed on the parent and the donor parent. In both the parent and the donor, a random node is selected, which is called the crossover point. An offspring is then created by copying the subtree of the donor node at the crossover point to the crossover point of the parent. Figure 5 gives an example of performing the *subtree crossover* operator.

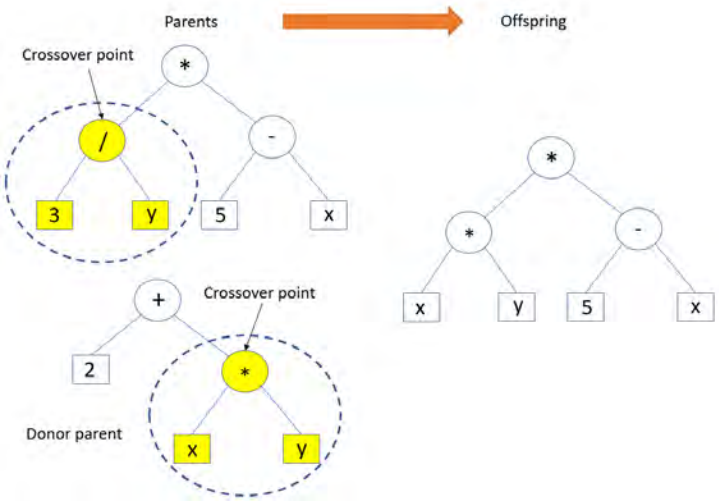


Figure 5: Subtree crossover on parent $\frac{3}{y} * (5 - x)$ and donor $2 + (x * y)$

Subtree mutation is an equivalent of *subtree crossover*. Instead of having a donor parent, the subtree under a mutation point is replaced by a random generated sub-tree. The probability of performing this operator is denoted by p_{sm}. Figure 6 gives an example *subtree mutation*.
Hoist mutation is a mutation operator that is a good solution to prevent bloat, which we will discuss in section 4.3.5. In this mutation operator, we select a sub-tree of the parent tree. This subtree is then removed from the parent tree. By this, the offspring tree will be smaller than its parent. The probability of performing this operator is denoted by p_{hm}. Figure 7 gives one example.

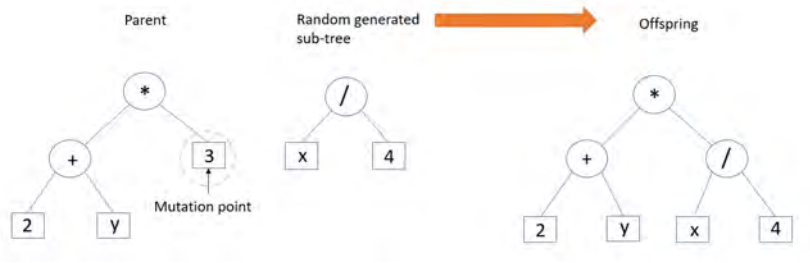


Figure 6: Subtree mutation on parent $3 * (2 + y)$

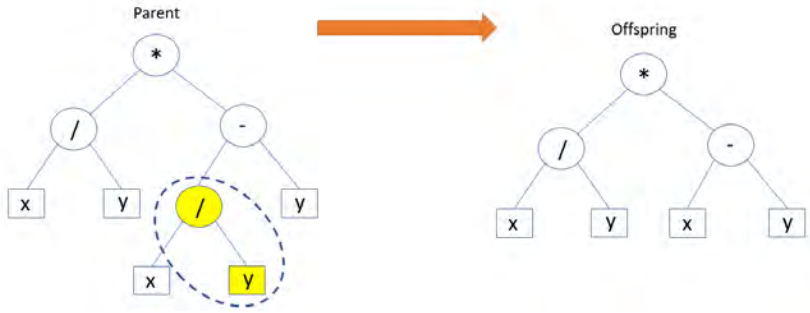


Figure 7: Hoist mutation on parent $(\frac{x}{y}) * (\frac{x}{y} - y)$

Point mutation is an operator where we select a random node and replace it with a different random primitive of the same arity. The probability of performing this operator is denoted by p_{pm}.

GP requires that the sum of the probabilities on the recombination and mutation does not exceed one. If the sum is less than one, the remainder belongs to the probability on reproduction of the parent to its offspring. This means that the offspring is identical to its parent. The name of this operator is *reproduction* and its probability it denoted as p_r.

There are more recombination and mutation operators available for GP. Poli et al. (2008) give a list of optional operators to create diversity in solutions. In the earlier works of Koza, mutation is seen by Koza as unnecessary to apply GP. Chellapilla (1997) shows that the combination of six mutation operators can outperform previously published GP on some simple problems. Luke and Spector (1997) state that it is not clear whether mutation or crossover is better and in their revised work (1998) this observation is confirmed.

4.3.5 Bloat

The phenomenon of trees becoming too big in size and shape is called *bloat*. We define *bloat* as program growth without significant return in terms of fitness. One method to prevent *bloat* is the *parsimony pressure* method. Here, we penalize the size of the program. All fitness functions are therefore replaced by $f(x) = f(x) - c * l(x)$ where $f(x)$ is the original fitness, $l(x)$ is the size of the program and constant c is known as the parsimony coefficient. Poli and McPhee (2014) suggest that c should be $c = \frac{Cov(l,f)}{Var(l)}$ in order to penalize in the correct way. Here, $Cov(l, f)$ is the covariance between the program size and the program fitness f in the population. The $Var(l)$ is the variance of the program size. This constant c should be updated for each new generation.

5 Data

In this section, we describe how data sets are created for the GP algorithm. First, the parameter space of the system parameters are discussed. Second, we discuss tree different sampling techniques to choose data points in the parameter space. Third, a dynamic scheme is given how single node queuing systems are simulated. At last, the batch-means method will be discussed.

5.1 Parameters

Different queuing instances are created by varying the system parameters. The distributions of interest for the service-time and inter-arrival time time are the exponential, the beta, the gamma and the log-normal. By this limitation, we only have to vary the first and second moment of these distributions. If we set A and S to a certain distribution, we only have to set the following parameters with the according restrictions:

- $A \sim P, S \sim Q; P, Q \in \{Beta, Gamma, Exponential, Lognormal\}$
- $E[A] \in R_{>0}; E[S] \in (0, E[A])$
- $Var(A) \in R_{>0}; Var(S) \in R_{>0}$
- $s \in Z_+$

The offered load per server in Erlang is defined as $\rho = \frac{ES}{EA}$. This offered load indicates the percentage time a server is busy. By the second restriction, $\rho \in (0, 1)$. For the exponential distribution only the first moment needs to be determined as $Var(X) = E(X)^2$.

5.2 Sampling

From section 5.1 it can be seen that there is an infinite number of different queuing systems possible. Even though there are a limited number of parameters to be determined, the number of instances is infinite. GP however requires a finite number of instances to search the mathematical expression space. This finite number of instances should represent the entire population. By limiting this this subset, selection is unavoidable. Selecting individuals from a population to gain information about the population is called by statisticians *sampling*. Normally, sampling is performed for finite population to perform statistical inference. In contrast to sampling on a finite population size, the population in this research is infinite by all different queuing systems that can be modelled by simulation. Therefore, while the population is infinite, we can still select a subset of all possible queuing instances by sampling to perform symbolic regression by GP.

In statistical sampling, there are several sampling methods available, listed in Liu and Motoda (2002) and Taherdoost (2016): probabilistic and non-probabilistic sampling methods. From one side, the probabilistic method, probabilities of selection are attached to individuals, while from the other, the non-probabilistic method is biased on choosing individuals based on characteristics. Problems occurs in both methods: the probabilistic is time consuming while non-probabilistic is seen as subjective and biased. In this research, there is a need for a sample that represents the population and generalization is required, the probabilistic method has preference. We will describe a subset of sampling methods which will be used to choose the parameters settings.

5.2.1 Simple random sampling

In simple random sampling, each individual has the same probability on being chosen. In a finite population of size n, each individual has probability $\frac{1}{n}$ on being selected. Suppose we want to draw samples in $y \in [a, b]$ where $b > a$, then we draw a random sample in $x \in U(0, 1)$ and apply the following operation $y = x * (b - a) + a$. Figure 8 gives an example of simple random sampling on two dimensions with 25 samples.

5.2.2 Systematic sampling

Systematic sampling is a method where each nth observation is chosen after a random start. In a finite population setting, if we order the population, we select for example each tenth individual. This technique is applicable to sampling in $y \in [a, b]$. Suppose n observations are drawn in this range, then the *skip* interval is $k = \frac{b-a}{n}$ and a random start r can be drawn from $U(0, k)$. The drawn sample set using these fixed parameters is then:

$$\{a + r + i * k : i \in \{0, 1, ..., (n - 1)\}\}$$

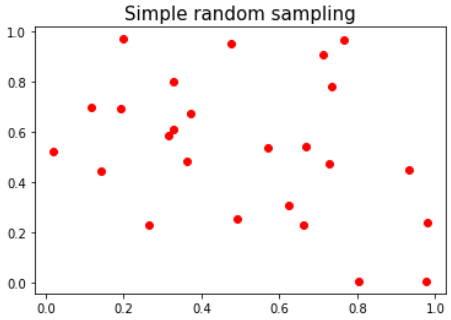


Figure 8: Example simple sampling

. Figure 9 gives an example of systematic sampling on two dimensions.

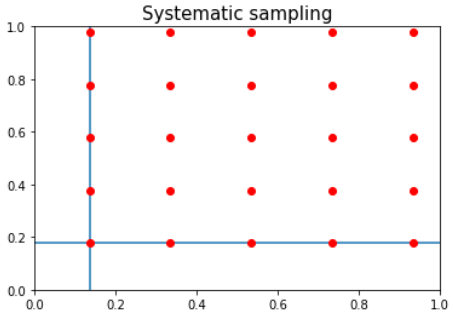


Figure 9: Example systematic sampling

5.2.3 Stratified sampling

In stratified sampling, the population is divided in equally sized subgroups and a random sample is taken from each subgroup. Choosing a random individual from every subgroups (or "stratum") results in representing each part of the population. Applying this technique to sampling on $[a, b]$ can be performed by dividing $[a, b]$ in n equal ranges where n is the number of observations. Each range has length $l = \frac{b-a}{n}$ and the drawn set is the following:

$$\{U(a + l * (i - 1), a + l * i) : i \in \{1, 2..., n\}\}$$

. Figure 10 gives an example of stratified sampling on two dimensions. The space is divided in 25 equal sub regions and from each region a sample is generated.

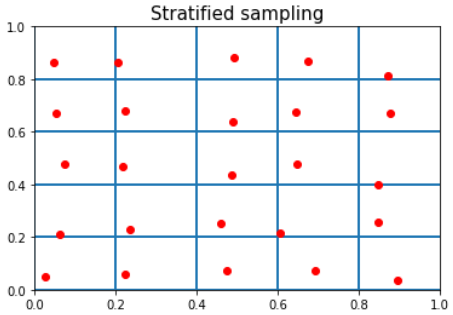


Figure 10: Example stratified sampling

5.3 Simulation

Using the sampling method, a set of queuing instances are generated. As there are no performance metrics for generic queues, performance metrics are obtained by simulation. Simulating a queue requires agents arriving at the system and servers helping these agents. First, a mathematical procedure is given to simulate a queue. Second, in order to get good approximations for the performance metrics the batch means method is used and discussed.

5.3.1 Procedure

In this section, a description will be given how a queue is simulated and results are mathematically obtained. Table 3 shows the variables required to model a queue.

Variable name	Description
a_k	Arrival time of customer k
s_k	Service start of customer k
c_k	Completion time of customer k
x_k	Inter arrival time between customer k and $k-1$
y_k	The service time of customer k

Table 3: Variable description for simulation

The simulation can be executed by generating N customer arrivals. We need to draw N inter-arrival time samples from distribution A and the same number for distribution S. Therefore, we have $x_1, x_2, ..., x_N \in A$ and $y_1, y_2, ..., y_N \in S$. The equations given in (1) show how the arrival times, the service times and the completion times can be calculated. The service time of customer k starts when a server is available. The arrival time of customer k is the sum of the inter arrival times of previous customers. The completion time is the start service time plus the service time itself.

$$
\begin{aligned}
a_k &= \sum_{n=1}^{k} x_k, s_k = max\{c_{k-s}, a_k\} \\
c_k &= \begin{cases} s_k + y_k, & \text{if } k \geq 1 \\ 0, & \text{otherwise} \end{cases}
\end{aligned} \tag{1}
$$

Using these variables and relations, performance measures can be calculated for each customer separately. Table 4 gives the description of the variables used to calculate the performance metrics. The equations given in (2) show how these variables can be determined. These results calculate the waiting time for each customer by looking at the time of arrival and the completion time. The number of customers when a customer arrives is calculated by looking at the completion times of previous customers.

Variable name	Description
w_k^Q	Waiting time of customer k in the queue
w_k	Total time of customer k in the system
l_k	The number of customer in the system when customer k arrives
l_k^Q	The number of customer in the queue when customer k arrives

Table 4: Performance indicator variables

$$
\begin{aligned}
w_k &= c_k - a_k, w_k^Q = s_k - a_k, \\
l_k &= \sum_{n=1}^{k-1} 1_{a_k \leq c_n}, l_k^Q = \sum_{n=1}^{k-s-1} 1_{a_k \leq c_n}
\end{aligned} \tag{2}
$$

The average waiting time (in the queue) is calculated by taking the average waiting time over all customers. The same can be done for the number of customers in the system (or in the queue).

5.3.2 Batch means method

Now that the performance metrics is determined for each agent, the goal is to get an estimate for the average performance metrics. There are several options to get a average. One idea is simulating a queue multiple times and taking the average over the simulation end-average. This however is time consuming and computational expensive. A more convenient choice is using the *batch means method* (Alexopoulos & Seila, 1996). In this method, we simulate a queue only once and divide the data in ordered batches which are equally sized. One issue when considering all data is the *warm-up period*. As queuing systems do not necessarily start in an equilibrium, the start of a queue needs to converge to this equilibrium. This is visually given in Figure 12. This issue is solvable by removing the first batch. One instance is created by simulate 100,000 customers and dividing their customers performance measure in 10 batches.

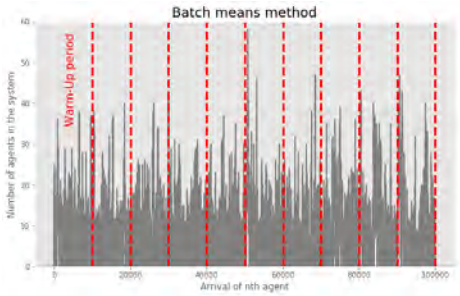


Figure 11: Batch means method

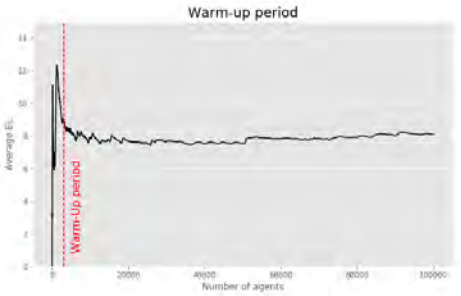


Figure 12: Warm-up period

6 Experiments and results

The experiments performed in this study are described in this section. First, simple queues are analysed to show that GP is able to obtain the performance metrics. Afterwards, GP is applied to the M/G/s queue to find the underlying performance metrics.

6.1 Case 1: Finding closed-form approximations of derivable performance metrics for single node queuing systems

In this section, we will apply GP to obtain closed-form approximations for performance metrics for single node queues. The rationale behind starting with simple queuing instances is that it is verifiable if GP has obtained the underlying function. It will serve as a benchmark. The performance metrics are used to obtain performance measures. First, we will show the parameter subspace. Secondly, the used parameters for the sampling methods will be stated. Thirdly, the GP algorithm settings that are used to search the parametric space are given. At last, the results for the M/M/1, the M/G/1 and the M/M/s queuing systems are shown.

6.1.1 Parameter bounds

Table 5 shows the used parameter bounds. It can be seen that the upper bound for ρ is set below the theoretical upper bound of 1. The rationale behind this choice is that if ρ is close to 1, the variance of the dependent variable (one performance metric) increases drastically compared to the change in ρ. As a consequence, the occurrence of bloat in GP is unavoidable when bloat preventing mechanisms are not sufficient. For the parameter ES, the lower bound is set to 0.1. A rationale here is the impact of ES on the squared coefficient of variation of the service time distributions. Other bounds are chosen in such way that the range can represent the whole space.

Queues	Parameter	Lower bound	Upper bound	Category
M/M/1,M/M/s,M/G/1	ρ	0	0.99	Continuous
M/M/1,M/M/s,M/G/1	ES	0.1	1	Continuous
M/G/1	var S	0	1	Continuous
M/M/s	s	1	50	Discrete

Table 5: Parameter bounds for case 1

6.1.2 Sampling settings

In this section, we describe the sampling method settings. For the simple random sampling, only the bounds in section 6.1.1 are required. For the stratified and the systematic sampling system, we require to determine some parameters. The settings for these parameter depends on the sample size. In this research, 10 data points are considered to be sufficient to represent one variable. This means that if there are p parameters to be varied, the total number of samples is 10^p. Table 5,6 and 7 show the parameters of the sampling method that satisfies the 10 data points per variable requirement.

Parameter	Skip k	Random start r
ρ	0.1	$U(0, 0.09)$
ES	0.1	$U(0, 0.1)$
var S	0.1	$U(0, 0.1)$
s	5	$\lfloor U(1, 5) \rfloor$

Table 6: Systematic sampling settings

Parameter	Interval length l
ρ	0.1
ES	0.1
var S	0.1
s	5

Table 7: Stratified sampling settings

6.1.3 Genetic programming parameter setting

The python library gplearn (Stephens, 2018) is used to perform GP. Table 8 gives the parameters used in the GP algorithm for this case. These settings are used to obtain performance metrics for the simple queues.

Parameter	Value	Parameter	Value	Parameter	Value
Population size	500	Init method	Ramped half and half	p_{sc}	0.9
Generations	10	Function set	[*,-,+,/]	p_{sm}	0.01
Tournament size	20	Metric	Mean Absolute Error	p_{hm}	0.01
Stopping criteria	0	Parsimony coefficient	0.001	p_{pm}	0.01
Init depth	(2,6)	Constant range	(1,1)	p_{pr}	0.05

Table 8: Genetic Programming parameter settings for case 1

6.1.4 Evaluation metric

To measure how well the function fits the data, we use the coefficient of determination (r^2). Suppose we have data $y_1, ..., y_n$ and prediction values $f_1, ..., f_n$. Then the r^2 can be calculated by the following formulas given in (3).

$$\begin{aligned} r^2 &= 1 - \frac{SS_{res}}{SS_{tot}}, & \hat{y} &= \sum_i y_i / n \\ SS_{res} &= \sum_i (y_i - f_i)^2, & SS_{tot} &= \sum_i (y_i - \hat{y}) \end{aligned} \tag{3}$$

6.1.5 M/M/1

For the M/M/1 queue, the objective is to find the relation between the input parameters λ and μ and the performance metrics. Table 9 shows the results of applying GP to different sampling methods and testing it with 10-fold cross-validation. In almost all folds, the actual underlying function was obtained. Table 10 shows which functions are obtained with the highest r^2. GP is able to get the underlying function in relatively few generations because the function is rather simple.

Sampling method	EL	ELQ	EW	EWQ
Simple	1.0	1.0	1.0	0.999927
Stratified	1.0	1.0	1.0	1.000000
Systematic	1.0	1.0	1.0	0.999991

Table 9: Average r^2 in 10-fold GP using different sampling methods M/M/1 queue

Metric	Best function	Exact
EL	$\frac{\lambda}{\mu-\lambda}$	$\frac{\rho}{1-\rho}$
ELQ	$\frac{\lambda}{\frac{\mu}{\lambda}*(\mu-\lambda)}$	$\frac{\rho^2}{1-\rho}$
EW	$\frac{1}{\mu-\lambda}$	$\frac{1}{\mu*(1-\rho)}$
EWQ	$\frac{\frac{\lambda}{\mu}}{\mu-\lambda}$	$\frac{\rho}{\mu*(1-\rho)}$

Table 10: Functions with $r^2 = 1.0$

Conclusion: For the M/M/1 queue, it can be observed that the closed-form approximation equals the exact function. As the function is rather simple, it is not hard for GP to obtain the actual closed-form expression.

6.1.6 M/G/1

Finding the closed-form expressions for the performance metrics of the M/G/1 queue is harder compared to the metrics of the M/M/1. In this setting, we try to find a relationship between λ, ES and $\sigma^2(S)$ and one of the performance metrics (EL, ELQ, EW, EWQ). 1000 samples are gathered by the sampling methods. Table 11 gives the r^2 results when applying GP.

Sampling method	EL	ELQ	EW	EWQ
Simple	0.667	0.725	0.635	0.711
Stratified	0.746	0.689	0.642	0.682
Systematic	0.657	0.666	0.661	0.698

Table 11: Average r^2 in 10-fold GP using different sampling methods M/G/1 queue

Table 12 shows two functions which have a high r^2 and are interpretive. We can observe that the structure of the function resembles the actual performance metrics of the M/G/1 queue.

Metric	Best function	r^2
EL	$\frac{1.0\lambda\left(ES^3\lambda^2 - ES - \lambda\sigma^2(S)\right)}{ES^2\lambda^2 - 1}$	0.999677
EW	$\frac{ES^2(ES\lambda - 1.0)(2ES\lambda - 2.0) + ES + \lambda\sigma^2(S)}{2ES\lambda - 2.0}$	0.925971

Table 12: Interpretable functions with the highest r^2

Only using the parameters to learn the relationship is a rather naive approach. Researchers can of course use professional knowledge on some part of the relationship (Lu, Ren, & Wang, 2016). The results from the M/M/1 queue gives us an indication that the load $\rho = \frac{\lambda}{\mu}$ is an important feature. Furthermore, the heavy traffic behaviour should also be included. When a system is heavily loaded (ρ goes to 1), the asymptotic behaviour around 1 can be used to speed up the convergence of obtaining a accurate approximation. Therefore, the asymptotic behaviour around 1 and the behaviour at 0 for all performance metrics can be approximated by using the relationship $R = \frac{1}{1-\rho}$. In addition to our original variables we use these expressions to learn the relationship, the underlying relationships can be easier obtained. Table 13 shows that using this prior knowledge, the underlying function can be obtained because the GP was pushed in the right direction. In the Appendix A is shown that indeed the best functions are the exact functions but in another format.

Conclusion: For the M/G/1 queue, at first sight, GP is not able to obtain the exact closed-form expression. As is shown in for this queue, some prior knowledge about the heavy-traffic behaviour R helped GP to actually obtain

Metric	Best function	Exact function	r^2
EL	$\lambda\left(0.5ESR\rho + ES + 0.5R\lambda\sigma^2(S)\right)$	$\frac{\rho^2(1+c^2(S))}{2(1-\rho)} + \rho$	1.0
ELQ	$\frac{0.5R\rho\left(ES\rho+\lambda\sigma^2(S)\right)}{ES}$	$\frac{\rho^2(1+c^2(S))}{2(1-\rho)}$	1.0
EW	$0.5ES\left(R+1\right) + 0.5R\lambda\sigma^2(S)$	$\frac{\rho ES(1+c^2(S))}{2(1-\rho)} + ES$	1.0
EWQ	$0.5R\left(ES\rho + \lambda\sigma^2(S)\right)$	$\frac{\rho ES(1+c^2(S))}{2(1-\rho)}$	1.0

Table 13: Interpretable functions with the highest r^2 using prior knowledge M/G/1

Sampling method	EL	ELQ	EW	EWQ
Simple	0.970	0.988	0.965	0.976
Stratified	0.982	0.997	0.932	0.975
Systematic	0.981	0.989	0.940	0.943

Table 14: Average r^2 in 10-fold GP using different sampling methods M/G/1 queue with prior knowledge

the actual closed-form expressions for the performance metrics. It is not clear that using a different sampling technique lead to better results.

6.1.7 M/M/s

The last queue analysed in this section is the M/M/s queue. Table 15 shows the 10-fold cross-validation r^2 score using different sampling techniques. It can be observed that the highest score is obtained for the ELQ metric, while the lowest score is obtained for the EWQ metric. One interesting function that is obtained in GP was the relation $ELQ = \frac{\lambda^5}{\mu^4 s^4(-\lambda+\mu s)}$. This function had a r^2 of 0.9999999 in one of the folds.

Sampling method	EL	ELQ	EW	EWQ
Simple	0.509	0.803	0.575	0.215
Stratified	0.267	0,899	0.592	0.090
Systematic	0.445	0.755	0.576	0.137

Table 15: Average r^2 in 10-fold GP using different sampling methods M/M/s queue

Similar to the M/G/1 queue, prior knowledge can be used for the M/M/s queue. For the M/M/s queue, the prior knowledge we use is the probability on waiting $C = P(W^Q > 0)$ and the load per machine in Erlang $\rho = \frac{\lambda}{\mu * s}$. In this setting with prior knowledge, the goal is to find the relationship between the independent variables λ, μ, s, ρ, C and the performance metrics

(EL,ELQ,EW,EWQ). Table 16 shows the accuracy that can be achieved when using this prior knowledge. As can be seen, the accuracy improved compared to the results in Table 15. Table 17 shows the functions with the highest r^2 in one of the folds. In Appendix B, the best functions are checked to show that these functions are indeed the exact functions. These functions actually achieved a r^2 of 1.0 and it can be easily verified that these obtained function are the actual functions.

Sampling method	EL	ELQ	EW	EWQ
Simple	0.825	0.874	0.710	0.717
Stratified	0.894	0,873	0.610	0.690
Systematic	0.850	0.816	0.741	0.657

Table 16: Average r^2 in 10-fold GP using different sampling methods M/M/s queue with prior knowledge

Metric	Best obtained function	Actual function
EL	$\frac{\lambda(-C\mu+\lambda-\mu s)}{\mu(\lambda-\mu s)}$	$\frac{\rho C}{1-\rho}+\rho s$
ELQ	$-\frac{C\rho}{\rho-1}$	$\frac{\rho C}{1-\rho}$
EWQ	$-\frac{C}{\mu s(\rho-1)}$	$\frac{C}{s\mu-\lambda}$

Table 17: Interpretable functions with the highest r^2 using prior knowledge M/M/s

Conclusion: Similar to the M/G/1 queue, using prior knowledge in the M/M/s system improves the speed of convergence to obtain the exact closed-form expressions for the EL, ELQ and EWQ performance metrics. Furthermore, there is no sampling technique that results in better approximations.

6.2 Case 2: Finding closed-form approximations of non-derivable performance metrics for single node queuing systems

In this second part, we try to obtain performance metrics for the M/G/s queue. First, we show the parameter bounds of the used parameters. In this case, we only use simple random sampling to narrow down the amount of computation. The performance metrics are here obtained by simulating queuing instances. Afterwards, we give the result of 10-fold cross-validation and the best obtained functions with the highest r^2.

6.2.1 Parameter bounds

Table 18 shows the bounds of the parameters used. For the service time distribution, we consider the gamma, beta and lognormal distribution. For the beta distribution it holds that the range is between $(0, 1)$ and the variance is

between $(0, ES(1-ES))$. Therefore, these are the bounds on which we sample for the $\sigma^2(S)$.

Parameter	Lower bound	Upper bound	Category
ρ	0	0.99	Continuous
ES	0.1	1	Continuous
var S	0	ES*(1-ES)	Continuous
s	1	50	Discrete

Table 18: Parameter bounds

6.2.2 Genetic programming parameter setting

Table 19 gives the parameters used in the GP algorithm for obtaining the performance metrics for the M/G/s queues. As the M/G/s is more complex, the initial depth is more varied.

Parameter	Value	Parameter	Value	Parameter	Value
Population size	1000	Init method	Ramped half and half	p_{sc}	0.9
Generations	20	Function set	[*,-,+,/]	p_{sm}	0.01
Tournament size	20	Metric	Mean Absolute Error	p_{hm}	0.01
Stopping criteria	0	Parsimony coefficient	0.001	p_{pm}	0.01
Init depth	(2,10)	Constant range	(0,0)	p_{pr}	0.05

Table 19: Genetic Programming parameter settings

6.2.3 M/G/s

For the M/G/s queue, we can use some prior knowledge to obtain the relationship. We use $\rho = \frac{\lambda}{ES*s}$, $C = P(W^Q > 0)$ and $R = \frac{1}{1-\rho}$, which are all gathered from the simulation. Another variable used is $c^2 = \frac{\sigma^2(S)}{ES^2}$. The independent variables are $\lambda, ES, \sigma^2(S), s, C, \rho, R, c^2$ and the dependent variables are the performance metrics. Table 20 shows the r^2 obtained over 10-fold Cross-validation. Table 21 shows the functions with the highest r^2. For these best functions, EWQ has a r^2 of 0.926042 and EW 0.872583 in one of the fold. As can be seen, the probability of waiting is a critical variable in determining these performance metrics.

Sampling method	EL	ELQ	EW	EWQ
Simple	0.524	0.685648	0.569392	0.534159

Table 20: Average r^2 in 10-fold GP M/G/s queue

Metric	Best function
EW	$\frac{ESs+PWQ^4\left(ESs+PWQ^2(ES+PWQ)\left(PWQ^3s+1\right)\right)+PWQ(ES+PWQ)}{s}$
EWQ	$\frac{PWQ\left(PWQ^2\left(PWQc^2+varS\right)\left(PWQs\left(PWQc^2+varS\right)(PWQ(PWQ-s)-varS)+c^2\rho(PWQ-s)\right)+c^2\rho(PWQ-s)\right)}{c^4s(PWQ-s)}$

Table 21: Interpretable functions with the highest r^2 using prior knowledge

Table 21 shows that the expected waiting time is highly dependent on the probability on waiting. The probability on waiting term is frequently present in the closed-form approximation. The r^2 scores in Table 20 show that GP is more able to obtain a better closed-form approximation for the ELQ compared to other performance metrics. As would be expected, as the function becomes more complex, it is harder to find a closed-form expression that has a high r^2. However, giving the GP enough prior knowledge (such as the heavy-traffic limit) can help GP in the right direction and, therefore, GP will converge faster.

Conclusion : For the M/G/s queue, finding a accurate closed-form approximation for the performance metrics is shown to be indeed hard.

7 Discussion

In this research paper, symbolic regression tried to find closed-form approximations for performance metrics of queuing instances. In the first experimentation case, GP was shown to be able to find the closed-form expressions for the performance metrics in the space of M/M/1 queues. Harder instances such as the M/G/1 and the M/M/s required some prior knowledge to derive these relationships. During the experimentation it was observed that giving GP the right input it can uncover the underlying function. By using some prior knowledge, the search is simplified, as complex parts of the equation are stored in variables.

What did not seem to have a significant impact was the sampling method. It was not clear that one sampling method outperformed other sampling methods. In this first experiment case, changing parameters of the GP is not investigated so varying the parameter could lead to an algorithm that is better in finding the closed-form expressions that belong to the performance metrics. The data used in the first case did not have a measurement error. Therefore, checking if the actual underlying function is obtained is easy as the error should be 0. Of course, in real life measurement errors are unavoidable.

In the last experiment, GP tried to find the performance metrics, the mean number of customers in the system and the mean waiting time distribution, for the M/G/s queue and found a function which might lead to more research. Only three different service time distributions are considered in this experi-

mentation and the subspace on which the parameters varied was rather limited. Increasing the subspace of the parameters does not necessarily push GP in the right direction as big errors can lead to over-fitting when the bloat mechanisms are not sufficient working. By lack of time, further research can be done on using different parameter settings for the GP. As was shown by giving the GP prior knowledge can give GP the right direction.

8 Conclusion

This research paper investigates the approximation of key performance metrics of single node queuing systems. Conclusions are:

- Derivable key performance metrics of single node queuing systems can be exactly reproduced by genetic programming.
- Giving prior knowledge, such as the heavy-traffic behaviour, can help genetic programming in the search for closed-form expressions of key performance metrics and improve the speed of convergence.
- Using different sampling techniques does not necessarily result in faster convergence.

9 Further research

Future research can investigate in for example the GI/M/s queue and the GI/G/1 queue. Finding out which prior knowledge is required to obtain a relationship can reduce the search in the form of the performance metric. Furthermore, in this research there is no study in which parameter setting for the GP might improve the speed of convergence and obtaining a closed-form expression with a higher r^2 for queuing systems where performance metrics are non-derivable. Experimenting with these parameters can result in better convergence.

References

Alexopoulos, C., & Seila, A. (1996). Implementing the batch means method in simulation experiments. In *Proceedings of the 28th conference on winter simulation* (pp. 214–221). Washington, DC, USA: IEEE Computer Society. Retrieved from `http://dx.doi.org/10.1145/256562.256608` doi: 10.1145/256562.256608

Brameier, M., & Banzhaf, W. (2007). *Linear genetic programming*. Springer Science+Business Media, LLC.

Chellapilla, K. (1997, 9). Evolving computer programs without subtree crossover. *IEEE Transactions on Evolutionary Computation*, *1*(3), 209-216. doi: 10.1109/4235.661552

Eiben, A. E., & Smith, J. E. (2015). *Introduction to evolutionary computing* (2nd ed.). Springer Publishing Company, Incorporated.

Gross, D., & Harris, C. M. (1998). *Fundamentals of queueing theory* (3rd ed.). John Wiley & Sons, Inc.

Gustafson, S. M. (2004). *An analysis of diversity in genetic programming* (Unpublished doctoral dissertation). University of Nottingham.

Hancock, P. J. B. (1994). An empirical comparison of selection methods in evolutionary algorithms. In T. C. Fogarty (Ed.), *Evolutionary computing* (pp. 80–94). Berlin, Heidelberg: Springer Berlin Heidelberg.

Hristov, A. (2018). *Performance models for analysis and control of it systems* (Unpublished doctoral dissertation). Vrije Universiteit Amsterdam. (Chapter 4)

Koole, G. (2014). *Optimization of business processes*. MG books.

Koza, J. (1998). *Genetic programming: On the programming of computers by means of natural selection* (6th ed.). The MIT Press.

Liu, H., & Motoda, H. (2002, 4). On issues of instance selection. *Data Mining and Knowledge Discovery*, *6*(2), 115–130. Retrieved from `https://doi.org/10.1023/A:1014056429969` doi: 10.1023/A:1014056429969

Lu, Q., Ren, J., & Wang, Z. (2016, 01). Using genetic programming with prior formula knowledge to solve symbolic regression problem. *Computational Intelligence and Neuroscience*, *2016*, 1-17. doi: 10.1155/2016/1021378

Luke, S., & Spector, L. (1997). A comparison of crossover and mutation in genetic programming. In J. R. Koza et al. (Eds.), *Genetic programming 1997: Proceedings of the second annual conference* (pp. 240–248). Stanford University, CA, USA: Morgan Kaufmann.

Luke, S., & Spector, L. (1998). A revised comparison of crossover and mutation in genetic programming. In *Genetic programming 1997: Proceedings of the second annual conference* (pp. 240–248). Morgan Kaufmann.

Miller, J. (2011). *Cartesian genetic programming*. Springer-Verlag Berlin Heidelberg.

Narayan Bhat, U., Shalaby, M., & J. Fischer, M. (1979, 06). Approximation techniques in the solution of queueing problems. *Naval Research Logistics Quarterly*, *26*, 311 - 326. doi: 10.1002/nav.3800260211

Onderwater, M., Bhulai, S., & van der Mei, R. (2016, 9). Value function discovery in markov decision processes with evolutionary algorithms. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *46*(9), 1190-1201. doi: 10.1109/TSMC.2015.2475716

Perkis, T. (1994, 07). Stack-based genetic programming. In *Proceedings of the first ieee conference on evolutionary computation. ieee world congress on computational intelligence* (Vol. 1, p. 148-153). doi: 10.1109/ICEC.1994.350025

Poli, R., B. Langdon, W., & Mcphee, N. (2008). *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`. (With contribution by J.R. Koza)

Poli, R., & McPhee, N. F. (2014). Parsimony pressure made easy: Solving the problem of bloat in gp. In Y. Borenstein & A. Moraglio (Eds.), *Theory and principled methods for the design of metaheuristics* (pp. 181–204). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from `https://doi.org/10.1007/978-3-642-33206-7_9` doi: 10.1007/978-3-642-33206-7_9

Raman, D., Nagalingam, S. V., & Gurd, B. W. (2009). A genetic algorithm and queuing theory based methodology for facilities layout problem. *International Journal of Production Research*, *47*(20), 5611-5635. Retrieved from `https://doi.org/10.1080/00207540802057352` doi: 10.1080/00207540802057352

Semen, P., Sergey, T., Viktor, B., & Sergey, S. (2006). Optimal scheduling of queueing networks with switching times using genetic algorithms. In *Proceedings of the 8th wseas international conference on automatic control, modeling & simulation* (pp. 287–292). Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS). Retrieved from `http://dl.acm.org/citation.cfm?id=1973458.1973510`

Stephens, T. (2018). *gplearn documentation.* Retrieved 2019-02-09, from `https://media.readthedocs.org/pdf/gplearn/latest/gplearn.pdf`

Taherdoost, H. (2016, 01). Sampling methods in research methodology; how to choose a sampling technique for research. *International Journal of Academic Research in Management*, *5*, 18-27. doi: 10.2139/ssrn.3205035

Van der Stap, I. (2016). *Approximating queueing functions with simulation and data analysis.* Retrieved from `https://beta.vu.nl/nl/Images/werkstuk-stap_tcm235-781190.pdf`

Xie, H. (2008). *An analysis of selection in genetic programming* (Unpublished doctoral dissertation). Victoria University of Wellington.

Appendix A: Validation for obtained approximations for M/G/1

In this section, it is shown that the best obtained functions for the performance metrics of the M/G/1 queue are indeed their exact closed-form expression.

EL:
$\lambda\left(0.5ESR\rho + ES + 0.5R\lambda\sigma^2(S)\right) = \lambda\frac{1}{2}ES\frac{\rho}{1-\rho} + \lambda ES + \frac{1}{2}\frac{\lambda^2\sigma^2(S)}{1-\rho} = \frac{\rho^2}{2(1-\rho)} + \frac{\lambda^2\sigma^2(S)}{2(1-\rho)} +$
$\rho = \frac{\rho^2+\lambda^2\sigma^2(S)}{2(1-\rho)} + \rho = \frac{\rho^2(1+\frac{\lambda^2\sigma^2(S)}{\rho^2})}{2(1-\rho)} + \rho = \frac{\rho^2(1+c^2(S))}{2(1-\rho)} + \rho$

ELQ:
$\frac{0.5R\rho\left(ES\rho+\lambda\sigma^2(S)\right)}{ES} = \frac{\rho(\rho+\frac{\lambda\sigma^2(S)}{ES})}{2(1-\rho)} = \frac{\rho^2(1+\frac{\lambda\sigma^2(S)}{ES\rho})}{2(1-\rho)} = \frac{\rho^2(1+c^2(S))}{2(1-\rho)}$

EW:
$0.5ES\left(R+1\right) + 0.5R\lambda\sigma^2(S) = \frac{1}{2}(\frac{ES}{1-\rho} + ES) + \frac{\lambda\sigma^2(S)}{2(1-\rho)} = \frac{1}{2}(\frac{ES}{1-\rho} + \frac{ES(1-\rho)}{1-\rho}) +$
$\frac{\lambda\sigma^2(S)}{2(1-\rho)} = \frac{2ES-ES\rho}{2(1-\rho)} + \frac{\lambda\sigma^2(S)}{2(1-\rho)} = \frac{2ES-ES\rho+\lambda\sigma^2(S)}{2(1-\rho)} = \frac{\rho ES(\frac{2}{\rho}-1+\frac{\lambda\sigma^2(S)}{\rho ES})}{2(1-\rho)} = \frac{\rho ES(\frac{2}{\rho}-2+1+c^2(S))}{2(1-\rho)} =$
$\frac{\rho ES(1+c^2(S))}{2(1-\rho)} + \frac{ES(2-2\rho)}{2(1-\rho)} = \frac{\rho ES(1+c^2(S))}{2(1-\rho)} + ES$

EWQ:
$0.5R\left(ES\rho + \lambda\sigma^2(S)\right) = \frac{ES\rho+\lambda\sigma^2(S)}{2(1-\rho)} = \frac{ES\rho(1+\frac{\lambda\sigma^2(S)}{\rho ES})}{2(1-\rho)} = \frac{\rho ES(1+c^2(S))}{2(1-\rho)}$

Appendix B: Validation for obtained approximations for M/M/s

In this section, it is shown that the best obtained functions for the performance metrics of the M/M/s queue are indeed their exact closed-form expression.

EL:
$\frac{\lambda(-C\mu+\lambda-\mu s)}{\mu(\lambda-\mu s)} = \frac{(-C\lambda\mu+\lambda(\lambda-\mu s))}{\mu(\lambda-\mu s)} = \frac{-C\lambda\mu}{\mu(\lambda-\mu s)} + \frac{\lambda}{\mu} = \frac{-C}{(1-\frac{\mu s}{\lambda})} + \frac{\lambda}{\mu}$
$= \frac{-C}{(1-\frac{1}{\rho})} + \rho s = \frac{\rho C}{(1-\rho)} + \rho s$

ELQ:
$-\frac{C\rho}{\rho-1} = \frac{\rho C}{1-\rho}$

EWQ:
$-\frac{C}{\mu s(\rho-1)} = \frac{C}{\mu s(1-\rho)} = \frac{C}{s\mu-\lambda}$