Hacking Continuous Probability Distributions into Pieces

Ramon Onis
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
r.onis@student.utwente.nl

ABSTRACT

When dealing with complex systems with probabilistic and real-time properties, an analysis is often needed in order to improve or understand certain aspects of the system (e.g. making machine maintenance more efficient or measure the chance that a network package gets transferred in time). In order to get insight in these systems, certain models can be constructed for analysis. The Modest Toolset supports the modelling and analysis of such systems. In order to analyse these systems, continuous probability distributions need to be converted to discrete probability distributions. The current implementation of this conversion does not scale well for complex systems and is not precise enough in many cases. A better method is required in order for the toolset to provide more accurate analysis. In this paper, we propose different methods to convert these continuous distributions into discrete distributions, report on the implementation and evaluate the performance.

Keywords

stochastic timed automata, probabilistic timed automata, modelling, continuous probability distributions

1. INTRODUCTION

There are many situations in which finite automata are a useful tool to model and analyse complex systems.
Finite state automata (FSA) are the most basic model that can be used to analyse a wide range of systems. For example, a vending machine counting deposited coins can perfectly be modelled using FSA.
However, FSA are not able to model all aspects of a system. For example, when modelling real-time systems, it might be a requirement that the model supports timers and delays. To model and analyse such systems, timed automata (TA) can be used [1].
Another requirement may be that the transitions might not be deterministic given a next symbol, but probabilistic. Probabilistic automata (PA) support these kind of transitions [10]. This allows for discrete probability distributions to be included in PA.
When combining both the timed and probabilistic properties of these types of automata, we are dealing with probabilistic timed automata (PTA) [7].
A final requirement we might consider is the ability to include stochastic variables (e.g. time between failure of a machine, propagation delay of a signal) in a model. The value of these variables gets to be determined by continuous probability distributions. For example, three frequently used probability distributions are uniform, exponential and normal distributions. When appending PTA with the feature to include stochastic variables based on continuous probability distributions (e.g. uniform, exponential and normal distributions), we are dealing with stochastic timed automata (STA) [3].

The Modest Toolset [6] already has a tool, mcpta, to perform an exact analysis of PTA [5]. However, in order to analyse STA, the toolset must convert them to PTA by replacing the stochastic decisions (based on continuous probability distributions) with probabilistic finite-state systems.
The tool, mcsta, does this by dividing the continuous probability distribution into sequential intervals, effectively constructing a discrete probability distribution that approximates the original distribution. The discrete distribution can be included into the newly constructed PTA. The constructed PTA is now a approximation of the original STA. How precise the approximation will be depends on how the continuous distribution is divided into finite intervals. Currently, the toolset divides the continuous distribution into equally sized intervals of length 1 (see Figure 1). However, this method does not scale well for more complex systems and lacks the precision that is desired.
In Section 5, we will describe new methods of approximating continuous distributions and provide the theory needed to implement the new methods. In Section 6, we

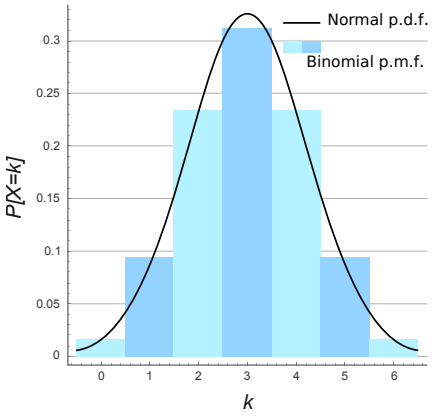

Figure 1. A discrete approximation of a normal distribution. Note that the discrete distribution is similar to the binomial distribution [11].

25^{th} *Twente Student Conference on IT* June 1^{st}, 2016, Enschede, The Netherlands.

provide details on how the described methods are implemented in the Modest Toolset. In Section 7, we evaluate the implementations using several models included in the toolset. Finally, in Section 8, we return to the research questions and suggest future work.

2. RESEARCH OBJECTIVES

The research in this paper is conducted in order to come up with the best method to approximate continuous probability distributions with discrete distributions which allows the Modest Toolset to convert STA to PTA.
There are two requirements for the proposed method that should be taken into account:

- The method should enable the toolset to construct a precise approximation of an STA in the form of a PTA.
- The analysis of this PTA should have a reasonable time/memory performance.

2.1 Research questions

Based on the above requirements, the following research question can be established. The research can be subdivided into three parts: developing, implementing and comparing possible methods. Each part corresponds to a subquestion:

- What is the best method, in terms of analysis speed/memory performance and precision, to convert continuous probability distributions to discrete probability distributions?
  - What are possible methods to convert continuous probability distributions to discrete probability distributions?
  - How can these methods be implemented into the Modest Toolset in an efficient way?
  - How can the implementations be compared in terms of analysis speed and precision.

3. BACKGROUND INFORMATION

This paper will focus mainly on continuous probability distributions and two types of automata, STA and PTA, and the tool that approximates STA with PTA to analyse STA.

3.1 Automaton types

In this paper, there are two types of automata to consider. The first type is the automaton that combines stochastic decisions, discrete probabilities and clocks/delays called stochastic timed automaton (STA).
The second type is the automaton that combines both discrete probabilities and clocks/delays called probabilistic timed automaton (PTA). It can be obtained from an STA by replacing the continuous probability distributions in a PTA with a discrete probability distribution. Currently, the Modest Toolset is able to only analyse PTA directly. To analyse STA, an approximating PTA has to be obtained from it.

3.2 mcsta

mcsta is the **M**odest **c**hecker for **s**tochastic **t**imed **a**utomata.
An important step in the model-checking of STA is to approximate them with PTA. In this step it is necessary to convert continuous probability distributions with approximating discrete distributions. The probability distributions that have to be converted can be constant, meaning that they have concrete values as parameters (e.g. $Exp(1.5)$ for an exponential distribution with a rate of 1.5), or variable, meaning that they can have mathematical expressions as parameters (e.g. $Uni(a+4, b*4)$ for a uniform distribution with bounds $a+4$ and $b*4$).
The resulting discrete distribution must be returned as a mapping of intervals to their corresponding probabilities. In the case of variable probability distributions, the bounds of the intervals and their corresponding probabilities can be mathematical expressions as well. Furthermore, because of how the toolset only allows PTA to have integer clock constraints, the bounds of the intervals are limited to integers [4].
The properties of a STA that can be analysed using mcsta are minimum and maximum reachability probabilities/rewards (e.g. maximum probability that a queue is full within 10s, minimum time before a packet is received). The results of the analysis are 'safe' approximations of these minimums/maximums, meaning that the resulting minimums/maximums are lower/upper bounds and do not violate the actual minimum/maximum (e.g. when the actual maximum of a property is 10, the resulting maximum can be higher than 10, but not lower).

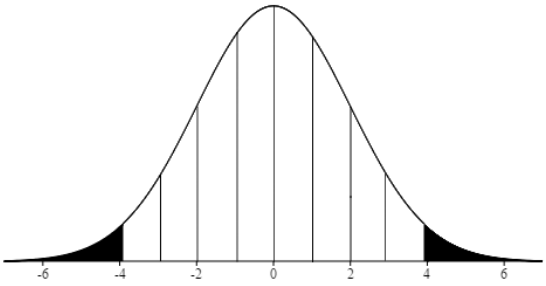

Figure 2. This is how the current implementation would unroll a normal distribution with mean 0 and standard deviation 2.

4. RELATED WORK

A simple method has already been described to approximate STA with a PTA which is the method that the Modest Toolset currently uses[4]. However, this method does not meet the requirements of allowing precise and fast analysis.

4.1 Current Method

The current implementation divides the probability distributions into sequential intervals of length 1. For probability distributions that are supported on (semi-) infinite intervals (e.g. normal/exponential distributions), the current implementation stops creating interval when the residual probability is smaller than a given threshold [4]. The residual probability is given as a parameter of the mcsta tool which can be set to a lower value in order to increase the amount of intervals that are created.
Figure 2 shows how the current implementation unrolls the normal distribution $Norm(0, 2)$. To achieve a residual probability of 0.05, only intervals within [-3.921, 3.921] are created (this is the white area under the curve). The first and last bound will be rounded down and up respectively as only integer bounds are supported. The black areas will be included as half open intervals.
The current implementation only allows exponential, normal (only with integer means and constant standard devi-

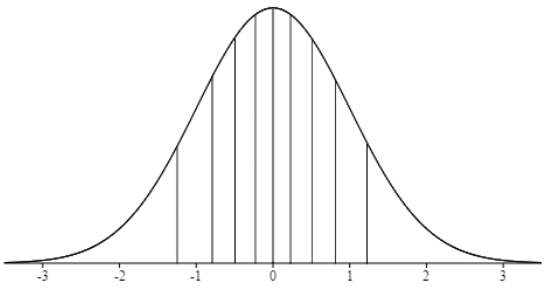

Figure 3. This is how a normal distribution with mean 0 and standard deviation 2 would be divided when 10 intervals are needed of equal probability mass.

ations) and uniform distributions.
The current method creates as many intervals of length 1 as necessary to cover a probability of at least $1 - \rho$, where ρ is the residual probability. However, this does not scale well with probability distributions with a large standard deviation. For example, the uniform distribution $U(0, 1000)$ will yield 1000 intervals, even when 100 intervals would be enough for a precise analysis.

5. RESEARCH - NEW METHODS

In this section, we will describe several methods to approximate continuous probability distributions with discrete probability distributions. We will first begin with describing how the current implementation works. After that, we will describe alternative methods.

5.1 New methods

We now present alternative methods to approximate continuous distributions.

5.1.1 Intervals with equal probability masses

We will now present a method that creates n intervals, where n is a number given by the user. By only asking for the amount of intervals to be created, this method should scale better when using distributions with a large standard deviation.
When it is specified how many intervals should be created, the probability mass of the intervals can be determined. To achieve intervals of equal probabilities, each interval must have a probability mass of $\frac{1}{n}$. To do this, we first define bound numbers:

Definition 1. The bound number of a bound is a number $i \in [0, 1, \ldots, n-1, n]$ that indicates the position of a bound. The leftmost bound has bound number 0, the following bound has bound number 1 and the rightmost bound has bound number n.

To determine what the value of a bound is, we use the following definition:

Definition 2. The value of a bound x is follows from the cumulative probability p assigned to it, such that $P(X < x) = p$.

To get intervals of equal probabilities, we assign the cumulative probability $p = \frac{i}{n}$ to every bound. This way each interval, whose bounds have bound number i and $i + 1$, has a probability mass of $\frac{1}{n}$.
Figure 3 visualizes how the distribution from Figure 2 would be divided into 10 intervals with equal probabilities of 0.1.

To determine the value of a bound that follows from probability p, we need to use the inverse cumulative distribution function (inverse CDF, or quantile function). For a random variable X and a probability $0 < p < 1$, the quantile function can be expressed as follows:

$$Q_X(p) = \{x : P(X \leq x) = p\}.$$

Or in words: given a probability p, the quantile function of a random variable returns the value x for which the probability of the random variable being less than or equal to x is equal to p.
However, the interval bounds are limited to integers. Therefore, the resulting bounds must be rounded in some way to satisfy this limitation. As a result, the actual probabilities of the rounded intervals may differ from the initial probability mass of $\frac{1}{n}$. To solve this problem, the probability masses corresponding to the rounded intervals must be recalculated. In order to do this, we must use the cumulative distribution function (CDF):

$$F_X(x) = P(X \leq x).$$

We can use the CDF to calculate the probability mass corresponding to the interval $[a, b]$ in the as follows:

$$\begin{aligned} P(a \leq X \leq b) &= P(a < X \leq b) \\ &= P(X \leq b) - P(X \leq a) \\ &= F_X(b) - F_X(a). \end{aligned}$$

5.1.2 Smaller intervals near median

The method in Section 5.1.1 is based on giving intervals equal probability masses. Another method might be to give intervals near the median smaller probability masses than further away from the median. This could result in more precision near the median of the probability distribution. We choose the median as a measure of central tendency to focus on because the median can be easily determined using the inverse CDF: the bound that has probability $\frac{1}{2}$ assigned to it corresponds to the median. To determine how we make intervals near the median more precise, we modify the method in 5.1.1. We look at the fraction $r = \frac{i}{n}$, and how r will be converted to a cumulative probability p assign to the bound with bound number i. For the method in 5.1.1, we use $p = r$ to let each interval have a equal probability mass of $\frac{i}{n}$. For the implementation we need to express p in terms of r such that the value of p increases more slowly near $r = \frac{1}{2}$, the probability corresponding to the median. In stead of assigning probabilities $p = r = \frac{i}{n}$ to each interval as in Section 5.1.1, we thus use a different equation to determine p.

5.2 Scaling

A important restriction to consider is the fact that the resulting intervals must have integer bounds. When dealing with distributions with small standard deviations it is therefore hard to return precise intervals due to the low resolution. A solution to this is to include a scaling factor that scales up the time in the original model to increase the resolution of the analysis. To make this work, the parameters of the probability distributions need to be edited accordingly. For example, when using a time scaling factor s, the distributions $Uni(a, b)$, $Exp(\lambda)$ and $Norm(\mu, \sigma)$ need to be changed to $Uni(as, bs)$, $Exp(\lambda/s)$ and $Norm(\mu s, \sigma s)$ respectively.

Function *GetIntervals(X, n)*
  Data: X: a random variable, n: amount of intervals
  Result: A mapping of intervals to their corresponding probability masses
  $result = []$
  `// first bound`
  $leftBound = \lfloor Q_X(0) \rfloor$
  $leftP = 0$
  foreach *i in [1..n-1]* **do**
    $r = i/n$
    $rightBound = \lfloor Q_X(r) + 0.5 \rfloor$ `// nearest int`
    $rightP = F_X(rightBound)$
    $interval = [leftBound, rightBound]$
    $p = rightP - leftP$
    $result.append((interval, p))$
    $leftP = rightP$
    $leftBound = rightBound$
  end
  `// last bound`
  $rightBound = \lceil Q_X(1) \rceil$
  `// last interval`
  $interval = [leftBound, rightBound]$
  $p = 1 - leftP$
  $result.append((interval, p))$
  return *result*

Algorithm 1: Determine the rounded intervals and their corresponding probabilities given a random variable X and the number of intervals n.

6. IMPLEMENTATION

The implementation is done in C#, the same language the Modest Toolset is written in. The Math.NET Numerics library[9] that the toolset already uses is used to compute several types of distribution functions (e.g. CDF or inverse CDF) for a given probability distribution.
In order to also support probability distributions with mathematical expressions as parameters, the implementations will use the expression library in the Modest Toolset in order to support expression based parameters. This library supports most mathematical operators/functions, rounding and conditional expressions.

6.1 Intervals with equal probability masses

As described in Section 5.1.1, inverse CDF can be used to determine the bounds of the resulting intervals. After that, the bounds must rounded to satisfy the requirement that the interval bounds must be integers. To guarantee that the union of the rounded intervals is not smaller than the union of the unrounded intervals, the first bound (with bound numer 0) and last bound (with bound number n) will be rounded down and up respectively. This ensures that, for example, the uniform distribution $Uni(2.7, 6.3)$ gets unrolled in intervals within the bounds of 2 and 7 instead of 3 and 6 when rounding to nearest integer. All other bounds (with bound numbers in $[1, \ldots, n-1]$) will be rounded to the nearest integer to approximate the original unrounded intervals as precise as possible.
Observe that this results in the possibility that, after rounding, an interval might have equal bounds (i.e. a singleton interval $[a, a] = \{a\}$). In this case, the corresponding probability mass will be zero and the the singleton interval can be excluded from the result. We could say that the bounds with equal values are merged and treated as one. This makes it possible that the actual list of given intervals may have a smaller length than requested.
Algorithm 1 shows how CDF and inverse CDF can be used to determine the rounded intervals and their corresponding probabilities as described above. Note that when X arises from distributions with support on (semi-) infinite intervals, such as the exponential and normal distributions, the quantile function might return $-\infty$ and $+\infty$ for the first and last bound respectively. In this case, the ceiling and floor function will not alter the value of the bound and thus will return $\pm\infty$.

Using the Math.NET Numerics library the following continuous probability distributions can be supported when only concrete numbers are used as parameters: beta, Cauchy, chi-squared, exponential, Fisher-Snedecor, gamma, log-normal, normal, Pareto, Rayleigh, Student's t, triangular and uniform. The implementation also supports expression based parameters for the exponential, normal and uniform distribution with the same parameter requirements as the original implementation: only the normal distribution must have integer means and a concrete value for a standard deviation. The reason for this limitation is that the CDF and inverse CDF of the normal distribution can not be expressed using the expressive power of the expression library. To still support normal distributions with expression based integer means, we first calculate the interval bounds for a mean of 0, after which we add the real mean to each bound. The requirement that the mean must be an integer guarantees that the intervals need not to be rounded after adding the mean.

```
var e = new NumericValue(new
    Rational(Math.E));
var minusRate = new UnaryMinus(rate,
    rate.Location);
var minusRateTimesX = new
    Multiplication(minusRate, x);
var ePow = new FunctionCall (
    FunctionSymbol.Pow, new Expression[] {
    e, minusRateTimesX });
var one = new NumericValue(1);
return new Subtraction(one, ePow);
```

Listing 1. How the CDF of the exponential distribution would be expressed using the expression library. Location parameters in the expression constructors are excluded for simplification.

In order to support other distributions, their CDF and inverse CDF must be expressible using the expressive power of the library. As an example, Listing 1 shows how the CDF of the exponential distribution $(1 - e^{-\lambda x})$ would be expressed using the expression library. Note that we approximate e with a rational number.

Table 1 shows the support level of each distribution in the Math.NET Numerics library. *Syntactically* indicates that the distribution is syntactically supported, but cannot be unrolled in the implementation (due to Math.NET not containing the inverse CDF of the distribution). *Expression based* means that the implementation fully supports expression based parameters. *Expression based possible* means that expression based support is possible, but not implementation is limited to only supporting concrete values as parameters. All other distributions are supported with concrete values as parameters, but are need addition expressional power in the expression library to support expression based parameters.

6.2 Smaller intervals near median

As a function that converts the $\frac{i}{n}$ ratio as described in

Table 1. List of distributions supported by the toolset.

Name	Support Level
Beta	Need beta function
Cauchy	Expression based possible
Chi	Syntactically
Chi-squared	Expression based possible
Erlang	Syntactically
Exponential	Expression based
Fisher-Snedecor	Need regularized beta function
Gamma	Need incomplete gamma function
Inverse gamma	Syntactically
Laplace	Syntactically
Log normal	Need (inverse) error function
Normal	Expression based(limited)
Pareto	Expression based possible
Rayleigh	Expression based possible
Stable	Syntactically
Student's t	Need regularized beta function
Uniform	Expression based
Weibull	Syntactically
Triangular	Expression based possible

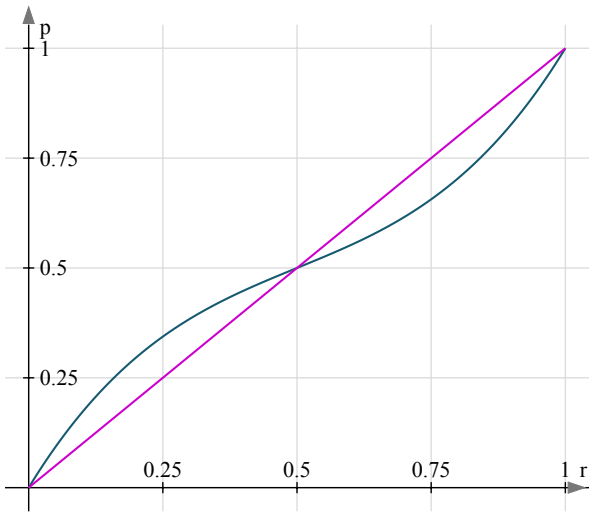

Figure 4. The equation $p = 2r^3 - 3r^2 + 2r$. (curved line) along with the equation $p = r$ that is used to create intervals with equal probability mass (straight line).

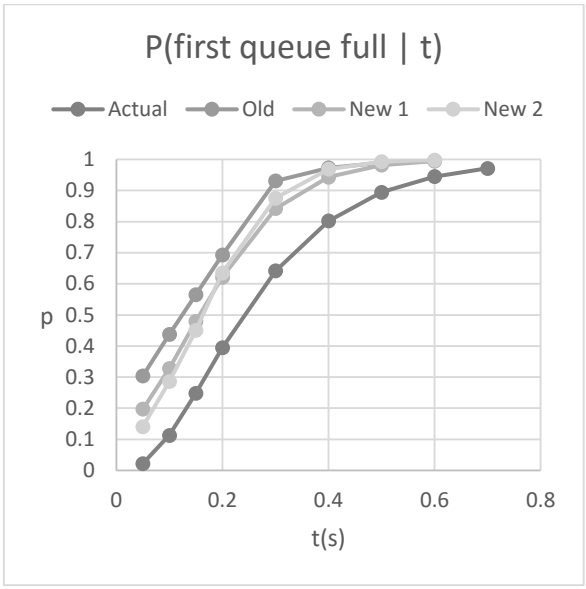

Figure 5. Results of evaluating the maximum probability of the first queue being full in time t.

5.1.2 to a cumulative probability p we use the equation

$$p = 2r^3 - 3r^2 + 2r.$$

See Figure 4 for a comparison with the equation that is used to create intervals with equal probability mass. Note other equations are also possible (other polynomials or equations based on trigonometric function), but these functions would give a similar curve as the function we use would give.

The implementation of this method is almost identical to the implementation in 6.1. The only adjustment lies in line 7 of Algorithm 1 where r has to be replaced with $2r^3 - 3r^2 + 2r$..

7. EVALUATION

In order to evaluate the performance of the implemented methods, we use different models. The important benchmarks that are used to compare the methods are how much memory they will use, how long the analysis will take and how close the results are to the actual value. In this section, method 1 and method 2 to refer to the methods implemented in Section 6.1 and Section 6.2 respectively. All of the measurements were done on a 2.6GHz (up to 3.2GHz max turbo frequency) Intel Core i5-3230M system with 8GB RAM (1600MHz) running 64-bit Windows 10.

7.1 Tandem queuing network

The tandem queuing network is a model that is included as a sample/case study in the Modest Toolset. It simulates two queues connected sequentially. It was used to evaluate the original implementation of the model checker [4]. The original model was part of the Prism benchmark suite[8] as a continuous-time Markov chain (CTMC). Due to the model being a CTMC, the Prism model checker can be used to perform precise analysis in order to provide the actual analysis values.

The model is very challenging for a model checker due to the effect of errors adding up with four exponential delays. Also the model already uses a scaling factor that increases the precision of the analysis, but requires more memory.

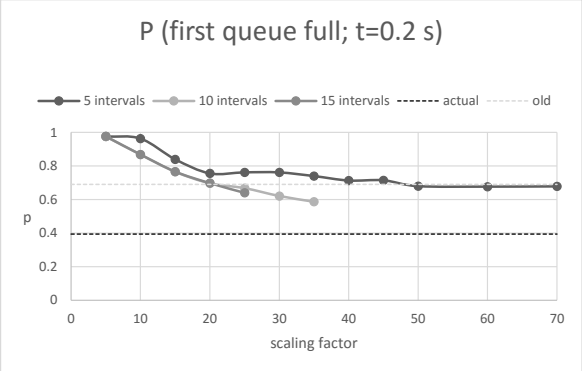

Figure 6. Results of evaluating the maximum probability of the first queue being full with different scaling factors and number intervals for equal probabilities.

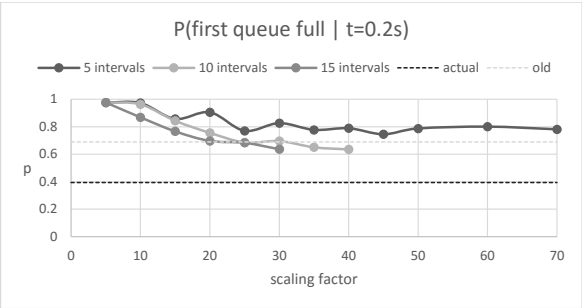

Figure 7. Results of evaluating the maximum probability of the first queue being full with different scaling factors and number intervals for narrower intervals near the median.

One property to analyse is the maximum probability of the first queue being full after time t. In Figure 5, we see the results when evaluating the maximum probability p of the first queue being full in a given time t. The actual values were computed using the Prism model checker. For the old implementation, we use a scaling factor of 20 and $\rho = 0.10$ for $t <= 0.2$ and a scaling factor of 10 and $\rho = 0.05$ for $t > 0.2$. For the new implementations, we experimented with different scaling factors and number of intervals. The settings that gave the best results while taking a similar amount of time as the old implementations are: 10 intervals and a scaling factor of 30 for creating intervals with equal probability mass (*New 1* in the figure), and 10 intervals and a scaling factor of 40 when making intervals narrower near the median (*New 2* in the figure). We can see that both new implementations get better results than the old implementations. The old implementation took 10 minutes to perform the analysis while the new implementation took around 8 minutes to perform the analysis. Note that the second method gave better results than the first method for $t < 0.2$.

In Figure 6 and Figure 7 we see how the probability of the first queue being full within a time bound of 0.2 s grow towards the actual value. As the time bounds after scaling must be integers, the scaling factors must be multiples of 5. For each amount of intervals, the scaling factor was increased until the analysis required more memory than the old method required (around 4600 MB) or took more time than the old method required (10 minutes). The result from the old method that is included was done with a scaling factor of 20 and $\rho = 0.10$. We see that for both methods, using 10 intervals yielded the most accurate results. Also, we can see that the method that makes narrower intervals near the median can handle a larger scaling factor without using too much memory or time. However, this does not result in more precise results.

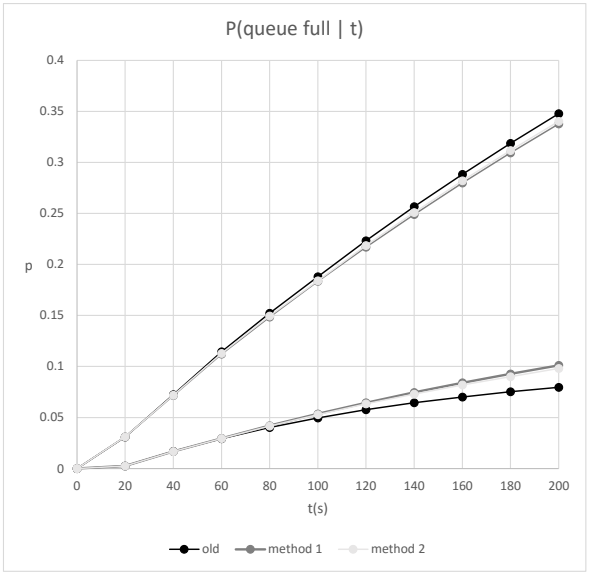

Figure 8. Results of evaluating the maximum and minimum probability of the fileserver queue being full within time t.

7.2 Fileserver queue

The fileserver queue is a model included in the Modest Toolset that combines many features of STA, including nondeterministic delays, nondeterministic choices, uniform distributions and exponential distributions.

The first property we are interested in is the probability p that the queue is full within time t. For the analysis, we use a time bound of 200. The old method with $\rho = 0.05$ could evaluate the minimum and maximum bound of p in $39s$. The settings for the two new methods that resulted in a similar run time ($\pm 1s$) are $n = 50$ and $n = 85$ for method 1 and 2 respectively.

The results for the different methods using the above settings are shown in Figure 8. We see that both new methods give tighter minimums/maximums than the old method, especially for the minimums. Also, method 1 gives slightly better results than method 2.

Next, we are interested in the time t before the queue is full for the first time. We use modes, using a scheduler to delay events as soon or as soon as posible (ASAP/ALAP) to resolve the nondeterministic delay, to approximate t. This gives us $t \approx 978$ for ASAP and $t \approx 694$ for ALAP. For the analysis with the old/new methods, we use the settings that give a run time of $20s \pm 1s$ when evaluating the minimum value for p. For the old method, $\rho = 0.04$ gives us a lower bound for t of 474. Method 1, with $n = 70$, gives us a lower bound of 489. Method 2, with $n = 115$, gives us a lower bound of 488.

7.3 M/G/1/c queue

The M/G/1/c queue is a model included in the Modest Toolset that representing a queue with size c, exponentially distributed time between customer arrivals and normally distributed processing time for each customer. For

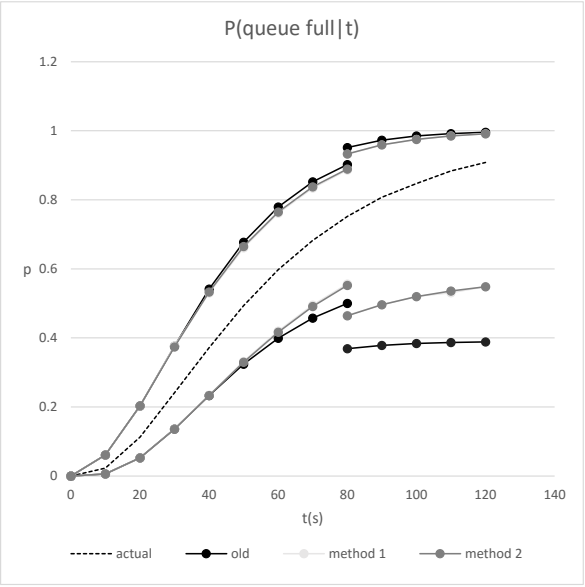

Figure 9. Minimum and maximum probabilities of the queue being within time t. Method 1 and 2 give almost equal results.

Table 2. Expected minimum values for m/g/1/5-queue of c and t.

Method	$E_{min}(c)$	$E_{min}(t)$
modes	6.19	61.1
old method, $\rho = 0.014$	4.28	46.9
new method 1, $n = 200$	4.50	47.4
new method 2, $n = 400$	4.50	47.4

this analysis, we set the queue to size 5, customer arrival rate $\frac{1}{6}$ and processing time for each customer follows the normal distribution $Norm(10, 2)$. The actual values of the properties we analyse can be approximated using the statistical model checker modes [2].

The first property we are interested in is the probability p that the queue is full in time t. For evaluation, we use 2 different settings for $t \leq 80$ and for $t \geq 80$. For the old implementation, we use $\rho = 0.05$ for $t \leq 80$ and $\rho = 0.10$ for $t \geq 80$. The run times are $26.7s$ for time bound 80 and $\rho = 0.05$ and $21.0s$ for time bound 120 and $\rho = 0.10$. We choose the amount of intervals for the 2 new methods such that the run time was equal ($\pm 1s$) to the run time of the old implementation. For method 1, this is 25 and 50 intervals for $t \leq 80$ and $t \geq 80$ respectively. For method 2, this is 45 and 85 intervals for $t \leq 80$ and $t \geq 80$ respectively.

The results of all methods using the above settings are shown in Figure 9. We can see that both new methods give tighter bounds for p, especially the lower bound for $t \geq 80$. There is no significant different between the results of the new methods.

The next properties we are interested in is the expected time t until the queue is full and the amount of customers served c before the queue is full. Using modes, we approximate the actual value which gives us $c \approx 6.19$ and $t \approx 61.1$.

For each method, we use the settings that resulted in a runtime of $20s \pm 0.5s$. The results are shown in Table 2. We see that the two new methods give marginally better results for c and t than the old method. We also see that, despite method 2 is done with twice as much intervals without a bigger run time, both new methods give the same results. This is probably because the calculated bounds in method 2 have a higher tendency to be merged near the median after rounding compared to method 1.

8. CONCLUSIONS AND FUTURE WORK

We successfully developed and implemented two methods for approximating continuous probability distributions with discrete probability distributions, both based on giving intervals in the discrete probability distributions certain probability masses. The implementation supports many different continuous probability distributions on different levels (see Table 1 for details). By evaluating the old and new methods with different models provided in the Modest Toolset, we saw that both new methods gave better analysis results for all models. However, the evaluation did not clearly tell us which of the two new methods gives better results in the same time frame. In general, giving each interval equal probability masses resulted in slightly better results, but one might say that the difference between the results of the two methods is not enough to tell with one is better.

8.1 Future Work

Although we did find methods that are better in terms of precision/speed than the old method, we did not explore all of the possible methods to approximate continuous probability distributions in STA. We could, for example, develop a method that can look at the model as a whole and determine how precise each probability distributions in a model need to be approximated in order to give precise analysis results in a short run time.

Also, more research is needed into the difference between the two methods that have been developed in this paper. An interesting question is what would happen when a different equation (see Section 6.2) is used to determine the probability assigned to the interval bounds.

9. REFERENCES

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[2] J. Bogdoll, A. Hartmanns, and H. Hermanns. Simulation and statistical model checking for modestly nondeterministic models. In *MMB/DFT*, volume 7201 of *Lecture Notes in Computer Science*, pages 249–252. Springer, March 2012.

[3] H. C. Bohnenkamp, P. R. D'Argenio, H. Hermanns, and J.-P. Katoen. Modest: A compositional modeling formalism for hard and softly timed systems. *IEEE Transactions on Software Engineering*, 32(10):812–830, 2006. ISSN: 0098-5589.

[4] A. Hartmanns. On the analysis of stochastic timed systems, 2015.

[5] A. Hartmanns and H. Hermanns. A Modest approach to checking probabilistic timed automata. In *QEST*, pages 187–196. IEEE Computer Society, September 2009.

[6] A. Hartmanns and H. Hermanns. *The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification*, pages 593–598. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[7] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101–150, 2002.

[8] M. Z. Kwiatkowska, G. Norman, D. Parker, et al. The PRISM benchmark suite. In *QEST*, pages 203–204, 2012.

[9] C. Ruegg and M. Cuda. Math.net numerics, 2009. Available at http://numerics.mathdotnet.com/.

[10] M. Stoelinga. An introduction to probabilistic automata. *Bulletin of the EATCS*, 78(176-198):2, 2002.

[11] Wikipedia user: Cflm001. Binomial distribution, 2013.